REVERSE-PRACTICE-BUUCTF-6

原创

P1umH0 于 2021-02-26 23:02:31 发布 36 收藏

分类专栏： Reverse-BUUCTF

本文链接：https://blog.csdn.net/weixin_45582916/article/details/114155800

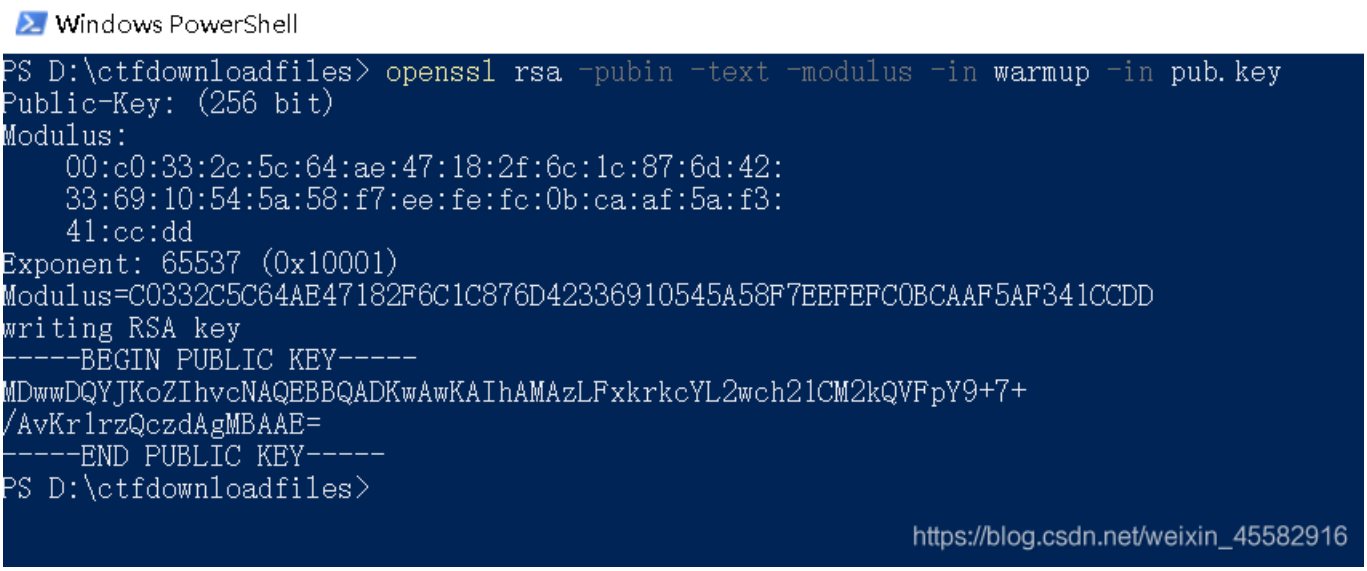
REVERSE-PRACTICE-BUUCTF-6

rsa

解压出来是.enc和.key两个文件，.enc是密文，.key存放着公钥信息

使用在线网站或者openssl解析.key文件中存放着的公钥信息

可获知rsa需要的模数n和公钥e



使用在线网站或者yafu分解n，结果为两个大素数

```
PS D:\ctfdownloadfiles\yafu> .\yafu-x64 "factor(@)" -batchfile n.txt


=== Starting work on batchfile expression ===
factor(0xC0332C5C64AE47182F6C1C876D42336910545A58F7EEFEFC0BCAAF5AF341CCDD)
==========================================
fac: factoring 869344822960481191906660620034948005889056560172030256172166540583783221035l7
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C77
rho: x^2 + 2, starting 1000 iterations on C77
rho: x^2 + 1, starting 1000 iterations on C77
pm1: starting B1 = 150K, B2 = gmp-ecm default on C77
ecm: 30/30 curves on C77, B1=2K, B2=gmp-ecm default
ecm: 74/74 curves on C77, B1=11K, B2=gmp-ecm default
ecm: 149/149 curves on C77, B1=50K, B2=gmp-ecm default, ETA: 0 sec

starting SIQS on c77: 869344822960481191906660620034948005889056560172030256172166540583783221035l7

==== sieving in progress (1 thread):   36224 relations needed ====
====           Press ctrl-c to abort and save state           ====
35999 rels found: 18788 full + 17211 from 185096 partial, (1897.45 rels/sec)

SIQS elapsed time = 109.1560 seconds.
Total factoring time = 122.3233 seconds


***factors found***

P39 = 304008741604601924494328155975272418463
P39 = 285960468890451637935629440372639283459

ans = 1

eof: done processing batchfile
PS D:\ctfdownloadfiles\yafu>
```


写脚本即可得到flag

```
#coding=utf-8
import gmpy2
from Crypto.Util.number import bytes_to_long,long_to_bytes
n=0xC0332C5C64AE47182F6C1C876D42336910545A58F7EEFEFC0BCAAF5AF341CCDD
e=0x10001
p=304008741604601924494328155975272418463
q=285960468890451637935629440372639283459
phin=(p-1)*(q-1)
d=gmpy2.invert(e,phin) #求e的逆元d，d即为私钥
with open("D:\\ctfdownloadfiles\\flag.enc") as f:
    c=f.read()
flag=bytes_to_long(c) #gmpy2.powmod()函数的参数不能是字符串，故将字符串转成整形
m=gmpy2.powmod(flag,d,n)
print(long_to_bytes(m))
```

```
est1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
  ♦ {zR♦♦    flag{decrypt_256}
```

CrackRTF

exe程序，运行后提示输入密码，输入错误直接退出，无壳，ida分析

main函数中，要求输入两次密码，先看第一个密码

第一个密码的部分逻辑清晰，重点是sub_40100A这个函数

```
int main_0()
{
  DWORD v0; // eax
  DWORD v1; // eax
  CHAR String; // [esp+4Ch] [ebp-310h]
  int v4; // [esp+150h] [ebp-20Ch]
  CHAR String1; // [esp+154h] [ebp-208h]
  BYTE input; // [esp+258h] [ebp-104h]

  memset(&input, 0, 0x104u);
  memset(&String1, 0, 0x104u);
  v4 = 0;
  printf("pls input the first passwd(1): ");
  scanf("%s", &input);                          // 获取输入
  if ( strlen((const char *)&input) != 6 )
  {
    printf("Must be 6 characters!\n");          // 输入必须是6个字符
    ExitProcess(0);
  }
  v4 = atoi((const char *)&input);
  if ( v4 < 100000 )                            // 简单的验证输入，输入字符串转成数字后必须大于等于100000且小于等于999999
    ExitProcess(0);
  strcat((char *)&input, "@DBApp");             // input_str+"@DBApp"
  v0 = strlen((const char *)&input);
  sub_40100A(&input, v0, &String1);             // 对拼接后的input做sha散列，结果存放到String1
  if ( !_strcmpi(&String1, "6E32D0943418C2C33385BC35A1470250DD8923A9") )
  {
    printf("continue...\n\n");
    printf("pls input the first passwd(2): ");
```


sub_40100A函数点进去，发现有个CryptCreateHash函数

CryptCreateHash函数详解

重要的是这个函数的第二个参数，决定了要使用的哈希算法

这里0x8004u使用的是sha（sha1）算法

```
int __cdecl sub_401230(BYTE *pbData, DWORD dwDataLen, LPSTR lpString1)
{
  int result; // eax
  DWORD i; // [esp+4Ch] [ebp-28h]
  CHAR String2; // [esp+50h] [ebp-24h]
  BYTE v6[20]; // [esp+54h] [ebp-20h]
  DWORD pdwDataLen; // [esp+68h] [ebp-Ch]
  HCRYPTHASH phHash; // [esp+6Ch] [ebp-8h]
  HCRYPTPROV phProv; // [esp+70h] [ebp-4h]

  if ( !CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
    return 0;
  if ( CryptCreateHash(phProv, 0x8004u, 0, 0, &phHash) )
  {
    if ( CryptHashData(phHash, pbData, dwDataLen, 0) )
    {
      CryptGetHashParam(phHash, 2u, v6, &pdwDataLen, 0);
      *lpString1 = 0;
      for ( i = 0; i < pdwDataLen; ++i )
      {
        wsprintfA(&String2, "%02X", v6[i]);
        lstrcatA(lpString1, &String2);
      }
      CryptDestroyHash(phHash);
      CryptReleaseContext(phProv, 0);
      result = 1;
```

```
    }
    else
    {
      CryptDestroyHash(phHash);
      CryptReleaseContext(phProv, 0);
      result = 0;
    }
  }
  else
```


写脚本爆破，得到第一个密码

```
#coding=utf-8
import hashlib
res="6E32D0943418C2C33385BC35A1470250DD8923A9"
res=res.lower()
pwd=100000
while pwd<=999999:
    s=str(pwd)+"@DBApp"
    h=hashlib.sha1()
    h.update(s.encode(encoding='utf-8'))
    hexdig=h.hexdigest() #这个返回的16进制摘要为小写字母，故已知的摘要也须先转成小写
    if hexdig==res:
        print(s)
        break
    else:
        pwd+=1
```

```
est1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
123321@DBApp
```


然后是第二个密码，和第一个密码的逻辑差不多，不过用的是md5散列，但是没有对第二个密码的6个字符做任何限制，爆破不可取，往下走，发现拼接后的input_2作为参数传入了sub_40100F函数

```
  printf("continue...\n\n");
  printf("pls input the first passwd(2): ");
  memset(&input_2, 0, 0x104u);
  scanf("%s", &input_2);                        // 获取第二个密码
  if ( strlen(&input_2) != 6 )
  {
    printf("Must be 6 characters!\n");          // 第二个密码也为6个字符
    ExitProcess(0);
  }
  strcat(&input_2, (const char *)&input);       // 在第二个密码后拼接“123321@DBApp”
  memset(&String1, 0, 0x104u);
  v1 = strlen(&input_2);
  sub_401019((BYTE *)&input_2, v1, &String1); // 对拼接后的input_2做md5散列，结果存放到String1
  if ( !_strcmpi("27019e688a4e62a649fd99cadaafdb4e", &String1) )// 比较已知的字符串和String1，但是之前对第二个密码没有任何限制，爆破不可取
  {
    if ( !sub_40100F(&input_2) )                // input_2作为参数传入该函数
    {
      printf("Error!!\n");
      ExitProcess(0);
    }
    printf("bye ~~\n");
  }
}
return 0;
```


sub_40100F函数的主要逻辑用红框标出，即exe程序带有的资源“AAA”和拼接后的input_2进行异或运算，结果写到程序创建的名为“dbapp.rtf”文件中

使用工具Resource Hacker可以获取资源“AAA”的数据

sub_401005函数就是进行异或运算

程序要创建并填充一个完整的rtf文件，rtf文件的文件头必不可少，搜索或自建一个空的rtf文件用010editor打开，可以得知rtf文件

的文件头

由于第二个密码拼接时放在input_2的前面，也就是说第二个密码和资源“AAA”的数据异或后的结果正是rtf文件的文件头

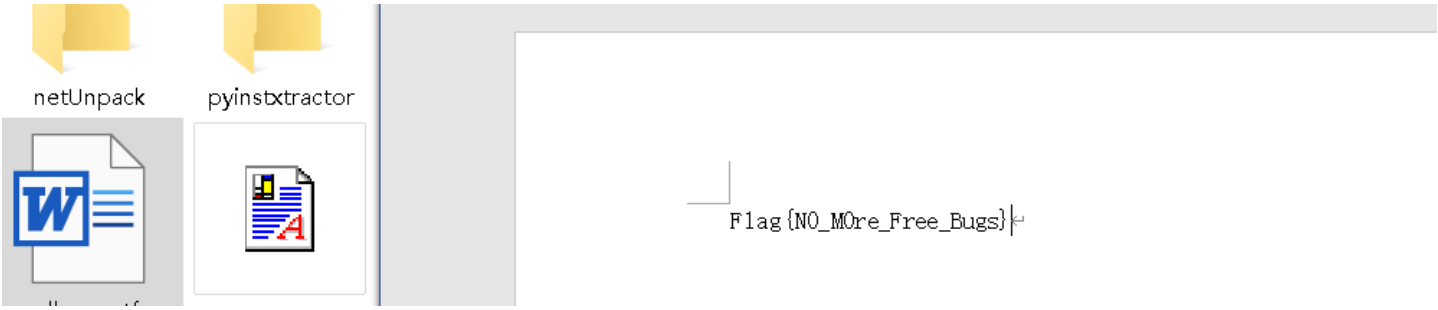
```
char __cdecl sub_4014D0(LPCSTR input_2)
{
  LPCVOID lpBuffer; // [esp+50h] [ebp-1Ch]
  DWORD NumberOfBytesWritten; // [esp+58h] [ebp-14h]
  DWORD nNumberOfBytesToWrite; // [esp+5Ch] [ebp-10h]
  HGLOBAL hResData; // [esp+60h] [ebp-Ch]
  HRSRC hResInfo; // [esp+64h] [ebp-8h]
  HANDLE hFile; // [esp+68h] [ebp-4h]

  hFile = 0;
  hResData = 0;
  nNumberOfBytesToWrite = 0;
  NumberOfBytesWritten = 0;
  hResInfo = FindResourceA(0, (LPCSTR)0x65, "AAA");
  if ( !hResInfo )
    return 0;
  nNumberOfBytesToWrite = SizeofResource(0, hResInfo);
  hResData = LoadResource(0, hResInfo);
  if ( !hResData )
    return 0;
  lpBuffer = LockResource(hResData);
  sub_401005(input_2, (int)lpBuffer, nNumberOfBytesToWrite);
  hFile = CreateFileA("dbapp.rtf", 0x10000000u, 0, 0, 2u, 0x80u, 0);
  if ( hFile == (HANDLE)-1 )
    return 0;
  if ( !WriteFile(hFile, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
    return 0;
  CloseHandle(hFile);
  return 1;
}
```


写脚本得到第二个密码，第二个密码的长度为6，故资源“AAA”和rtf文件头都取前6个字节异或即可得到第二个密码，注意“\r”在python中的转义语义，再加一个“\”

```
head="{\\rtf1"
res=[0x05,0x7d,0x41,0x15,0x26,0x01]
str=""
for i in range(len(head)):
    str+=chr(ord(head[i])^res[i])
print(str)
```

```
est2
  D:\python27-x64\python2.exe D:/Python/pycharm/pyc
  ~!3a@0
```


再次运行exe程序，输入正确的第一个和第二个密码，在当前目录下生成一个“dbapp.rtf”文件，内容即为flag



dbapp.rtf re.exe

[2019红帽杯]easyRE

elf文件，无壳，ida分析
左侧函数窗找不到主逻辑函数，shift+F12打开字符串窗口
发现一长段很像base64的字符串和base64字符表

Address	Length	Type	String
.rodata:…	000002E9	C	VmOwd2VHUXhTWGhpUm1SWVYwZDRWVll3Wkc5WFJsbDNXa1pPVlUxV2NIcFhhM…
.rodata:…	0000000A	C	continue!
.rodata:…	00000010	C	You found me!!!
.rodata:…	00000009	C	bye bye~
.rodata:…	00000041	C	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345678…
.rodata:…	00000014	C	../csu/libc-start.c
.rodata:…	00000017	C	FATAL: kernel too old\n

一路交叉引用base64字符串，来到sub_4009C6函数
先看该函数中用到base64字符串的片段
逻辑是v56经过10次base64变换，结果为已知的那个base64字符串

```
if ( v5 == 39 )
{
  v6 = (const __m128i *)sub_400E44((const __m128i *)&v56);
  v7 = (const __m128i *)sub_400E44(v6);
  v8 = (const __m128i *)sub_400E44(v7);
  v9 = (const __m128i *)sub_400E44(v8);
  v10 = (const __m128i *)sub_400E44(v9);
  v11 = (const __m128i *)sub_400E44(v10);
  v12 = (const __m128i *)sub_400E44(v11);
  v13 = (const __m128i *)sub_400E44(v12);
  v14 = (const __m128i *)sub_400E44(v13);
  v15 = sub_400E44(v14);
  v0 = base64_str;
  v1 = (char *)v15;
  if ( !(unsigned int)sub_400360(v15, base64_str) )
  {
    sub_410CC0("You found me!!!");
    v1 = "bye bye~";
    sub_410CC0("bye bye~");
  }
  result = 0LL;
}
```


写脚本得到v56，结果为一个url，打开没发现和flag相关的内容，应该是误导选手了

```
import base64
s="Vm0wd2VHUXhTWGhpUm1SWVYwZDRWVll3Wkc5WFJsbDNXa1pPVlUxV2NIcFhhMk0xVmpKS1NHVkdXbFp
i=1
while i<=10:
    s=base64.b64decode(s)
    i+=1
print(s)
```

```
test1
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
https://bbs.pediy.com/thread-254172.htm
```


继续看sub_4009C6函数的其他内容，该函数一开始给一堆变量赋值，然后有一个异或后比较的运算，用红框标出来了

```
109  if ( v2 == 36 )
110  {
111    for ( i = 0; ; ++i )
112    {
113      v1 = v53;
114      LODWORD(v4) = sub_424BA0((const __m128i *)v53);
115      if ( i >= v4 )
116        break;
117      if ( (unsigned __int8)(v53[i] ^ i) != *(&v17 + i) )
118      {
119        result = 0xFFFFFFFFLL;
120        goto LABEL_13;
121      }
122    }
123    sub_410CC0("continue!");
124    memset(&v56, 0, 0x40uLL);
125    v58 = 0;
126    v0 = &v56;
127    sub_4406E0(0LL, (__int64)&v56);
128    v57 = 0;
129    v1 = &v56;
130    LODWORD(v5) = sub_424BA0((const __m128i *)&v56);
131    if ( v5 == 39 )
```


写脚本，提示说前四个字符为“flag”，也没有对flag具体内容的提示

```
arr=[73,111,100,108,62,81,110,98,40,111,99,
     121,127,121,46,105,127,100,96,51,119,
     125,119,101,107,57,123,105,121,61,126,
     121,76,64,69,67]
s=""
for i in range(len(arr)):
    s+=chr(arr[i]^i)
print(s)
```

```
test1
D:\python27-x64\python2.exe D:/Python/pychar
Info:The first four chars are `flag`
```


sub_4009C6函数分析完后，并没有对flag具体内容的判断
于是再次去字符串窗口找找有没有其他提示的内容
在那串base64字符串下面，有一段没有在sub_4009C6函数中用到的数据

```
.data:00000000006CC090 base64_str      dq offset aVm0wd2vhuxhtwg
.data:00000000006CC090                                         ; DATA XREF: sub_4009C6+31B↑r
.data:00000000006CC090                                         ; "Vm0wd2VHUXhTWGhpUm1SWVYwZDRWVll3Wkc5WFJ"...
.data:00000000006CC098                 align 20h
.data:00000000006CC0A0 byte_6CC0A0     db 40h                  ; DATA XREF: sub_400D35+95↑r
.data:00000000006CC0A0                                         ; sub_400D35+C1↑r
.data:00000000006CC0A1                 db  35h ; 5
```

```
.data:00000000006CC0A2                   db  20h
.data:00000000006CC0A3 byte_6CC0A3       db 56h                  ; DATA XREF: sub_400D35+A6↑r
.data:00000000006CC0A4                   db  5Dh ; ]
.data:00000000006CC0A5                   db  18h
.data:00000000006CC0A6                   db  22h ; "
.data:00000000006CC0A7                   db  45h ; E
.data:00000000006CC0A8                   db  17h
.data:00000000006CC0A9                   db  2Fh ; /
.data:00000000006CC0AA                   db  24h ; $
.data:00000000006CC0AB                   db  6Eh ; n
.data:00000000006CC0AC                   db  62h ; b
.data:00000000006CC0AD                   db  3Ch ; <
.data:00000000006CC0AE                   db  27h ; '
.data:00000000006CC0AF                   db  54h ; T
.data:00000000006CC0B0                   db  48h ; H
.data:00000000006CC0B1                   db  6Ch ; l
.data:00000000006CC0B2                   db  24h ; $
.data:00000000006CC0B3                   db  6Eh ; n
.data:00000000006CC0B4                   db  72h ; r
.data:00000000006CC0B5                   db  3Ch ; <
.data:00000000006CC0B6                   db  32h ; 2
.data:00000000006CC0B7                   db  45h ; E
.data:00000000006CC0B8                   db  5Bh ; [
.data:00000000006CC0B9                   db    0
```


交叉引用来到sub_400D35函数

v5和v8相同，v8和byte_6CC0A0数组前4个字符异或的结果为“flag”，前面也提示到前4个字符为“flag”

然后v8再和byte_6CC0A0数组的全部元素异或

```
  v9 = __readfsqword(0x28u);
  v2 = 0LL;
  v5 = sub_43FD20(0LL) - qword_6CEE38;
  for ( i = 0; i <= 1233; ++i )
  {
    v2 = v5;
    sub_40F790(v5);
    sub_40FE60(v5);
    sub_40FE60(v5);
    v5 = (unsigned __int64)sub_40FE60(v5) ^ 2557891634;
  }
  v8 = v5;
  if ( ((unsigned __int8)v5 ^ byte_6CC0A0[0]) == 'f' && (HIBYTE(v8) ^ (unsigned __int8)byte_6CC0A3) == 'g' )
  {
    for ( j = 0; j <= 24; ++j )
    {
      v2 = (unsigned __int8)(byte_6CC0A0[j] ^ *((_BYTE *)&v8 + j % 4));
      sub_410E90(v2);
    }
  }
  v4 = __readfsqword(0x28u);
  result = v4 ^ v9;
  if ( v4 != v9 )
    sub_444020(v2, a2);
  return result;
}
```


写脚本，先解出v8，再循环异或得到flag

```
arr=[0x40, 0x35, 0x20, 0x56, 0x5D, 0x18, 0x22, 0x45, 0x17, 0x2F,
  0x24, 0x6E, 0x62, 0x3C, 0x27, 0x54, 0x48, 0x6C, 0x24, 0x6E,
  0x72, 0x3C, 0x32, 0x45, 0x5B]
v8=[0]*4
v8[0]=ord('f')^arr[0]
v8[1]=ord('l')^arr[1]
v8[2]=ord('a')^arr[2]
v8[3]=ord('g')^arr[3]
s=""
```

```
for i in range(len(arr)):
    s+=chr(arr[i]^v8[i%len(v8)])
print(s)
```

test1

```
D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
flag{Act1ve_Defen5e_Test}
```


[ACTF新生赛2020]easyre

exe程序，运行后提示输入，输入错误直接退出，有upx壳，脱壳后ida分析

main函数逻辑清晰，flag的内容减1后作为下标，从_data_start_这个数组中取值，与v4到v15比较，验证flag的内容

```
28  __main();
29  v4 = 42;
30  v5 = 70;
31  v6 = 39;
32  v7 = 34;
33  v8 = 78;
34  v9 = 44;
35  v10 = 34;
36  v11 = 40;
37  v12 = 73;
38  v13 = 63;
39  v14 = 43;
40  v15 = 64;
41  printf("Please input:");
42  scanf("%s", &input);
43  if ( (_BYTE)input != 'A' || HIBYTE(input) != 'C' || v20 != 'T' || v21 != 'F' || v22 != '{' || v26 != '}' )
44    return 0;
45  v16 = v23;
46  v17 = v24;
47  v18 = v25;
48  for ( i = 0; i <= 11; ++i )
49  {
50    if ( *(&v4 + i) != _data_start__[*((char *)&v16 + i) - 1] )
51      return 0;
52  }
53  printf("You are correct!");
54  return 0;
55}
```


写逆脚本即可得到flag

```
arr=[42, 70, 39, 34, 78, 44, 34, 40, 73, 63, 43, 64]
data_start=[0x7E, 0x7D, 0x7C, 0x7B, 0x7A, 0x79, 0x78, 0x77, 0x76, 0x75,
  0x74, 0x73, 0x72, 0x71, 0x70, 0x6F, 0x6E, 0x6D, 0x6C, 0x6B,
  0x6A, 0x69, 0x68, 0x67, 0x66, 0x65, 0x64, 0x63, 0x62, 0x61,
  0x60, 0x5F, 0x5E, 0x5D, 0x5C, 0x5B, 0x5A, 0x59, 0x58, 0x57,
  0x56, 0x55, 0x54, 0x53, 0x52, 0x51, 0x50, 0x4F, 0x4E, 0x4D,
  0x4C, 0x4B, 0x4A, 0x49, 0x48, 0x47, 0x46, 0x45, 0x44, 0x43,
  0x42, 0x41, 0x40, 0x3F, 0x3E, 0x3D, 0x3C, 0x3B, 0x3A, 0x39,
  0x38, 0x37, 0x36, 0x35, 0x34, 0x33, 0x32, 0x31, 0x30, 0x2F,
  0x2E, 0x2D, 0x2C, 0x2B, 0x2A, 0x29, 0x28, 0x27, 0x26, 0x25,
  0x24, 0x23, 0x20, 0x21, 0x22]
s=""
for i in range(len(arr)):
    for j in range(len(data_start)):
        if data_start[j]==arr[i]:
            s+=chr(j+1)
print(s)
```

```
est1
  D:\python27-x64\python2.exe D:/Python/pycharm/pycfile/test1.py
  U9X_1S_W6@T?
```