# Writeup——郑翰浓（zhn）

http://162.105.23.109/Who_am_I?StudentID=2101212849&UID=143&date=20221126&QQ=774889315&phone=15070403317

# 目录



# Tutorial

## †签到†

网上找了个在线解密pdf的网站，解密后全选复制出来，两行字母上下各取一个就是flag

## 小北问答 · 极速版

b站视频av号有api；gStore搜一下就能找到doi；中文域名按F12就看到转义后的了；pkurunner下载apk包后用zip打开，随便找个小的配置文件看看；质数每次不一样，而且答案不能确定，只能每次临时算每次猜第一个；火狐版本网上找得到；电子游戏概论找不到，但我去年也参赛了所以有印象，知道每关是整百数字，于是硬着头皮试，把每关的金币给试出来了；mac地址查邮编的题我实在不会，所以只能祈祷这题不出现。

于是我能过关的概率 = P(质数猜对) * P(mac题不出现)

总概率几十分之一吧。事实上我确实也试了几十次

2秒之内的话写程序答题就行，题目顺序随机就识别关键字

```
from pwn import *
import re
import math
r = remote('prob01.geekgame.pku.edu.cn', 10001)
r.interactive()
r.sendline(b"143:MEQCIHAAre_UyqgIRU9_OPdSB-yausunfZTrJyMcMP-
wwb2GAiBcpb3Oa4uiaRur2feiizpLT4W5fAZfgaoy3Jo4JzKyGQ==")
r.recvline()
print(r.recvline().decode('utf8'))
print(r.recvline().decode('utf8'))
r.sendline("急急急".encode('utf8'))
print(r.recvline().decode('utf8'))
print(r.recvline().decode('utf8'))

for i in range(7):
    r.recvline().decode('utf8')
    t = r.recvline().decode('utf8')
    print(t)
    r.recvline().decode('utf8')
    if t.find('Androi') > 0:
        r.sendline(b"cn.edu.pku.pkurunner")
    elif t.find('质数') > 0:
        list = re.findall("\d+", t)
        for i in range(int(list[1]), int(list[2])):
            prime = 1
            for k in range(2, int(math.sqrt(i))):
                if i % k == 0:
                    prime = 0
                    break
            if prime == 1:
                r.sendline(str(i).encode('utf8'))
                break
    elif t.find('Fire') > 0:
        r.sendline(b"65")
    elif t.find('bili') > 0:
        r.sendline(b"418645518")
    elif t.find('Host') > 0:
        r.sendline(b"ctf.xn--4gqwbu44czhc7w9a66k.com")
    elif t.find('gStore') > 0:
        r.sendline(b"10.14778/2002974.2002976")
    elif t.find('GeekGame') > 0:
        list = re.findall("\d+", t)
        lv = int(list[1])
        c = 2000
        if lv == 4:
            c = 1100
        if lv == 5:#  这些数字是我一个个试出来的，打了井号的是确定对了的
            c = 1400
        if lv == 6:#
            c = 1700
        if lv == 7:#
            c = 2100
```

```
        if lv == 8:#
            c = 2500
        if lv == 9:#
            c = 3000
        if lv == 10:#
            c = 3400
        if lv == 11:#
            c = 3900
        if lv == 12:#
            c = 4400
        if lv == 13:
            c = 4900
        if lv == 14:#
            c = 5500
        r.sendline(str(c).encode('utf8'))
    else:
        print("???")
        r.sendline(b'123')
    print(r.recvline().decode('utf8'))
print(r.recv().decode('utf8'))
```

# Misc

## 编原译理习题课

第一题设置一个巨大的常量数组就行，因为占着静态空间的

```
from pwn import *
import re
import math
r = remote('prob04.geekgame.pku.edu.cn', 10004)
r.interactive()
r.sendline(b"143:MEQCIHAAre_UyqgIRU9_OPdSB-yausunfZTrJyMcMP-
wwb2GAiBcpb3Oa4uiaRur2feiizpLT4W5fAZfgaoy3Jo4JzKyGQ==")
print(r.recv().decode('utf8'))
r.sendline(b'1')
print(r.recv().decode('utf8'))
r.sendline(b'const int a[9999999]={0};int main(){}')
r.sendline(b'//EOF')
print(r.recv().decode('utf8'))
print(r.recv().decode('utf8'))
print(r.recv().decode('utf8'))
```

第二题用宏嵌套，生成一堆语法错误就行

```
from pwn import *
import re
import math
r = remote('prob04.geekgame.pku.edu.cn', 10004)
r.interactive()
r.sendline(b"143:MEQCIHAAre_UyqgIRU9_OPdSB-yausunfZTrJyMcMP-
wwb2GAiBcpb3Oa4uiaRur2feiizpLT4W5fAZfgaoy3Jo4JzKyGQ==")
print(r.recv().decode('utf8'))
r.sendline(b'2')
print(r.recv().decode('utf8'))
```

```
r.sendline(b'#define A ;\n#define B A+A\n#define C B+B\n#define D C+C\n#define E
D+D\n#define F E+E\n#define G F+F\n'
           +b'#define H G+G\n#define I H+H\n#define J I+I\n#define K
J+J\n#define L K+K\n#define M L+L\n#define N M+M\nint main(){N}')
r.sendline(b'//EOF')
print(r.recv().decode('utf8'))
print(r.recv().decode('utf8'))
```

第三题我不会。试过左括号用宏爆炸，不行

---

第二阶段在提示下才知道原来是利用bug，而且才知道G++有官方bug发布平台，之前一直没找到这种的。于是找到个发布时间很近的https://gcc.gnu.org/bugzilla/show_bug.cgi?id=105300

```
void operator""_x(const char *, unsigned long);
static_assert(false, "foo"_x);
```

提交直接成功

有点小后悔这么简单的办法一开始没想到，但是也没办法，这属于我的知识盲区，确实一开始不知道，也认了

## Flag Checker

反编译一下，flag1用rot13函数反向转换一下就得到了

flag2的话得到这一串脚本代码

```
function checkflag2(_0xa83ex2){
            var _0x724b=['charCodeAt','map','','split',c,'Correct','Wrong','j-
'];
            return (JSON[_0x724b[4]](_0xa83ex2[_0x724b[3]](_0x724b[2])
[_0x724b[1]](function(_0xa83ex3){
                return _0xa83ex3[_0x724b[0]](0)}))== JSON[_0x724b[4]]
([0,15,16,17,30,105,16,31,16,67,3,33,5,60,4,106,6,41,0,1,67,3,16,4,6,33,232]
['charCodeAt'](function(_0xa83ex3){
                    return (checkflag2+ _0x724b[2])[_0x724b[0]](_0xa83ex3)}))?
_0x724b[5]:_0x724b[6])}
```

虽然我看不懂，但整个好像是一个相等判定的问号表达式，像是判定flag正确性的。于是我灵机一动把这个问号表达式的错值改成了前面这一串

```
        s = "function checkflag2(_0xa83ex2){var _0x724b=
['charCodeAt','map','','split','stringify','Correct','Wrong','j-'];"
                + "return (JSON[_0x724b[4]](_0xa83ex2[_0x724b[3]](_0x724b[2])
[_0x724b[1]](function(_0xa83ex3){"
                + "return _0xa83ex3[_0x724b[0]](0)}))== JSON[_0x724b[4]]
([0,15,16,17,30,105,16,31,16,67,3,33,5,60,4,106,6,41,0,1,67,3,16,4,6,33,232]
[_0x724b[1]](function(_0xa83ex3){return (checkflag2+ _0x724b[2])[_0x724b[0]]
(_0xa83ex3)}))?_0x724b[5]:JSON[_0x724b[4]]
([0,15,16,17,30,105,16,31,16,67,3,33,5,60,4,106,6,41,0,1,67,3,16,4,6,33,232]
[_0x724b[1]](function(_0xa83ex3){return (checkflag2+ _0x724b[2])[_0x724b[0]]
(_0xa83ex3)})))}\r\n";
        try {
            scriptEngine.eval(s);
        } catch (ScriptException e) {
            e.printStackTrace();
        }
```

运行后输出

```
[102,108,97,103,123,106,97,118,97,115,99,114,105,112,116,45,111,98,102,117,115,9
9,97,116,111,114,125]
```

一看就全是正常范围ASCII字符，就是flag2了

## 智慧检测器

乱按几下就崩了，获得flag1【狗头】~~实际上是研究了好一会儿才发现一次性输入多个方向会出异常的~~

flag2的话，也研究了很久，首先想的是从flag1的bug原因上入手，发现原因在于每次移动后创建新对象的语句`NewPos = list(CurPos)`被写在了同一行字符串的for循环之外，导致对象保留的引用延迟了一个周期更新，所以判定非法移动会不及时。另外考虑第三个模式有一个特殊之处就是有传送机制，其实传送终点的更新也存在对象引用的问题，也可以卡bug让传送终点变得不一样，比如按TU。

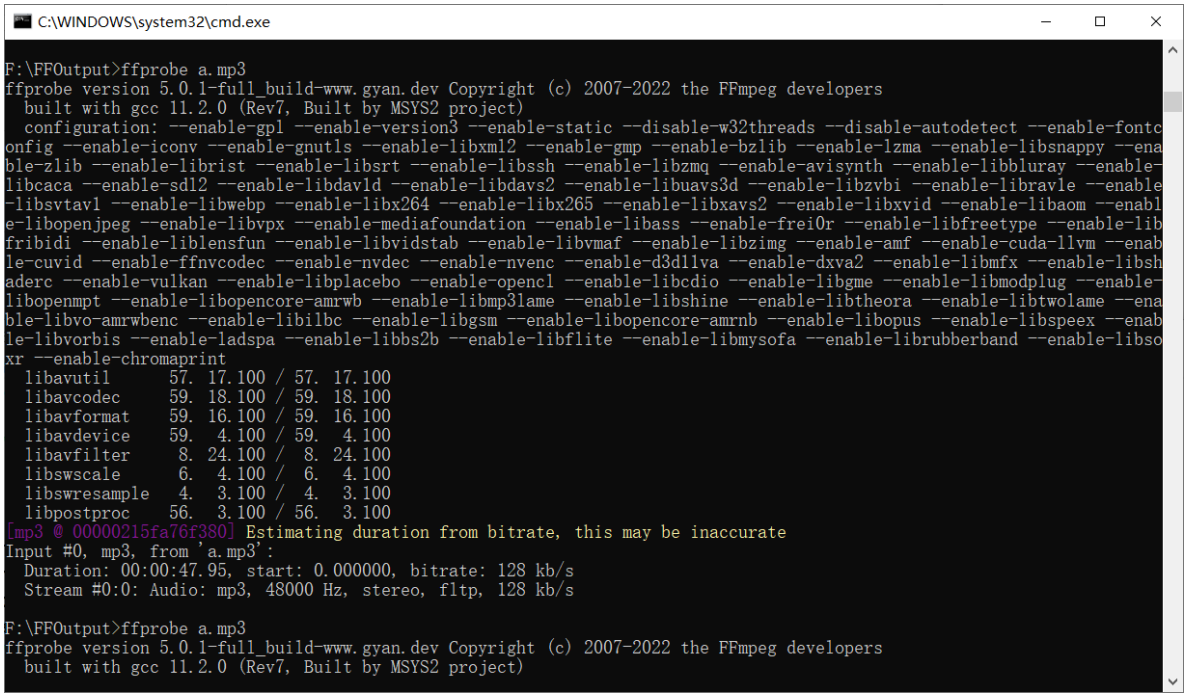
正是深入研究了程序逻辑问题，反而绕过了正解耽误了很多时间。其实就利用移动非法判定延迟就可以了，先RD去-1层看看终点位置，然后一个劲NU/SU/WU/EU，边往上走边往终点靠。核心点在于：这四种操作都只算作一步，第二步判定非法于是计数器没加。以及要注意别U过头了。。。至于T传送存在的bug有啥用，我不知道。~~大概是出题者不小心写出来的bug~~

## 我用108天录了个音

> 嘎嘎~ 机械~ 嘎嘎~

~~可惜我终究是乌鸦嘴，不是神之嘴，录了两天音还是通不过flag2~~

很巧，我正好做视频编码的。看了代码，文件长度是实打实的长度，时长是ffprobe输出的时长。我正好做视频编码的，所以这块还有些基础知识。先是录了段正常的音频试试效果，我用CoolEdit来剪辑静音时间的，它保存的mp3格式仿佛有问题：没有ID3文件头，只有帧头。我用ffprobe本地计时，有一句话引起了我的注意




”由比特率估计的时长，可能不准确“

好，那就把每个帧的比特率全给改了，看看效果

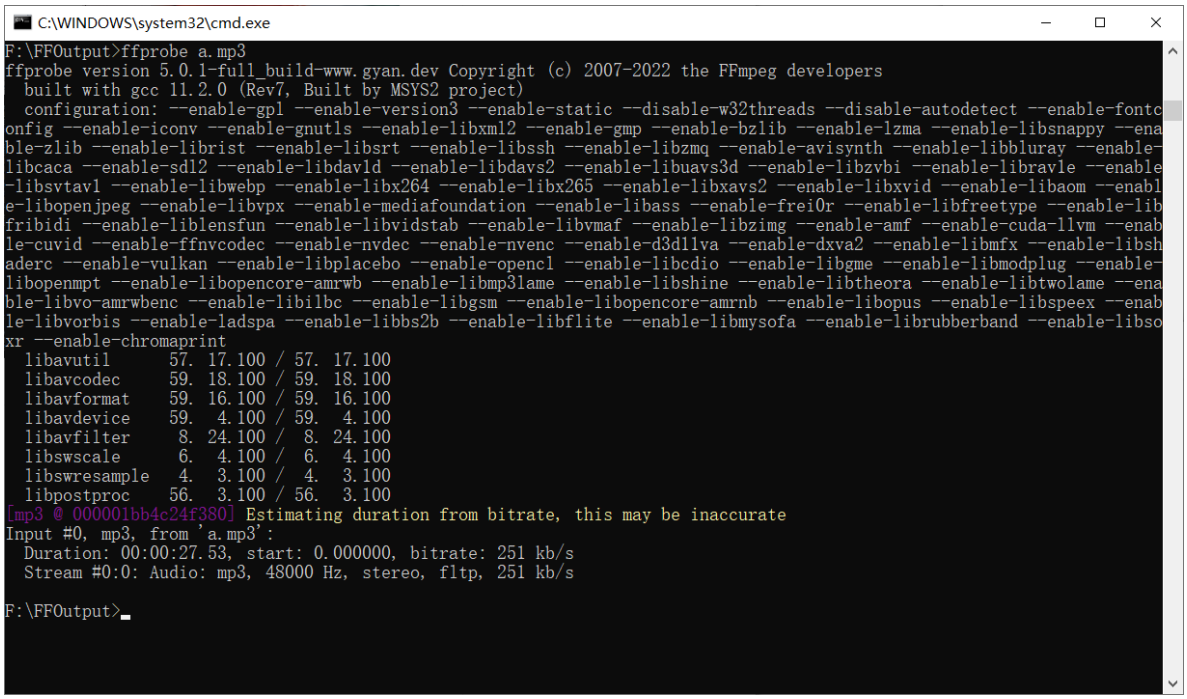
结果音频播放得鬼畜起来，开头很快结尾很快。不过ffprobe输出的时长确实变短了——实际上也变短了

后来琢磨半天，发现只要把音频头部一百来个帧头的比特率字段给改了就行，ffprobe有可能只采样了前面一些帧，这样就骗过了ffprobe了

```
import java.io.*;

public class Main2 {
    static int o[] = {255,250,149,64,126,165};
    static int a[] = {255,250,213,64,131,164};
    public static void main(String[] args)  throws Exception{
        FileInputStream fin = new FileInputStream("F:\\FFOutput\\123.mp3");
        FileOutputStream fo = new FileOutputStream("F:\\FFOutput\\a.mp3");
//      for (int i = 0 ; i < 100; i++) {
//          System.out.print(fin.read() + " ");
//      }
        int length;
        byte buf[] = new byte[9999999];
        length = fin.read(buf);
        int count = 0;
a:      for (int i = 0; i < length - 6; i++) {
            for (int k = 0; k < 4; k++) {
                if ((byte)o[k] != buf[i + k]) continue a;
            }
            count++;
            System.out.println(count);
            if (count > 100) continue a;
            for (int k = 0; k < 4; k++) {
                buf[i + k] = (byte) a[k];
            }
        }
        fo.write(buf, 0, length);
    }
```

```
}
```




为了前面不会播放得很快，我额外加了两秒的静音。flag1就大功告成

flag2放弃了，文件大小要求太苛刻，我录了一天音还下了个语音生成软件，都还是不行

试过用aac格式最低码率

`ffmpeg -i 1.mp3 -acodec aac -ar 8k -ac 1 -ab 1k 1.aac`

还是不行；也研究了aac结构看看能不能进一步优化码率，也不太行

---

第二阶段折腾了好久做出了大小符合的aac文件，但时长不知为啥，服务器的判定和我本机的ffrobe不一致就很奇怪。（难道正解真的不是aac而是ogg吗）

## 小Z的服务器

不会

# Web

## 企鹅文档

试了各种转存的方式，都不允许。按F12研究网络包，因为太多了也没啥收获。后来干脆保存了整个网页到本地，一个叫opendoc的文件引起了我的注意，然后就不用解释了。~~其实中间还有个小意外：opendoc里只有前60行，https://geekgame.pku.edu.cn/service/template/prob_kAiQcWHobs，网址不全说模板不存在，怎么也找不到后面的部分。于是我研究了好久才在F12抓包里的sheet里找到了60行之后的部分~~

https://geekgame.pku.edu.cn/service/template/prob_kAiQcWHobsBzRJEs_next

第二关的话我也研究了蛮久，一开始没看到提示，以为是要通过某个api之类的强行找出来，失败了。后来看到提示后自己建了个文档涂黑一些格子来研究，但似乎和那个文档不太一样。。。后来直接在har文件里搜索`Below is your flag`什么的，定位到了之后，下面那一些数字我看了半个多小时才知道不是连续的，那么写程序打印出来就得到了

```
for i in range(2621):
    if i % 11 == 0:
        print()
    if i in a:
        print("X", end='')
    else:
        print(" ", end='')
```

## 给钱不要!

~~这不是燕园大厦贴在厕所小便池上方的标语吗~~

我能做到safe级别的跳转，用类似这样的网址。~~其实是我第一次知道ip还能用一段数字表示~~

```
\/2724796269\1
```

然后就没然后了，虽然成功访问到了我的服务器，但这做不到very_safe，我也没XSS基础

---

第二阶段根据提示以及根据源码注释中的

```
  // * If the user typed four distinct components, this is an IP for sure.
  // * If the user typed two or three components, this is almost certainly a
  //   query, especially for two components (as in "13.5/7.25"), but we'll
  //   allow navigation for an explicit scheme or trailing slash below.
  // * If the user typed one component, this is likely a query, but could be
  //   a non-dotted-quad version of an IP address.
```

怪不得之前用一个数字总识别为unknown了。然后试了下两个数字。~~其实这也是我第一次知道ip也能用两段数字表示~~

```
\/162.6887277\1
```

成功转到我的服务器上而且是very_safe的，拿到了flag1

（我发现162.6888888也在北大网段内，是不是可以考虑搞个靓号？）

后面因为不会XSS，就不做了

## 私有笔记

~~此即，智慧之殿堂？~~这题我不会，智慧之殿堂我不配

---

第二阶段，搜索MediaWiki bug report system找到了平台，定位到 T297322 CVE-2021-44857, CVE-2021-44858: Unauthorized users can undo edits on any protected page and view contents of private wikis using mcrundo (wikimedia.org)然后用`title=首页&action=mcrundo&undo=1&undoafter=2`成功拿下flag1

flag2定位到 T257062 Lilypond seemingly not subject to restrictions (CVE-2020-29007) (wikimedia.org)，但因为我知识盲点太多，我完全不是做安全这块的，所以没办法只能直接放弃

## 企业级理解

不会

## 这也能卷

第一个flag简单啦，把网一断发现还能判断提交是否正确，所以一看就知道是本地代码判断的，一看js代码一大坨没法读，我也不懂js也不会调试，于是就在调试面板乱点暂停、继续按钮，然后一不小心就发现flag显示出来了（

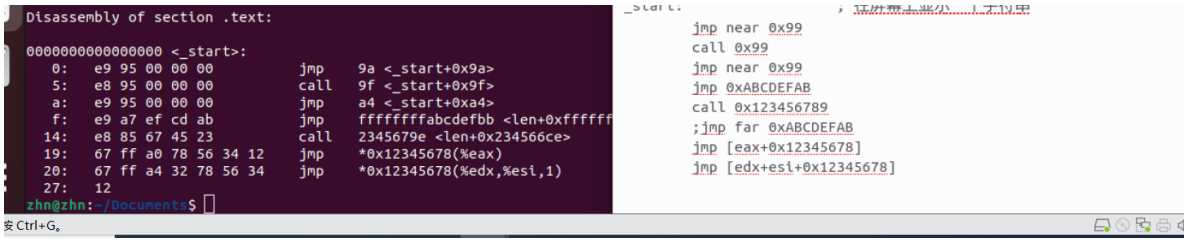
后面两个不会

# Binary

## 简单题

确实简单，大概是Binary分类里最简单的一个了。一开始我下了个nasm，用mov来做，然后发现Linux下代码段不可写，数据段和其他段又不可执行，否则会报段错误，没办法。~~其实后来才知道在mmap里代码段也是可写的。~~然后睡了一觉后想到用jmp，只要我瞎跳把指令调断，不就成功了吗

研究了jmp的指令结构，使用基址加变址时最长占8字节，其中开头的67 ff a4最好别变，后面随便改，指令都被认为是8字节的。所以后面5字节可以用来藏指令，需要把正式代码的机器码都控制在5字节之内才好




试过了好多方案，试了调用中断让系统执行命令行，试了int 0x80调用系统功能，结果都不行

https://blog.csdn.net/zmhDD/article/details/109125102

(51条消息) linux汇编读取文件操作系统,【Linux x86汇编踩坑】文件读写（一）读取文件并输出... weixin_39526706的博客-CSDN博客

~~后来才知道是因为调用系统功能的话得调用64位，也就是用syscall调用才行。~~然后开始试着写代码用fopen、fread、printf来弄，因为load.c里正好本来就有这些函数。结果这个函数转成汇编后，直接在load.c里写好了再编译运行是没问题，但用编好的load来输入运行就报段错误，定位问题在call这些函数时。试过通过栈帧把返回地址也就是原始rip找出来然后计算偏移量来间接寻址调这些函数，还是不行。

一筹莫展时，偶然发现系统调用该用64位，一切都解决了。

转成全jmp的代码的话，就用真伪jmp交替就行。真jmp跳到下一条伪jmp中的真指令，这个指令执行完了下一句就是下一个真jmp了，如此循环。指令不够长的部分随便填充掉，只要真指令顶着下一个真jmp之前放，然后上一个真jmp写好跳转到的位置，就可以了。以下是生成这些伪jmp的代码

```
public static void main(String[] args) throws IOException {
    FileOutputStream fo = new FileOutputStream("D:\\a.bin");
    String code = "48 31 c0      \r\n" +
        "b8 10 02 00 00\r\n" +
        "48 29 c4      ";  // 一个例子
    String num[] = code.split("\r\n");
    byte[] buf = new byte[num.length*10];
    for (int i = 0; i < num.length; i++) {
        num[i] = num[i].replaceAll("\\s*", "");
        int pad = 0;
        while (num[i].length() < 10) {
```

```
            pad++;
            num[i] = "00" + num[i];
        }
        num[i] = "eb0" + (3 + pad) + "67ffa4" + num[i];
        for (int j = 0; j < 10 && j*2 < num[i].length(); j++) {
            int kk = Integer.parseInt(num[i].substring(j*2, j*2+2), 16);
            buf[i*10+j] = (byte) kk;
        }
    }
    String enc = Base64.getEncoder().encodeToString(buf);
    System.out.println(enc);
    fo.write(buf);
}
```

其中有个小难点是字符串"./flag.txt"放哪，因为有点长，没法藏在jmp里面了。于是只能通过立即数存储到内存中。说到立即数，得吐槽一句为了把指令都藏在jmp里，好多看起来很简单的指令都得改，包括所有的变址寻址，甚至对64位寄存器赋值立即数，也得拆成好几条语句来写，搞了我好久。。。

汇编代码的话因为一开始我是尝试用fopen那些的，所以用那个开始写的，后来改了syscall了懒得重写，就接着写了，所以代码写的很乱。。。这是原始C代码

```
char c[] = {'%','s','\0', 'r','w','\0',
      '/','f','l','a','g','.','t','x','t','\0'};
char buf[512];
FILE *f = fopen(c+6, c+3);
fread(buf, 1, 512, f);
printf(c, buf);
```

这是转换的汇编然后用syscall代替掉了读写文件函数。注释里是原始C代码以及汇编代码，我改了之后每个指令的机器码都在5字节之内，可以直接处理了

```
  //char c[] = {'%','s','\0', 'r','w','\0',
  //    '/','f','l','a','g','.','t','x','t','\0'};
  //char buf[512];
  asm(
    "nop\n"
    "pop %rcx\n"
    "push %rcx\n"

  // sub $0x210,%rsp
    "xor %rax, %rax\n"
    "mov $0x210, %eax\n"
    "sub %rax, %rsp\n"

  // ??
  //   "mov %fs:0x28,%rax\n"
  //   "mov %rax,-0x8(%rbp)\n"

  // xor %eax,%eax  &  movabs $0x662f007772007325,%rax
  // &  movabs $0x7478742e67616c,%rdx  &  mov %rax,-0x210(%rbp)
  // &  mov %rax,-0x210(%rbp)
     "mov %rbp, %rbx\n"
     "xor %rax, %rax\n"
     "mov $0x210, %eax\n"
     "sub %rax, %rbx\n"
     "mov $0x72007325, %eax\n"
```

```
    "mov %eax, (%rbx)\n"
    "mov $0x4, %eax\n"
    "add %rax, %rbx\n"
    "mov $0x662f0077, %eax\n"
    "mov %eax, (%rbx)\n"
    "mov $0x4, %eax\n"
    "add %rax, %rbx\n"
    "mov $0x2e67616c, %eax\n"
    "mov %eax, (%rbx)\n"
    "mov $0x4, %eax\n"
    "add %rax, %rbx\n"
    "mov $0x00747874, %eax\n"
    "mov %eax, (%rbx)\n"
);


//FILE *f = fopen(c+6, c+3);
asm(
// lea -0x210(%rbp),%rdx  &  add $0x3,%rdx  &  mov %rdx,%rsi
  "mov %rbp, %rsi\n"
  "xor %rax, %rax\n"
  "mov $0x20D, %eax\n"
  "sub %rax, %rsi\n"

//lea -0x210(%rbp),%rax  &  add $0x6,%rax  &  mov %rax,%rdi
  "mov %rbp, %rdi\n"
  "xor %rax, %rax\n"
  "mov $0x20a, %eax\n"
  "sub %rax, %rdi\n"

  "mov $0,%esi\n"
  "mov $2,%eax\n"
  "syscall\n"
  /*
  "mov $0, %edx\n"
  "mov %rdi,%rsi\n"
  "mov $0,%edi\n"
  "mov $2,%eax\n"
  "syscall\n"
  */
);


//fread(buf, 1, 512, f);
asm(
// mov %rax,-0x228(%rbp)  &  mov -0x228(%rbp),%rdx  &  mov %rdx,%rcx
  "mov %rax, %rcx\n"

//lea -0x210(%rbp),%rax  &  mov %rax,%rdi
  "mov %rbp, %rdi\n"
  "xor %rax, %rax\n"
  "mov $0x200, %eax\n"
  "sub %rax, %rdi\n"

  "mov $0x200,%edx\n"
  "mov $0x1,%esi\n"
 // "call 0x10c0\n"
  "mov $0x100, %edx\n"
```

```
    "mov %rdi,%rsi\n"
    "mov %ecx,%edi\n"
    "mov $0,%eax\n"
    "syscall\n"
    "mov %eax, %edx\n"
  );

  //printf(c, buf);
  asm(
  // lea -0x210(%rbp),%rdx  &  mov %rdx,%rsi
 //   "mov %rbp, %rsi\n"
//   "xor %rax, %rax\n"
 //   "mov $0x200, %eax\n"
 //   "sub %rax, %rsi\n"
 //           "mov $0,%ebx\n"
  //  "mov $1, %eax\n"
  //  "int $0x80\n"
  //  "nop\n"
  // lea -0x220(%rbp),%rax  &  mov %rax,%rdi
  //  "mov %rbp, %rdi\n"
  //  "xor %rax, %rax\n"
  //  "mov $0x210, %eax\n"
  //  "sub %rax, %rdi\n"

  //  "xor %eax, %eax\n"
   // "call 0x1100\n"
  //  "mov $0x100, %edx\n"
   // "mov %rdi,%rsi\n"
    "mov $1,%edi\n"
    "mov $1,%eax\n"
    "syscall\n"


    "mov $0,%ebx\n"
    "mov $1, %eax\n"
    "int $0x80\n"
    "nop"
  );
```

后面的生成结果啥的就不细说了

## TTOWRSS & 次世代立方计算机 & 混淆器 & 编原译理习题课 · 实验班

全都不会。写标题是为了万一有人在我这个Writeup里搜解法，方便搜到并确认我没写这些题

TTOWRSS的话我找到了一个奇怪数组

```
19 9 1 25 30
33 5 20 11 7
14 4 23 3 18
8 28 32 38 22
0 21 16 6 12
15 37 13 36 27
31 26 29 10 17
35 24 39 34 2
```

像是一个0~39到0~39的一一映射，但是还是没能看懂代码。。。~~（靠%rip来间接寻址好像是我第一次见，我记得我本科那会儿学8086的时候不是这样的呀，看不明白了）~~

# Algorithm

## 381654729

先确定flag长度，然后一位一位暴力试，合法就往更深层试，试出来为止。因为越到深层，符合条件的比例越少，随着剪枝所以其实复杂度并不高，几分钟就跑出来了。~~以下代码是我写了好久才这么精简的~~

```
flag="flag{\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0}".encode()
ff = int.from_bytes(flag,"big")

def test(nn, level):
    for a in range(1, 128):
        if level == 18:
            print(a)
        nnn = (nn | (a << (level << 3)))
        num = nnn ^ 25114135107867476148236799399243749964567773476153369860002
        num_len=len(hex(num))-2
        right = 1
        for i in range(37-level*2, 39-level*2):
            if (num>>((num_len-i)*4))%i>0:
                right = 0
                break
        if right == 1:
            if (level <= 0):
                print("!!!", chr(a))
                return 1
            if test(nnn, level - 1) == 1:
                print(chr(a))
                return 1
    return 0

flag="flag{\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0}".encode()
ff = int.from_bytes(flag,"big")
test(ff, 18)
```

## 乱码还原

一开始没看python代码，乱码恢复 (mytju.com)这个网站让我知道了是utf-8和shift_jis的乱码。于是用java写了半天代码，还有bug，后来还是我手动一个一个字符调正确的flag1。~~初始的java代码写太乱了而且有bug，就不放了~~

flag2数据量太大没法这么搞，于是python重写了代码，很成功。实际上就是个暴力测试，带两个字符的lookahead，这样就已经足够唯一确定解了，然后还原正确了的部分加个截断，免得长串处理耗时

```
dict = [
    '滅', '苦', '婆', '娑', '耶', '陀', '跋', '多', '漫', '都', '殿', '悉', '夜',
'燦', '帝', '吉',
    '利', '阿', '無', '南', '那', '怛', '喝', '羯', '勝', '摩', '伽', '謹', '波',
'者', '穆', '僧',
    '室', '藝', '尼', '瑟', '地', '彌', '菩', '提', '蘇', '醯', '盧', '呼', '舍',
'佛', '參', '沙',
```

```
    '伊', '隸', '麼', '遮', '闍', '度', '蒙', '孕', '薩', '夷', '迦', '他', '姪',
'豆', '特', '逝',
    '朋', '輪', '楞', '栗', '寫', '數', '曳', '諦', '羅', '曰', '咒', '即', '密',
'若', '般', '故',
    '不', '實', '真', '訶', '切', '一', '除', '能', '等', '是', '上', '明', '大',
'神', '知', '三',
    '藐', '耨', '得', '依', '諸', '世', '槃', '涅', '竟', '究', '想', '夢', '倒',
'顛', '離', '遠',
    '怖', '恐', '有', '礙', '心', '所', '以', '亦', '智', '道', '。', '集', '盡',
'死', '老', '至',
    '冥', '奢', '梵', '吶', '俱', '哆', '怯', '諳', '罰', '侄', '缽', '皤']
file = open("C:\\Users\\ZHN\\Downloads\\flag2.enc","rb")
f = file.read().decode("utf-8")
s = '佛曰：'
pos = 0
pos1 = 0
pos2 = 0
cnt = 0
while 1:
    cnt += 1
    if cnt % 10 == 0:
        while 1:
            ss = s[pos1+1:]
            ss = ss.encode("utf-8").decode("shift_jis",errors="ignore")
            ok = 0
            for i in range(5):
                p2 = pos2 - i
                if len(ss) <= p2 or p2 < 0:
                    break
                if ss[p2] == f[pos] and ss[p2-1] == f[pos-1] and ss[p2-2] ==
f[pos-2] and ss[p2-3] == f[pos-3]:
                    ok = 1
                    pos2 = p2
                    pos1 = pos1 + 1
                    break
            if ok == 1:
                continue
            else:
                break

    if cnt % 20 == 0:
        print(pos1)
        print(s)
    if cnt % 1000 == 0:
        print(pos1)
        print(s)
        fw = open("C:\\Users\\ZHN\\Downloads\\flagb"+str(cnt)+".txt","w")
        fw.write(s)
    if len(s) >= len(f):
        print(s)
        break
    common = -1
    maxi = '?'
    ok = 1
    prob = []
    for i in dict:
        ss = s[pos1:] + i
        ss = ss.encode("utf-8").decode("shift_jis",errors="ignore")
```

```
            ss = ss[pos2:]
            c = 0
            for k in range(len(ss)):
                if pos + k < len(f) and ss[k] == f[pos+k]:
                    c += 1
                else:
                    break
            if c > common:
                ok = 1
                maxi = i
                common = c
                prob = [i]
            elif c == common:
                ok = 0
                prob += i
        if ok == 1:
            s += maxi
            pos += 1
            pos2 += 1
            continue

        common = -1
        maxi = '?'
        maxj = '?'
        for i in prob:
            for j in dict:
                ss = s[pos1:] + i + j
                ss = ss.encode("utf-8").decode("shift_jis",errors="ignore")
                ss = ss[pos2:]
                c = 0
                for k in range(len(ss)):
                    if pos + k < len(f) and ss[k] == f[pos+k]:
                        c += 1
                    else:
                        break
                if c > common:
                    maxi = i
                    if maxj != '!':
                        maxj = j
                    common = c
                elif c == common:
                    if maxi == i:
                        maxj = '!';
        if maxj == '!':
            s += maxi
            pos += 1
            pos2 += 1
        else:
            s += maxi + maxj
            pos += 2
            pos2 += 2
```

不过我这个程序还是跑的不算快，跑了1/4左右就开始明显变慢，还原1kB需要好几分钟，也不知为啥这个程序会越跑越慢，也许是python的问题比如说垃圾回收机制？~~主要是写到这里离第一阶段结束只有几个小时了我怕来不及~~

然后把输出的东西放进网站里解，解不开；然后用程序里的Decrypt函数解

(52条消息) No module named “Crypto” 解决方案_iO快到碗里来的博客-CSDN博客

也报错了；于是试了下那个串重新模拟乱码，然后输出和flag2.enc完全一致。所以唯一的可能就是乱码过程有损，复原答案不唯一

找所有答案不唯一的地方

```
for i in range(len(s)):
    if i % 100 == 0:
        print(i)
    for c in dict:
        if c == s[i]:
            continue
        s1 = s[:i]+c+s[i+1:]
        s1 = s1.encode("utf-8").decode("shift_jis",errors="ignore")
        if s1 == t:
            print(i, c, s[i])
```

找出来这么一堆

```
pos = [299,381,754,819,998,1670,1670,2418,2518,2817,
       4470,5152,5527,6055,6687,7547,7991,8320,9934,10113,
       11583,12075,12479,12620,12926,13454,15128,15735,16028,16041,
       17321,18187,18320,18403,18764,18882,19224,19242,19260,19920,
       19970,20176,20179,20207,20236,20236,20269,22196,22503,22641,
       23483,24224,24451,24520,25330,25917,26555,27433,27468]
test = ['蘇','蘇','蘇','蘇','老','蘇','老','蘇','蘇','蘇',
        '蘇','蘇','蘇','蘇','蘇','老','蘇','蘇','蘇','蘇',
        '蘇','蘇','蘇','蘇','老','蘇','蘇','蘇','老','蘇',
        '蘇','老','蘇','蘇','蘇','蘇','蘇','蘇','蘇','蘇',
        '蘇','蘇','蘇','蘇','蘇','老','蘇','蘇','蘇','蘇',
        '蘇','蘇','蘇','蘇','蘇','蘇','蘇','蘇','老']
```

共59个，大约要试2的59次方次这可如何是好呢？好在我非常机智，发现`decode('utf-16le',error)`有第二个选项`error`，可以取`'ignore'`。果然呢，ignore之后解出来了，中间有些片段很乱，其余部分只有A-Z2-7，像是base32。~~其实我不懂AES加密的原理，以下都是在瞎jb试。~~首先我试了把一一两个词替换成另一种，发现基本上该不能解的还是不能解。然后我偶然想到一招：每次在每个位置随机选择是否把答案不唯一的地方换成另一种答案，随机概率是1/2。

```
for i in range(0,100):
    tt = "佛曰：逝罰世僧悉罰離侄伊耨皤吉恐瑟......." # 此处省略上万字
    for j in range(0,59):
        if random.randint(0,1) == 1:
            tt = tt[:pos[j]] + test[j] + tt[pos[j]+1:]
    try:
        Decrypt(tt)
    except UnicodeDecodeError as e:
        i = i
    except ValueError as e1:
        continue
    print(Decrypt(tt, 'ignore'))
```

这样一来，每次生成的错误的地方就不一样了，这样东拼西凑把每次的有效片段拼在一起，能把整个都凑起来了，完美！实际上最后有几个字符总是没法完美复原的，不过在我后面发现无伤大雅

原先代码是编码了10层每次用b16encode, b32encode, b64encode, b85encode, a85encode中的随机一个的，那么就decode还原吧，每层猜用的是哪种编码。偶尔报错说长度不对，那么大概就是最后几个字符没完美复原的原因，那我就强行补全它们了比如全补0

依次b32decode，a85decode，b85decode，b16decode，a85decode，b64decode，a85decode，b64decode，a85decode，b64decode，然后得到一串奇怪的东西

```
b'\xe5\xb0\x86\xe9\x9c\x80\xe8\xa6\x81\xe6\x89\x93\xe7\xa0\x81\xe7\x9a\x84\xe6\x
96\x87\xe5\xad\x97\xe8\xbe\x93\xe5\x85\xa5\xe5\x9c\xa8\xe4\xb8\x8a\xe9\x9d\xa2\x
e7\x9a\x84\xe6\x96\x87\xe6\x9c\xac\xe6\xa1\x86\xe9\x87\x8c\xef\xbc\x8c\xe7\x82\x
b9\xe5\x87\xbb\xe3\x80\x8e\xe5\x90\xac\xe4\xbd\x9b\xe8\xaf\xb4\xe5\xae\x87\xe5\x
ae\x99\xe7\x9a\x84\xe7\x9c\x9f\xe8\xb0\x9b\xe3\x80\x8f\xe6\x8c\x89\xe9\x92\xae\x
ef\xbc\x8c\xe5\xb0\xb1\xe8\x83\xbd\xe5\x9c\xa8\xe4\xb8\x8b\xe9\x9d\xa2\xe5\xbe\x
97\xe5\x88\xb0\xe6\x89\x93\xe7\xa0\x81\xe5\x90\x8e\xe7\x9a\x84\xe6\x96\x87\xe5\x
ad\x97\xe3\x80\x82\r\nflag{AES_1s_b10ck_cipher}\r\n\xe5\xb0\x86\xe9\x9c\x80\xe8\
xa6\x81\xe8\xa7\xa3\xe7\xa0\x81\xe7\x9a\x84\xe6\x96\x87\xe5\xad\x97\xe8\xbe\x93\
xe5\x85\xa5\xe5\x9c\xa8\xe4\xb8\x8b\xe9\x9d\xa2\xe7\x9a\x84\xe6\x96\x87\xe6\x9c\
xac\xe6\xa1\x86\xe9\x87\x8c\xef\xbc\x8c\xe8\xae\xb0\xe5\xbe\x97\xe5\xb8\xa6\xe4\
xb8\x8a\xe3\x80\x8e\xe4\xbd\x9b\xe6\x9b\xb0\xef\xbc\x9a\xe3\x80\x8f\xe6\x88\x96\
xe3\x80\x8e\xe5\xa6\x82\xe6\x98\xaf\xe6\x88\x91\xe9\x97\xbb\xef\xbc\x9a\xe3\x80\
x8f\xe7\x9a\x84\xe6\x96\x87\xe5\xad\x97\xef\xbc\x8c\xe7\x82\xb9\xe5\x87\xbb\xe3\
x80\x8e\xe5\x8f\x82\xe6\x82\x9f\xe4\xbd\x9b\xe6\x89\x80\xe8\xa8\x80\xe7\x9a\x84\
xe7\x9c\x9f\xe6\x84\x8f\xe3\x80\x8f\xe6\x8c\x89\xe9\x92\xae\xef\xbc\x8c\xe5\xb0\
xb1\xe8\x83\xbd\xe5\x9c\xa8\xe4\xb8\x8a\xe9\x9d\xa2\xe7\x9a\x84\xe6\x96\x87\xe6\
x9c\xac\xe6\xa1\x86\xe9\x87\x8c\xe5\xbe\x97\xe5\x88\xb0\xe8\xa7\xa3\xe7\xa0\x81\
xe5\x90\x8e\xe7\x9a\x84\xe6\x96\x87\xe5\xad\x97\xe3\x80\x82
```

本以为搞错了什么，怎么看起来全乱了。结果研究了五分钟发现flag就在里面了

虽然这题折腾了好久，不过一直猜编码的过程确实非常好玩，就像游戏闯关一样的成就感。出题人牛逼

## 奇怪的加密

线索是单字母单词主要就只有a，而且这个加密是循环的，每一种字母一定在小于26次之内回到自己到自己的映射。那么就猜a->a的循环次数，发现22是个可能的解，而11的话会出现很多奇怪的单字母单词，看起来不太正常，那就是22了。于是找在在模22余1位置出现的单字母单词发现只有y，那么确定A->Y，如此循环找下去找出如下映射表

```
letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key=    'YNK_F__AW_QP_TZCH_EG__XOBS'
# G->?->I
# F->?->L
```

为了能正常解码呢，就把这些格子都猜上吧。以下标了井号的几行被我改成解密逻辑了

```
import sys, random

letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key=    'YNKIFJDAWLQPMTZCHREGUVXOBS'

key=key.strip().upper()
print(key)
if ''.join(sorted(key))!=letters:
    print("Invalid key")

keymap={letters[i]:key[i] for i in range(26)}
```

```
current_key={letters[i]:letters[i] for i in range(26)}
plain_text='Cinqwmzewtxs kn f kiepagkuf umpd op hsoert trsjbo lxmlurzyrzmke
enpariq dtseeimrw areslyy, '

cipher_text=""
for i in plain_text:
    if i not in letters and i not in letters.lower():
        cipher_text=cipher_text+i
    else:
        current_key1={current_key[i]:i for i in current_key} #
        if i in letters:
            cipher_text=cipher_text+current_key1[i] #
        else:
            cipher_text=cipher_text+current_key1[i.upper()].lower() #
        current_key={i:keymap[current_key[i]] for i in current_key}

print(cipher_text)
```

输出为

```
Cdyptmgdaphy is a technique umey to enabre srcqde comjuricrtmon betwren
oiffedmrt prdties,
```

这已经能猜到了吧，然后稍微调一下映射表

```
letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key=    'YNKDFMRAWUQPLTZCHIEGJVXOBS'
```

完美了，解码全文后这句话很显眼

```
The flag is foxtrot lima alpha golf left bracket foxtrot romeo echo nine uniform
echo november charlie yankee underscore four november alpha lima yankee five
india sierra underscore one sierra underscore uniform sierra echo foxtrot
uniform lima right bracket.
```

根据flag范围提示就可以猜出来了，不解释了

flag2，第一阶段被那个提示“此部分和 Flag 1 没有关联，也即不需要先解出 Flag 1”迷惑了，以为是说第二个不是那个字母替换的密码算法，搞得我无从下手于是放弃了

---

第二阶段才知道第二个文本的解码后范围是0-9a-f，而且也是一开始那个算法。那就只能猜a-f的自映射循环周期一样了呗，不一样的话还真不好办呢。

```
for i in range(len(c)):
    pos = i % 26
    if pos == 0:
        print()
    print(c[i],end='')
```

这每一行开头都是a-f，改成13的话就不是了。说明假设成功，循环周期是26。按此猜想，每一列的字母应该是有限的集合，应该都只有7个字母。统计一下看

```
st = [[] for i in range(26)]
for i in range(len(c)):
    pos = i % 26
    if c[i] not in st[pos]:
        st[pos] += c[i]
print(st)
```

```
[['b', 'e', 'd', 'a', 'f', 'c'],
 ['p', 'h', 'j', 'l', 'v', 'd'],
 ['y', 'g', 'z', 'j', 'b', 'e'],
 ['i', 'd', 'p', 'z', 'u', 'w'],
 ['f', 'u', 'j', 'r', 'g', 'a'],
 ['i', 'x', 'v', 'z', 'l', 'a'],
 ['u', 'n', 'r', 'e', 'v', 'b'],
 ['e', 'd', 'a', 'p', 'k', 'x'],
 ['v', 'm', 'n', 'j', 'p', 'g'],
 ['k', 's', 'i', 'z', 'g', 'e'],
 ['i', 'u', 'm', 'p', 'r', 't'],
 ['g', 'x', 'r', 's', 'a', 'o'],
 ['n', 'q', 't', 'v', 'x', 'i'],
 ['n', 'e', 'k', 'c', 'o', 'r'],
 ['x', 'm', 'k', 'h', 'p', 'q'],
 ['c', 's', 'y', 'm', 'g', 'n'],
 ['k', 'w', 't', 's', 'h', 'i'],
 ['m', 'f', 'o', 't', 'y', 'r'],
 ['q', 'x', 's', 'l', 'o', 'w'],
 ['n', 't', 'q', 'f', 'c', 'b'],
 ['k', 'c', 'l', 'o', 'd', 'h'],
 ['h', 'q', 'm', 'b', 'j', 'y'],
 ['z', 'w', 'y', 's', 'd', 'c'],
 ['u', 't', 'f', 'w', 'j', 'h'],
 ['z', 'o', 'l', 'y', 'a', 'f'],
 ['v', 'q', 'u', 'w', 'b', 'l']]
```

没错。那么随便找几个a，看下一行都出现了的字母，就是a映射到的字母。如此找26个字母的映射，因为工作量也不算大所以直接手动操作了懒得写程序。得出这个映射表

```
letters='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key=    'VDHJPLIYRZMBSKQGCXTOAEFNWU'
```

解密后每行为32个0-9a-f字符，接下来怎么办呢？试了base16解码是乱码，那么猜这些全是md5，网上找了个md5解码查询，前几行对上了没问题，那么开始写代码暴力解，穷举所有两个字符组合的md5

```
mdd = cipher_text.split('\n')
import hashlib
tab1 = [['',''] for i in range(128)]
for i in range(128):
    md5hash = hashlib.md5(bytes(chr(i), "ascii"))
    tab1[i][0] = md5hash.hexdigest()
    tab1[i][1] = chr(i)

tab2 = [[['',''] for i in range(128)] for j in range(128)]
for i in range(128):
    for j in range(128):
        md5hash = hashlib.md5(bytes(chr(i)+chr(j), "ascii"))
```

```
        tab2[i][j][0] = md5hash.hexdigest()
        tab2[i][j][1] = chr(i)+chr(j)
for m in mdd:
    ok = 0
    for i in range(128):
        if tab1[i][0] == m:
            print(tab1[i][1])
            ok=1
    for i in range(128):
        for j in range(128):
            if tab2[i][j][0] == m:
                print(tab2[i][j][1])
                ok=1
    if ok == 0:
        print("???")
```

一会儿就在输出里找到了有意思的东西了

```
......
???
ag
{
m
d5
_
1
s_
re
4
1l
y_
1n
5e
cu
r3
}
......
```

## 扫雷 II

flag1我还真写了个Go程序来猜随机种子。随机种子的精度是毫秒，猜5秒范围内的足够了，玩死一次然后用雷的位置就知道下一局的雷位置了，简单得很

```
func main() {
    m := [16]int{}
    t0 := time.Now().UnixMilli() - 5000
    println(t0)
    for i := 0; i < 16; i++ {
        fmt.Scanf("%d", &m[i])
    }
a:
    for i := t0; i < t0+5000; i++ {
        rand.Seed(i)
        bd := genBoard1()
        for j := 15; j < 16; j++ {
            for k := 0; k < 16; k++ {
```

```
                if nearCount(bd, j, k) != m[k] {
                    continue a
                }
            }
        }
        println(i)
        bd = genBoard1()
        for j := 0; j < 16; j++ {
            for k := 0; k < 16; k++ {
                print(nearCount(bd, j, k))
                print(" ")
            }
            println()
        }
    }
}
```

其中nearCount被我改了一下写成了直接返回棋盘每个格子是（1）不是（0）雷了。具体就不展示了

至于flag2和flag3呢，只能说出题人白给了吧。网页版只要没死之前按个Reset然后随便开一个难度，那么就会把之前扫出来了的部分全都显示出来，而且可以一直这么重开直到能继续扫下去为止，重开几十次总能扫完（当然，更优解还是先用flag1的方法点出一大片区域来，然后重开其他难度扫其余区域，省时间），正如某句话叫做

> 生而无赖，逆风重开

只要还没输，就能重来

于是flag2、3轻而易举拿下了。~~虽然感觉这好像不是出题者的本意，但利用bug也是规则允许的一部分不是吗，应该不算犯规吧~~

## 方程组

不就是个线性代数问题嘛，第一题矩阵求逆

```
primes=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239,
241, 251, 257, 263, 269, 271]
key1 = [16404, 16416, 16512, 16515, 16557, 16791, 16844, 16394, 15927, 15942,
15896, 15433, 15469, 15553, 15547, 15507, 15615, 15548, 15557, 15677, 15802,
15770, 15914, 15957, 16049, 16163]
key2 = [19106.6119577929, 19098.1846041713, 19124.6925013201, 19072.8591005901,
19063.3797914261, 19254.8741381550, 19410.9493230296, 18896.7331405884,
19021.3167024024, 18924.6509997019, 18853.3351082021, 18957.2296714145,
18926.7035797566, 18831.7182995672, 18768.8192204100, 18668.7452791590,
18645.9207293335, 18711.1447224940]
key3 =
25800.359843622375482317741765092423108740749704076506674391637220601256480076793833725266596491145653469234638681214279142266384627498702292519864562549230222347690184575651985867669548991937988156542
```

```
import numpy as np
p = primes[:26]
q = [p[-i:]+p[:-i] for i in range(26)]
qq = np.array(q)
qq = np.sqrt(qq)
flag1 = np.matmul(np.linalg.inv(qq), key1)
flag1 = np.round(flag1)
print(flag1)
for c in flag1:
    print(chr(int(c)), end='')
```

```
[102. 108.  97. 103. 123. 103.  48.  48. 100.  95.  49. 105. 110. 101.
  97. 114.  95. 101. 113. 117.  97. 116. 105. 111. 110. 125.]
flag{g00d_1inear_equation}
```

第二题信息量小于方程数，但是可以利用ASCII字符范围来穷举这些缺少的部分来猜答案。一开始不知道`getcontext().prec=15`只能对Decimal设置，结果搞得计算精度似乎是有问题，输出一堆奇怪解。首先已知开头是'flag{'结尾是'}'，写如下代码

```
import os
from decimal import *
getcontext().prec=15
p = primes[:28]
dict = [i for i in range(48,58)]+[i for i in range(64,91)]+[95]+[i for i in
range(97,123)]
q = [p[-i:]+p[:-i] for i in range(18)]
q0 = [q[i][:5]+q[i][-5:] for i in range(len(q))]
k = np.sqrt([q[i][5:-5] for i in range(len(q))])
ki = np.linalg.inv(k)
for m in dict:
    print(m)
    for n in dict:
        for s in dict:
            for t in dict:
                base = np.matmul(np.sqrt(q0),
np.array([102,108,97,103,123,m,n,s,t,125]))
                k2 = key2 - base
                sol = np.matmul(ki, k2)
                rd = np.round(sol)
                ok = 1
                for ww in rd:
                    if ww not in dict:
                        ok = 0
                        break
                if ok == 1:
                    print('flag{', end='')
                    for c in sol:
                        print(chr(int(c)), end='')
                    print(chr(m),chr(n),chr(s),chr(t),'}',sep='')
```

部分输出如下

```
......
flag{x/u_`qd^`_gnli`bhcuxew}
flag{y/t_ard^`^gnkcfhoZuxfu}
```

```
flag{x/t_`qd^`^gnkfcel^uxfv}
flag{x/u_`qd^`_gnliabhcuxfw}
flag{x/u_`qd^`_gnkiabhbuxgw}
flag{x/u^`qd^`_fnkm__eeuxhx}
flag{xOu_`qe^a_fnrmV_azuyWz}
flag{xOu_`qe^a_fnqjYadtuyXy}
flag{xOu_`qe^a_fnrmV^ayuyXz}
flag{xOu_`qe^a_fnqjYadsuyYy}
flag{xOu_`qe^a_fnqmW^axuyYz}
flag{x/u_`qe^a_fnqkZadsuyZy}
flag{y/t_`qe^`^gnmcchmauy_v}
flag{x/t_`qe^`^gnnf`ejeuy_w}
flag{x/t_`qd^`^gnmgaejcuyaw}
......
```

于是猜开头是'flag{y0u_are'结尾是'}'，改了下代码增加了已知条件

```
import os
from decimal import *
getcontext().prec=15
p = primes[:28]
dict = [i for i in range(48,58)]+[i for i in range(64,91)]+[95]+[i for i in
range(97,123)]
q = [p[-i:]+p[:-i] for i in range(15)]
q0 = [q[i][:12]+q[i][-1:] for i in range(len(q))]
k = np.sqrt([q[i][12:-1] for i in range(len(q))])
ki = np.linalg.inv(k)
base = np.matmul(np.sqrt(q0),
np.array([102,108,97,103,123,121,48,117,95,97,114,101,  125]))
k2 = key2[:15] - base
sol = np.matmul(ki, k2)
rd = np.round(sol)
ok = 1
for ww in rd:
    if ww not in dict:
        ok = 0
        break
if ok == 1:
    print('flag{', end='')
    for c in sol:
        print(chr(int(c)), end='')
```

```
flag{_a^good_gudrsdr
```

这很明显了呀，最后一个词根据字母数量和语意，怎么看都是guesser

第三题想过把小数部分拆成每10个十进制位一组，这样构造了20组模10的10次方空间下的线性方程组，因为误差来源于下一个方程的向上进位嘛，很小可以忽略。本以为这个设计很完美，然而发现这个矩阵没法求逆，因为模空间下只能做加减或者整数乘，没法做除法。于是自己写了个矩阵三角化，只用乘法和加法来消元，结果到了最后一行，数字就相当大了（具体好像是10的几十次方还是上百次方），等于说求解`ax ≡ b (mod 10^10)`其中a非常非常大而且是有误差的值。主要是误差的积累导致这个式子毫无意义，完全没法得到正解。其他办法实在想不出来，只能放弃这题

---

第二阶段的话，我能猜到应该是构造Lattice然后规约，但不知为什么我的python就是装不上Lattice库

```
ValueError: numpy.ndarray size changed, may indicate binary incompatibility.
Expected 96 from C header, got 88 from PyObject
```

计算了下这题就算做出来了分数也不够，干脆及时摆烂算了

# 总结

最后我第六名，第二阶段上分太难，最后半天离第五名还差一百来分需要再追两三题，但没做出来的题目全是我的知识盲区，已经没办法追了。虽然还是很盼着前五被锤一个，不过还是尊重对手。有几个题是本该第一阶段做出来的，但肯定不止我如此，别人也会这么想，就像当年考试时候总觉得自己有不该丢的分一样，其实每个人都有（你们北大本科生或许都是学霸不一定有这情况，我本科华科的，习惯了那种到处丢分的感觉了），所以就承认自己技不如人得了，毕竟我本来也不是做安全方向的。我本科是计科的，现在研二做视频编码的，能打这个比赛纯属吃本科的老本~~（毕竟华科教8086汇编、系统结构以及硬件是出了名的，算是复习了一遍本科知识了）~~，以及我编程能力确实还行吧，所以能拿第六也很知足了。

没有遗憾，因为每个阶段都表现出了最好的自己（其实还是有点遗憾的，毕竟是废寝忘食拼了命的。我住校外的，这些天疫情没法入校所以宅家打这比赛，每天日夜颠倒，连续一个星期每天只睡6小时，为了省时间每两三天要做核酸了才去中关园食堂正经吃一次饭，平时就只自己吃吃零食的，所以这个星期必定是瘦了不少。接下来要赶论文截稿以及继续干实验室的活）。

感谢主办方，感谢每一位出题人和工作人员。最后也感谢能读到这里的你，不管你是完整读的还是跳着读的还是直接拉到底看到这里的，至少没让我白写这么多吧。

// EOF