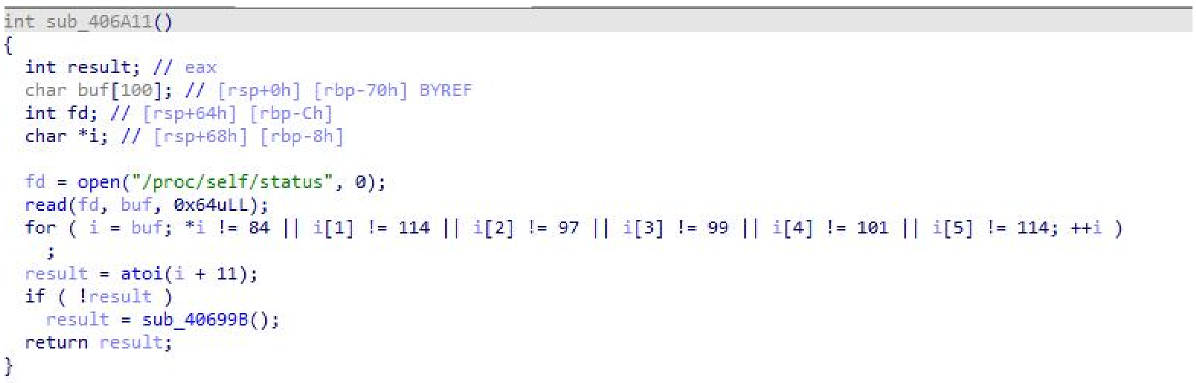
WEEK3-RogerThat WP

萎了萎了，没时间做pwn了(游戏玩多了)

2.Re

2.FAKE

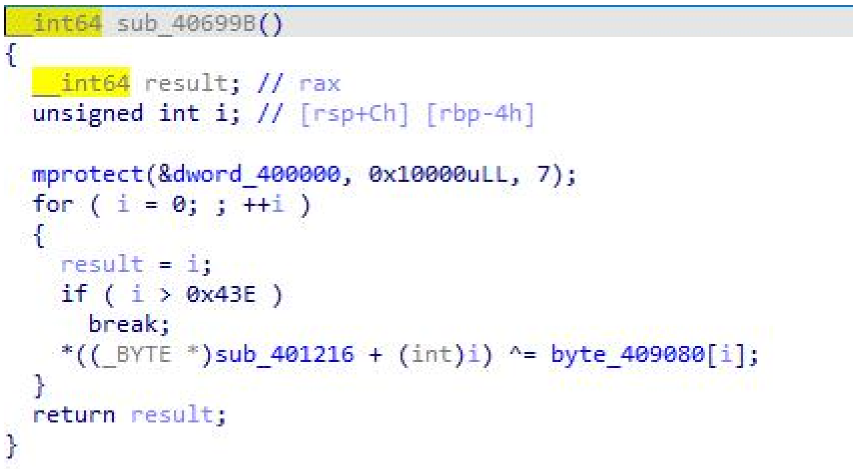
乱打一通，拿了一个假flag,没注意到init函数对调试器操作了一波。



```
int sub_406A11()
{
  int result; // eax
  char buf[100]; // [rsp+0h] [rbp-70h] BYREF
  int fd; // [rsp+64h] [rbp-Ch]
  char *i; // [rsp+68h] [rbp-8h]

  fd = open("/proc/self/status", 0);
  read(fd, buf, 0x64uLL);
  for ( i = buf; *i != 84 || i[1] != 114 || i[2] != 97 || i[3] != 99 || i[4] != 101 || i[5] != 114; ++i )
    ;
  result = atoi(i + 11);
  if ( !result )
    result = sub_40699B();
  return result;
}
```

这里应该是检测了是否有debugger，通过文件系统来达成，在windows中估计就是IsDebugger了，也可能有别的骚操作，暂且没见着，大胆猜测可以有多进程通信检测什么的。



```
__int64 sub_40699B()
{
  __int64 result; // rax
  unsigned int i; // [rsp+Ch] [rbp-4h]

  mprotect(&dword_400000, 0x10000uLL, 7);
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i > 0x43E )
      break;
    *((_BYTE *)sub_401216 + (int)i) ^= byte_409080[i];
  }
  return result;
}
```

针对这个比较简单的加密函数有脚本

```
#include <idc.idc>

static main(){
    auto addr = 0x401216;
    auto i = 0;
    for(i = 0;i<0x43E;i++){
        PatchByte(addr+i,Byte(addr+i)^Byte(0x409080+i));
    }
}
```

解密后得到真正的比较函数

```
__int64 __fastcall sub_401216(__int64 a1)
{
  int v2[36]; // [rsp+8h] [rbp-1D0h]
  int v3[36]; // [rsp+98h] [rbp-140h]
  int v4[38]; // [rsp+128h] [rbp-B0h] BYREF
  int m; // [rsp+1C0h] [rbp-18h]
  int l; // [rsp+1C4h] [rbp-14h]
  int k; // [rsp+1C8h] [rbp-10h]
  int j; // [rsp+1CCh] [rbp-Ch]
  int i; // [rsp+1D0h] [rbp-8h]
  unsigned int v10; // [rsp+1D4h] [rbp-4h]

  memset(v4, 0, 0x90uLL);
  v3[0] = 55030;v3[1] = 61095;v3[2] = 60151;v3[3] = 57247;v3[4] = 56780;v3[5] =
55726;
  v3[6] = 46642;v3[7] = 52931;v3[8] = 53580;v3[9] = 50437;v3[10] = 50062;v3[11]
= 44186;
  v3[12] = 44909;v3[13] = 46490;v3[14] = 46024;v3[15] = 44347;v3[16] =
43850;v3[17] = 44368;
  v3[18] = 54990;v3[19] = 61884;v3[20] = 61202;v3[21] = 58139;v3[22] =
57730;v3[23] = 54964;
  v3[24] = 48849;v3[25] = 51026;v3[26] = 49629;v3[27] = 48219;v3[28] =
47904;v3[29] = 50823;
  v3[30] = 46596;v3[31] = 50517;v3[32] = 48421;v3[33] = 46143;v3[34] =
46102;v3[35] = 46744;
  v2[0] = 104;v2[1] = 103;v2[2] = 97;v2[3] = 109;v2[4] = 101;v2[5] = 123;
  v2[6] = 64;v2[7] = 95;v2[8] = 70;v2[9] = 65;v2[10] = 75;v2[11] = 69;
  v2[12] = 95;v2[13] = 102;v2[14] = 108;v2[15] = 97;v2[16] = 103;v2[17] = 33;
  v2[18] = 45;v2[19] = 100;v2[20] = 111;v2[21] = 95;v2[22] = 89;v2[23] = 48;
  v2[24] = 117;v2[25] = 95;v2[26] = 107;v2[27] = 111;v2[28] = 110;v2[29] = 119;
  v2[30] = 95;v2[31] = 83;v2[32] = 77;v2[33] = 67;v2[34] = 63;v2[35] = 125;
  v10 = 1;
  for ( i = 0; i <= 5; ++i )
  {
    for ( j = 0; j <= 5; ++j )
    {
      for ( k = 0; k <= 5; ++k )
        v4[6 * i + j] += v2[6 * k + j] * *(_DWORD *)(4LL * (6 * i + k) + a1);
    }
  }
  for ( l = 0; l <= 5; ++l )
  {
    for ( m = 0; m <= 5; ++m )
    {
      if ( v4[6 * l + m] != v3[6 * l + m] )
        v10 = 0;
    }
  }
  return v10;
}
```

而假的函数却也像真的一样，要是解析不出来数据段嘛，那我倒是会先去想SMC，这点看真的是非常巧妙！！！后来仔细想了一下，应该是encrypt(f1(x)) == f2(2),只要满足能转换f1和f2，任何加密方式都可以。而异或恰恰是比较简单方便的加密方式。

基本是同一穿代码，异或以下（加密以下）能性成一种看似很像真代码的代码。

最后经过一些探索，发现是一个线性的加密，矩阵乘法。因此得到解密脚本

```
import numpy as np

v2 = [0]*36
v3 = [0]*36

v3[0] = 55030;v3[1] = 61095;v3[2] = 60151;v3[3] = 57247;v3[4] = 56780;v3[5] =
55726;
v3[6] = 46642;v3[7] = 52931;v3[8] = 53580;v3[9] = 50437;v3[10] = 50062;v3[11] =
44186;
v3[12] = 44909;v3[13] = 46490;v3[14] = 46024;v3[15] = 44347;v3[16] =
43850;v3[17] = 44368;
v3[18] = 54990;v3[19] = 61884;v3[20] = 61202;v3[21] = 58139;v3[22] =
57730;v3[23] = 54964;
v3[24] = 48849;v3[25] = 51026;v3[26] = 49629;v3[27] = 48219;v3[28] =
47904;v3[29] = 50823;
v3[30] = 46596;v3[31] = 50517;v3[32] = 48421;v3[33] = 46143;v3[34] =
46102;v3[35] = 46744;
v2[0] = 104;v2[1] = 103;v2[2] = 97;v2[3] = 109;v2[4] = 101;v2[5] = 123;
v2[6] = 64;v2[7] = 95;v2[8] = 70;v2[9] = 65;v2[10] = 75;v2[11] = 69;
v2[12] = 95;v2[13] = 102;v2[14] = 108;v2[15] = 97;v2[16] = 103;v2[17] = 33;
v2[18] = 45;v2[19] = 100;v2[20] = 111;v2[21] = 95;v2[22] = 89;v2[23] = 48;
v2[24] = 117;v2[25] = 95;v2[26] = 107;v2[27] = 111;v2[28] = 110;v2[29] = 119;
v2[30] = 95;v2[31] = 83;v2[32] = 77;v2[33] = 67;v2[34] = 63;v2[35] = 125;

v2_matrix = []
for i in range(6):
    temp_list = []
    for j in range(6):
        temp_list.append(v2[6*j+i])
    v2_matrix.append(temp_list)

v3_matrix = []
for i in range(6):
    temp_list = []
    for j in range(6):
        temp_list.append(v3[6*i+j])
    v3_matrix.append(temp_list)

# print(v2_matrix)
# print(v3_matrix)

v2_matrix = np.array(v2_matrix)
v3_matrix_compute = []
for i in range(6):
    v3_matrix_compute.append(np.array(v3_matrix[i]))

# print(v2_matrix)
# print(np.linalg.inv(v2_matrix))
# print(v3_matrix_compute[0])
for i in range(6):
    v3_matrix_compute[i] = np.array([v3_matrix_compute[i]]).T
    #print(v3_matrix_compute[i])
    result = np.matmul(np.linalg.inv(v2_matrix),v3_matrix_compute[i])
    for i in np.array(result):
        for j in i:
            #print(j,end = '')
            print(chr(round(j)),end = '')
```

对于脚本的一点mark，希望下次能快点写出来，踩了大坑，不要拿4血，真的淦。不过，最重要的应该还是知识点，其实这种操作复杂化一点就变成了加壳，那些脱壳的操作，正是这个进化版本，可能还会用到什么动态库，在seh上设置一个加壳函数，抛出异常再脱壳等等(仅猜测)

```
#初始化矩阵
matrix = np.array(<list>)
#矩阵乘法
result = np.matmul()
#矩阵点乘
result = np.dot()
#一维矩阵转置
matrix = np.array([matrix]).T
#矩阵求逆
matrix_inv = np.linalg.inv(matrix_inv)
```

再mark一波idc https://blog.csdn.net/jazrynwong/article/details/84875699

其实看文档也可以,不过还是中文舒服点https://www.hex-rays.com/products/ida/support/idadoc/162.shtml

3.helloRe3

跟踪StringOutDebug，也就是debugger函数就会发现这个东西，可以看见

1

框框中对一个全局变量进行了改变，可见，这里是关键，但是发现我一旦把这地方变成1，它马上就变回0了。
当我们跟踪内存时候发现这个&(&addr)[72*i][4*v16+1]其实在内存中代表着对应字符对应的一个数字&(&addr)[72*i][4*v16]则代表内存中的一个指向对应字符的指针

直接在这个地方下硬件写入断点，就能的到被改变的原因了，原来是一个线程搞的鬼。不过不下断点也能猜测到是别的线程下的手脚。不过仔细观察一下线程的一些特征也能找到一些不对劲的地方。

2

除了主线程以外，还有一个线程在用户代码上花了很多时间，调试过后会发现这个线程的时间比主线程还长，由此可见应该去分析一下那个线程的入口，不出所料。

3

看见那个地方为0，那么就会在循环里面不断重复。一旦变成1，他就开始了！！！

4

在右边的方框中先把每个地方取反然后再进入左边，经过动态调试可以发现第一个函数是加密，静态分析后会发现第二个是比较两个是否相等。跟进第一个函数后发现，里面有两个函数，第一个生成一个key，第二个是加密，而key以明文256位存在内存中，直接dump下来就可以了。很奇怪的是为什么我dump了好几次，但是有两个结果？？第二天的时候dump下来的key才能用，很奇怪。可能是我之前dump错了地方，搞得我想半天以为是反调试，比较这个代码

最后解密脚本：

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

void exchange_a_byte(uint8_t * a,uint8_t * b){
    char v = *a;
    *a = *b;
```

```c
    *b = v;
}
int main() {
    uint8_t string_0[256] =
{0x90,0x1D,0x7B,0xA2,0x53,0xA5,0x73,0xB6,0xCC,0x32,0x20,0x21,0x28,0x7A,0xE4,0x97

,0xEF,0x41,0x66,0x04,0xF3,0x24,0xA9,0xBB,0x44,0x3E,0xCD,0xA4,0xD5,0x7D,0xFE,0x4B

,0x3A,0x3C,0x8F,0x08,0xB8,0x23,0x2B,0x54,0x6D,0x25,0x47,0xBD,0x4D,0xDC,0xDA,0x2A

,0xBF,0x75,0xB1,0xC3,0x05,0xD2,0x17,0x3F,0x30,0x83,0xFD,0x96,0xE0,0xAD,0xCF,0x57

,0x0E,0x85,0x18,0xB4,0x52,0x86,0x6C,0xE1,0x8C,0x77,0x38,0x8E,0xB0,0x40,0xDF,0x7E

,0xD4,0x78,0xC7,0xF2,0x5C,0x5D,0x39,0x67,0x62,0x27,0x79,0x2E,0x58,0x82,0x2F,0x01

,0x69,0x59,0xAC,0x22,0x76,0x8D,0xDD,0xC1,0xB2,0xE9,0xFA,0x2C,0xE2,0xC4,0x1A,0x0D

,0xC8,0x09,0xD7,0x46,0x02,0x0C,0xE7,0xA8,0xB5,0x64,0x5A,0xA6,0x1E,0x5F,0x6B,0xC6

,0xAA,0x91,0xB9,0xD8,0xD6,0x48,0x49,0xA7,0x11,0x56,0xFB,0xEC,0x8B,0x1B,0xE3,0x45

,0xAB,0xF6,0x4A,0xDB,0xB7,0x81,0x10,0x42,0x71,0x0F,0x89,0x14,0xD3,0x1F,0x50,0xEE

,0x60,0x33,0x4F,0x7C,0x98,0xAE,0x9A,0xEA,0x8A,0xC5,0x9C,0xF8,0xBA,0xD9,0x1C,0x31

,0xB3,0x68,0x5B,0x80,0x03,0xF9,0x19,0x92,0x3B,0x29,0x3D,0x5E,0xDE,0x51,0xED,0x70

,0x7F,0x0A,0x36,0xD1,0x87,0xAF,0x35,0x16,0x9E,0xC9,0xEB,0xF5,0x37,0xF1,0x61,0x43

,0xE5,0x95,0xA3,0xE8,0x34,0x9D,0xBC,0xCA,0x65,0x13,0xA1,0xF0,0xF7,0xCE,0x94,0x55

,0x6F,0x99,0x84,0x26,0x88,0x00,0xFC,0xD0,0xCB,0x9F,0x93,0xC2,0x6A,0x9B,0x4E,0x2D

,0x74,0xE6,0xBE,0x15,0x72,0xFF,0xF4,0x07,0x4C,0x12,0x6E,0x06,0xC0,0x63,0xA0,0x0B
};

    uint8_t result[20] = {0x4D,0xAF,0x27,0xAD,
                          0xE1,0xEC,0x6D,0xDA,
                          0xF0,0x31,0x5E,0x9A,
                          0x9E,0x29,0xFA,0xBE,
                          0x6B,0x08,0xC8,0x49};

    uint8_t flag[20] = {0};
    int v6 = 0,v7 = 0;
    for(int i = 0;i < 20;i++){
        v7 = (v7+1)%256;
        v6 = (string_0[v7]+v6)%256;
        exchange_a_byte(string_0+v7,string_0+v6);
        flag[i] = result[i] ^ *(uint8_t*)(string_0 + (*(uint8_t*)(string_0+v6)+*
(uint8_t*)(string_0+v7))%256);
    }

//    for(int i = 0;i < 256;i++){
//        if(i%16 == 0 && i!= 0)
//            printf("\n");
//        printf("%x ",string_0[i]);
//    }
//    putchar('\n');
//    for(int i = 0;i<20;i++){
//        printf("%x ",flag[i]);
//    }

    printf("\n");
    for(int i = 0;i<20;i++){
        printf("%02d ",flag[i]^0xFF);
    }
}
//HGAME{6-K4K.4R+3C4T}
```

在这个题目中也有很多可以讨论的地方，比如在TLS函数中可以做手脚，检测int断点等等，对于dump那块内存最稳妥的办法是修改程序流，似乎可以用dll注入的方法，不过那个徒手操作确实比较复杂到现在我也掌握的不好（老是算错地址。。。）在WndProc中我们也可以看到其中对输入的一些操作，但是我终究不清楚它的WM_COMMOND跑哪去了，那些按钮究竟在哪实现的，这点很奇怪。

5

我还找到了另外的一个switch分支***sub_1401CC620***，但是很迷惑，只是经过动态调试知道了它在按钮按下的时候走的default路线。在其中还有对xmm0，xmm1这些寄存器的操作，应该是经过优化后得到的结果。

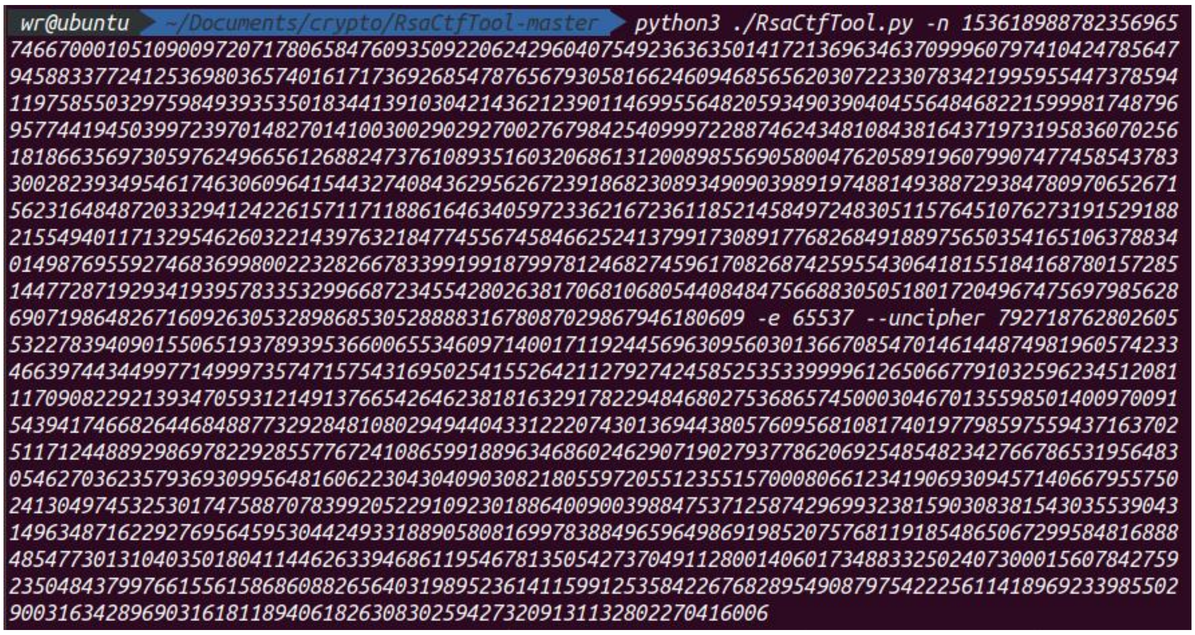
mark：rep stosb就是从EDI所指的内存开始，将连续的ECX个字节写成AL的内容，多用于清零等，每次遇到这个都会忘，然后重新查，就像movsx，movss这些指令一样。

还有一点，程序中很容易产生[addr] = ??? 也就是地址写入的时候出异常，这种时候x64dbg可以用shift+f9忽略异常，也可以在选项中直接把对应异常的代码忽略掉（建议这样做）

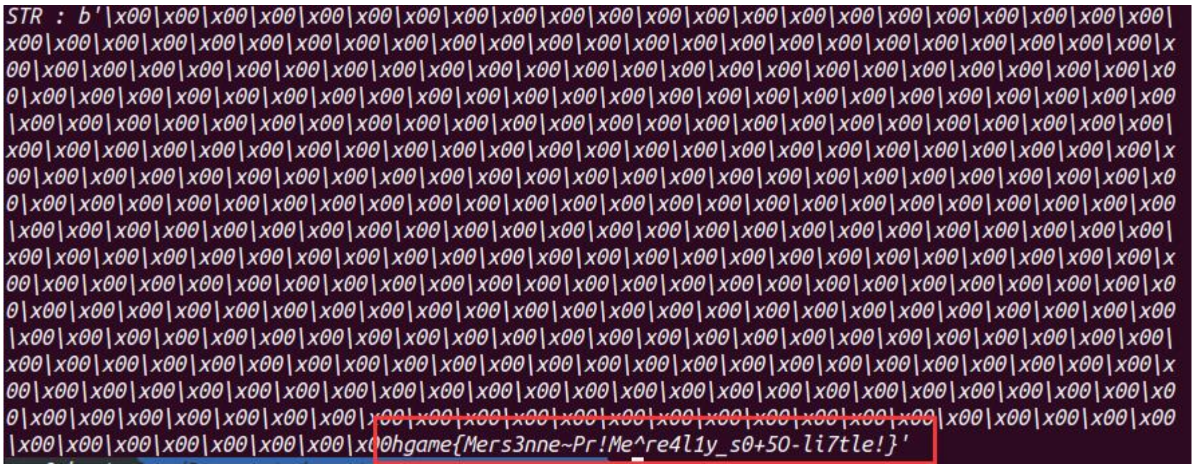
4.Crypto

1.rsa

和之前一样,,,这算是白嫖flag嘛,,,都有点不好意思了



```
 wr@ubuntu  ~/Documents/crypto/RsaCtfTool-master  python3 ./RsaCtfTool.py -n 153618988782356965
74667000105109009720717806584760935092206242960407549236363501417213696346370999607974104247856479
45883377241253698036574016171736926854787656793058166246094685656203072233078342199595544737859
41197585503297598493935350183441391030421436212390114699556482059349039040455648468221599981748796
95774419450399723970148270141003002902927002767984254099972288746243481084381643719731958360702561
81866356973059762496656126882473761089351603206861312008985569058004762058919607990747745854378330
02823934954617463060964154432740843629562672391868230893490903989197488149388729384780970652671
56231648487203329412422615711711886164634059723362167236118521458497248305115764510762731915291882
15549401171329546260322143976321847745567458466252413799173089177682684918897565035416510637883401
49876955927468369980022328266783399199187997812468274596170826874259554306418155184168780157285
144772871929341939578335329966872345542802638170681068054408484756688305051801720496747569798562869
07198648267160926305328986853052888831678087029867946180609 -e 65537 --uncipher 79271876280260553
22783940901550651937893953660065534609714001711924456963095603013667085470146144874981960574233466
39744344997714999735747157543169502541552642112792742458525353399996126506677910325962345120811170
90822921393470593121491376654264623818163291782294846802753686574500030467013559850140097009154394
17466826446848877329284810802949440433122207430136944380576095681081740197798597559437163702511712
44889298697822928557767241086599188963468602462907190279377862069254854823427667865319564830546270
36235793693099564816062230430409030821805597205512355157000806612341906930945714066795575024130497
45325301747588707839920522910923018864009003988475371258742969932381590308381543035539043149634871
62292769564595304424933188905808169978388496596498691985207576811918548650672995848168884854773013
10403501804114462633946861195467813505427370491128001406017348833250240730001560784275923504843799
76615561586860882656403198952361411599125358422676828954908797542225611418969233985502900316342896
90316181189406182630830259427320913111328022704160 06
```



```
STR : b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00hgame{Mers3nne~Pr!Me^re4l1y_s0+5O-li7tle!}'
```

不过那个网站也分出了素数