HGAME Week3 WriteUp

RE

Math 简单

```
v23 = __readfsqword(0x28u);
v3 = std::operator<<<std::char_traits<char>>(
       &std::cout,
       "to continue, you have to guess the value of my dice first!",
       envp);
std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
v21 = rolling_dice();
std::operator<<<std::char_traits<char>>(&std::cout, "now the dice have been rolled, guess what it is: ", v4);
std::istream::operator>>(&std::cin, &v20);
v5 = v20;
v7 = std::operator<<<std::char_traits<char>>(&std::cout, "expected: ", v6);
v8 = std::ostream::operator<<(v7, v21);
v10 = std::operator<<<std::char_traits<char>>(v8, ", guess: ", v9);
v11 = std::ostream::operator<<(v10, v5);
std::ostream::operator<<(v11, &std::endl<char,std::char_traits<char>>);
if ( v20 != v21 )
{
  v13 = std::operator<<<std::char_traits<char>>(&std::cout, "you are bad at guessing dice", v12);
  std::ostream::operator<<(v13, &std::endl<char,std::char_traits<char>>);
  exit(0);
}
std::operator<<<std::char_traits<char>>(
  &std::cout,
  "wow, you are good at dice-guessing, now give me your flag: ",
  v12);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(&v22);
std::operator>><char,std::char_traits<char>,std::allocator<char>>(&std::cin, &v22);
if ( std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(&v22) != 32 )
{
  v15 = std::operator<<<std::char_traits<char>>(&std::cout, "assert len(flag) == 32", v14);
  std::ostream::operator<<(v15, &std::endl<char,std::char_traits<char>>);
  exit(0);
}
v16 = std::operator<<<std::char_traits<char>>(&std::cout, "now the math part...", v14);
std::ostream::operator<<(v16, &std::endl<char,std::char_traits<char>>);
if ( (unsigned __int8)math_part(&v22) )
  v18 = std::operator<<<std::char_traits<char>>(
          &std::cout,
          "wow, you are good at doing math too, you deserve to have the flag, just submit it!",
          v17);
else
  v18 = std::operator<<<std::char_traits<char>>(&std::cout, "you are bad at doing math", v17);
std::ostream::operator<<(v18, &std::endl<char,std::char_traits<char>>);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&v22);
return 0;
```

前面是一个随机生成骰子然后检验的代码，其实管不管都无所谓（反正最早调试的时候我都是随便蒙的，概率还挺高【笑】）

后面就是输入一个32位的flag，点进math_path函数以后就是一个32元方程组，这个跟第一周的一道re题差不多，至少我的思路差不多

可是我还是想弄个系数方程组出来，所以费时费力还出错了，直到我发现mma也可以像oyeye的z3题解一样Solve解方程

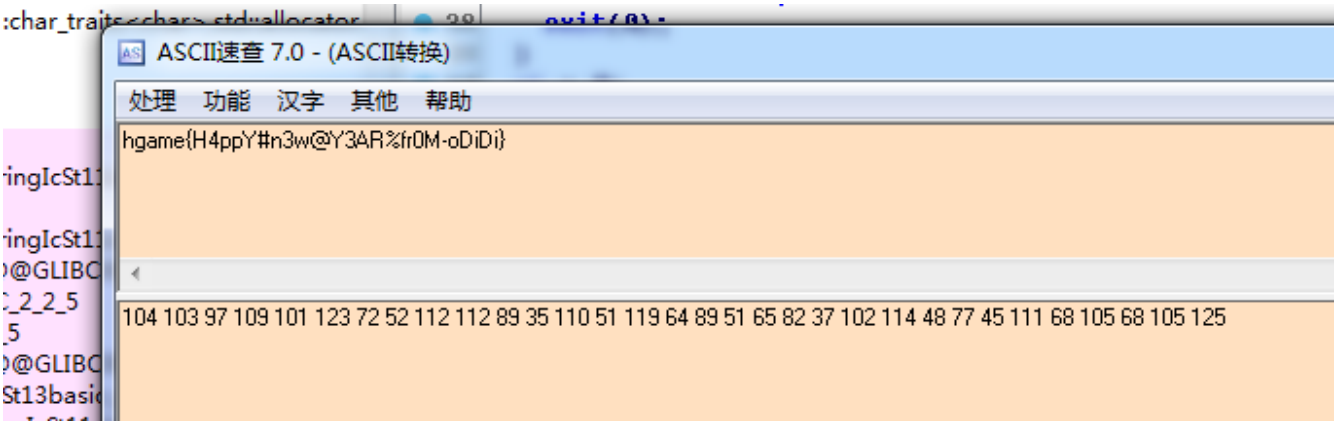
最后大概就是这个样子

In[5]:=
```
Solve[76 * v34[21]
 |解方程
 + 31 * v34[9]
 + 87 * v34[28]
 + 54 * v34[2]
 + 74 * v34[5]
 + 99 * v34[26]
 + 94 * v34[3]
 + 84 * v34[19]
 + 32 * v34[15]
 + 90 * v34[27]
 + 16 * v34[14]
 + 19 * v34[8]
 + 33 * v34[20]
 + 35 * v34[31]
 + 65 * v34[29]
 + 47 * v34[12]
 + 3 * v34[1]
 + 57 * v34[7]
 + 5 * v34[17]
 + 70 * v34[13]
 + 28 * v34[24]
 + 79 * v34[11]
 + 63 * v34[23]
 + 66 * v34[30]
 + 28 * v34[10]
 + v34[4]
 + 82 * v34[16] + 58 * v34[25] + 81 * v34[6] + 61 * v34[18] + 31 * v34[22] + 71 * v34[0] == 145 397 &&
   55 * v34[6]
 + 38 * v34[9]
 + 39 * v34[18]
 + 73 * v34[24]
```

... ...

```
+ 89 * v34[1]
+ 88 * v34[18]
+ 3 * v34[3]
+ 59 * v34[20]
+ 80 * v34[23]
+ 49 * v34[17]
+ 56 * v34[21]
+ 32 * v34[27]
+ 24 * v34[2] + 13 * v34[14] + 73 * v34[19] + 99 * v34[7] + 76 * v34[12] + 77 * v34[30] +
    18 * v34[6] == 138403
]
```

```
Out[5]= {{v34[0] → 104, v34[1] → 103, v34[2] → 97, v34[3] → 109, v34[4] → 101,
  v34[5] → 123, v34[6] → 72, v34[7] → 52, v34[8] → 112, v34[9] → 112, v34[10] → 89,
  v34[11] → 35, v34[12] → 110, v34[13] → 51, v34[14] → 119, v34[15] → 64,
  v34[16] → 89, v34[17] → 51, v34[18] → 65, v34[19] → 82, v34[20] → 37, v34[21] → 102,
  v34[22] → 114, v34[23] → 48, v34[24] → 77, v34[25] → 45, v34[26] → 111,
  v34[27] → 68, v34[28] → 105, v34[29] → 68, v34[30] → 105, v34[31] → 125}}
```



得到flag

Say-Muggle-Code a.k.a. SMC

这道题一开始并没看到smc，不过也没啥关系，把不懂的函数搜一下就知道它在干什么了

```
else
{
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(&v15, &v12, 6LL, 16LL);
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::substr(&v16, &v12, 22LL, 16LL);
  v17 = 0LL;
  v18 = 0LL;
  v17 = *(unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
                              &v15,
                              0LL);
  for ( i = 1; i <= 15; ++i )
  {
    v7 = *(unsigned __int8 *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
                               &v15,
                               i - 1);
    *((_BYTE *)&v17 + i) = *(_BYTE *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](
                                       &v15,
                                       i) ^ v7;
  }
  v8 = (unsigned __int8)check1(&v15) ^ 1 || !check2((__int64)&v16, &v17);
  if ( v8 )
    v9 = std::operator<<<std::char_traits<char>>(&std::cout, "your flag is good, but mine is better, muggle!");
  else
    v9 = std::operator<<<std::char_traits<char>>(
           &std::cout,
           "wow, your flag is exactly the same as mine, congratulations, just submit it!");
  std::ostream::operator<<(v9, &std::endl<char,std::char_traits<char>>);
```

输入39位的flag，然后判断前后是"hgame{"和"}"，之后就是将中间的32位字串分成两组

第一组进check1，第二组进check2，当然check2的另一个参数是由前半段字串处理得来的

```
1 signed __int64 __fastcall check1(__int64 a1)
2 {
3   int v1; // eax
4   int i; // [rsp+1Ch] [rbp-14h]
5
6   for ( i = 0;
7         i < (unsigned __int64)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::length(a1);
8         ++i )
9   {
10    v1 = *(char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::operator[](a1, i);
11    LOBYTE(v1) = v1 ^ 0xE9;
12    if ( v1 != (unsigned __int8)data1[i] )
13      return 0LL;
14  }
15  return 1LL;
16 }
```

check1就是一个xor

```
int a[] = { 0xDE, 0xD1, 0xD8 , 0x8C , 0x8F , 0xD9 , 0xDF , 0xDE  , 0xDF , 0x8C , 0xD8 ,
0xDA , 0x8C , 0xDC , 0xDD , 0xD8 };
    int b[16];
    for(int i =0;i<16;i++)
    {
        b[i] = a[i] ^ 0xe9;
        printf("%c", b[i]);
    }
```

得到前半段为 781ef0676e13e541

check2的函数如图

```
bool __fastcall check2(__int64 a1, _DWORD *a2)
{
  const char *v2; // rax
  char dest[8]; // [rsp+10h] [rbp-20h]
  __int64 v5; // [rsp+18h] [rbp-18h]
  char v6; // [rsp+20h] [rbp-10h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  *(_QWORD *)dest = 0LL;
  v5 = 0LL;
  v6 = 0;
  v2 = (const char *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(a1);
  strcpy(dest, v2);
  mprotect(encrypt, 0x200uLL, 7);
  modify(encrypt, 0x200uLL);
  encrypt((__int64)dest, a2);
  return strcmp(dest, data2) == 0;
}
```

把不懂的函数查了一下，发现是mprotect()是linux函数，大意就是改变一段内存的属性，就是可不可以执行啊之类的，然后modify()函数点进去发现是对encrypt段内容的处理

然后后面又调用了encrypt函数，看来就是将encrypt的值做了些处理然后再通过encrypt函数来处理后半段字串了

然而我很懒，我懒得管这些处理，我决定直接动态调试

```
{
  _DWORD *result; // rax
  int v3; // [rsp+10h] [rbp-10h]
  signed int i; // [rsp+14h] [rbp-Ch]
  signed int j; // [rsp+18h] [rbp-8h]

  v3 = 0;
  for ( i = 0; i <= 31; ++i )
  {
    result = (_DWORD *)2654435769LL;
    v3 -= 1640531527;
    for ( j = 0; j <= 3; j += 2 )
    {
      *(_DWORD *)(a1 + 4LL * j) = *(_DWORD *)(4LL * j + a1)
                                + ((*(_DWORD *)(4 * (j + 1LL) + a1) + v3) ^ (16 * *(_DWORD *)(4 * (j + 1LL) + a1) + *a2) ^ ((*(_DWORD *)(4 * (j + 1LL) + a1) >> 5) + a2[1]));
      result = (_DWORD *)(4 * (j + 1LL) + a1);
      *result += (*(_DWORD *)(4LL * j + a1) + v3) ^ (16 * *(_DWORD *)(4LL * j + a1) + a2[2]) ^ ((*(_DWORD *)(4LL * j + a1) >> 5)
                                                                                              + a2[3]);
    }
  }
  return result;
}
```

得到encrypt函数。这个加密是真的不懂，后来搜索了1640531527才发现是一个Tea加密

最后解密是这样的

```
    DWORD a2[4] = { 0x54090f37,0x01065603,0x02545301,0x05015056};
    DWORD a1[4] = { 0xbc12926d,0xf604bc33,0xe22c59d6,0xcc87ed7d };
    unsigned int v3;
    unsigned int delta = 0x9e3779b9;
    v3 = delta * 32;
    for (int i = 0; i <= 31; ++i)
    {
        for (int j = 0; j <= 3; j += 2)
        {
            a1[j + 1] -= 16 * a1[j] + a2[2]^a1[j] + v3^(a1[j] >> 5) + a2[3];
            a1[j] -= 16 * a1[j+1] + a2[0] ^ a1[j+1] + v3 ^ (a1[j+1] >> 5) + a2[1];
        }
        v3 -= delta;
    }
    for (int i = 0; i < 4; i++)
    {
        printf("%x\n", a1[i]);
    }
```

作为一个懒惰的人，我上面a[2]的值是动态调试的时候直接看的，所以没有去管前面的处理XD

```
>>> from libnum import *
>>> n2s(0x6662303561303764646232636565323 4)
'fb05a07ddb2cee24'
```

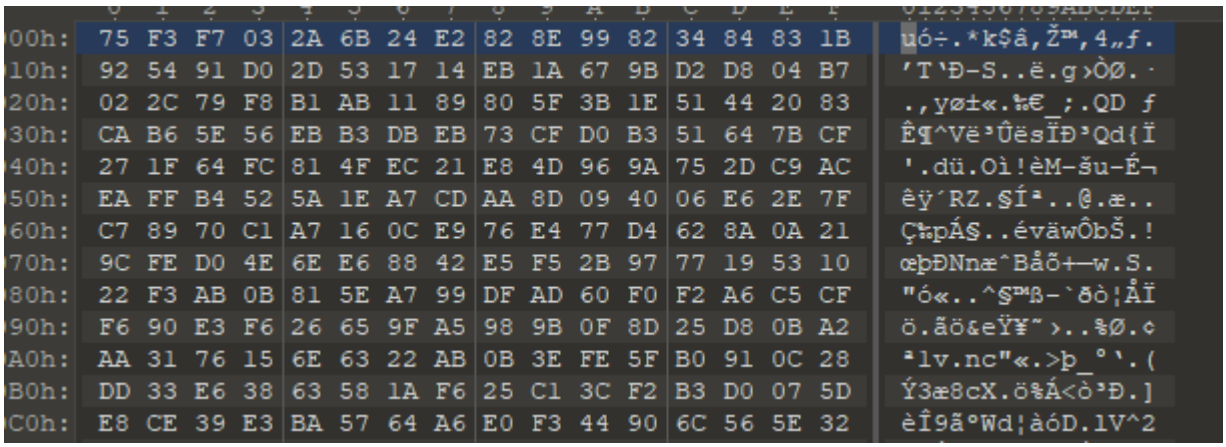
后半段也就出来了，最后的flag就是 hgame{781ef0676e13e541fb05a07ddb2cee24}

MISC

时至今日，你仍然是我的光芒

下载了hint中的两个软件，先是用 DeEgger Embedder 得到了一张图片

根据hint是使用outguess，但是我用outguess直接读隐写失败了，搜了一下outguess是可以有Key的，一开始以为密码就是'sec.*'，结果生成了一个实在看不懂的东西



本来打算放弃了，最后一天忍不住问了BrownFly学长，才知道sec.*是正则表达式，意思是'sec'开头的字串，才总算搞懂这道题的意思，就是用rockyou.txt这个字典去爆破outguess的密码。因为我没有Kali和rockyou.txt，好心的BrownFly学长帮我提取出来了所有'sec'开头的字典

接下来就是跑脚本咯，第一次用pyhton去调用其它程序，这个感觉还是非常新奇的

下面附上脚本

```
import subprocess

def tryOutguess(key):
    popen = subprocess.Popen(['outguess','-k',key,'-r','flag.jpg','flag.txt'])
    flagFile = open('flag.txt','r')
    flag = flagFile.readline()
    a = flag[0:5]
    if a == 'hgame':
        print 'flag is: ' + flag
        flagFile.close()
        return 1
    print 'There is no flag'
    flagFile.close()
    return 0
```

```
def tryKey():
    keyFile = open('passwd.txt','r')
    for line in keyFile.readlines():
        key = line.strip('\n')
        print 'Try key: ' + key
        if tryOutguess(key):
            break
    keyFile.close()

if __name__ == '__main__':
    tryKey()
```

```
Try key: security2
Reading flag.jpg....
Extracting usable bits:   46681 bits
Steg retrieve: seed: 17533, len: 28935
Extracted datalen is too long: 28935 > 5836
There is no flag
Try key: security101
Reading flag.jpg....
Extracting usable bits:   46681 bits
Steg retrieve: seed: 58, len: 27
Extracting usable bits:   46681 bits
Steg retrieve: seed: 55666, len: 25523
Extracted datalen is too long: 25523 > 5836
Extracting usable bits:   46681 bits
Steg retrieve: seed: 44884, len: 32486
Extracted datalen is too long: 32486 > 5836
flag is: hgame{Whataya_Want_From_Me}
Reading flag.jpg....
danis@ubuntu:~$ Extracting usable bits:   46681 bits
```

得到flag

另外，这道题最后的提交时间

> 幼稚园 2019-02-15T19:59:25+08:00

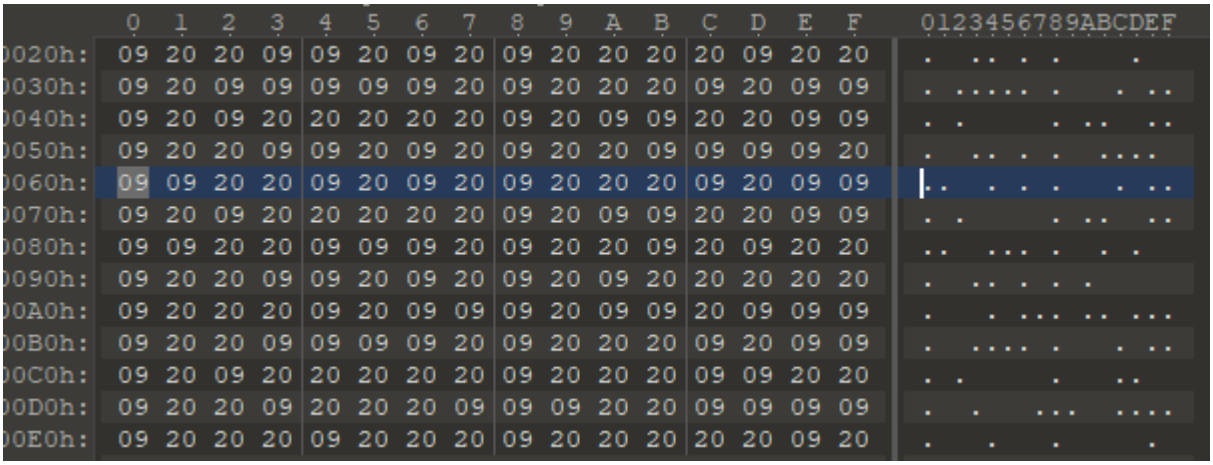
真是太紧张刺激了【笑】

至少像那雪一样

下载到的是一张图片，这次binwalk不准，把jpg文件结尾的ff d9找到然后下面的都是.zip压缩文件了

这个压缩文件的特点是里面有一张名字也是至少像那雪一样的文件，看了下大小跟我从下载下来的文件中提取出来的jpg一样，所以可以想到是明文攻击，用工具就行啦

之后就得到了一个flag.txt，但是是空的，用hex方式打开，看到一堆".'和' '，搜索了很久才发现是“反着的”ASCII编码，就是'.'代表0，' '代表1



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
0020h:	09	20	20	09	09	20	09	20	09	20	20	20	20	09	20	20	. .. . . .
0030h:	09	20	09	09	09	09	09	20	09	20	20	20	09	20	09	09	. ...
0040h:	09	20	09	20	20	20	20	20	09	20	09	09	20	20	09	09	. . . .
0050h:	09	20	20	09	09	20	09	20	09	20	20	09	09	09	09	20	. .. .
0060h:	09	09	20	20	09	20	09	20	09	20	20	20	09	20	09	09	.. . .
0070h:	09	20	09	20	20	20	20	20	09	20	09	09	20	20	09	09	. . . .
0080h:	09	09	20	20	09	09	09	20	09	20	20	09	20	09	20	20	.. ..
0090h:	09	20	20	09	09	20	09	20	09	20	09	20	20	20	20	20	. .. . . .
00A0h:	09	20	20	20	09	20	09	09	09	20	09	09	20	09	09	09	. . ..
00B0h:	09	20	20	09	09	09	09	20	09	20	20	20	09	20	09	09	. ...
00C0h:	09	20	09	20	20	20	20	20	09	20	20	20	09	09	20	20	. . . ..
00D0h:	09	20	20	09	20	20	20	09	09	09	20	20	09	09	09	09	. . ..
00E0h:	09	20	20	20	09	20	20	20	09	20	20	20	20	20	09	20	. . . .

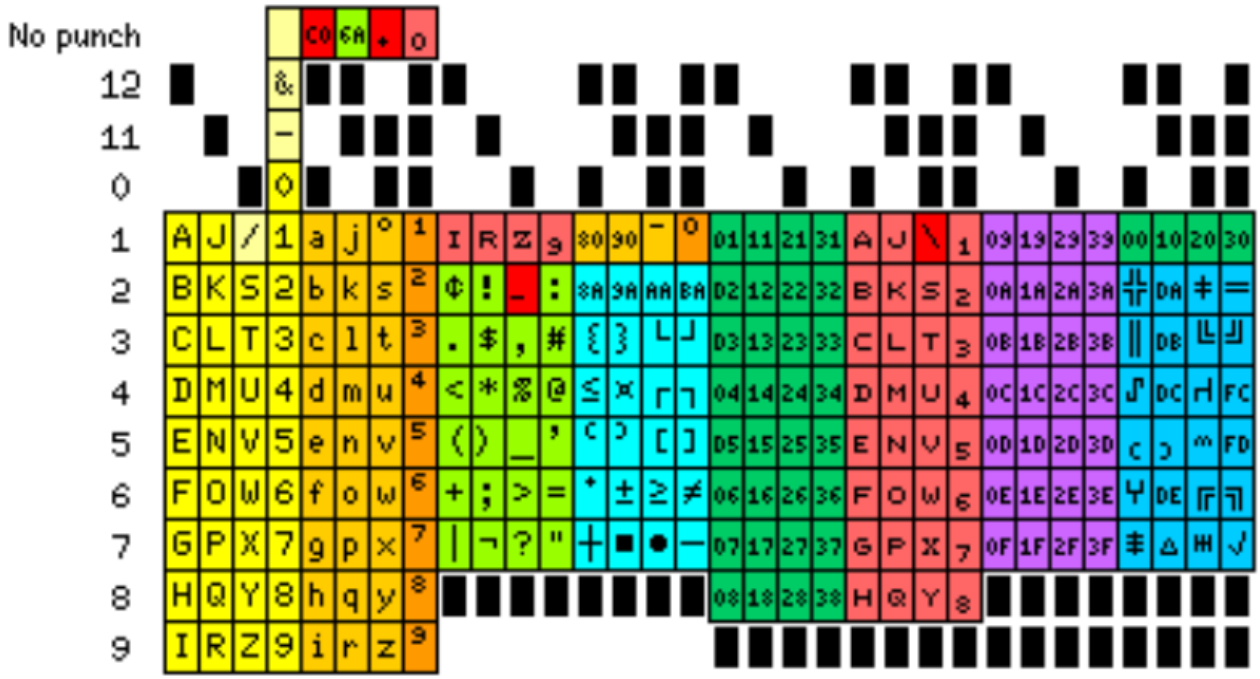
然后就用ASCII码编码，得到flag

hgame{At_Lea5t_L1ke_tHat_sn0w}

旧时记忆

这道题一开始看不懂，所以我仔细研读了题目，提炼出了几个关键词：“学历史”，"memory"，“存储器”（后面两个是hint）

那就很好搜索了，搜索存储器的历史这类的就能搜出来，这个是IBM的打孔卡，之后在google上找到了一张对比图



对照就能得到flag了

hgame{0LD_DAY5%M3MORY}

听听音乐?

这道题还是比较明了的，将音乐文件载入到Au里就能发现后面一堆是摩斯电码，其实我觉得直接听应该也可以

```
FLAG:1T_JU5T_4_EASY_WAV
```

得到flag

CRYPTO

babyRSA

这道题一开始试了很多网上的脚本都失败了，各种报错，我很是不解，无奈只能去理解RSA

其实看了半天也不太理解，所以我打算手算一波，结果用辗转相除法一除我就懵逼了，没余数，直接就除干净了！遂发现原来e和n不互质，那怎么办呢?

还好我的信息收集能力还是很强的，我竟然搜到了一个类似的问题!

看看大神是怎么理解这类题的

> 这里的e和phi又不是互素的，有公约数2，乍一看非常头疼
>
> 实际上，这里的公约数2和14比实在太小了，所以我们可以直接破解：
>
> 按照之前的思路
>
> $$c \equiv m^e \ mod \ q_1q_2$$
> $$e = 2 * 7$$
> $$2 * 7 * d \equiv 1 \ mod \ q_1q_2$$
> $$m^2 \equiv c^{2d} \ mod \ q_1q_2$$
>
> 2d可以通过7的逆元求得，由于2次方太小，所以直接对m开方即可

我个人是这么理解的，这个相当于标准RSA的一个变式，相当于先把m做一个幂运算，然后把这个m^4当做新的m来做标准的RSA加密

最后的脚本如下

```
from libnum import *
import gmpy2

p =
gmpy2.mpz(583800044303078033678069964607731236037903057890983844889520562066157682745 27)
```

```
q =
gmpy2.mpz(818595269757200606493800981936716128012005050291270765394576801554876696228 67)
e = gmpy2.mpz(3)
phi_n = (p - 1) * (q - 1)
d = gmpy2.invert(e, phi_n)
print (d)

c =
gmpy2.mpz(206087215323690202467878926681944491769659156726458690815919286163630886447291 5705101961715856261436089883846151859217524093807880064765763374101364474 60)
m = pow(c,d,p*q)
print m
print n2s(gmpy2.iroot(m, 4)[0])
```

```
from libnum import *

a = 0x6867616d657b656173795f43727970746f217d
print n2s(a)
```

```
>>> hex(232828321890052373500842
'0x6867616d657b656173795f4372797
>>> quit()
danis@ubuntu:~$ python ./rsa.py
hgame{easy_Crypto!}
```

得到flag

basicmath

这道题一开始以为又是一个变相的RSA

后来我发现它就是一个变相的RSA，但是它叫二次剩余，它的一个用途是Rabin加密——就是RSA的e为2的一种类似情况，但这个不是Rabin加密，因为它的n是质数，Rabin和RSA一样n=p*q

那么怎么办呢？？

这时候我重新审视了题目，我提炼出了一个关键点，那就是这个质数不是一般的质数，它必须要除4余3

```
p = getPrime(256)
while(p % 4 != 3):
    p = getPrime(256)
a = pow(m, 2, p)
```

这样一来搜索就有了切入点，还好百度和谷歌非常的强大，我搜索到了一本书http://book.51cto.com/art/200812/102580.htm

我仔细阅读了这本书的相关内容后我发现一个重点

3. 解二次等式模一个素数

虽然Euler标准告诉我们，如果一个整数a是 Z_p^* 中的一个QR或QNR，它不能求出 $x^2 \equiv a \pmod p$ 的解。为了求出这个二次等式的解，我们注意到一个素数既可以是p = 4k + 1也可以是p = 4k + 3，其中k是一个正整数。这个二次方程的解涉及第一种情况，第二种情况就非常容易了。我们只讨论第二种情况，这种情况我们将在第10章讨论Rabin密码系统时用到。

特殊情况：p = 4k + 3　如果p的形式是4k + 3(就是p　　mod 4)，　a是 Z_p^* 中的一个QR，那么

$$x \equiv a^{(p+1)/4} \pmod p \quad \text{且} \quad x \equiv -a^{(p+1)/4} \pmod p$$

例9.43

解下列二次方程：

$$(1)\ x^2 \equiv 3 \pmod{23}$$
$$(2)\ x^2 \equiv 2 \pmod{11}$$
$$(3)\ x^2 \equiv 7 \pmod{19}$$

解答

(1) 在第一个方程中，3是 Z_{23} 中的一个QR，解是 $x \equiv \pm 16 \pmod{23}$ 。也就是说，$\sqrt{3} \equiv \pm 16 \pmod{23}$ 。

这个知识点完美契合了这道题目，所以我就这么做了

```
In[ ]:= PowerMod[5 491 280 935 375 344 696 344 639 339 035 431 520 073 311 126 446 116 169 370 534 450 549 651 945 232,
        幂模
         (96 844 604 612 122 594 734 846 587 450 751 002 272 823 339 993 969 599 631 517 516 290 673 675 281 347 + 1) / 4,
         96 844 604 612 122 594 734 846 587 450 751 002 272 823 339 993 969 599 631 517 516 290 673 675 281 347]

Out[ ]= 96 844 604 612 122 594 734 846 587 450 748 673 989 604 439 470 234 591 202 189 447 038 449 025 024 582

In[ ]:= PowerMod[-5 491 280 935 375 344 696 344 639 339 035 431 520 073 311 126 446 116 169 370 534 450 549 651 945 232,
        幂模
         (96 844 604 612 122 594 734 846 587 450 751 002 272 823 339 993 969 599 631 517 516 290 673 675 281 347 + 1) / 4,
         96 844 604 612 122 594 734 846 587 450 751 002 272 823 339 993 969 599 631 517 516 290 673 675 281 347]

Out[ ]= 2 328 283 218 900 523 735 008 429 328 069 252 224 650 256 765
```

这两种都是可能情况，所以都要试一下

```
danis@ubuntu:~$ python ./rsa.py
hgame{easy_Crypto!}
```

得到flag

这周得益于上周的划水，我还是稍微适应了一下节奏。但是随之而来的问题是我落下了两周的PWN题，真的一道都没有做，中间知识点的空缺可以说是超级大了，无奈，只能摆正心态。第四周打算不肝misc和crypto了，还是学习pwn更加的迫切，把第四周的题目做出来怕是太难了，我打算先把前两周的pwn题做一做学一学，其它东西先放一放

另外就是我又累了，真的是不争气，从小水到大，不太习惯这种紧张的节奏。想想高考的那阵子疯狂打CS GO的我【笑哭】

加油吧，希望自己能够真的在比赛中学到知识，而不是整天玩游戏