buuctf刷题记录

原创

mackilo 于 2022-02-19 23:21:50 发布 51 收藏

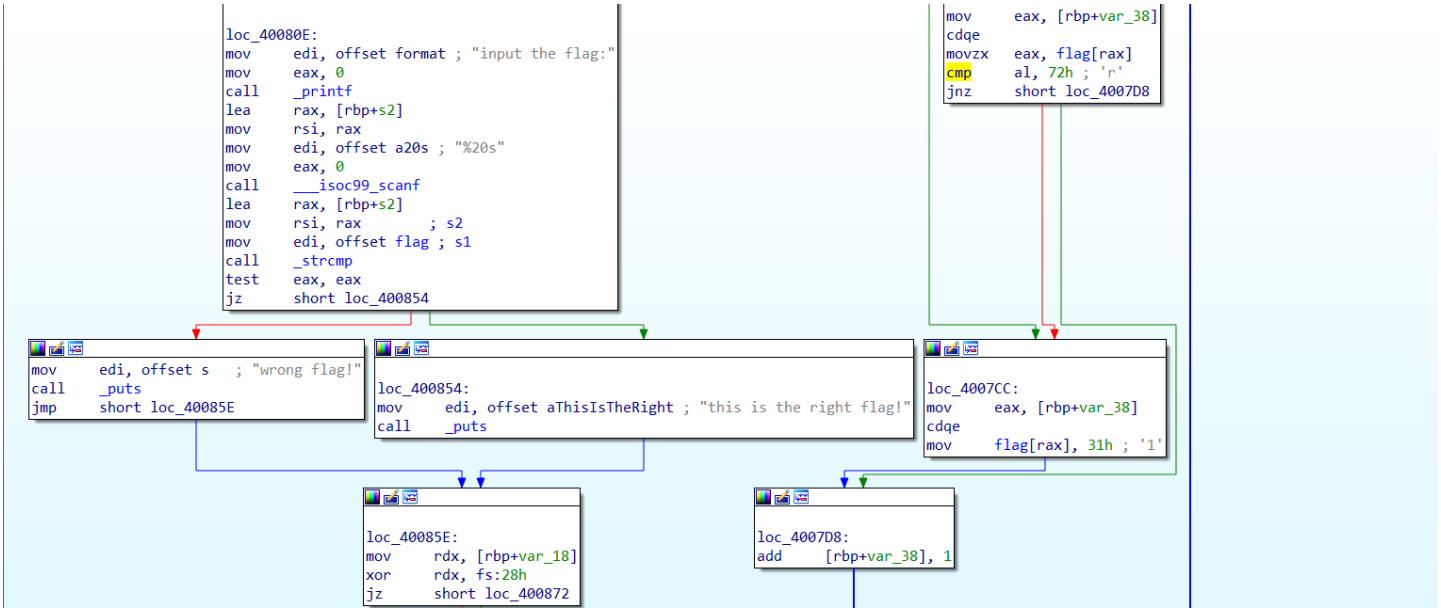
文章标签： 安全

本文链接：https://blog.csdn.net/mackilo/article/details/123025588

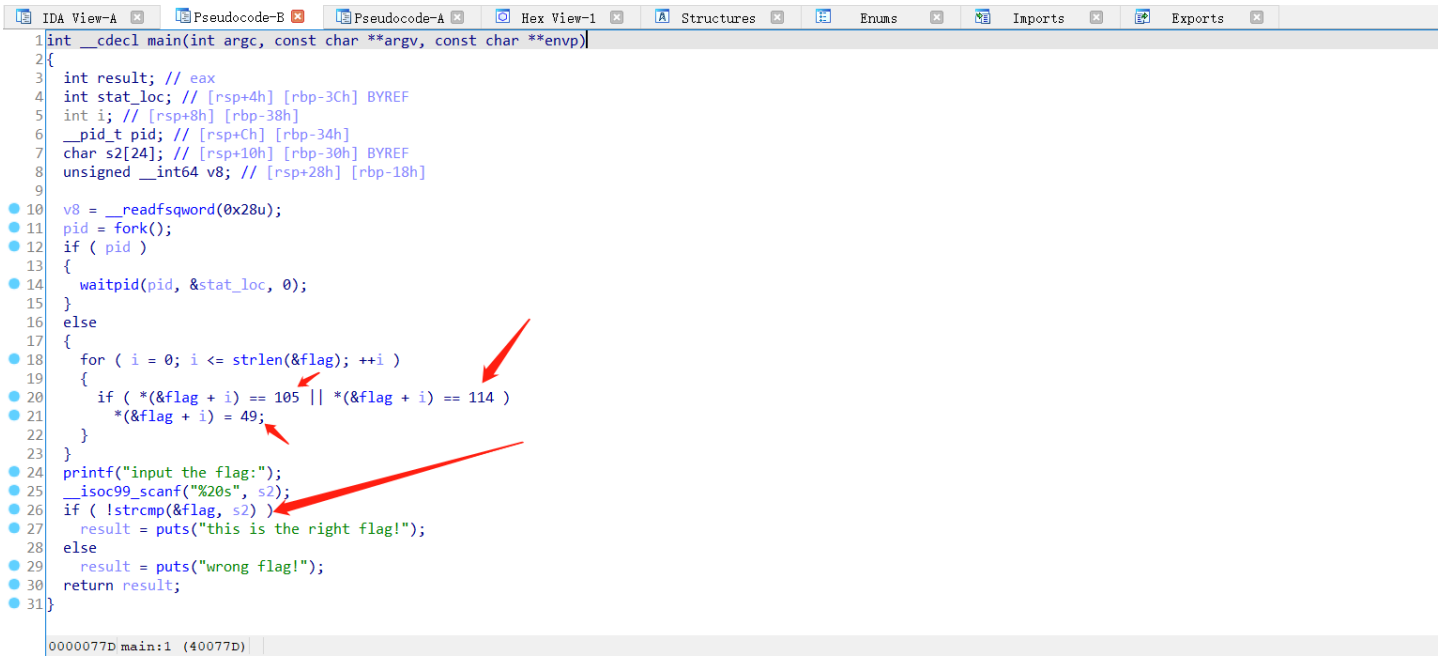
2022-1-16

reverse2

扔到IDA64里解析，

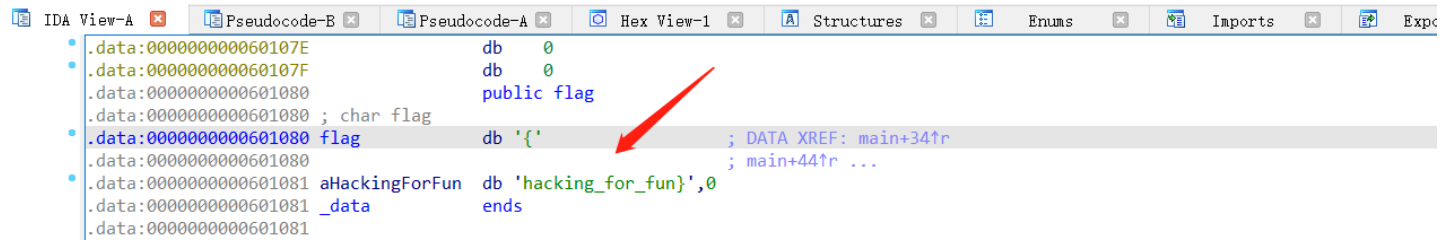


F5生成伪代码

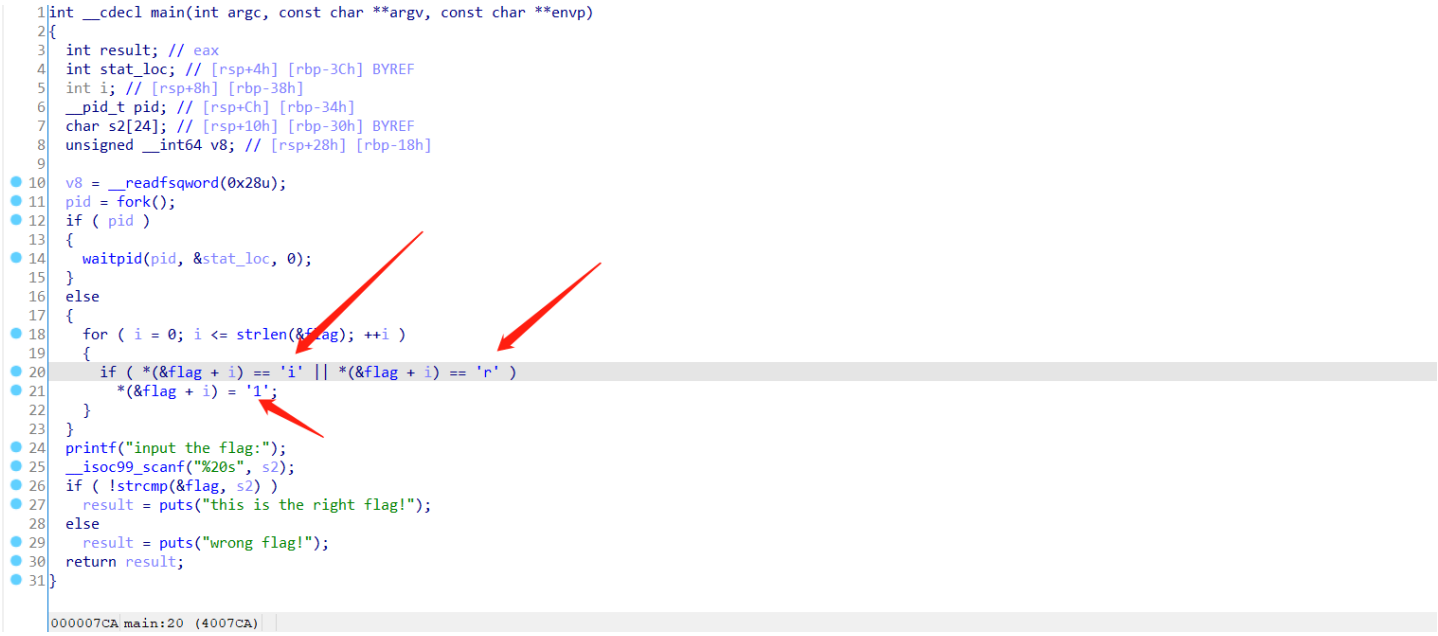


通过分析得出经过一定的变换后与flag进行比较的

查看flag的值，是{hacking_for_fun}



再返回前面看如何对flag进行变换的，将105、114、49转换成值，



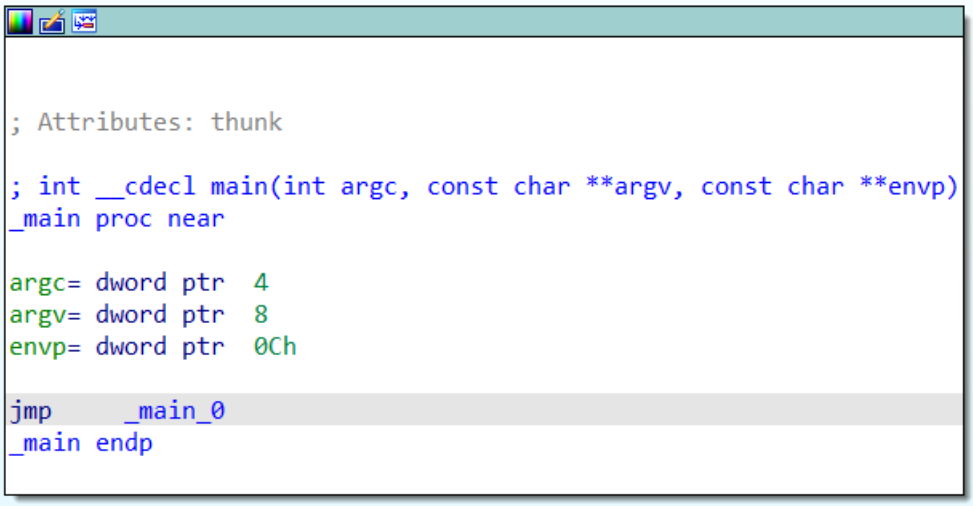
箭头所指的函数功能为，把flag中存在的'i'以及'r'转换成'1'，进而得出正确flag。

2022-1-17

内涵的软件

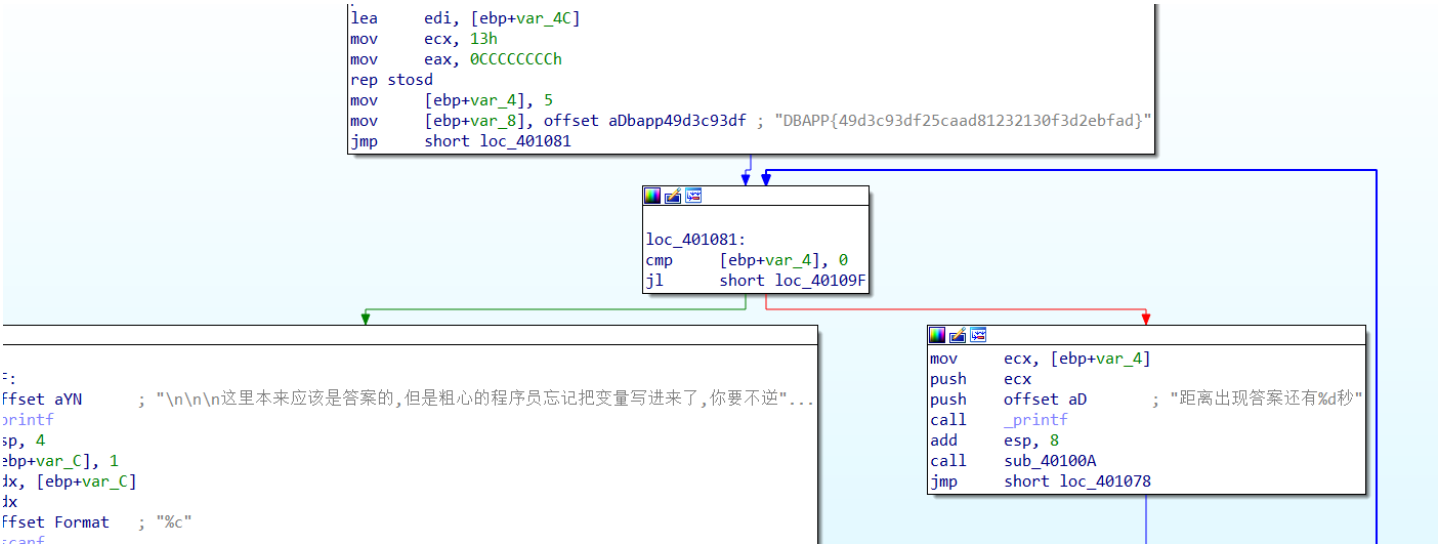
下载好以后发现是乱码的exe文件，拖到ExeinfoPE里查看信息，

没有加壳，用IDA打开，查看main函数



```
; Attributes: thunk

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

argc= dword ptr  4
argv= dword ptr  8
envp= dword ptr  0Ch

jmp     _main_0
_main endp
```

可以看到跳转到main0函数里 打开main0函数



F5看伪代码，

IDA View-A | Pseudocode-B | Pseudocode-A | Hex View-1 | Structures | Enums | Import

```
int __cdecl main_0(int argc, const char **argv, const char **envp)
{
  int result; // eax
  char v4[4]; // [esp+4Ch] [ebp-Ch] BYREF
  const char *v5; // [esp+50h] [ebp-8h]
  int v6; // [esp+54h] [ebp-4h]

  v6 = 5;
  v5 = "DBAPP{49d3c93df25caad81232130f3d2ebfad}";
  while ( v6 >= 0 )
  {
    printf(aD, v6);
    sub_40100A();
    --v6;
  }
  printf(
    "\n"
    "\n"
    "\n"
    "这里本来应该是答案的,但是粗心的程序员忘记把变量写进来了,你要不逆向试试看:(Y/N)\n");
  v4[0] = 1;
  scanf("%c", v4);
  if ( v4[0] == 89 )
  {
    printf(aOdIda);
    result = sub_40100A();
  }
  else
  {
    if ( v4[0] == 78 )
      printf(asc_425034);
    else
```

看到一串字符串，直接尝试，就是了，不得不吐槽一下，这字符串我以为还是加密的呢，然而并不是。

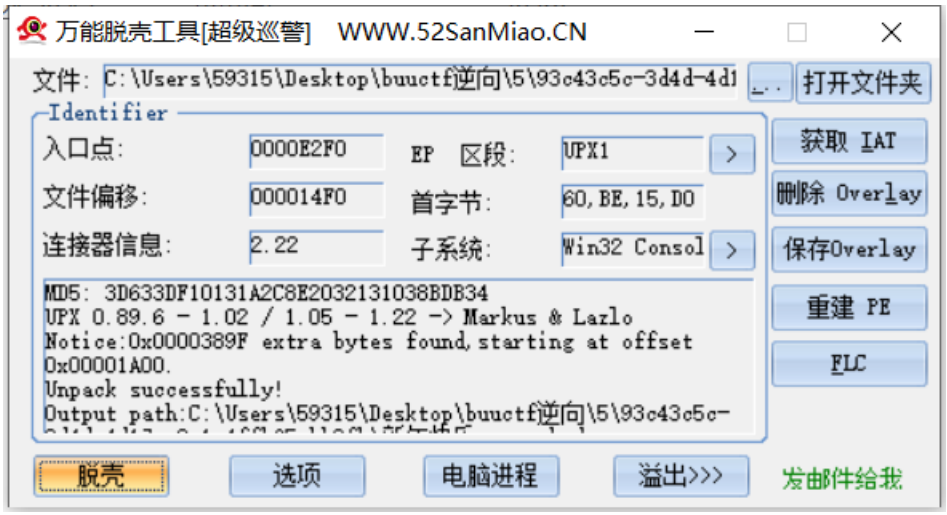
2022-1-18

新年快乐

先放到ExeinfoPE里查看

发现是加壳的

使用脱壳工具脱一下壳



脱壳后再扔到IDA里，F5查看，

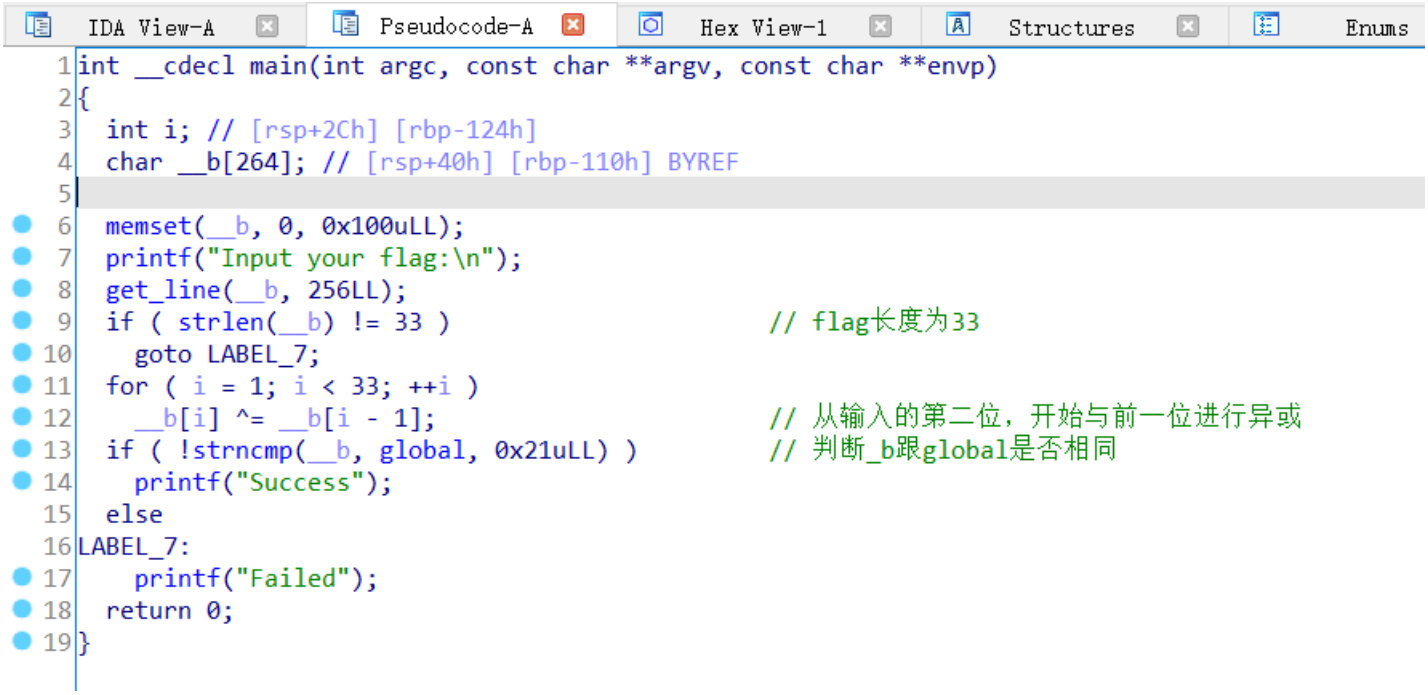
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax
  char Str2[14]; // [esp+12h] [ebp-3Ah] BYREF
  __int16 Str1; // [esp+20h] [ebp-2Ch] BYREF
  _BYTE v6[30]; // [esp+22h] [ebp-2Ah] BYREF

  sub_401910();
  strcpy(Str2, "HappyNewYear!");
  Str1 = word_40306B;
  memset(v6, 0, sizeof(v6));
  printf("please input the true flag:");
  scanf("%s", &Str1);
  if ( !strncmp((const char *)&Str1, Str2, strlen(Str2)) )
    result = puts(aThisIsTrueFlag);
  else
    result = puts(Buffer);
  return result;
}
```

代码逻辑如图，比对str1和str2，str2的值直接就给了。就是"HappyNewYear!"。

2022-1-19

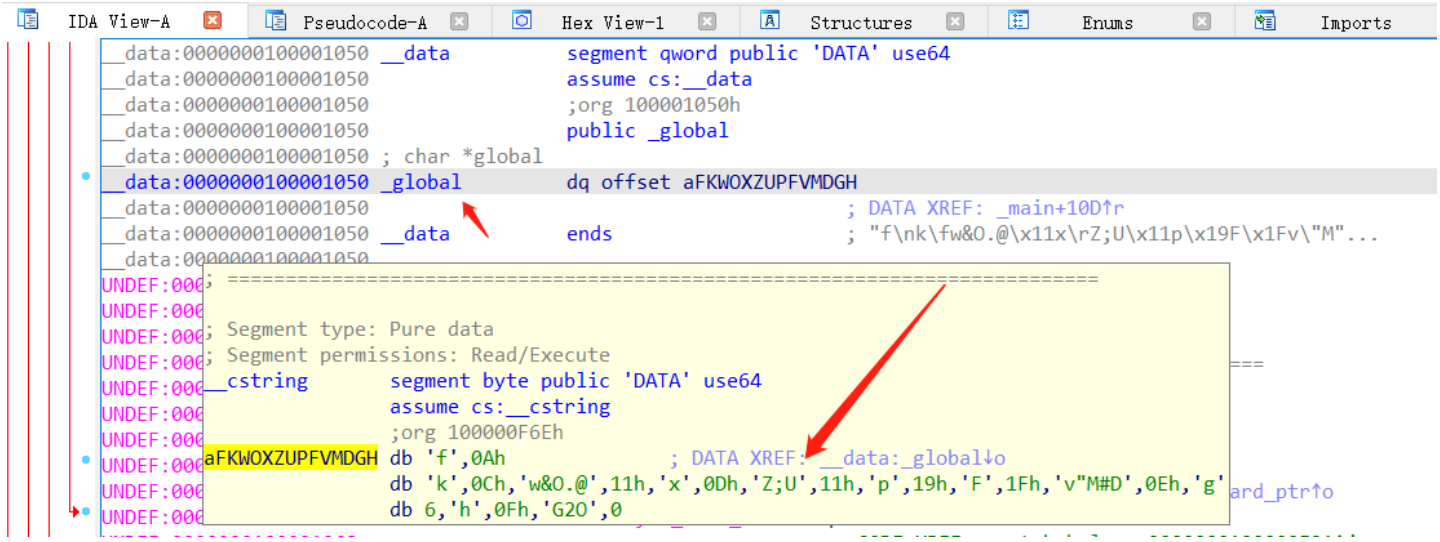
XOR

下载下来是个无后缀文件，扔到IDA64里解析，看到main函数，直接F5，



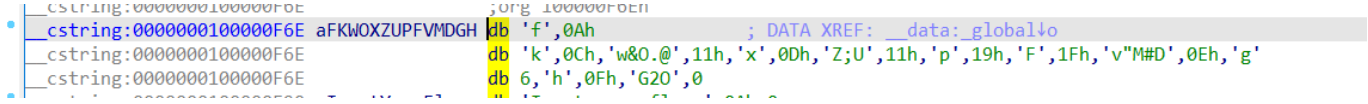
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int i; // [rsp+2Ch] [rbp-124h]
  char __b[264]; // [rsp+40h] [rbp-110h] BYREF

  memset(__b, 0, 0x100uLL);
  printf("Input your flag:\n");
  get_line(__b, 256LL);
  if ( strlen(__b) != 33 )                    // flag长度为33
    goto LABEL_7;
  for ( i = 1; i < 33; ++i )
    __b[i] ^= __b[i - 1];                     // 从输入的第二位，开始与前一位进行异或
  if ( !strncmp(__b, global, 0x21uLL) )       // 判断_b跟global是否相同
    printf("Success");
  else
LABEL_7:
    printf("Failed");
  return 0;
}
```

看到是对_b跟global做对比，



```
__data:0000000100001050 __data          segment qword public 'DATA' use64
__data:0000000100001050                 assume cs:__data
__data:0000000100001050                 ;org 100001050h
__data:0000000100001050                 public _global
__data:0000000100001050 ; char *global
__data:0000000100001050 _global         dq offset aFKWOXZUPFVMDGH
__data:0000000100001050                                         ; DATA XREF: _main+10D↑r
__data:0000000100001050 __data          ends                    ; "f\nk\fw&O.@\x11x\rZ;U\x11p\x19F\x1Fv\"M"...
__data:0000000100001050
; ===========================================================================

; Segment type: Pure data
; Segment permissions: Read/Execute
__cstring       segment byte public 'DATA' use64
                assume cs:__cstring
                ;org 100000F6Eh
aFKWOXZUPFVMDGH db 'f',0Ah              ; DATA XREF: __data:_global↓o
                db 'k',0Ch,'w&O.@',11h,'x',0Dh,'Z;U',11h,'p',19h,'F',1Fh,'v"M#D',0Eh,'g'
                db 6,'h',0Fh,'G2O',0
```

所以看一下global什么内容，跟进是字符串。



```
__cstring:0000000100000F6E aFKWOXZUPFVMDGH db 'f',0Ah              ; DATA XREF: __data:_global↓o
__cstring:0000000100000F6E                 db 'k',0Ch,'w&O.@',11h,'x',0Dh,'Z;U',11h,'p',19h,'F',1Fh,'v"M#D',0Eh,'g'
__cstring:0000000100000F6E                 db 6,'h',0Fh,'G2O',0
```

写一个python脚本

```
s = ['f',0xA,'k',0xC,'w&O.@',0x11,'x',0xD,'Z;U',0x11,'p',0x19,'F',0x1F,'v"M#D',0xE,'g',6,'h',0xF,'G2O',0]

for i in range(1,len(s)):

    if(isinstance(s[i],int)):            #如果为int类型，转换为char类型

        s[i]=chr(s[i])

s = ''.join(s)                           #将列表拼接成字符串

flag = 'f'                               #flag的第一位不进行异或，所以直接写为f

for i in range(1,len(s)):

    flag += chr(ord(s[i])^ord(s[i-1]))    #异或操作，两次异或会还原

print(flag)
```

2022-1-20

reverse3

下载下来是reverse_3.exe，扔到ExeinfoPE里，没加壳，扔到IDA里，看见了main0，直接F5，

```
int __cdecl main_0(int argc, const char **argv, const char **envp)

{

  size_t v3; // eax

  const char *v4; // eax

  size_t v5; // eax

  char v7; // [esp+0h] [ebp-188h]

  char v8; // [esp+0h] [ebp-188h]

  signed int j; // [esp+DCh] [ebp-ACh]

  int i; // [esp+E8h] [ebp-A0h]

  signed int v11; // [esp+E8h] [ebp-A0h]

  char Destination[108]; // [esp+F4h] [ebp-94h] BYREF

  char Str[28]; // [esp+160h] [ebp-28h] BYREF

  char v14[8]; // [esp+17Ch] [ebp-Ch] BYREF



  for ( i = 0; i < 100; ++i )

  {

    if ( (unsigned int)i >= 0x64 )

      j____report_rangecheckfailure();
```

```
  Destination[i] = 0;
}
sub_41132F("please enter the flag:", v7);
sub_411375("%20s", (char)Str);
v3 = j_strlen(Str);
v4 = (const char *)sub_4110BE(Str, v3, v14);
strncpy(Destination, v4, 0x28u);
v11 = j_strlen(Destination);
for ( j = 0; j < v11; ++j )
  Destination[j] += j;
v5 = j_strlen(Destination);
if ( !strncmp(Destination, Str2, v5) )
  sub_41132F("rigth flag!\n", v8);
else
  sub_41132F("wrong flag!\n", v8);
return 0;
}
```

看出来是输入一个字符串然后经过sub_4110BE函数进行加密，然后再通过一个for循环进行变换，然后与str2进行比较。先看一下str2的值，

```
.data:0041A030 dword_41A030    dd 1                    ; DATA XREF: ___scrt_is_ucrt_dll_in_use+4
.data:0041A034 ; char Str2[]
.data:0041A034 Str2            db 'e3nifIH9b_C@n@dH',0 ; DATA XREF: _main_0+142↑o
```

看一下sub_4110BE函数，

```
void *__cdecl sub_4110BE(char *a1, unsigned int a2, int *a3)
{
  int v4; // [esp+D4h] [ebp-38h]
  int v5; // [esp+D4h] [ebp-38h]
  int v6; // [esp+D4h] [ebp-38h]
  int v7; // [esp+D4h] [ebp-38h]
  int i; // [esp+E0h] [ebp-2Ch]
  unsigned int v9; // [esp+ECh] [ebp-20h]
```

```
unsigned int v9; // [esp+ECh] [ebp-20h]
int v10; // [esp+ECh] [ebp-20h]
int v11; // [esp+ECh] [ebp-20h]
void *v12; // [esp+F8h] [ebp-14h]
char *v13; // [esp+104h] [ebp-8h]

if ( !a1 || !a2 )
  return 0;
v9 = a2 / 3;
if ( (int)(a2 / 3) % 3 )
  ++v9;
v10 = 4 * v9;
*a3 = v10;
v12 = malloc(v10 + 1);
if ( !v12 )
  return 0;
j_memset(v12, 0, v10 + 1);
v13 = a1;
v11 = a2;
v4 = 0;
while ( v11 > 0 )
{
  byte_41A144[2] = 0;
  byte_41A144[1] = 0;
  byte_41A144[0] = 0;
  for ( i = 0; i < 3 && v11 >= 1; ++i )
  {
    byte_41A144[i] = *v13;
    --v11;
    ++v13;
```

```
  }
  if ( !i )
    break;
  switch ( i )
  {
    case 1:
      *((_BYTE *)v12 + v4) = aAbcdefghijklmn[(int)(unsigned __int8)byte_41A144[0] >> 2];
      v5 = v4 + 1;
      *((_BYTE *)v12 + v5) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | (16 * (byte_41A144[0] & 3))];
      *((_BYTE *)v12 + ++v5) = aAbcdefghijklmn[64];
      *((_BYTE *)v12 + ++v5) = aAbcdefghijklmn[64];
      v4 = v5 + 1;
      break;
    case 2:
      *((_BYTE *)v12 + v4) = aAbcdefghijklmn[(int)(unsigned __int8)byte_41A144[0] >> 2];
      v6 = v4 + 1;
      *((_BYTE *)v12 + v6) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | (16 * (byte_41A144[0] & 3))];
      *((_BYTE *)v12 + ++v6) = aAbcdefghijklmn[((byte_41A144[2] & 0xC0) >> 6) | (4 * (byte_41A144[1] & 0xF))];
      *((_BYTE *)v12 + ++v6) = aAbcdefghijklmn[64];
      v4 = v6 + 1;
      break;
    case 3:
      *((_BYTE *)v12 + v4) = aAbcdefghijklmn[(int)(unsigned __int8)byte_41A144[0] >> 2];
      v7 = v4 + 1;
      *((_BYTE *)v12 + v7) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | (16 * (byte_41A144[0] & 3))];
      *((_BYTE *)v12 + ++v7) = aAbcdefghijklmn[((byte_41A144[2] & 0xC0) >> 6) | (4 * (byte_41A144[1] & 0xF))];
      *((_BYTE *)v12 + ++v7) = aAbcdefghijklmn[byte_41A144[2] & 0x3F];
      v4 = v7 + 1;
      break;
  }
```

```
  }

  *((_BYTE *)v12 + v4) = 0;

  return v12;

}
```

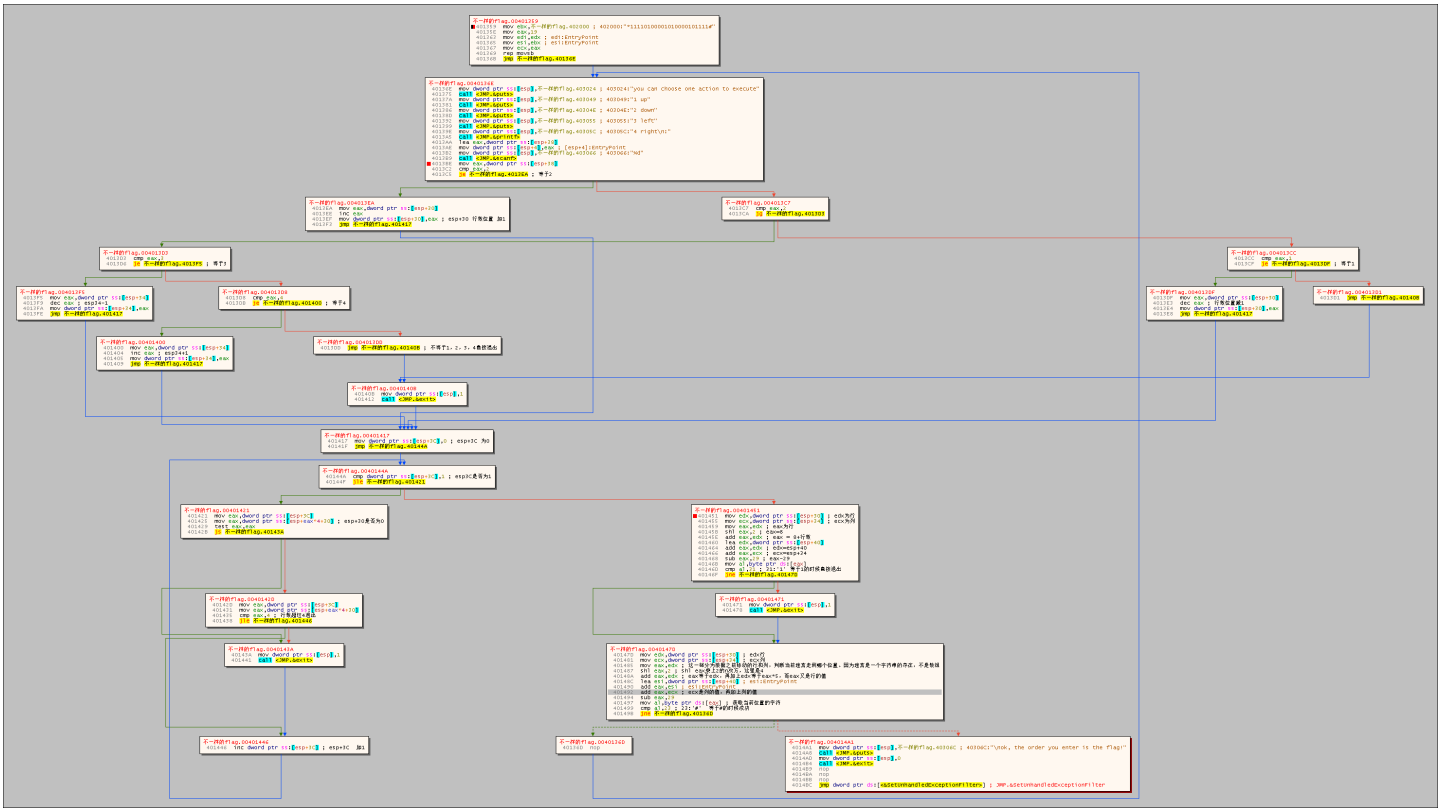
看上去是base64加密，写个python解一下。

```
import base64

str = 'e3nifIH9b_C@n@dH'

flag = ''

for i in range(len(str)):

    flag += chr(ord(str[i])-i)



print(base64.b64decode(flag))
```

2022-1-21

不一样的flag

下载下来是"不一样的flag.exe"，扔到ExeinfoPE，未加壳，扔到IDA里，看一下伪代码，此处贴一下看的别人的伪代码分析，



说实话一开始没看明白，乱七八糟的，后来看了解析发现是个迷宫题，将49跟35转成值，发现是1跟#，大概想了一下，结尾是#，就输出正确，然后就看一下字符串是以#结尾，但是还是不知道是个啥，不过说起来做这种题，也从来没有想过直接打开xxx.exe，直接打开发现是可以操纵上下左右的，看了解析才知道是一个迷宫，将字符串分为5行5列（留个疑问，为什么是5行5列）。

```
*1111

01000

01010

00010

1111#
```

贴一下伪代码

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)

{

  char v3[29]; // [esp+17h] [ebp-35h] BYREF

  int v4; // [esp+34h] [ebp-18h]

  int v5; // [esp+38h] [ebp-14h] BYREF

  int i; // [esp+3Ch] [ebp-10h]

  char v7[12]; // [esp+40h] [ebp-Ch] BYREF
```

```
  __main();
  v4 = 0;
  strcpy(v3, "*11110100001010000101111#");
  while ( 1 )
  {
    puts("you can choose one action to execute");
    puts("1 up");
    puts("2 down");
    puts("3 left");
    printf("4 right\n:");
    scanf("%d", &v5);
    if ( v5 == 2 )
    {
      ++*(_DWORD *)&v3[25];
    }
    else if ( v5 > 2 )
    {
      if ( v5 == 3 )
      {
        --v4;
      }
      else
      {
        if ( v5 != 4 )
LABEL_13:
          exit(1);
        ++v4;
      }
    }
    else
```

```
  {
    if ( v5 != 1 )
      goto LABEL_13;
    --*(_DWORD *)&v3[25];
  }
  for ( i = 0; i <= 1; ++i )
  {
    if ( *(int *)&v3[4 * i + 25] < 0 || *(int *)&v3[4 * i + 25] > 4 )
      exit(1);
  }
  if ( v7[5 * *(_DWORD *)&v3[25] - 41 + v4] == 49 )
    exit(1);
  if ( v7[5 * *(_DWORD *)&v3[25] - 41 + v4] == 35 )
  {
    puts("\nok, the order you enter is the flag!");
    exit(0);
  }
 }
}
```

*号为开头，只能走0，走1直接就退出，操纵的顺序即为flag，222441144222。

2022-1-22

SimpleRev

下载下来是SimpleRev，甩到IDA64里进行分析。直接F5，

IDA View-A | Pseudocode-A | Hex View-1 | Structures | Enums | Imports

```
1  int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
2  {
3    int v3; // eax
4    char v4; // [rsp+Fh] [rbp-1h]
5
6    while ( 1 )
7    {
8      while ( 1 )
9      {
10       printf("Welcome to CTF game!\nPlease input d/D to start or input q/Q to quit this program: ");
11       v4 = getchar();
12       if ( v4 != 100 && v4 != 68 )
13         break;
14       Decry("Welcome to CTF game!\nPlease input d/D to start or input q/Q to quit this program: ", argv);
15     }
16     if ( v4 == 113 || v4 == 81 )
17       Exit("Welcome to CTF game!\nPlease input d/D to start or input q/Q to quit this program: ", argv);
18     puts("Input fault format!");
19     v3 = getchar();
20     putchar(v3);
21   }
22 }
```

发现输入d/D就可以进入，紧接着是一个Decry函数，我们对Decry函数进行分析

```
unsigned __int64 Decry()

{

  char v1; // [rsp+Fh] [rbp-51h]

  int v2; // [rsp+10h] [rbp-50h]

  int v3; // [rsp+14h] [rbp-4Ch]

  int i; // [rsp+18h] [rbp-48h]

  int v5; // [rsp+1Ch] [rbp-44h]

  char src[8]; // [rsp+20h] [rbp-40h]

  __int64 v7; // [rsp+28h] [rbp-38h]

  int v8; // [rsp+30h] [rbp-30h]

  __int64 v9; // [rsp+40h] [rbp-20h]

  __int64 v10; // [rsp+48h] [rbp-18h]

  int v11; // [rsp+50h] [rbp-10h]

  unsigned __int64 v12; // [rsp+58h] [rbp-8h]



  v12 = __readfsqword(0x28u);

  *(_QWORD *)src = 0x534C43444ELL;

  v7 = 0LL;
```

```
v8 = 0;
v9 = 0x776F646168LL;                            //数据在内存中是小端顺序，高位在高地址处，低位在低地址处
                                                //故实际的字符顺序应为'0x4e44434c53'转为字符为'NDCLS'
v10 = 0LL;
v11 = 0;
text = join(key3, (const char *)&v9);           // 让text等于key3+v9
                                                // key3 = "kills"
                                                // v9   = "hadow"
                                                // 因为小端序存储
                                                // 则text = "killshadow"
strcpy(key, key1);                              // 将key1复制给key
                                                // key = "ADSFK"
strcat(key, src);                               // 将src处的字符串拼接到key后
                                                // key = "ADSFKNDCLS"
v2 = 0;
v3 = 0;
getchar();                                      // 获取输入（清空缓冲区？）
v5 = strlen(key);                               // v5 = key的长度   v5 = 10
for ( i = 0; i < v5; ++i )
{
  if ( key[v3 % v5] > 64 && key[v3 % v5] <= 90 )// if(key[v3]>64 && key[v3]<=90)
                                                // key[v3] = key[v3]+32
                                                // : 将大写字母转换成小写字母
    key[i] = key[v3 % v5] + 32;                 // key = "adsfkndcls"
  ++v3;
}
printf("Please input your flag:", src);
while ( 1 )
{
  v1 = getchar();
```

```
  if ( v1 == '\n' )                             // 如果输入的为换行符，则退出
    break;
  if ( v1 == ' ' )
  {
    ++v2;                                       // 如果输入的为空格，则v2加一
  }
  else
  {
    if ( v1 <= 96 || v1 > 122 )                 // 如果输入的v1不为小写字母
    {
      if ( v1 > 64 && v1 <= 90 )                // 如果v1为大写字母
        str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;// 对str2[v2]进行处理（v2为0每次加1）
                                                // str2[v2] = (v1-key[v3]+58)%26 + 97
                                                // 变换后str2[v2]存放小写字母
    }
    else
    {                                           // 如果输入的值v1为小写字母
      str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;// 做同样处理
    }                                           // 如果不为大小写字母，则不进行处理
    if ( !(v3 % v5) )                           // 如果循环到key的最后一位
      putchar(' ');                             // 打印处一个空格
    ++v2;
  }
}
if ( !strcmp(text, str2) )                      // 如果text和str2存储的相同，则成功
                                                // text = "killshadow"
  puts("Congratulation!\n");
else
  puts("Try again!\n");
return __readfsqword(0x28u) ^ v12;
```

```
}
```

其中在处理text时，有一个函数为join，即text = join

&v9);，我们去查看一下

```
char *__fastcall join(const char *a1, const char *a2)

{

  size_t v2; // rbx

  size_t v3; // rax

  char *dest; // [rsp+18h] [rbp-18h]



  v2 = strlen(a1);                      //v2为a1的长度，即key的长度

  v3 = strlen(a2);                      //v3为a2的长度，即v9的长度

  dest = (char *)malloc(v2 + v3 + 1); //动态分配一个能存下key3和v9的空间dest

  if ( !dest )

    exit(1);

  strcpy(dest, a1);                     //将a1的值赋给dest

  strcat(dest, a2);                     //将a2的值拼接到dest

  return dest;                          //实则返回a1+a2,即key+v9

}
```

其中对于str2做的处理为

```
str2[v2] = (v1 - 39 - key[v3 % v5] + 97) % 26 + 97;
```

所以我们要做的就是对于str2进行逆向处理，写出脚本

```
#第一种 逆推

text = 'killshadow'  # str2

key = 'adsfkndcls'

v3 = 0

v5 = len(key)

n = 0

flag = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

for i in range(0, 10):
```

```
    for j in range(0, 10):
        v1 = (ord(text[j])-97)+26*i+ord(key[v3 % v5])-58
        if(65 < v1 <= 90) or (v1 >= 97 and v5 <= 122):
            flag[j] = chr(v1)
            n = n+1
            if n == 10:
                print(''.join(flag))
                break
        v3 = v3 + 1
#第二种 暴力字典破解
text = 'killshadow'  # str2
key = 'adsfkndcls'
v3 = 0
v5 = len(key)
dict1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
flag = ""
for i in range(0, 10):
    n = 0
    for char in dict1:
        x = (ord(char) - 39 - ord(key[v3 % v5])+97) % 26 + 97
        if chr(x) == text[i]:
            n = n + 1
            if n == 1:
                print(char, end="")
    v3 = v3 + 1
```

2022-1-23

[GXYCTF2019]luck_guy

下载下来是名为luck_guy的文件，扔到IDA里，直接F5,

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  welcome(argc, argv, envp);

  puts("_________________");

  puts("try to patch me and find flag");

  puts("please input a lucky number");

  __isoc99_scanf("%d");

  patch_me(0LL);

  puts("OK,see you again");

  return 0;

}
```

可以看到，存在一个patch_me的函数，跟踪

```
int __fastcall patch_me(int a1)

{

  int result; // eax



  if ( a1 % 2 == 1 )

    result = puts("just finished");

  else

    result = get_flag();

  return result;

}
```

肯定是get_flag

有用，跟踪

```
unsigned __int64 get_flag()

{

  unsigned int v0; // eax

  int i; // [rsp+4h] [rbp-3Ch]

  int j; // [rsp+8h] [rbp-38h]
```

```
int j; // [rsp+8h] [rbp-38h]
__int64 s; // [rsp+10h] [rbp-30h] BYREF
char v5; // [rsp+18h] [rbp-28h]
unsigned __int64 v6; // [rsp+38h] [rbp-8h]

v6 = __readfsqword(0x28u);
v0 = time(0LL);                //得到时间
srand(v0);                     //使用时间作为种子生成随机数字
for ( i = 0; i <= 4; ++i )
{
  switch ( rand() % 200 )      // 产生1-199之间的随机数
  {
    case 1:
      puts("OK, it's flag:");
      memset(&s, 0, 0x28uLL);
      strcat((char *)&s, f1);
      strcat((char *)&s, &f2);
      printf("%s", (const char *)&s);
      break;
    case 2:
      printf("Solar not like you");
      break;
    case 3:
      printf("Solar want a girlfriend");
      break;
    case 4:
      s = 0x7F666F6067756369LL;
      v5 = 0;
      strcat(&f2, (const char *)&s);
      break;
```

```
    case 5:
      for ( j = 0; j <= 7; ++j )
      {
        if ( j % 2 == 1 )
          *(&f2 + j) -= 2;
        else
          --*(&f2 + j);
      }
      break;
    default:
      puts("emmm,you can't find flag 23333");
      break;
  }
 }
 return __readfsqword(0x28u) ^ v6;
}
```

可以看出来是随机组合，最后得出flag。flag的拼接是在case1里进行的，flag=f1+f2，

```
.data:0000000000601078 ; char f1[]
.data:0000000000601078 f1              db 'GXY{do_not_',0      ; DATA XREF: get_flag+9E↑o
.data:0000000000601078 _data           ends
```

可以看到f1='GXY{do_not_'，f2的值是由case4进行赋值，将s赋给f2，case5对f2进行了一定的处理，所以正确的运行顺序应该是4->5->1。

写一下脚本

```
flag = 'GXY{do_not_'

f2 = [0x7F, 0x66, 0x6F, 0x60, 0x67, 0x75, 0x63, 0x69][::-1]

s = ''

for i in range(8):

    if i % 2 == 1:

        s = chr(int(f2[i]) - 2)

    else:

        s = chr(int(f2[i]) - 1)

    flag += s

print(flag)
```

结果是GXY{do_not_hate_me}

> flag{do_not_hate_me}

2022-1-24

[ACTF新生赛2020]easyre

下载下来easy_re.exe，扔到ExeinfoPE里分析，发现是加壳的。

脱一下壳，扔到IDA里，直接F5，

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  _BYTE v4[12]; // [esp+12h] [ebp-2Eh] BYREF

  _DWORD v5[3]; // [esp+1Eh] [ebp-22h]

  _BYTE v6[5]; // [esp+2Ah] [ebp-16h] BYREF

  int v7; // [esp+2Fh] [ebp-11h]

  int v8; // [esp+33h] [ebp-Dh]

  int v9; // [esp+37h] [ebp-9h]

  char v10; // [esp+3Bh] [ebp-5h]

  int i; // [esp+3Ch] [ebp-4h]



  sub_401A10();

  qmemcpy(v4, "*F'\"N,\"(I?+@", sizeof(v4));

  printf("Please input:");

  scanf("%s", v6);

  if ( v6[0] != 65 || v6[1] != 67 || v6[2] != 84 || v6[3] != 70 || v6[4] != 123 || v10 != 125 )

    return 0;

  v5[0] = v7;

  v5[1] = v8;

  v5[2] = v9;

  for ( i = 0; i <= 11; ++i )

  {

    if ( v4[i] != byte_402000[*((char *)v5 + i) - 1] )

      return 0;

  }

  printf("You are correct!");

  return 0;

}
```

根据代码可以看到，flag是由v6+v5+v10组成的，v6为'ACTF{'，v10为'}'，此处主要分析v5，下面有个for循环函数，我们来分析一下。首先i为12位，那么v5应该也为12位，其主要意思是查找字符，从_data_start字符串中查找v4（v4的值在前面有给出），此处v7、v8、v9均为int型变量，可以分别存储4个字符型数据，通过循环在_data_start_中找到v7、v8、v9中存储的12个字符型变量的ASCII码对应的元素，与v4数组进行比较。其中_data_start的下标就是flag的ascll码-1，v4=byte_402000[输入的数组的每一位值-1]。

写一下脚本

```
key = '~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(\'&%$# !"'

v4 = "*F'\"N,\"(I?+@"

v4_list = []

for i in v4:

    v4_list.append(ord(i))

x = []

flag = ''

for i in v4_list:

    x.append(key.find(chr(i))+1)

for i in x:

    flag += chr(i)

print(flag)
```

这个地方自行调整,有的要求是写flag{U9X_1S_W6@T?}

2022-1-25

[2019红帽杯]EASYRE

这题太坑了
一开始找到一串base64加密的字符串，解密之后是个网址，被坑了。
后面看了别人的writeup才知道，原来不在那个函数里，在主函数下面的函数里面。

```
unsigned __int64 sub_400D35()

{

  unsigned __int64 result; // rax

  unsigned int v1; // [rsp+Ch] [rbp-24h]

  int i; // [rsp+10h] [rbp-20h]

  int j; // [rsp+14h] [rbp-1Ch]

  unsigned int v4; // [rsp+24h] [rbp-Ch]

  unsigned __int64 v5; // [rsp+28h] [rbp-8h]



  v5 = __readfsqword(0x28u);

  v1 = sub_43FD20(0LL) - qword_6CEE38;

  for ( i = 0; i <= 1233; ++i )

  {

    sub_40F790(v1);

    sub_40FE60();

    sub_40FE60();

    v1 = sub_40FE60() ^ 0x98765432;

  }

  v4 = v1;

  if ( ((unsigned __int8)v1 ^ byte_6CC0A0[0]) == 102 && (HIBYTE(v4) ^ (unsigned __int8)byte_6CC0A3) == 103 )
  // 关键代码

  {

    for ( j = 0; j <= 24; ++j )

      sub_410E90(byte_6CC0A0[j] ^ *((_BYTE *)&v4 + j % 4));

  }

  result = __readfsqword(0x28u) ^ v5;

  if ( result )

    sub_444020();

  return result;

}
```

v4^byte_6CC0A0[0] == ‘f’

v7 = v4

HIBYTE=HIBYTE=(((_BYTE)&

+1))

也就是v4的下一位

byte_6CC0A3 也就是byte_6CC0A0[3]

第一次开始解出来flag开头是flag

可以推测就是判断v4的四位是不是与byte_6CC0A0前四位异或后等于’flag’

然后下面的异或应该是出flag的

写一个脚本：

```
s = [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]

s1 = 'flag'

flag = ''

key = ''



for i in range(4):

    key += chr(s[i] ^ ord(s1[i]))



for i in range(len(s)):

    flag += chr(s[i] ^ ord(key[i % 4]))



print(flag)
```

2022-1-26

[ACTF新生赛2020]rome

老规矩，先拖ExeinfoPE，没壳，再扔到IDA里，进到main里，发现有个func函数，进到func函数里,v3-v6是’ACTF{’，大致意思就是分别对大小写字母进行处理，直接暴力破解。

```
int func()

{

  int result; // eax
```

```
int v1[4]; // [esp+14h] [ebp-44h]

unsigned __int8 v2; // [esp+24h] [ebp-34h] BYREF

unsigned __int8 v3; // [esp+25h] [ebp-33h]

unsigned __int8 v4; // [esp+26h] [ebp-32h]

unsigned __int8 v5; // [esp+27h] [ebp-31h]

unsigned __int8 v6; // [esp+28h] [ebp-30h]

int v7; // [esp+29h] [ebp-2Fh]

int v8; // [esp+2Dh] [ebp-2Bh]

int v9; // [esp+31h] [ebp-27h]

int v10; // [esp+35h] [ebp-23h]

unsigned __int8 v11; // [esp+39h] [ebp-1Fh]

char v12[29]; // [esp+3Bh] [ebp-1Dh] BYREF



strcpy(v12, "Qsw3sj_lz4_Ujw@l");

printf("Please input:");

scanf("%s", &v2);

result = v2;

if ( v2 == 65 )

{

  result = v3;

  if ( v3 == 67 )

  {

    result = v4;

    if ( v4 == 84 )

    {

      result = v5;

      if ( v5 == 70 )

      {

        result = v6;

        if ( v6 == 123 )
```

```
    {
      result = v11;
      if ( v11 == 125 )
      {
        v1[0] = v7;
        v1[1] = v8;
        v1[2] = v9;
        v1[3] = v10;
        *(_DWORD *)&v12[17] = 0;
        while ( *(int *)&v12[17] <= 15 )
        {
          if ( *((char *)v1 + *(_DWORD *)&v12[17]) > 64 && *((char *)v1 + *(_DWORD *)&v12[17]) <= 90 )
            *((_BYTE *)v1 + *(_DWORD *)&v12[17]) = (*((char *)v1 + *(_DWORD *)&v12[17]) - 51) % 26 + 65;
          if ( *((char *)v1 + *(_DWORD *)&v12[17]) > 96 && *((char *)v1 + *(_DWORD *)&v12[17]) <= 122 )
            *((_BYTE *)v1 + *(_DWORD *)&v12[17]) = (*((char *)v1 + *(_DWORD *)&v12[17]) - 79) % 26 + 97;
          ++*(_DWORD *)&v12[17];
        }
        *(_DWORD *)&v12[17] = 0;
        while ( *(int *)&v12[17] <= 15 )
        {
          result = (unsigned __int8)v12[*(_DWORD *)&v12[17]];
          if ( *((_BYTE *)v1 + *(_DWORD *)&v12[17]) != (_BYTE)result )
            return result;
          ++*(_DWORD *)&v12[17];
        }
        result = printf("You are correct!");
      }
    }
  }
}
```

```
    }
  }
  return result;
}
```

写个脚本

```
key = "Qsw3sj_lz4_Ujw@l"
flag = ""
for i in range(len(key)):
    for a in range(128):
        x = a
        if 64 < x <= 90:
            x = (x - 51) % 26 + 65
        if 96 < a <= 122:
            x = (x - 79) % 26 + 97
        if x == ord(key[i]):
            flag += chr(a)
print(flag)
```

2022-1-27

[WUSTCTF2020]level1

下载下来扔到IDA里，F5看一下伪代码

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  int i; // [rsp+4h] [rbp-2Ch]

  FILE *stream; // [rsp+8h] [rbp-28h]

  char ptr[24]; // [rsp+10h] [rbp-20h] BYREF

  unsigned __int64 v7; // [rsp+28h] [rbp-8h]



  v7 = __readfsqword(0x28u);

  stream = fopen("flag", "r");

  fread(ptr, 1uLL, 0x14uLL, stream);

  fclose(stream);

  for ( i = 1; i <= 19; ++i )

  {

    if ( (i & 1) != 0 )

      printf("%ld\n", (unsigned int)(ptr[i] << i));

    else

      printf("%ld\n", (unsigned int)(i * ptr[i]));

  }

  return 0;

}
```

对下标索引为奇数进行移位，偶数进行乘法。输出的数字在外层的output.txt里。写一下脚本

```
key = [198, 232, 816, 200, 1536, 300, 6144, 984, 51200, 570, 92160,

       1200, 565248, 756, 1474560, 800, 6291456, 1782, 65536000]



for i in range(19):

    if (i + 1) & 1:

        print(chr(key[i] >> (i + 1)), end='')

    else:

        print(chr(key[i] // (i + 1)), end='')
```

2022-1-28

[WUSTCTF2020]level2

下载下来扔到ExeinfoPE里发现有壳，

下用工具去不了壳，于是下载upx，upx -d，脱壳。[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-16LaJEyT-1645283525940)(http://skyofstars.cn/usr/uploads/2021/06/2116134172.png)] 扔到IDA里，直接看到flag。

```
.text:08048889                push    ecx
.text:0804888A                sub     esp, 14h
.text:0804888D                mov     [ebp+var_C], offset flag ; "wctf2020{Just_upx_-d}"
.text:08048894                sub     esp, 0Ch
.text:08048897                push    offset aWhereIsIt ; "where is it?"
```

2022-1-29

[GWCTF 2019]xxor

老规矩，先ExeinfoPE，没加壳，扔到IDA里

```
__int64 __fastcall main(int a1, char **a2, char **a3)

{

  int i; // [rsp+8h] [rbp-68h]

  int j; // [rsp+Ch] [rbp-64h]

  __int64 v6[6]; // [rsp+10h] [rbp-60h] BYREF

  __int64 v7[6]; // [rsp+40h] [rbp-30h] BYREF



  v7[5] = __readfsqword(0x28u);

  puts("Let us play a game?");

  puts("you have six chances to input");

  puts("Come on!");
```

```
v6[0] = 0LL;
v6[1] = 0LL;
v6[2] = 0LL;
v6[3] = 0LL;
v6[4] = 0LL;
for ( i = 0; i <= 5; ++i )
{
  printf("%s", "input: ");
  a2 = (char **)((char *)v6 + 4 * i);
  __isoc99_scanf("%d", a2);
}
v7[0] = 0LL;
v7[1] = 0LL;
v7[2] = 0LL;
v7[3] = 0LL;
v7[4] = 0LL;
for ( j = 0; j <= 2; ++j )
{
  dword_601078 = v6[j];
  dword_60107C = HIDWORD(v6[j]);
  a2 = (char **)&unk_601060;
  sub_400686(&dword_601078, &unk_601060);
  LODWORD(v7[j]) = dword_601078;
  HIDWORD(v7[j]) = dword_60107C;
}
if ( (unsigned int)sub_400770(v7, a2) != 1 )
{
  puts("NO NO NO~ ");
  exit(0);
}
```

```
  }
  puts("Congratulation!\n");
  puts("You seccess half\n");
  puts("Do not forget to change input to hex and combine~\n");
  puts("ByeBye");
  return 0LL;
}
```

先分析一下main函数，

```
    printf("%s", "input: ");
    a2 = (char **)((char *)v6 + 4 * i);
```

v6应该就是输入的6个数，v6的每个元素占4个字节。跟踪v6的行踪，仔细分析与v6有关的加密代码。

```
    dword_601078 = v6[j];
    dword_60107C = HIDWORD(v6[j]);
    a2 = (char **)&unk_601060;
    sub_400686(&dword_601078, &unk_601060);
    LODWORD(v7[j]) = dword_601078;
    HIDWORD(v7[j]) = dword_60107C;
```

dword_601078是取了每个v6元素四个字节中的后两个字节中的值，dword_60107c则是取了每个v6元素四个字节中前两个字节的值（详细去看hidword和lodword的定义）。

这其实有个疑问，问什么dword——601078那一条语句没有lodword函数却依然和有lodword函数的结果相同，查了别人写的WriteUp才知道这是由于dword_601078这个变量只能存储2个字节的数据，因此在读取v6[j]时只能读取其前两个字节，那么效果其实就跟lodword一样了。

其中主要是两个函数，sub_400686和sub_400770

```
__int64 __fastcall sub_400686(unsigned int *a1, _DWORD *a2)
{
  __int64 result; // rax
  unsigned int v3; // [rsp+1Ch] [rbp-24h]
  unsigned int v4; // [rsp+20h] [rbp-20h]
  int v5; // [rsp+24h] [rbp-1Ch]
  unsigned int i; // [rsp+28h] [rbp-18h]

  v3 = *a1;
  v4 = a1[1];
  v5 = 0;
  for ( i = 0; i <= 0x3F; ++i )
  {
    v5 += 1166789954;
    v3 += (v4 + v5 + 11) ^ ((v4 << 6) + *a2) ^ ((v4 >> 9) + a2[1]) ^ 0x20;
    v4 += (v3 + v5 + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 0x10;
  }
  *a1 = v3;
  result = v4;
  a1[1] = v4;
  return result;
}
```

看一下这几行代码

```
  *a1 = v3;
  result = v4;
  a1[1] = v4;
  return result;
```

v3的值赋值到a1地址处，v4的值赋值到a1头部之后的元素上，考虑到a1其实是dword_601078，a1的一个元素大小为4字节，a1[1]所指向的地址是dword_60107c。这样这段代码的意义就是对输入的数据两个两个一组进行异或加密。

```
__int64 __fastcall sub_400770(_DWORD *a1)

{

  __int64 result; // rax



  if ( a1[2] - a1[3] == 2225223423LL

    && a1[3] + a1[4] == 4201428739LL

    && a1[2] - a1[4] == 1121399208LL

    && *a1 == -548868226

    && a1[5] == -2064448480

    && a1[1] == 550153460 )

  {

    puts("good!");

    result = 1LL;

  }

  else

  {

    puts("Wrong!");

    result = 0LL;

  }

  return result;

}
```

这一段是对a1这个数组进行了赋值

```
  if ( (unsigned int)sub_400770(v7, a2) != 1 )
```

这里要使得异或之后的值与v7相同，才能成功。

v7可以通过逻辑运算得到值，然后逆一下就出来结果了。
贴一下Python脚本

```
from Crypto.Util.number import long_to_bytes

from ctypes import *



a1 = [3746099070, 550153460, 3774025685, 1548802262, 2652626477, 2230518816]

key = [2, 2, 3, 4]

for i in range(0, 5, 2):

    v5 = (c_int(1166789954*0x40))

    v3 = c_uint(a1[i])

    v4 = c_uint(a1[i + 1])

    for j in range(64):

        v4.value -= (v3.value + v5.value + 20) ^ ((v3.value << 6) + key[2]) ^ ((v3.value >> 9) + key[3]) ^ 0x10

        v3.value -= (v4.value + v5.value + 11) ^ ((v4.value << 6) + key[0]) ^ ((v4.value >> 9) + key[1]) ^ 0x20

        v5.value -= 1166789954

    a1[i] = v3.value

    a1[i + 1] = v4.value



for k in range(6):

    print(long_to_bytes(a1[k]).decode(), end="")
```

2022-2-6

[SUCTF2019]SignIn

说实话，我刚开始看这题的时候感觉很简单，但是越看越没思路，去查了writeup，才知道这是RSA加密，这没接触过RSA加密做这题我觉得就离谱。

```
__int64 __fastcall main(int a1, char **a2, char **a3)

{

  char v4[16]; // [rsp+0h] [rbp-4A0h] BYREF

  char v5[16]; // [rsp+10h] [rbp-490h] BYREF

  char v6[16]; // [rsp+20h] [rbp-480h] BYREF

  char v7[16]; // [rsp+30h] [rbp-470h] BYREF

  char v8[112]; // [rsp+40h] [rbp-460h] BYREF

  char v9[1000]; // [rsp+B0h] [rbp-3F0h] BYREF

  unsigned __int64 v10; // [rsp+498h] [rbp-8h]



  v10 = __readfsqword(0x28u);

  puts("[sign in]");

  printf("[input your flag]: ");

  __isoc99_scanf("%99s", v8);

  sub_96A(v8, v9);

  __gmpz_init_set_str(v7, "ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35", 16LL);

  __gmpz_init_set_str(v6, v9, 16LL);

  __gmpz_init_set_str(v4, "10346103590081691412139010129904904441395040517371217043416168653987816098454
9", 10LL
);

  __gmpz_init_set_str(v5, "65537", 10LL);

  __gmpz_powm(v6, v6, v5, v4);

  if ( (unsigned int)__gmpz_cmp(v6, v7) )

    puts("GG!");

  else

    puts("TTTTTTTTTql!");

  return 0LL;

}
```

sub_96A函数是把用户输入的字符串转换为16进制存入v9中。

程序调用了 __gmpz_init_set_str 函数，通过搜索得知这是一个 GNU 高精度算法库，官方文档地址：https://gmplib.org/manual/

```
__gmpz_init_set_str 其实就是 mpz_init_set_str



int mpz_init_set_str (mpz_t rop, const char *str, int base) [Function]

Initialize rop and set its value like mpz_set_str



int mpz_set_str (mpz_t rop, const char *str, int base) [Function]

Set the value of rop from str, a null-terminated C string in base base. White space is allowed

in the string, and is simply ignored.
```

这三个参数分别是多精度整数变量，字符串，进制。

这个函数的作用就是将 str 字符数组以 base 指定的进制解读成数值并写入 rop 所指向的内存。然后后面还调用了一个__gmpz_powm函数，官方文档中的定义是这样。

```
void mpz_powm (mpz_t rop, const mpz_t base, const mpz_t exp, const mpz_t mod) [Function]

Set rop to base^exp mod mod.
```

其实就是计算 base 的 exp 次方，并对 mod 取模，最后将结果写入 rop 中这个运算的过程和RSA的加密过程一样。

接下来就是__gmpz_cmp函数，看这个函数名就知道这是比较函数。

```
if ( (unsigned int)__gmpz_cmp(v6, v7) )
```

v6是用户输入，v7是程序的硬编码数据

RSA的加解密算法

RSA加密是对明文的E次方后除以N后求余数的过程

加密算法
C=MEmodN
C是密文，M是明文，E是公钥（E和 φ互为质数），N是公共模数（质数 P 、Q相乘得到N），MOD就是模运算。
解密算法
M=CDmodN
C是密文，M是明文，D是私钥（私钥由这个公式计算得出E * D % φ = 1），N是公共模数（质数 P 、Q相乘得到N），MOD就是模运算，φ是欧拉函数（由这个公式计算得出φ =

）。

下面写解密过程

通过main

函数，我们可以得到

```
C = ad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35

N = 10346103590081691412139010129904904441395040517371217043416168653987816098454 9

E = 65537
```

知道这3条数据我们就可以开始解密密文了，首先我们要分解N得到P、Q。

分解网站：http://www.factordb.com/

```
P = 282164587459512124844245113950593348271

Q = 366669102002966856876605669837014229419
```

现在我们有了P、Q和E，我们就可以计算出欧拉函数，然后我们就可以通过欧拉函数φ

和公钥E计算出私钥D。

```
D = gmpy2.invert(e,(p-1)*(q-1))
```

计算出私钥d后我们就可以对密文C进行解密，解密算法是（密文C的私钥D次方对公共模数N取余）

```
M = gmpy2.powmod(c,d,n)
```

解密脚本

```
import gmpy2

import binascii



p = 282164587459512124844245113950593348271

q = 366669102002966856876605669837014229419

c = 0xad939ff59f6e70bcbfad406f2494993757eee98b91bc244184a377520d06fc35

n = 10346103590081691412139010129904904441395040517371217043416168653987816098454 9

e = 65537

d = gmpy2.invert(e, (p-1)*(q-1))

m = gmpy2.powmod(c, d, n)

print(binascii.unhexlify(hex(m)[2:]).decode(encoding="utf-8"))
```

得到flag

2022-2-7

[MRCTF2020]Transform

下载下来扔到ExeinfoPE里，发现没有加壳，拖到IDA64里

```
```

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  char Str[104]; // [rsp+20h] [rbp-70h] BYREF

  int j; // [rsp+88h] [rbp-8h]

  int i; // [rsp+8Ch] [rbp-4h]



  sub_402230(argc, argv, envp);

  sub_40E640("Give me your code:\n");

  sub_40E5F0("%s", Str);

  if ( strlen(Str) != 33 )                  //33位

  {

    sub_40E640("Wrong!\n");

    system("pause");

    exit(0);

  }

  for ( i = 0; i <= 32; ++i )

  {

    byte_414040[i] = Str[dword_40F040[i]];      //将输入的flag使用dword_40F040数组作为索引，打乱顺序

    byte_414040[i] ^= LOBYTE(dword_40F040[i]);  //再将打乱后的flag数组，与dword_40F040数组异或，得到byte_40F0E0

  }

  for ( j = 0; j <= 32; ++j )

  {

    if ( byte_40F0E0[j] != byte_414040[j] )

    {

      sub_40E640("Wrong!\n");

      system("pause");

      exit(0);

    }

  }

  sub_40E640("Right!Good Job!\n");
```

```
  sub_40E640("Here is your flag: %s\n", Str);
  system("pause");
  return 0;
}
.data:000000000040F040 ; int dword_40F040[40]
.data:000000000040F040 dword_40F040    dd 9, 0Ah, 0Fh, 17h, 7, 18h, 0Ch, 6, 1, 10h, 3, 11h, 20h
.data:000000000040F040                                         ; DATA XREF: main+79↑o
.data:000000000040F040                                         ; main+B8↑o
.data:000000000040F040                 dd 1Dh, 0Bh, 1Eh, 1Bh, 16h, 4, 0Dh, 13h, 14h, 15h, 2, 19h
.data:000000000040F040                 dd 5, 1Fh, 8, 12h, 1Ah, 1Ch, 0Eh, 8 dup(0)
.data:000000000040F0E0 ; _BYTE byte_40F0E0[96]
.data:000000000040F0E0 byte_40F0E0     db 67h, 79h, 7Bh, 7Fh, 75h, 2Bh, 3Ch, 52h, 53h, 79h, 57h
.data:000000000040F0E0                                         ; DATA XREF: main+EF↑o
.data:000000000040F0E0                 db 5Eh, 5Dh, 42h, 7Bh, 2Dh, 2Ah, 66h, 42h, 7Eh, 4Ch, 57h
.data:000000000040F0E0                 db 79h, 41h, 6Bh, 7Eh, 65h, 3Ch, 5Ch, 45h, 6Fh, 62h, 4Dh
.data:000000000040F0E0                 db 3Fh dup(0)
```

整体思路很见到简单，就是将输入的flag使用dword_40F040数组作为索引，打乱顺序，再将打乱后的flag数组，与dword_40F040数组异或，得到的值与byte_40F0E0相等。（此处还有个坑，8dup

=0，不等于8）

加密方式：

```
a = flag[b[i]];
a ^= b;
c == a
即c = a ^ b
```

解密就反过来，因为a是不知道的，所以就先解出a：

```
a = a ^ b
即 a = c ^ b
然后 flag[b[i]] = a。
```

贴上脚本

```
dword_40F040 = [0x9, 0x0A, 0x0F, 0x17, 0x7, 0x18, 0x0C, 0x6, 0x1,

                0x10, 0x3, 0x11, 0x20, 0x1D, 0x0B, 0x1E, 0x1B, 0x16, 0x4,

                0x0D, 0x13, 0x14, 0x15, 0x2, 0x19, 0x5, 0x1F, 0x8, 0x12, 0x1A, 0x1C, 0x0E, 0]

byte_40F0E0 = [0x67, 0x79, 0x7B, 0x7F, 0x75, 0x2B, 0x3C, 0x52, 0x53, 0x79,

               0x57, 0x5E, 0x5D, 0x42, 0x7B, 0x2D, 0x2A, 0x66, 0x42, 0x7E,

               0x4C, 0x57, 0x79, 0x41, 0x6B, 0x7E, 0x65, 0x3C, 0x5C, 0x45, 0x6F, 0x62, 0x4D, 0]



flag = [0] * 33



for i in range(33):

    byte_40F0E0[i] ^= dword_40F040[i]



for i in range(33):

    flag[dword_40F040[i]] = byte_40F0E0[i]

print(''.join([chr(x) for x in flag]))
```

2022-2-8

[ACTF新生赛2020]usualCrypt

下载下来，先扔到ExeinfoPE里，无壳，然后再放到IDA里。

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  int v3; // esi

  int result; // eax

  int v5[3]; // [esp+8h] [ebp-74h] BYREF

  __int16 v6; // [esp+14h] [ebp-68h]

  char v7; // [esp+16h] [ebp-66h]

  char v8[100]; // [esp+18h] [ebp-64h] BYREF



  sub_403CF8((int)&unk_40E140);                     //输出

  scanf("%s", v8);                                  //输入
```

```
  v5[0] = 0;
  v5[1] = 0;
  v5[2] = 0;
  v6 = 0;
  v7 = 0;
  sub_401080((int)v8, strlen(v8), (int)v5);         //加密函数
  v3 = 0;
  while ( *((_BYTE *)v5 + v3) == byte_40E0E4[v3] )  //比较函数
  {
    if ( ++v3 > strlen((const char *)v5) )
      goto LABEL_6;
  }
  sub_403CF8((int)aError);
LABEL_6:
  if ( v3 - 1 == strlen(byte_40E0E4) )
    result = sub_403CF8((int)aAreYouHappyYes);
  else
    result = sub_403CF8((int)aAreYouHappyNo);
  return result;
}
```

打开sub_401080函数，可以看出来主体基本上是base64加密

```
int __cdecl sub_401080(int a1, int a2, int a3)
{
  int v3; // edi
  int v4; // esi
  int v5; // edx
  int v6; // eax
  int v7; // ecx
  int v8; // esi
```

```
int v9; // esi

int v10; // esi

int v11; // esi

_BYTE *v12; // ecx

int v13; // esi

int v15; // [esp+18h] [ebp+8h]



v3 = 0;

v4 = 0;

sub_401000();

v5 = a2 % 3;

v6 = a1;

v7 = a2 - a2 % 3;

v15 = a2 % 3;

if ( v7 > 0 )

{

  do

  {

    LOBYTE(v5) = *(_BYTE *)(a1 + v3);

    v3 += 3;

    v8 = v4 + 1;

    *(_BYTE *)(v8 + a3 - 1) = byte_40E0A0[(v5 >> 2) & 0x3F];

    *(_BYTE *)(++v8 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(a1 + v3 - 3) & 3)

                                          + (((int)*(unsigned __int8 *)(a1 + v3 - 2) >> 4) & 0xF)];

    *(_BYTE *)(++v8 + a3 - 1) = byte_40E0A0[4 * (*(_BYTE *)(a1 + v3 - 2) & 0xF)

                                          + (((int)*(unsigned __int8 *)(a1 + v3 - 1) >> 6) & 3)];

    v5 = *(_BYTE *)(a1 + v3 - 1) & 0x3F;

    v4 = v8 + 1;

    *(_BYTE *)(v4 + a3 - 1) = byte_40E0A0[v5];

  }
```

```
    while ( v3 < v7 );
    v5 = v15;
  }
  if ( v5 == 1 )
  {
    LOBYTE(v7) = *(_BYTE *)(v3 + a1);
    v9 = v4 + 1;
    *(_BYTE *)(v9 + a3 - 1) = byte_40E0A0[(v7 >> 2) & 0x3F];
    v10 = v9 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3)];
    *(_BYTE *)(v10 + a3) = 61;
LABEL_8:
    v13 = v10 + 1;
    *(_BYTE *)(v13 + a3) = 61;
    v4 = v13 + 1;
    goto LABEL_9;
  }
  if ( v5 == 2 )
  {
    v11 = v4 + 1;
    *(_BYTE *)(v11 + a3 - 1) = byte_40E0A0[((int)*(unsigned __int8 *)(v3 + a1) >> 2) & 0x3F];
    v12 = (_BYTE *)(v3 + a1 + 1);
    LOBYTE(v6) = *v12;
    v10 = v11 + 1;
    *(_BYTE *)(v10 + a3 - 1) = byte_40E0A0[16 * (*(_BYTE *)(v3 + a1) & 3) + ((v6 >> 4) & 0xF)];
    *(_BYTE *)(v10 + a3) = byte_40E0A0[4 * (*v12 & 0xF)];
    goto LABEL_8;
  }
LABEL_9:
  *(_BYTE *)(v4 + a3) = 0;
```

```
  return sub_401030(a3);
}
```

加密函数里还有两个函数，分别是sub_401000跟sub_401030

。分别打开看一下

```
int sub_401000()
{
  int result; // eax
  char v1; // cl

  for ( result = 6; result < 15; ++result )
  {
    v1 = byte_40E0AA[result];
    byte_40E0AA[result] = byte_40E0A0[result];
    byte_40E0A0[result] = v1;
  }
  return result;
}
```

将两个数组里的数据进行了交换，看地址两个数组是连在一起的，其实也可以连在一起当成一个数组看，从下标为6开始到下标为15，往后偏移了10（0xA）位，也就是QRSTUVWXY和GHIJKLMNOP相互交换了一下，所以原始用来加密的base64密码表是‘ABCDEFQRSTUVWXYGHIJKLMNOPZabcdefghijklmnopqrstuvwxyz0123456789+/’。

再来看一下sub_401030函数，可以看出是对字符串进行了大小写转换。

```
int __cdecl sub_401030(const char *a1)
{
  __int64 v1; // rax
  char v2; // al

  v1 = 0i64;
  if ( strlen(a1) )
  {
```

```
    do
    {
      v2 = a1[HIDWORD(v1)];
      if ( v2 < 97 || v2 > 122 )
      {
        if ( v2 < 65 || v2 > 90 )
          goto LABEL_9;
        LOBYTE(v1) = v2 + 32;
      }
      else
      {
        LOBYTE(v1) = v2 - 32;
      }
      a1[HIDWORD(v1)] = v1;
LABEL_9:
      LODWORD(v1) = 0;
      ++HIDWORD(v1);
    }
    while ( HIDWORD(v1) < strlen(a1) );
  }
  return v1;
}
```

逆向思路就是：–>结果大小写互换–>修改base64转换表–>加密结果通过转换表得到正常的加密后的结果–>base64解密。

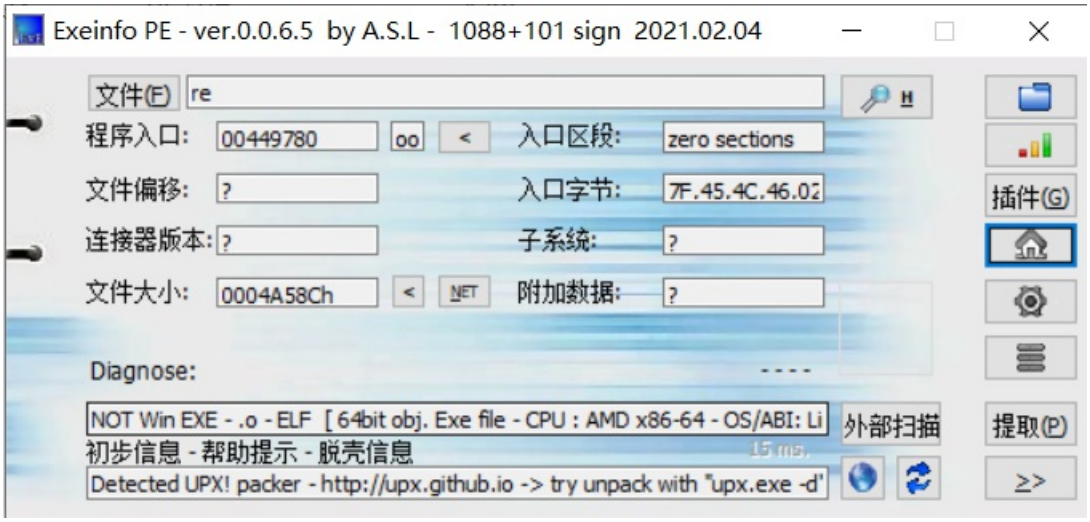
贴上脚本：

```
import base64



result = 'zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9'.swapcase()  # 大小写转换

base64key = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'

base64keylist = list(base64key)

translist = list(base64key)

offset = 10

flag = ''



for i in range(6, 15):  # 置换密码表

    translist[i], translist[i + offset] = translist[i + offset], translist[i]

base64dict = dict(zip(base64keylist, translist))  # 生成旧密码表跟新密码表的映射

for i in range(len(result)):

    flag += base64dict[result[i]]

flag = base64.b64decode(flag)

print(flag)
```

2022-2-9

[GUET-CTF2019]re

先下载下来，拖到ExeinfoPE里，发现有壳，用upx去一下壳。



拖到IDA64里

```
__int64 __fastcall sub_400E28(__int64 a1, int a2, int a3, int a4, int a5, int a6)
{
  int v6; // edx
  int v7; // ecx
  int v8; // er8
  int v9; // er9
  __int64 result; // rax
  __int64 v11; // [rsp+0h] [rbp-30h] BYREF
  unsigned __int64 v12; // [rsp+28h] [rbp-8h]

  v12 = __readfsqword(0x28u);
  sub_40F950((unsigned int)"input your flag:", a2, a3, a4, a5, a6, 0LL, 0LL, 0LL, 0LL);
  sub_40FA80((unsigned int)"%s", (unsigned int)&v11, v6, v7, v8, v9, v11);
  if ( (unsigned int)sub_4009AE(&v11) )
    sub_410350("Correct!");
  else
    sub_410350("Wrong!");
  result = 0LL;
  if ( __readfsqword(0x28u) != v12 )
    sub_443550();
  return result;
}
```

打开sub_4009AE函数：

```
_BOOL8 __fastcall sub_4009AE(char *a1)
{
  if ( 1629056 * *a1 != 166163712 )
    return 0LL;
  if ( 6771600 * a1[1] != 731332800 )
    return 0LL;
```

```
if ( 3682944 * a1[2] != 357245568 )
  return 0LL;
if ( 10431000 * a1[3] != 1074393000 )
  return 0LL;
if ( 3977328 * a1[4] != 489211344 )
  return 0LL;
if ( 5138336 * a1[5] != 518971936 )
  return 0LL;
if ( 7532250 * a1[7] != 406741500 )
  return 0LL;
if ( 5551632 * a1[8] != 294236496 )
  return 0LL;
if ( 3409728 * a1[9] != 177305856 )
  return 0LL;
if ( 13013670 * a1[10] != 650683500 )
  return 0LL;
if ( 6088797 * a1[11] != 298351053 )
  return 0LL;
if ( 7884663 * a1[12] != 386348487 )
  return 0LL;
if ( 8944053 * a1[13] != 438258597 )
  return 0LL;
if ( 5198490 * a1[14] != 249527520 )
  return 0LL;
if ( 4544518 * a1[15] != 445362764 )
  return 0LL;
if ( 3645600 * a1[17] != 174988800 )
  return 0LL;
if ( 10115280 * a1[16] != 981182160 )
  return 0LL;
```

```
  if ( 9667504 * a1[18] != 493042704 )
    return 0LL;
  if ( 5364450 * a1[19] != 257493600 )
    return 0LL;
  if ( 13464540 * a1[20] != 767478780 )
    return 0LL;
  if ( 5488432 * a1[21] != 312840624 )
    return 0LL;
  if ( 14479500 * a1[22] != 1404511500 )
    return 0LL;
  if ( 6451830 * a1[23] != 316139670 )
    return 0LL;
  if ( 6252576 * a1[24] != 619005024 )
    return 0LL;
  if ( 7763364 * a1[25] != 372641472 )
    return 0LL;
  if ( 7327320 * a1[26] != 373693320 )
    return 0LL;
  if ( 8741520 * a1[27] != 498266640 )
    return 0LL;
  if ( 8871876 * a1[28] != 452465676 )
    return 0LL;
  if ( 4086720 * a1[29] != 208422720 )
    return 0LL;
  if ( 9374400 * a1[30] == 515592000 )
    return 5759124 * a1[31] == 719890500;
  return 0LL;
}
```

其实就是很简单的挨个计算问题，贴上脚本：

```
a = [0] * 32

flag = ''

a[0] = chr(166163712//1629056)

a[1] = chr(731332800 // 6771600)

a[2] = chr(357245568 // 3682944)

a[3] = chr(1074393000 // 10431000)

a[4] = chr(489211344 // 3977328)

a[5] = chr(518971936 // 5138336)

a[7] = chr(406741500 // 7532250)

a[8] = chr(294236496 // 5551632)

a[9] = chr(177305856 // 3409728)

a[10] = chr(650683500 // 13013670)

a[11] = chr(298351053 // 6088797)

a[12] = chr(386348487 // 7884663)

a[13] = chr(438258597 // 8944053)

a[14] = chr(249527520 // 5198490)

a[15] = chr(445362764 // 4544518)

a[16] = chr(981182160 // 10115280)

a[17] = chr(174988800 // 3645600)

a[18] = chr(493042704 // 9667504)

a[19] = chr(257493600 // 5364450)

a[20] = chr(767478780 // 13464540)

a[21] = chr(312840624 // 5488432)

a[22] = chr(1404511500 // 14479500)

a[23] = chr(316139670 // 6451830)

a[24] = chr(619005024 // 6252576)

a[25] = chr(372641472 // 7763364)

a[26] = chr(373693320 // 7327320)

a[27] = chr(498266640 // 8741520)

a[28] = chr(452465676 // 8871876)
```

```
a[29] = chr(208422720 // 4086720)

a[30] = chr(515592000 // 9374400)

a[31] = chr(719890500 // 5759124)

for i in range(len(a)):

    print(a[i], end='')
```

但是a[6]我们是不知道的，所以我们只能爆破，得出a[6]=1。

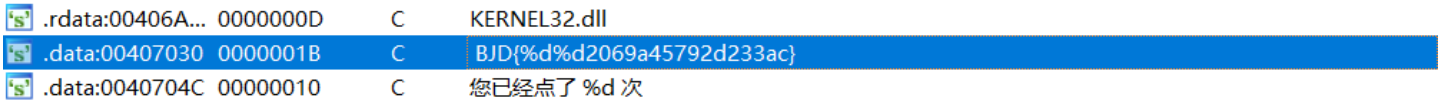
2022-2-10

[BJDCTF2020]JustRE

先扔到ExeinfoPE里，没有壳，扔到IDA里。

没看明白main函数，shift+F12看一下字符串。

看到有一串像flag的字符串。



Ctrl+X交叉引用。

```
    {
      sprintf(String, " BJD{%d%d2069a45792d233ac}", 19999, 0);
      SetWindowTextA(hWnd, String);
```

看出来就是一个赋值操作，把19999跟0分别代入到%d里得到flag

```
sprintf(String, " BJD{%d%d2069a45792d233ac}", 19999, 0);
```

2022-2-11

[FlareOn4]IgniteMe

扔ExeinfoPE里，没壳，扔IDA里。

```
void __noreturn start()

{

  DWORD NumberOfBytesWritten; // [esp+0h] [ebp-4h] BYREF



  NumberOfBytesWritten = 0;

  hFile = GetStdHandle(0xFFFFFFF6);

  dword_403074 = GetStdHandle(0xFFFFFFF5);

  WriteFile(dword_403074, aG1v3M3T3hFl4g, 0x13u, &NumberOfBytesWritten, 0);// 输出提示

  sub_4010F0(NumberOfBytesWritten);             // 读取数据

  if ( sub_401050() )                           // 处理判断

    WriteFile(dword_403074, aG00dJ0b, 0xAu, &NumberOfBytesWritten, 0);

  else

    WriteFile(dword_403074, aN0tT00H0tRWe7r, 0x24u, &NumberOfBytesWritten, 0);

  ExitProcess(0);

}
```

可以看出主要就是两个处理函数，分别是sub_4010F0跟sub_401050

。

分别点开看看

```
int sub_4010F0()

{

  unsigned int v0; // eax

  char Buffer[260]; // [esp+0h] [ebp-110h] BYREF

  DWORD NumberOfBytesRead; // [esp+104h] [ebp-Ch] BYREF

  unsigned int i; // [esp+108h] [ebp-8h]

  char v5; // [esp+10Fh] [ebp-1h]



  v5 = 0;

  for ( i = 0; i < 0x104; ++i )

    Buffer[i] = 0;

  ReadFile(hFile, Buffer, 0x104u, &NumberOfBytesRead, 0);// 读取输入

  for ( i = 0; ; ++i )

  {

    v0 = sub_401020(Buffer);                    // 长度

    if ( i >= v0 )

      break;

    v5 = Buffer[i];

    if ( v5 != 10 && v5 != 13 )

    {

      if ( v5 )

        byte_403078[i] = v5;

    }

  }

  return 1;

}
```

可以看出这个函数是用来去除'n'和'r'的，那就再看一下另外一个函数

```
int sub_401050()

{

  int v1; // [esp+0h] [ebp-Ch]

  int i; // [esp+4h] [ebp-8h]

  unsigned int j; // [esp+4h] [ebp-8h]

  char v4; // [esp+Bh] [ebp-1h]



  v1 = sub_401020(byte_403078);

  v4 = sub_401000();

  for ( i = v1 - 1; i >= 0; --i )

  {

    byte_403180[i] = v4 ^ byte_403078[i];

    v4 = byte_403078[i];

  }

  for ( j = 0; j < 0x27; ++j )

  {

    if ( byte_403180[j] != (unsigned __int8)byte_403000[j] )

      return 0;

  }

  return 1;

}
```

大体逻辑是这样的：

v0是长度，v4是一个给定的数值。循环处理flag，将其倒序与v4亦或的结果保存，替换v4内容，继续下一位。
由于结果已经给出了，我们要做的就是逆向写一遍这个算法。但是我们缺少v4的内容，不知道v4就不知道其他位置的flag。

这个地方看别人动态调试出来的是v4=4，不会动态调试，所以先直接用。

写个逆向脚本

```
byte_403000 = [0x0D, 0x26, 0x49, 0x45, 0x2A, 0x17, 0x78, 0x44, 0x2B, 0x6C, 0x5D, 0x5E, 0x45, 0x12, 0x2F, 0x17,
               0x2B, 0x44, 0x6F, 0x6E, 0x56, 0x09, 0x5F, 0x45, 0x47, 0x73, 0x26, 0x0A, 0x0D, 0x13, 0x17, 0x48,
               0x42, 0x01, 0x40, 0x4D, 0x0C, 0x02, 0x69]



flag = []
v4 = 4
for i in range(len(byte_403000) - 1, -1, -1):
    flag.append(byte_403000[i] ^ v4)
    v4 = flag[-1]
print('flag{' + ''.join([chr(x) for x in flag[::-1]]) + '}')
```

2022-2-12

[MRCTF2020]Xor

无壳，一开始拖到IDA里反编译不出来，退出来，直接点函数，再F5就可以了。

```
int __cdecl main(int argc, const char **argv, const char **envp)

{

  unsigned int i; // eax



  sub_401020((int)"Give Me Your Flag String:\n");

  sub_401050("%s", byte_4212C0);

  if ( strlen(byte_4212C0) != 27 )

  {

LABEL_6:

    sub_401020((int)"Wrong!\n");

    sub_404B7E("pause");

    _loaddll(0);

    __debugbreak();

  }

  for ( i = 0; i < 0x1B; ++i )

  {

    if ( ((unsigned __int8)i ^ (unsigned __int8)byte_4212C0[i]) != byte_41EA08[i] )

      goto LABEL_6;

  }

  sub_401020((int)"Right!\n");

  sub_404B7E("pause");

  return 0;

}
```

其实就是一个简单的异或。

看一下byte_41EA08的值。

```
.rdata:0041EA08 byte_41EA08     db 'M'                  ; DATA XREF: _main+48↑r
.rdata:0041EA09 aSawbFxzJTqjNBp db 'SAWB~FXZ:J:`tQJ"N@ bpdd}8g',0
```

写一下逆向脚本

```
a = 'MSAWB~FXZ:J:`tQJ"N@ bpdd}8g'

flag = []

for i in range(len(a)):

    flag.append(chr(i ^ ord(a[i])))

print(''.join(flag))
```

[GWCTF 2019]pyre

21.5.8

下载文件，是一个pyc文件，找某度有个pyc在线反编译

得到源码

```
#!/usr/bin/env python
# visit http://tool.lu/pyc/ for more information
print 'Welcome to Re World!'
print 'Your input1 is your flag~'
l = len(input1)
for i in range(l):
    num = ((input1[i] + i) % 128 + 128) % 128
    code += num

for i in range(l - 1):
    code[i] = code[i] ^ code[i + 1]

print code
code = [
    '\x1f',
    '\x12',
    '\x1d',
    '(',
    '0',
    '4',
    '\x01',
    '\x06',
    '\x14',
    '4',
    ',',
    '\x1b',
    'U',
    '?',
    'o',
    '6',
    '*',
    ':',
    '\x01',
    'D',
    ';',
    '%',
    '\x13']
```

然后根据源码写出解密脚本

```
code = [
    '\x1f',
    '\x12',
    '\x1d',
    '(',
    '0',
    '4',
    '\x01',
    '\x06',
    '\x14',
    '4',
    ',',
    '\x1b',
    'U',
    '?',
    'o',
    '6',
    '*',
    ':',
    '\x01',
    'D',
    ';',
    '%',
    '\x13']

flag=''

for i in reversed(range(22)):
    code[i]=chr(ord(code[i+1])^ord(code[i]))

for j in range(23):
    flag+=chr(((ord(code[j])-128)%128-j)%128)

print flag
```

其中有一个注意的就是，就是取余的逆运算
具体可以参考这个：求余逆运算+负数求余

[ACTF新生赛2020]Universe_final_answer

直接用z3就行了

存个学z3的教程https://firmianay.gitbooks.io/ctf-all-in-one/content/doc/5.8.1_z3.html

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  __int64 v4; // [rsp+0h] [rbp-A8h] BYREF
  char v5[104]; // [rsp+20h] [rbp-88h] BYREF
  unsigned __int64 v6; // [rsp+88h] [rbp-20h]

  v6 = __readfsqword(0x28u);
  __printf_chk(1LL, "Please give me the key string:", a3);
  scanf("%s", v5);
  if ( (unsigned __int8)sub_860(v5) )
  {
    sub_C50(v5, &v4);
    __printf_chk(1LL, "Judgement pass! flag is actf{%s_%s}\n", v5);
  }
  else
  {
    puts("False key!");
  }
  return 0LL;
}
```

打开之后，在这里进入目标函数

```
bool __fastcall sub_860(char *a1)
{
  int v1; // ecx
  int v2; // esi
  int v3; // edx
  int v4; // er9
  int v5; // er11
  int v6; // ebp
  int v7; // ebx
  int v8; // er8
  int v9; // er10
  bool result; // al
  int v11; // [rsp+0h] [rbp-38h]

  v1 = a1[1];
  v2 = *a1;
  v3 = a1[2];
  v4 = a1[3];
  v5 = a1[4];
  v6 = a1[6];
  v7 = a1[5];
  v8 = a1[7];
  v9 = a1[8];
  result = 0;
  if ( -85 * v9 + 58 * v8 + 97 * v6 + v7 + -45 * v5 + 84 * v4 + 95 * v2 - 20 * v1 + 12 * v3 == 12613 )
  {
    v11 = a1[9];
    if ( 30 * v11 + -70 * v9 + -122 * v6 + -81 * v7 + -66 * v5 + -115 * v4 + -41 * v3 + -86 * v1 - 15 * v2 - 30 * v8 == -54400
      && -103 * v11 + 120 * v8 + 108 * v7 + 48 * v4 + -89 * v3 + 78 * v1 - 41 * v2 + 31 * v5 - (v6 << 6) - 120 * v9 == -10283
      && 71 * v6 + (v7 << 7) + 99 * v5 + -111 * v3 + 85 * v1 + 79 * v2 - 30 * v4 - 119 * v8 + 48 * v9 - 16 * v11 == 22855
      && 5 * v11 + 23 * v9 + 122 * v8 + -19 * v6 + 99 * v7 + -117 * v5 + -69 * v3 + 22 * v1 - 98 * v2 + 10 * v4 == -2944
      && -54 * v11 + -23 * v8 + -82 * v3 + -85 * v2 + 124 * v1 - 11 * v4 - 8 * v5 - 60 * v7 + 95 * v6 + 100 * v9 == -2222
      && -83 * v11 + -111 * v7 + -57 * v2 + 41 * v1 + 73 * v3 - 18 * v4 + 26 * v5 + 16 * v6 + 77 * v8 - 63 * v9 == -13258
      && 81 * v11 + -48 * v9 + 66 * v8 + -104 * v6 + -121 * v7 + 95 * v5 + 85 * v4 + 60 * v3 + -85 * v2 + 80 * v1 == -1559
      && 101 * v11 + -85 * v9 + 7 * v6 + 117 * v7 + -83 * v5 + -101 * v4 + 90 * v3 + -28 * v1 + 18 * v2 - v8 == 6308 )
    {
      return 99 * v11 + -28 * v9 + 5 * v8 + 93 * v6 + -18 * v7 + -127 * v5 + 6 * v4 + -9 * v3 + -93 * v1 + 58 * v2 == -1697;
    }
  }
  return result;
}
```

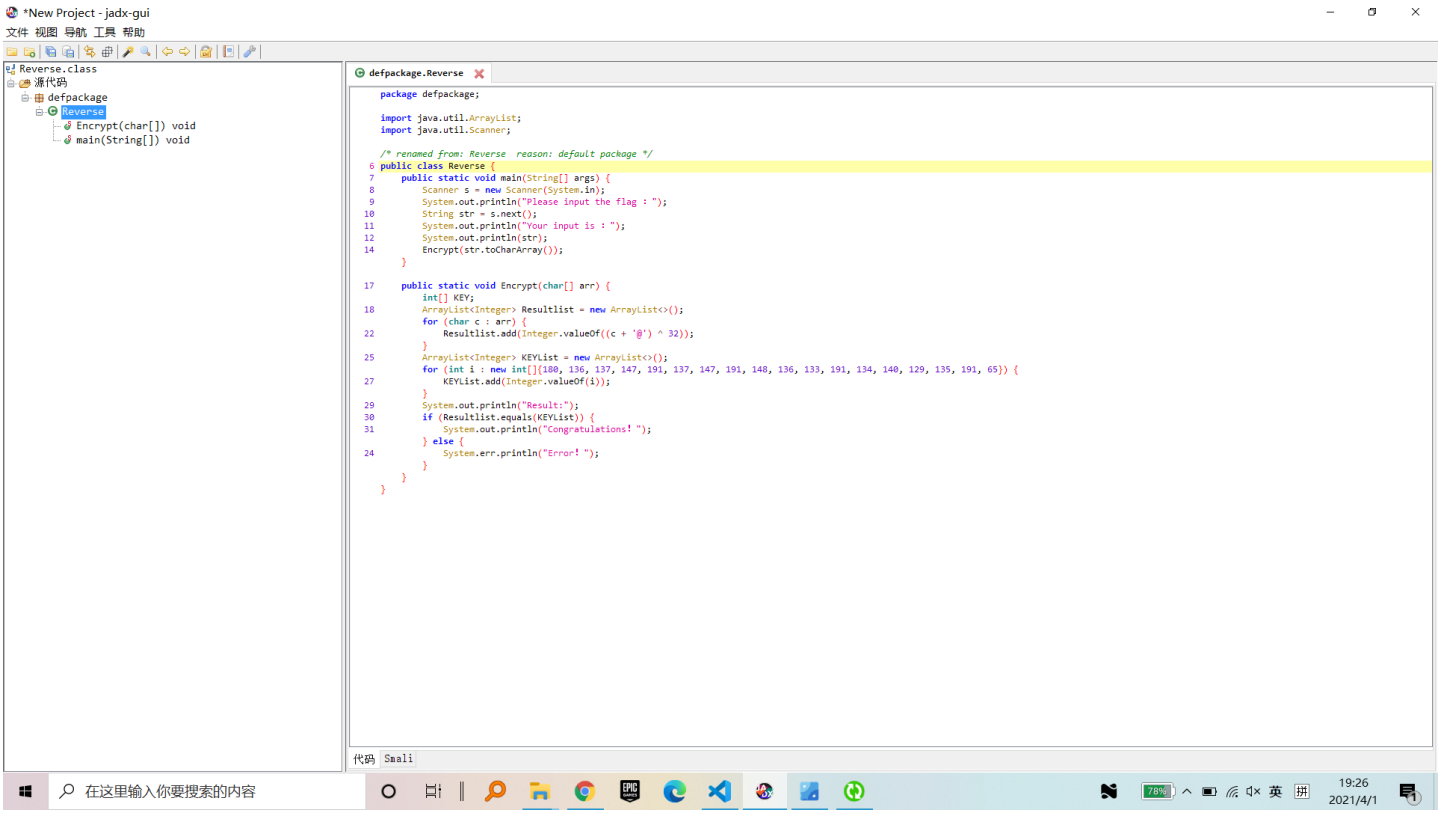
一堆公式，丢到python里面用z3求解一下

```
from z3 import *
solver = Solver()
a1=[BitVec("a1[%d]" % i,16) for i in range(10)]
solver.add(-85 * a1[8] + 58 * a1[7] + 97 * a1[6] + a1[5] + -45 * a1[4] + 84 * a1[3] + 95 * a1[0] - 20 * a1[1] +
12 * a1[2] == 12613)
solver.add(30 * a1[9] + -70 * a1[8] + -122 * a1[6] + -81 * a1[5] + -66 * a1[4] + -115 * a1[3] + -41 * a1[2] + -8
6 * a1[1] - 15 * a1[0] - 30 * a1[7] == -54400)
solver.add(-103 * a1[9] + 120 * a1[7] + 108 * a1[5] + 48 * a1[3] + -89 * a1[2] + 78 * a1[1] - 41 * a1[0] + 31 *
a1[4] - (a1[6] << 6) - 120 * a1[8] == -10283)
solver.add(71 * a1[6] + (a1[5] << 7) + 99 * a1[4] + -111 * a1[2] + 85 * a1[1] + 79 * a1[0] - 30 * a1[3] - 119 *
a1[7] + 48 * a1[8] - 16 * a1[9] == 22855)
solver.add(5 * a1[9] + 23 * a1[8] + 122 * a1[7] + -19 * a1[6] + 99 * a1[5] + -117 * a1[4] + -69 * a1[2] + 22 * a
1[1] - 98 * a1[0] + 10 * a1[3] == -2944)
solver.add(-54 * a1[9] + -23 * a1[7] + -82 * a1[2] + -85 * a1[0] + 124 * a1[1] - 11 * a1[3] - 8 * a1[4] - 60 * a
1[5] + 95 * a1[6] + 100 * a1[8] == -2222)
solver.add(-83 * a1[9] + -111 * a1[5] + -57 * a1[0] + 41 * a1[1] + 73 * a1[2] - 18 * a1[3] + 26 * a1[4] + 16 * a
1[6] + 77 * a1[7] - 63 * a1[8] == -13258)
solver.add(81 * a1[9] + -48 * a1[8] + 66 * a1[7] + -104 * a1[6] + -121 * a1[5] + 95 * a1[4] + 85 * a1[3] + 60 *
a1[2] + -85 * a1[0] + 80 * a1[1] == -1559)
solver.add(101 * a1[9] + -85 * a1[8] + 7 * a1[6] + 117 * a1[5] + -83 * a1[4] + -101 * a1[3] + 90 * a1[2] + -28 *
 a1[1] + 18 * a1[0] - a1[7] == 6308)
solver.add(99 * a1[9] + -28 * a1[8] + 5 * a1[7] + 93 * a1[6] + -18 * a1[5] + -127 * a1[4] + 6 * a1[3] + -9 * a1[
2] + -93 * a1[1] + 58 * a1[0] == -1697)
if solver.check() == sat:
    m = solver.model()
flag1=''
for i in [str(m.eval(a1[i])) for i in range(10)]:
    if int(i)>32768:
        i=int(i)-32768
    flag1+=chr(int(i))
print flag1
```

然后解出来的东西再放到题目里面运行一下就可以出来flag

Java逆向解密

下载来是个.class文件，用jadx打开来是这个样子



看到26有一堆奇奇怪怪的，看着就像ascii的东西，有个加密函数

那么就写个解密脚本

```
key=[180, 136, 137, 147, 191, 137, 147, 191, 148, 136, 133, 191, 134, 140, 129, 135, 191, 65]
flag=''
for i in key:
    s=(i^32)-ord('@')
    flag+=chr(s)
print flag
```

运行结果:

buu re11.py

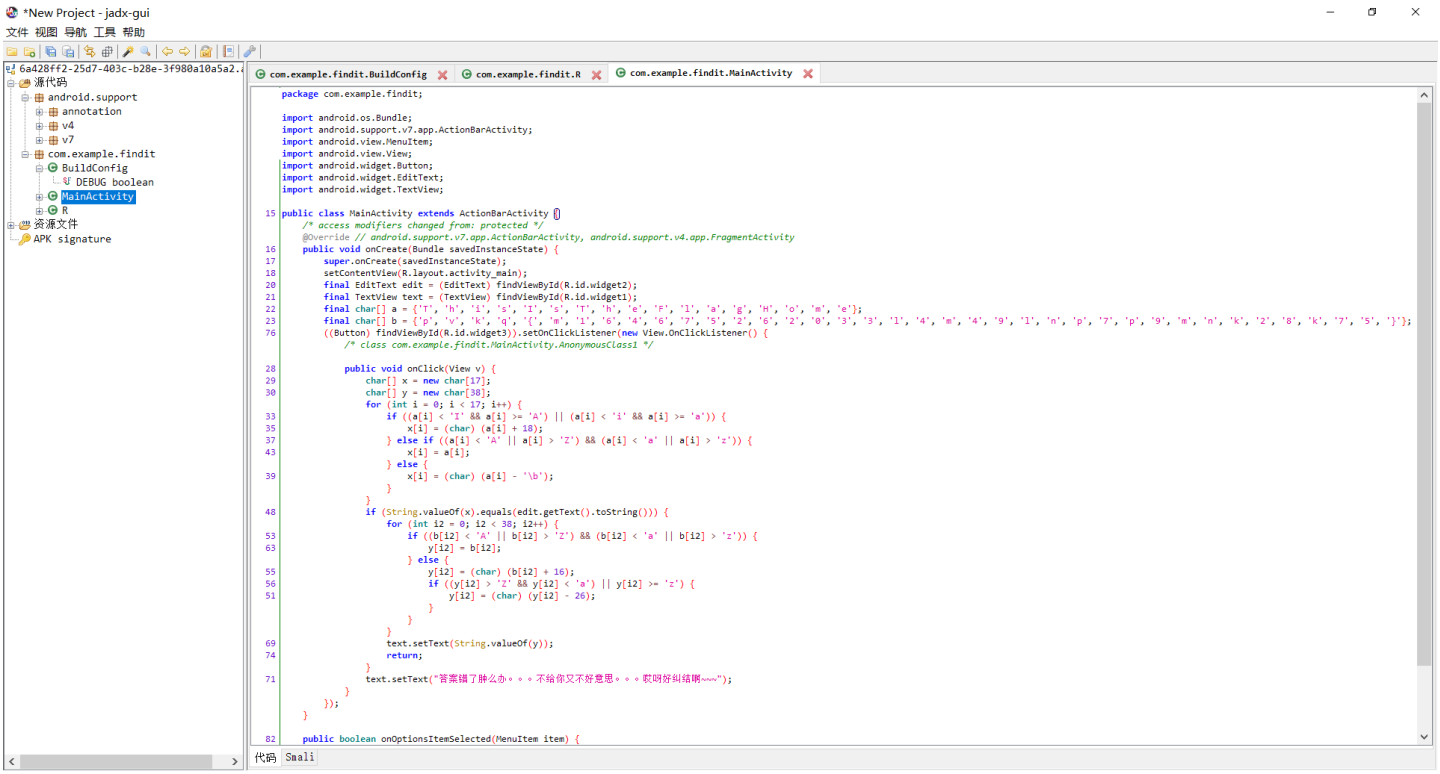
```python
key=[180, 136, 137, 147, 191, 137, 147, 191, 148, 136, 133, 191, 134, 140, 129, 135, 191, 65]
flag=''
for i in key:
    s=(i^32)-ord('@')
    flag+=chr(s)
print flag

```

```
问题  输出  调试控制台  终端                                  2: Python Debug Consc

尝试新的跨平台 PowerShell https://aka.ms/pscore6

加载个人及系统配置文件用了 1052 毫秒。
Xunflash@LAPTOP-CAQITUFD D: > py2  & 'C:\Python27\python2.exe' 'c:\Users\Xunflash\.vscode\extensions\ms-python.python-2021.3.680753044\pythonFiles\lib\python\debugpy\launcher' '61234' '--' 'd:\py2\buu re11.py'
This_is_the_flag_!
```

findit

21.5.1

下载下来发现是个apk，直接jadx打开，源码直接出来了



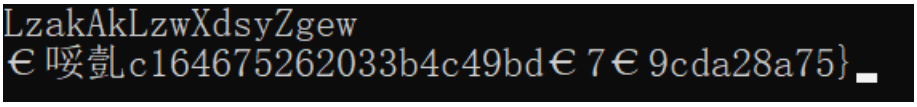
代码都不用写，直接挑出关键代码稍作修改放到vscode里面运行一下就可以了

```
#include<stdio.h>

int main(){
    char a[] = {'T', 'h', 'i', 's', 'I', 's', 'T', 'h', 'e', 'F', 'l', 'a', 'g', 'H', 'o', 'm', 'e'};
    char b[] = {'p','v','k','q','{', 'm', '1', '6', '4', '6', '7', '5', '2', '6', '2', '0', '3', '3', '1', '4',
'm', '4', '9', '1', 'n', 'p', '7', 'p', '9', 'm', 'n', 'k', '2', '8', 'k', '7', '5', '}'};
    char x[17];
    char y[38];

    for (int i = 0; i < 17; i++) {
        if ((a[i] < 'I' && a[i] >= 'A') || (a[i] < 'i' && a[i] >= 'a')) {
            x[i] = (char) (a[i] + 18);
        } else if ((a[i] < 'A' || a[i] > 'Z') && (a[i] < 'a' || a[i] > 'z')) {
            x[i] = a[i];
        } else {
            x[i] = (char) (a[i] - '\b');
        }
    }
```

```
for (int i2 = 0; i2 < 38; i2++) {
if ((b[i2] < 'A' || b[i2] > 'Z') && (b[i2] < 'a' || b[i2] > 'z')) {
y[i2] = b[i2];
} else {
y[i2] = (char) (b[i2] + 16);
if ((y[i2] > 'Z' && y[i2] < 'a') || y[i2] >= 'z') {
y[i2] = (char) (y[i2] - 26);
}
}
}
for(int i =0;i<17;i++){
printf("%c",x[i]);
```

```
    }
    printf("\n");
    for(int j=0;j<38;j++){
        printf("%c",y[j]);
    }
}
```

不知道为啥输出y会乱码



于是就交给手机，把x输进去框框里面，得到flag

[ ](https://cdn.jsdelivr.net/gh/Xunflash/pic/Xunflash-pic/ctf re find it.jpg)

[Zer0pts2020]easy strcmp

说实话，这个题挺折腾人的，不过也是因为自己python掌握的不好，类型转换老是搞不明白

不过动调能解决的话就很不错了

拿到题目，上来直接一个strcmp，真的很怪

```
__int64 __fastcall main(int a1, char **a2, char **a3)
{
  if ( a1 > 1 )
  {
    if ( !strcmp(a2[1], "zer0pts{********CENSORED********}") )
      puts("Correct!");
    else
      puts("Wrong!");
  }
  else
  {
    printf("Usage: %s <FLAG>\n", *a2);
  }
  return 0LL;
}
```

flag也不是这个东西。边上还有几个奇怪的函数，于是找到了这两个

```
// write access to const memory has been detected, the output may be wrong!
int (**sub_5570C4E00795())(const char *s1, const char *s2)
{
  int (**result)(const char *, const char *); // rax

  result = &strcmp;
  realstrcmp = (__int64 (__fastcall *)(_QWORD, _QWORD))&strcmp;
  off_5570C5001028 = sub_5570C4E006EA;
  return result;
}
```

这个把strcmp的地址改了，为了方便，我重新命名了一下

```
__int64 __fastcall sub_5570C4E006EA(_BYTE *a1, __int64 a2)
{
  int i; // [rsp+18h] [rbp-8h]
  int len_input; // [rsp+18h] [rbp-8h]
  int j; // [rsp+1Ch] [rbp-4h]

  for ( i = 0; a1[i]; ++i )
    ;
  len_input = (i >> 3) + 1;
  for ( j = 0; j < len_input; ++j )
    *(_QWORD *)&a1[8 * j] += qword_5570C5001060[j];
  return realstrcmp(a1, a2);
}
```

（忽略那个+=，我改过，原来是-=的）

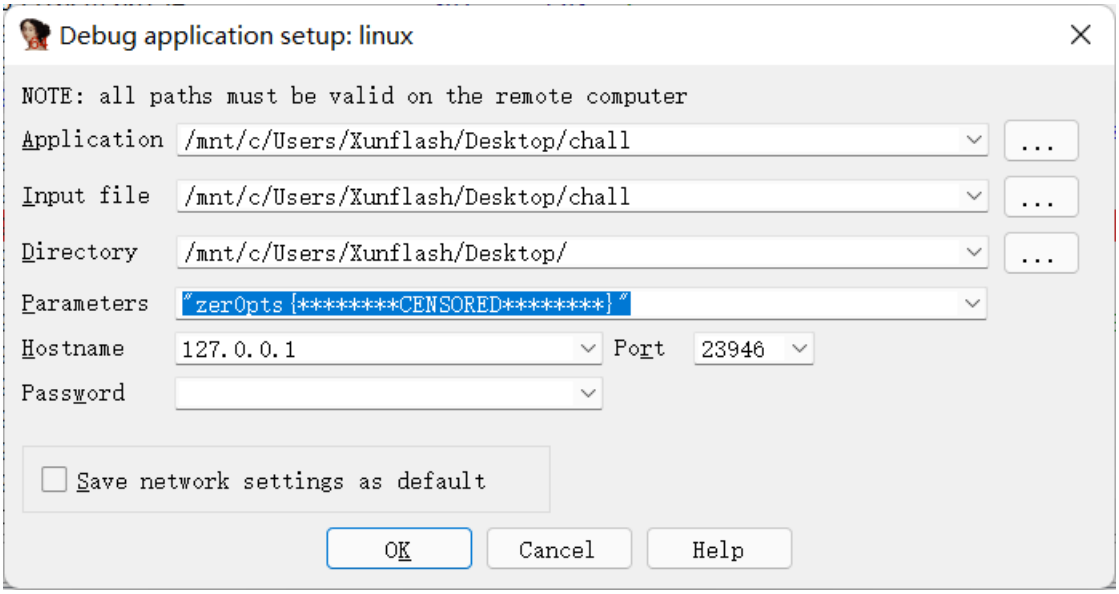
这个“假的”strcmp就是这么个逻辑，把一坨东西和输入进去的相减，最后得到`zer0pts{********CENSORED********}`这个东西

那就加回来就行了

python脚本不会类型转换，c语言写也报错，离谱了，于是我直接动调，算是一种新思路吧

```
.text:00005570C4E0072C ; 10:     *(_QWORD *)&a1[8 * j] -= qword_55D075601060[j];
.text:00005570C4E0072C
.text:00005570C4E0072C loc_5570C4E0072C:                        ; CODE XREF: sub_5570C4E006EA+90↓j
.text:00005570C4E0072C                  mov     eax, [rbp+var_4]
.text:00005570C4E0072F                  shl     eax, 3
.text:00005570C4E00732                  movsxd  rdx, eax
.text:00005570C4E00735                  mov     rax, [rbp+var_18]
.text:00005570C4E00739                  add     rax, rdx
.text:00005570C4E0073C                  mov     rcx, [rax]
.text:00005570C4E0073F                  mov     eax, [rbp+var_4]
.text:00005570C4E00742                  cdqe
.text:00005570C4E00744                  lea     rdx, ds:0[rax*8]
.text:00005570C4E0074C                  lea     rax, qword_5570C5001060
.text:00005570C4E00753                  mov     rdx, [rdx+rax]
.text:00005570C4E00757                  mov     eax, [rbp+var_4]
.text:00005570C4E0075A                  shl     eax, 3
.text:00005570C4E0075D                  movsxd  rsi, eax
.text:00005570C4E00760                  mov     rax, [rbp+var_18]
.text:00005570C4E00764                  add     rax, rsi
.text:00005570C4E00767                  add     rcx, rdx        ; Keypatch modified this from:
.text:00005570C4E00767                                          ;   sub rcx, rdx
.text:00005570C4E0076A                  mov     rdx, rcx
.text:00005570C4E0076D                  mov     [rax], rdx
.text:00005570C4E00770                  add     [rbp+var_4], 1
.text:00005570C4E00774
```

这个地方原本是sub的，改成add，然后动调，参数直接写成这样



然后断点在图上那个地方，按几次f4，然后点进去看flag就行了

```
DEE8 db  6Ch ; l
DEE9 db    0
DEEA aZer0ptsL3tsM4k db 'zer0pts{l3ts_m4k3_4_DETOUR_t0d4y}',0
DF0C db  48h ; H
DF0D db  4Fh ; O
```

整个就是属于是一个脑洞题...

[UTCTF2020]basic-re

都不想放上来...以后这种太简单的题就不放出来了
直接string或者shift+f12就有的

flag:flag{str1ngs_1s_y0ur_fr13nd}

[FlareOn6]Overlong

动调改一下就好了1C改成7F

```
.text:002311C0 ; int __stdcall start(int, int, int, int)
.text:002311C0                 public start
.text:002311C0 start           proc near
.text:002311C0
.text:002311C0 Text            = byte ptr -84h
.text:002311C0 var_4           = dword ptr -4
.text:002311C0
.text:002311C0                 push    ebp
.text:002311C1                 mov     ebp, esp
.text:002311C3                 sub     esp, 84h
.text:002311C9                 push    7Fh
.text:002311CB                 push    offset unk_232008
.text:002311D0                 lea     eax, [ebp+Text]
.text:002311D6                 push    eax
.text:002311D7                 call    sub_231160
.text:002311DC                 add     esp, 0Ch
.text:002311DF                 mov     [ebp+var_4], eax
.text:002311E2                 mov     ecx, [ebp+var_4]
.text:002311E5                 mov     [ebp+ecx+Text], 0
.text:002311ED                 push    0               ; uType
.text:002311EF                 push    offset Caption  ; "Output"
.text:002311F4                 lea     edx, [ebp+Text]
.text:002311FA                 push    edx             ; lpText
.text:002311FB                 push    0               ; hWnd
.text:002311FD                 call    ds:MessageBoxA
.text:00231203                 xor     eax, eax
.text:00231205                 mov     esp, ebp
.text:00231207                 pop     ebp
.text:00231208                 retn    10h
.text:00231208 start           endp
```

因为字符串很长，而for循环只循环前面28位，改长一点就行了。然后断在start函数的return 0，再点进text复制栈上内容就行了。