BUUCTF-逆向

原创

Jokermans 于 2021-04-11 22:11:04 发布 337 收藏 5

文章标签： 深度学习

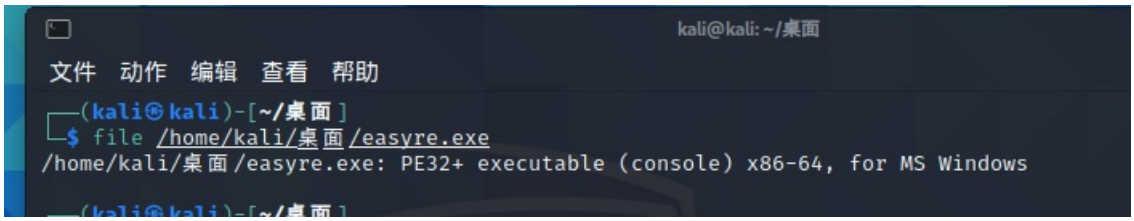
本文链接：https://blog.csdn.net/weixin_52125240/article/details/115583889

BUUCTF-Re

BUUCTF-逆向

1.easyre

先用虚拟机看看是多少位的文件；

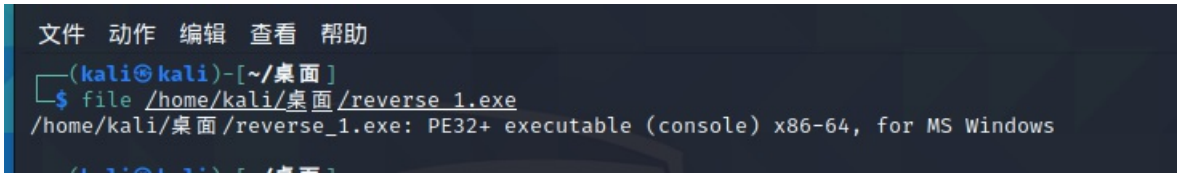


拖进IDA看看，一下子就出来了；

```
sub     rsp, 30h
call    __main
lea     rdx, [rbp+b]
lea     rax, [rbp+a]
mov     r8, rdx
mov     rdx, rax
lea     rcx, aDD        ; "%d%d"
call    scanf
mov     edx, [rbp+a]
mov     eax, [rbp+b]
cmp     edx, eax
jnz     short loc_40152F
lea     rcx, aFlagThis_is_a_ ; "flag{this_Is_a_EaSyRe}"
call    printf
jmp     short loc_40153B
```

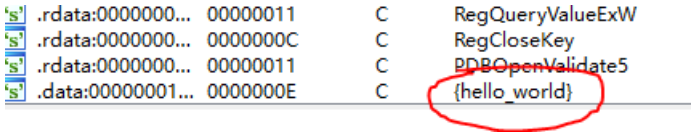

2.reverse1

先用虚拟机看看是多少位的文件；



打开IDA，查找字符串；

点击View->Open subviews->Strings



然后观察伪代码

```
for ( *(&v6 + 1) = 0; ; ++*(&v6 + 1) )
{
  v8 = *(&v6 + 1);
  v2 = j_strlen(Str2);
  if ( v8 > v2 )
    break;
  if ( Str2[(signed __int64)*(&v6 + 1)] == 'o' )
    Str2[(signed __int64)*(&v6 + 1)] = '0';
}
```

所有的 ' o ' 全都变成 ' 0 ' ；

> flag{hell0_w0rld}

3.reverse2

一样的看看文件是多少位的；

```
┌──(kali㉿kali)-[~/桌面]
└─$ file /home/kali/桌面/reverse_2
/home/kali/桌面/reverse_2: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter
/lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=4957efbd7aceb0b4e234593bb09f7af414ddfc22, not s
tripped
```

在IDA中F5反汇编，得到伪代码；

```
  for ( i = 0; i <= strlen(&flag); ++i )
  {
    if ( *(&flag + i) == 'i' || *(&flag + i) == 'r' )
      *(&flag + i) = '1';
  }
}
printf("input the flag:", argv);
__isoc99_scanf();
if ( !strcmp(&flag, &s2) )
  result = puts("this is the right flag!");
else
  result = puts("wrong flag!");
v4 = *MK_FP(__FS__, 40LL) ^ v9;
return result;
```


大概意思是如果字符串里面有 ' i ' 或者 ' r ',则变更成 ' 1 ' ；

在IDA中，找字符串，找到了flag；

.rodata:000000...	00000010	C	input the flag:
.rodata:000000...	00000005	C	%20s
.rodata:000000...	0000000C	C	wrong flag!
.rodata:000000...	00000018	C	this is the right flag!
.eh_frame:0000...	00000006	C	;*3$\"
.data:00000000...	00000011	C	hacking_for_fun}

所以根据代码提示
最终flag为

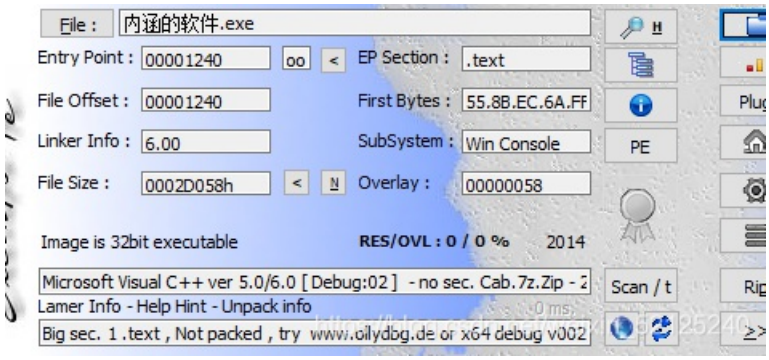
> flag{hack1ng_fo1_fun}

4.内涵的软件

查看位数

```
┌──(kali㉿kali)-[~/桌面]
└─$ file /home/kali/桌面/内涵的软件.exe
/home/kali/桌面/内涵的软件.exe: PE32 executable (console) Intel 80386, for MS Windows
```

工具查看一下，无壳；



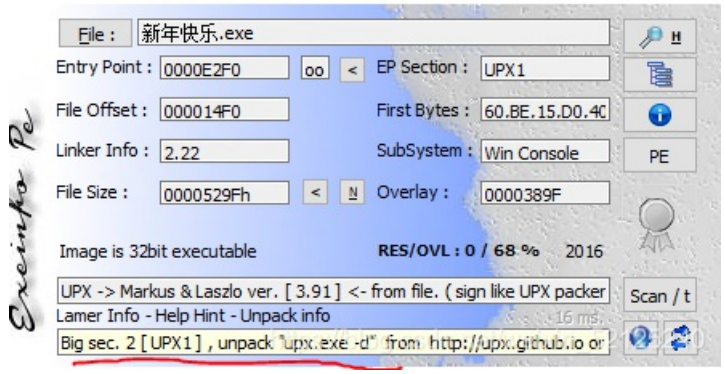
IDA32位打开就有，简单的考验眼力；

```
x, 13h
x, 0CCCCCCCCh

bp+var_4], 5
bp+var_8], offset aDbapp49d3c93df ; "DBAPP{49d3c93df25caad81232130f3d2ebfad}"
ort loc_401081
-------------------------------------------------
```

> flag{49d3c93df25caad81232130f3d2ebfad}

5.新年快乐

查壳一查，发现有壳；



拖到虚拟机里面去脱壳；

然后file一下，看出是32位的，再拖到IDA里面去看看

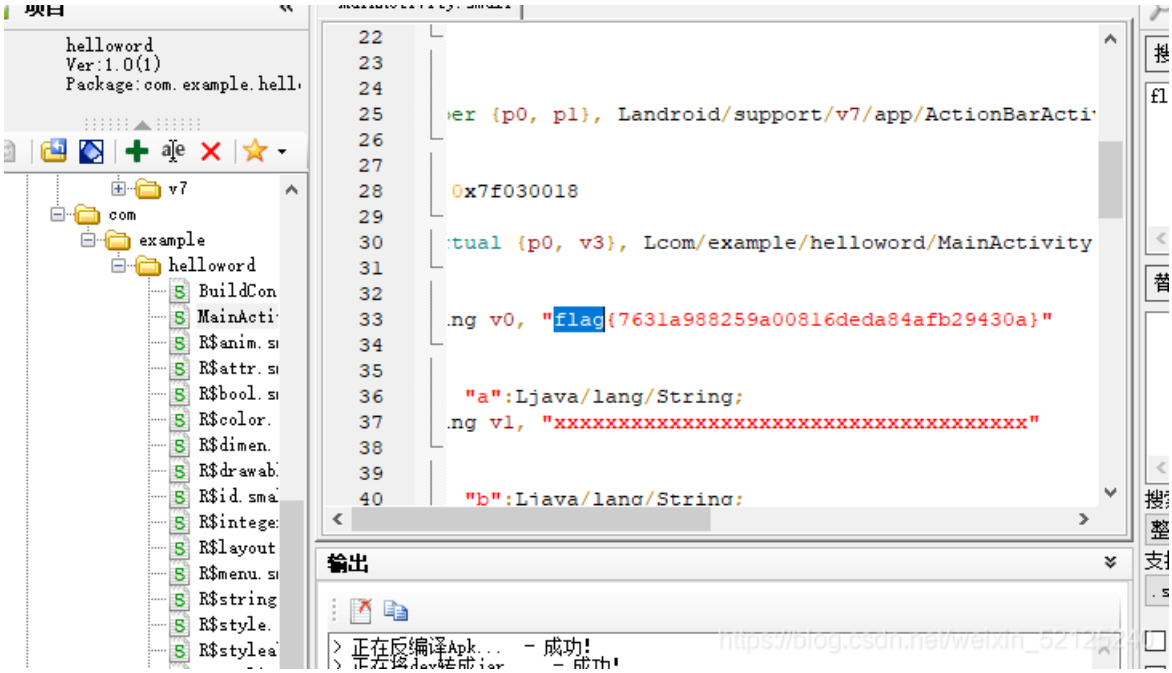
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int result; // eax@2
  char v4; // [sp+12h] [bp-3Ah]@1
  __int16 v5; // [sp+20h] [bp-2Ch]@1
  __int16 v6; // [sp+22h] [bp-2Ah]@1

  __main();
  qmemcpy(&v4, "HappyNewYear!", '\x0E');
  v5 = 0;
  memset(&v6, 0, 0x1Eu);
  printf("please input the true flag:");
  scanf("%s", &v5);
  if ( !strncmp((const char *)&v5, &v4, strlen(&v4)) )
    result = puts("this is true flag!");
  else
    result = puts("wrong!");
  return result;
}
```

主函数很简单，就不用细说了，就是一个比较，
得到flag为：

> flag{HappyNewYear!}

6.helloword

是一个apk文件，android逆向；
用APKIDE打开附件
搜索flag



> flag{7631a988259a00816deda84afb29430a}

7.xor

异或运算，没有加壳；
拖进IDA看一下；

```
memset(v7, 0, 0x100uLL);
v3 = 256LL;
printf("Input your flag:\n", 0LL);
get_line(v7, 256LL);
if ( strlen(v7) != 33 )
  goto LABEL_13;
for ( i = 1; i < 33; ++i )
  v7[i] ^= v7[i - 1];
v3 = (signed __int64)global;
if ( !strncmp(v7, global, 0x21uLL) )
  printf("Success", v3);
else
LABEL_13:
  printf("Failed", v3);
```

逻辑很简单，就是把下一个和之前的一个异或，然后保存。

```
aXZUPFVMD 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',6,'h',0Fh,'G2O',0
YourFlag  db 'Input your flag:',0Ah,0 ; DATA XREF: _main+B↑o
```

这里的数据比较奇怪了，既有数据，又有字符。所以不要单纯的复制下来，要转换为ascii码，然后进行异或，最后贴上脚本。

```
a=[0x66,0x0A,0x6b,0x0C,0x77,0x26,
   0x4f,0x2e,0x40,0x11,0x78,0x0D,
   0x5a,0x3b,0x55,0x11,
   0x70,0x19,0x46,0x1F,0x76,0x22,
   0x4d,0x23,0x44,0x0E,0x67,
   0x06,0x68,0x0F,0x47,0x32,0x4f]
s=' '
s+='f'
for i in range(1,len(a)):
    s+=chr(a[i]^a[i-1])
print(s)
```

得出来flag；

> flag{QianQiuWanDai_YiTongJiangHu}

8.reverse3

首先打开IDA返汇编得到伪代码；

```
sub_41132F("please enter the flag:", v4);
sub_411375("%20s", (unsigned int)&Str);       // 输入str
v0 = j_strlen(&Str);
v1 = sub_4110BE((int)&Str, v0, (int)&v11);    // str进入sub_4110BE加密
strncpy(Dest, (const char *)v1, 0x28u);
sub_411127();
i = j_strlen(Dest);
for ( j = 0; (signed int)j < (signed int)i; ++j )// 进入for循环得到Dest
  Dest[j] += j;
v2 = j_strlen(Dest);
strncmp(Dest, Str2, v2);                      // Str2与Dest进行比较
if ( sub_411127() )
  sub_41132F("wrong flag!\n", v4);
else
  sub_41132F("rigth flag!\n", v4);
```


点击Str2，看一下其中的内容；

```
34 ; char Str2[]
34 Str2            db 'e3nifIH9b_C@n@dH',0 ;
45                 db    0
```

然后点击sub_4110BE，看一下加密运算；

```
{
  *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[(signed int)(un
  *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[((byte_41A144[1
  *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[64];
  *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[64];
}
```

发现使用了这个进行数组变换，跟进发现就是之前的base64。
首先for循环将Dest每一位都加了j，所以写脚本还原v1，然后进行base64解码

```
import base64

str='e3nifIH9b_C@n@dH'
flag=' '
for i in range(len(str)):
    flag+=chr(ord(str[i])-i)
flag=base64.b64decode(flag)
print(flag)
```

得到flag

flag{i_l0ve_you}

9.不一样的flag

使用IDA一打开就看到一个可以数据；

```
sub     esp, 40h
call    ___main
mov     dword ptr [esp+30h], 0
mov     dword ptr [esp+34h], 0
lea     edx, [esp+17h]
mov     ebx, offset __data_start__ ; "*11110100001010000101111#"
mov     eax, 19h
mov     edi, edx
mov     esi, ebx
mov     ecx, eax
```

以为是十六进制编码啥的，发现都不对；

发现反汇编出来的伪代码也有点奇奇怪怪的；

```
v2 = 0;
qmemcpy(&v0, _data_start__, 0x19u);
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", &v3);
  if ( v3 == 2 )
  {
    ++v1;
  }
  else if ( v3 > 2 )
  {
    if ( v3 == 3 )
```


可以发现有上下左右四个方向，并且碰到1退出，碰到#输出flag，应该是一个迷宫，再查看01串长度正好25；

```
*1111
01000
01010
00010
1111#
```

得到222441144222

flag{222441144222}

10.SimpleRev

首先就是一个普普通通的64位文件，简单题就不看有没有加壳了，大概率是没有的。

老样子拖进IDA看看情况。

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  int v3; // eax@7
  char v4; // [sp+Fh] [bp-1h]@1

  while ( 1 )
  {
    while ( 1 )
    {
      printf("Welcome to CTF game!\nPlease input d/D to start or input q/Q to quit this program: ", argv, envp);
      v4 = getchar();
      if ( v4 != 100 && v4 != 68 )
        break;
      Decry();
    }
    if ( v4 == 113 || v4 == 81 )
      Exit();
    puts("Input fault format!");
    v3 = getchar();
    putchar(v3);
  }
}
```


这个伪代码一看就不简单。

关键就在于Dercy()这个函数，点进去看一下。

```
v12 = *MK_FP(__FS__, 40LL);
 *(_QWORD *)src = 'SLCDN';     //NDCLS
 v7 = 0LL;
 v8 = 0;
 v9 = 'wodah';         //hadow
 v10 = 0LL;
 v11 = 0;
 text = join(key3, (const char *)&v9);
 strcpy(key, key1);
 strcat(key, src);
 v2 = 0;
 v3 = 0;
 getchar();
 v5 = strlen(key);
 for ( i = 0; i < v5; ++i )
 {
   if ( key[v3 % v5] > '@' && key[v3 % v5] <= 'Z' )
     key[i] = key[v3 % v5] + 32;
   ++v3;
 }
 printf("Please input your flag:", src);
 while ( 1 )
 {
   v1 = getchar();
   if ( v1 == '\n' )
     break;
   if ( v1 == ' ' )
   {
     ++v2;
   }
   else
   {
     if ( v1 <= 96 || v1 > 122 )
     {
       if ( v1 > '@' && v1 <= 'Z' )
         str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
     }
     else
     {
       str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
     }
     if ( !(v3 % v5) )
       putchar(32);
     ++v2;
   }
 }
 if ( !strcmp(text, str2) )
   puts("Congratulation!\n");
 else
   puts("Try again!\n");
 return *MK_FP(__FS__, 40LL) ^ v12;
}
```

```
v12 = *MK_FP(__FS__, 40LL);
*(_QWORD *)src = 0x534C43444ELL;
v7 = 0LL;
v8 = 0;
v9 = 0x776F646168LL;
v10 = 0LL;
v11 = 0;
```

将src和v9的值换成十六进制，可以到这两个是大端序，但是数据在内存中都是小端序，所以我们要将其转一下，一般在CPU，x86都是小端序，但是IDA将其转换为大端序。（具体如何判断我也不是很清楚，查了很多也没有一个所以然，大家就先这样记忆吧）

src = NDCLS, v9 = hadow;

join连接key3和v9，key3点进去得到kills，所以text = killshadow。

strcpy(key, key1); //复制函数，将key1的字符串复制给key。

key1点进去得到ADSFK。

strcat函数，是将key 和 src 拼接在一起的函数， 所以key = ADSFKNDCLS。

```
v5 = strlen(key);
for ( i = 0; i < v5; ++i )
{
  if ( key[v3 % v5] > '@' && key[v3 % v5] <= 'Z' )
    key[i] = key[v3 % v5] + 32;
  ++v3;
}
```

这个key[v3%v5]是大写字母，则把它变为小写字母。**大小写字母的ASCII码，相差32。**

```
    else
    {
      if ( v1 <= '`' || v1 > 'z' )
      {
        if ( v1 > '@' && v1 <= 'Z' )
          str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
      }
      else
      {
        str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97;
      }
      if ( !(v3 % v5) )
        putchar(' ');
      ++v2;
    }
  }
  if ( !strcmp(text, str2) )
    puts("Congratulation!\n");
  else
    puts("Try again!\n");
  return __readfsqword(0x28u) ^ v12;
}
```


看到最下面的if语句，如果！strcmp（text，str2），则正确，strcmp函数，是比较函数，如果两个字符串相同，则等于0，所以text = str2才成功。

而得到str2的关键就是：
str2[v2] = (v1 - 39 - key[v3++ % v5] + 97) % 26 + 97

接下来，写脚本。

```
lt='ABCDEFGHIJKLMNOPQRSTUVWXYZ'
key=list('ADSFKNDCLS'.lower())
klens=len(key)

text='killshadow'
flag=''
for i in range(len(text)):
    str2=text[i]
    for c in lt:
        if str2== chr((ord(c) - 39 - ord(key[i  % klens]) + 97) % 26 + 97):
            flag+=c
print('flag{'+flag+'}')
```

lt可以试试大写也可以试试小写，这里我个人不确定是大写还是小写，不过大写答案正确

> flag{KLDQCUDFZO}

11.Java逆向解密

文件下载完成后打开是.class文件。

.class文件是.java文件编译后生成的字节码文件，我们使用一般的文本编辑工具打开的话，里面的内容是乱码。 使用专业的集成开发工具（IDE）可以打开.class文件，如eclipse，idea等。

下载了打开反汇编Java的工具

```
import java.util.ArrayList;
import java.util.Scanner;

public class Reverse {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.println("Please input the flag : ");
    String str = s.next();
    System.out.println("Your input is : ");
    System.out.println(str);
    char[] stringArr = str.toCharArray();
    Encrypt(stringArr);
  }

  public static void Encrypt(char[] arr) {
    ArrayList<Integer> Resultlist = new ArrayList<>();
    for (int i = 0; i < arr.length; i++) {
      int result = arr[i] + 64 ^ 0x20;
      Resultlist.add(Integer.valueOf(result));
    }
    int[] KEY = {
        180, 136, 137, 147, 191, 137, 147, 191, 148, 136,
        133, 191, 134, 140, 129, 135, 191, 65 };
    ArrayList<Integer> KEYList = new ArrayList<>();
    for (int j = 0; j < KEY.length; j++)
      KEYList.add(Integer.valueOf(KEY[j]));
    System.out.println("Result:");
    if (Resultlist.equals(KEYList)) {
      System.out.println("Congratulations! ");
    } else {
      System.err.println("Error! ");
    }
  }
}
```


逻辑清晰简单，就是我们输入一个字符串
然后经过一个for循环进行异或
然后将得到的新字符串与KEY进行比较，看看是否相等。

解码的python脚本如下

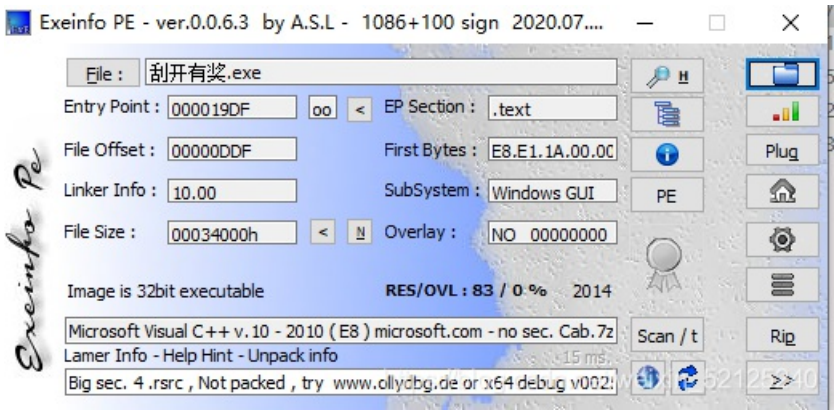
```
str = [180, 136, 137, 147, 191, 137, 147, 191, 148,
       136, 133, 191, 134, 140, 129, 135, 191, 65 ]

flag=' '

for i in range(len(str)):
    flag+=chr(str[i]-ord('@')^0x20)
print(flag)
```

得到flag为

> flag{ This_is_the_flag_!}

12.刮开有奖

首先还是例行检查，看看有没有壳。



正常的文件，没有加壳。

然后我试了试x64dbg,不过并没有什么用处。

用IDA试了试，看到了DialogFunc函数，反汇编得到伪代码。

```
BOOL __stdcall DialogFunc(HWND hDlg, UINT a2, WPARAM a3, LPARAM a4)
{
  const char *v4; // esi@5
  const char *v5; // edi@5
  BOOL result; // eax@14
  int v7; // [sp+8h] [bp-20030h]@5
  int v8; // [sp+Ch] [bp-2002Ch]@5
  int v9; // [sp+10h] [bp-20028h]@5
  int v10; // [sp+14h] [bp-20024h]@5
  int v11; // [sp+18h] [bp-20020h]@5
  int v12; // [sp+1Ch] [bp-2001Ch]@5
  int v13; // [sp+20h] [bp-20018h]@5
  int v14; // [sp+24h] [bp-20014h]@5
  int v15; // [sp+28h] [bp-20010h]@5
  int v16; // [sp+2Ch] [bp-2000Ch]@5
  int v17; // [sp+30h] [bp-20008h]@5
  CHAR String; // [sp+34h] [bp-20004h]@4
  char v19; // [sp+35h] [bp-20003h]@6
  char v20; // [sp+36h] [bp-20002h]@5
  char v21; // [sp+37h] [bp-20001h]@5
  char v22; // [sp+38h] [bp-20000h]@5
  char v23; // [sp+39h] [bp-1FFFFh]@5
  char v24; // [sp+3Ah] [bp-1FFFEh]@5
  char v25; // [sp+3Bh] [bp-1FFFDh]@5
  char v26; // [sp+10034h] [bp-10004h]@5
  char v27; // [sp+10035h] [bp-10003h]@5
  char v28; // [sp+10036h] [bp-10002h]@5

  if ( a2 == 272 )
  {
    result = 1;
  }
  else
  {
    if ( a2 != 273 )
      return 0;
    if ( (_WORD)a3 == 1001 )
    {
      memset(&String, 0, 0xFFFFu);
      GetDlgItemTextA(hDlg, 1000, &String, 0xFFFF);
```

```
    if ( strlen(&String) == 8 )
    {
      v7 = 'Z';
      v8 = 'J';
      v9 = 'S';
      v10 = 'E';
      v11 = 'C';
      v12 = 'a';
      v13 = 'N';
      v14 = 'H';
      v15 = '3';
      v16 = 'n';
      v17 = 'g';
      sub_4010F0((int)&v7, 0, 10);
      memset(&v26, 0, 65535u);
      v26 = v23;
      v28 = v25;
      v27 = v24;
      v4 = (const char *)sub_401000((int)&v26, strlen(&v26));
      memset(&v26, 0, 0xFFFFu);
      v27 = v21;
      v26 = v20;
      v28 = v22;
      v5 = (const char *)sub_401000((int)&v26, strlen(&v26));
      if ( String == v7 + 34
        && v19 == v11
        && 4 * v20 - 141 == 3 * v9
        && v21 / 4 == 2 * (v14 / 9)
        && !strcmp(v4, "ak1w")
        && !strcmp(v5, "V1Ax") )
        MessageBoxA(hDlg, "U g3t 1T!", "@_@", 0);
    }
    return 0;
  }
  if ( (_WORD)a3 != 1 && (_WORD)a3 != 2 )
    return 0;
  EndDialog(hDlg, (unsigned __int16)a3);
  result = 1;
}
return result;
}
```

大概可以看出我们的flag是8位长度的字符串。

```
  memset(&String, 0, 0xFFFFu);
  GetDlgItemTextA(hDlg, 1000, &String, 0xFFFF);
  if ( strlen(&String) == 8 )
  {
```

看伪代码分析太麻烦了，所以转成C语言代码运行一下。记得把*(_DWORD*) 删掉，因为这是汇编的表示，然后将各种基址+偏移的表示也换成数组的寻址,如下

```c
#include <stdio.h>
#include <string.h>

int sub_4010F0(char*a1,int a2,int a3)
{
 int result;
 int i;
 int v5;
 int v6;

 result=a3;
 for(i=a2;i<=a3;a2=i)
 {
  v5=i;
  v6=a1[i];
  if(a2<result&&i<result)
  {
   do
   {
    if(v6>a1[result])
    {
     if(i>=result)
      break;
     ++i;
     a1[v5]=a1[result];
     if(i>=result)
      break;
     while(a1[i]<=v6)
     {
      if(++i>=result)
       goto LABEL_13;
     }
     if(i>=result)
      break;
     v5=i;
     a1[result]=a1[i];
    }
    --result;
   }
   while(i<result);
  }
 LABEL_13:
  a1[result]=v6;
  sub_4010F0(a1,a2,i-1);
  result=a3;
  ++i;
 }
 return result;
}

int main(void)
{
 char str[]="ZJSECaNH3ng";
 sub_4010F0(str,0,10);
 printf("%s",str);
 return 0;
}
```

得到结果为3CEHJNSZagn

分析代码，转到汇编处。

```
.text:004012B0                push    0FFFFh          ; size_t
.text:004012B5                lea     edx, [ebp+var_10004]
.text:004012BB                push    0               ; int
.text:004012BD                push    edx             ; void *
.text:004012BE                call    _memset
.text:004012C3                 mov     al, [ebp+var_1FFFF]
.text:004012C9                 mov     dl, [ebp+var_1FFFD]
.text:004012CF                 mov     cl, [ebp+var_1FFFE]
.text:004012D5                mov     [ebp+var_10004], al
.text:004012DB                lea     eax, [ebp+var_10004]
.text:004012E1                mov     [ebp+var_10002], dl
.text:004012E7                add     esp, 18h
.text:004012EA                mov     [ebp+var_10003], cl
.text:004012F0                lea     edx, [eax+1]
.text:004012F3
.text:004012F3 loc_4012F3:                            ; CODE XREF: DialogFunc+158↓j
.text:004012F3                mov     cl, [eax]
.text:004012F5                inc     eax
.text:004012F6                test    cl, cl
.text:004012F8                jnz     short loc_4012F3
.text:004012FA                sub     eax, edx
.text:004012FC                push    eax
.text:004012FD                lea     eax, [ebp+var_10004]
.text:00401303                push    eax
.text:00401304                call    sub_401000
.text:00401309                push    0FFFFh          ; size_t
.text:0040130E                lea     ecx, [ebp+var_10004]
.text:00401314                push    0               ; int
.text:00401316                push    ecx             ; void *
.text:00401317                mov     esi, eax
.text:00401319                call    _memset
.text:0040131E                 mov     al, [ebp+var_20001]
.text:00401324                 mov     dl, [ebp+var_20002]
.text:0040132A                 mov     cl, [ebp+var_20000]
.text:00401330                mov     [ebp+var_10003], al
.text:00401336                lea     eax, [ebp+var_10004]
.text:0040133C                mov     [ebp+var_10004], dl
.text:00401342                add     esp, 14h
.text:00401345                mov     [ebp+var_10002], cl
.text:0040134B                lea     edx, [eax+1]
.text:0040134E                mov     edi, edi
```



```
-00020004 String db ?
-00020003 var_20003 db ?
-00020002 var_20002 db ?
-00020001 var_20001 db ?
-00020000 var_20000 db ?
-0001FFFF var_1FFFF db ?
-0001FFFE var_1FFFE db ?
-0001FFFD var_1FFFD db ?
```

看加粗加红处（下面是对应字符串的信息）

我们可以知道，v6使用sub_4010F0函数后的字符串的6,7,8位，调用sub_401000函数，v7使用sub_4010F0函数后的字符串的3,4,5位，调用sub_401000函数。（这一步我自己其实一直不能理解。）

然后进入sub_401000,观察里面的加密方式。

```
void *__cdecl sub_401000(int a1, signed int a2)
{
  int v2; // ecx@1
```

```
int v2; // edx@1
signed int v3; // esi@1
size_t v4; // ebx@3
void *v5; // eax@3
void *v6; // edi@3
signed int v7; // eax@5
void *v8; // ebx@5
int v9; // edi@8
signed int v10; // edx@8
signed int v11; // edi@11
signed int v12; // eax@11
signed int v13; // esi@11
void *result; // eax@18
void *v15; // [sp+Ch] [bp-10h]@3
void *v16; // [sp+10h] [bp-Ch]@5
signed int v17; // [sp+14h] [bp-8h]@11
int v18; // [sp+18h] [bp-4h]@8

v2 = a2 / 3;
v3 = 0;
if ( a2 % 3 > 0 )
  ++v2;
v4 = 4 * v2 + 1;
v5 = malloc(v4);
v6 = v5;
v15 = v5;
if ( !v5 )
  exit(0);
memset(v5, 0, v4);
v7 = a2;
v8 = v6;
v16 = v6;
if ( a2 > 0 )
{
  while ( 1 )
  {
    v9 = 0;
    v10 = 0;
    v18 = 0;
    do
    {
      if ( v3 >= v7 )
        break;
      ++v10;
      v9 = *(_BYTE *)(v3++ + a1) | (v9 << 8);
    }
    while ( v10 < 3 );
    v11 = v9 << 8 * (3 - v10);
    v12 = 0;
    v17 = v3;
    v13 = 18;
    do
    {
      if ( v10 >= v12 )
      {
        *((_BYTE *)&v18 + v12) = (v11 >> v13) & 0x3F;
        v8 = v16;
      }
      else
      {
```

```
          *((_BYTE *)&v18 + v12) = 64;
        }
        *(_BYTE *)v8 = byte_407830[*((_BYTE *)&v18 + v12)];
        v8 = (char *)v8 + 1;
        v13 -= 6;
        ++v12;
        v16 = v8;
      }
      while ( v13 > -6 );
      v3 = v17;
      if ( v17 >= a2 )
        break;
      v7 = a2;
    }
    v6 = v15;
  }
  result = v6;
  *(_BYTE *)v8 = 0;
  return result;
}
```

又是一个很复杂的代码，然后我们看到函数byte_407830，点进去看到

```
; char byte_407830[]
byte_407830     db 41h                  ; DATA XREF: sub_401000+C0↑r
aBcdefghijklmno db 'BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=',0
                align 4
aAk1w            db 'ak1w',0            ; DATA XREF: DialogFunc+24D↑o
                align 4
aV1ax            db 'V1Ax',0            ; DATA XREF: DialogFunc+27D↑o
                align 4
```

一看也知道是base64加密。

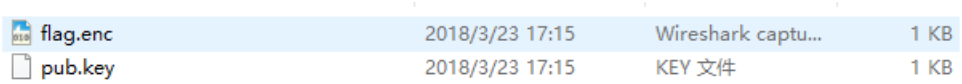
```
    if ( String == v9 + 34                // sub_4010F0函数后的第一位等于51+34=85-->'U'
      && v21 == v13                       // 第2位，等于v13,即sub_4010F0函数返回值的第5位值-->'J'
      && 4 * v22 - 141 == 3 * v11
      && v23 / 4 == 2 * (v16 / 9)
      && !strcmp(v6, "ak1w")              // 第6,7,8行代码base64之后，需要等于"ak1w"
      && !strcmp(                         // 第3,4,5行代码，加密之后等于V1Ax
           v7,
           "V1Ax") )
    {
      MessageBoxA(hDlg, "U g3t 1T!", "@_@", 0);
    }
```

最后得到flag为

> flag{UJWP1jMp}

13.rsa

首先看到这道题目的时候真的是一头雾水，不知道给的文件是用来干什么的；



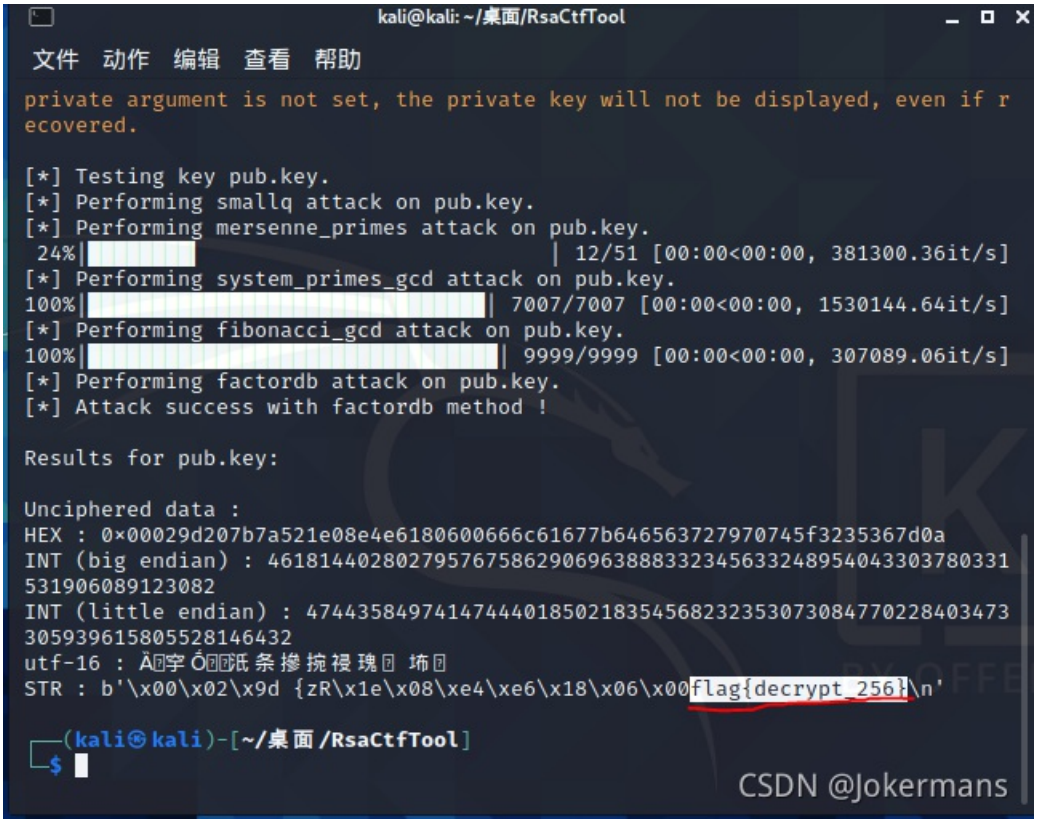
然后就去搜索enc文件，想着可以把它转换成代码，然后去分析代码，果然这条路没有走通，然后就在网上看到别人介绍RsaCtfTool这个工具，下载了好久才下载完成；

看这篇文章，来下载RsaCtfTool；
安装RsaCtfTool

然后把文件复制到Kali中，使用命令

```
python3 RsaCtfTool.py --publickey pub.key --uncipherfile flag.enc
```

得到flag

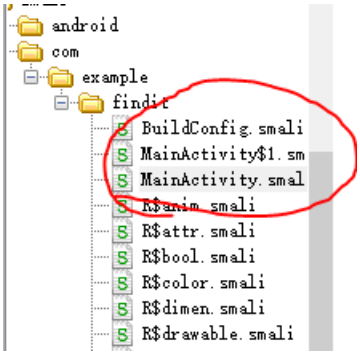


所以flag为

```
flag{decrypt_256}
```

14.findit

文件下载好了，.apk文件，安卓逆向。
用apk改之理看一看



找到主函数。
打开后看到一串十六进制数。

```
.array-data 2
    0x70s
    0x76s
    0x6bs
    0x71s
    0x7bs
    0x6ds
    0x31s
    0x36s
    0x34s
    0x36s
    0x37s
    0x35s
    0x32s
    0x36s
    0x32s
    0x30s
    0x33s
    0x33s
    0x6cs
    0x34s
    0x6ds
    0x34s
    0x39s
    0x6cs
    0x6es
    0x70s
    0x37s
    0x70s
    0x39s
    0x6ds
    0x6es
    0x6bs
    0x32s
    0x38s
    0x6bs
    0x37s
    0x35s
    0x7ds
.end array-data
```

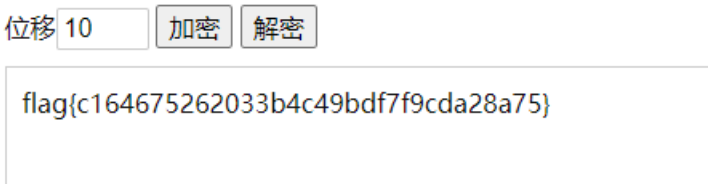

转换成字符的形式。

```
str=[0x70,0x76,0x6B,0x71,0x7b,0x6d,0x31,0x36,0x34,0x36,0x37,0x35,0x32,0x36,0x32,0x30,0x33,0x33,0x6c,
0x34,0x6d,0x34,0x39,0x6c,0x6e,0x70,0x37,0x70,0x39,0x6d,0x6e,0x6b,0x32,0x38,0x6b,0x37,0x35,0x7d]

flag=''
for i in range(0,len(str)):
    flag+=chr(str[i])
print(flag)
```

得到一串字符

> pvkq{m164675262033l4m49lnp7p9mnk28k75}

以为这就是flag，提交后发现不对，仔细观察后感觉像是凯撒密码
解密一下得到



> flag{c164675262033b4c49bdf7f9cda28a75}

15.简单注册器

文件下载好后是一个.apk文件，安卓逆向。
用Jadx-gui反编译，得到主要代码。

```
public void onClick(View v) {
    int flag = 1;
    String xx = editview.getText().toString();
    if (!(xx.length() == 32 && xx.charAt(31) == 'a' && xx.charAt(1) == 'b' && (xx.charAt(0) + xx.charAt(2)) - 48 == 56)) {
        flag = 0;
    }
    if (flag == 1) {
        char[] x = "dd2940c04462b4dd7c450528835cca15".toCharArray();
        x[2] = (char) ((x[2] + x[3]) - 50);
        x[4] = (char) ((x[2] + x[5]) - 48);
        x[30] = (char) ((x[31] + x[9]) - 48);
        x[14] = (char) ((x[27] + x[28]) - 97);
        for (int i = 0; i < 16; i++) {
            char a = x[31 - i];
            x[31 - i] = x[i];
            x[i] = a;
        }
        textview.setText("flag{" + String.valueOf(x) + "}");
        return;
    }
    textview.setText("输入注册码错误");
}
```


然后写脚本把flag跑出来

```
flagtrue = "dd2940c04462b4dd7c450528835cca15"
x = [i for i in flagtrue]
x[2] = chr(ord(x[2]) + ord(x[3]) - 0x32)
x[4] = chr(ord(x[2]) + ord(x[5]) - 0x30)
x[0x1e] = chr(ord(x[0x1f]) + ord(x[0x9]) - 0x30)
x[0xe] = chr(ord(x[0x1b]) + ord(x[0x1c]) - 0x61)

for i in range(16):
    x[i],x[31-i] = x[31-i],x[i]

print ("flag{"+ ''.join(x) + "}")
```

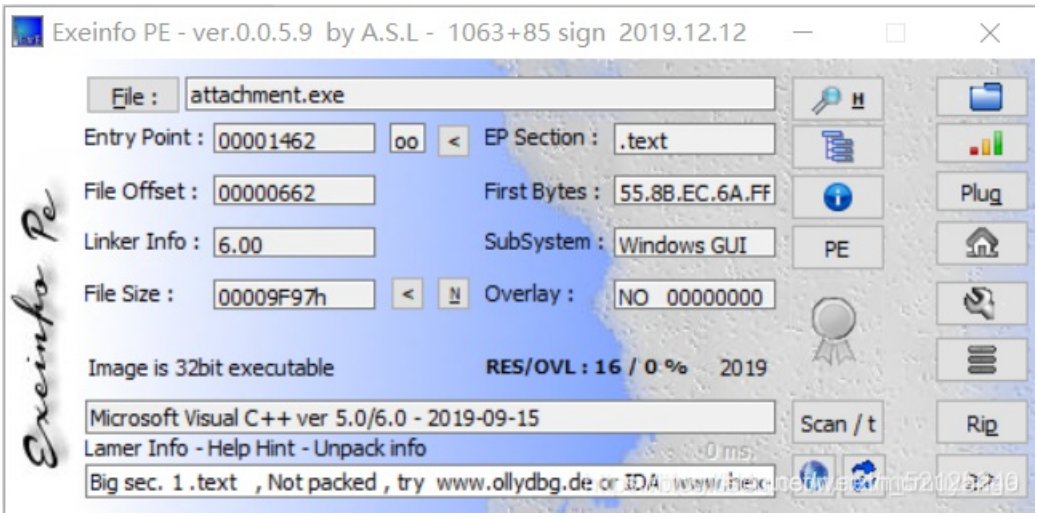
这是别人写的脚本，python没学好，我自己手算的，这么简单都不会，还不赶紧学python。

得到flag为：

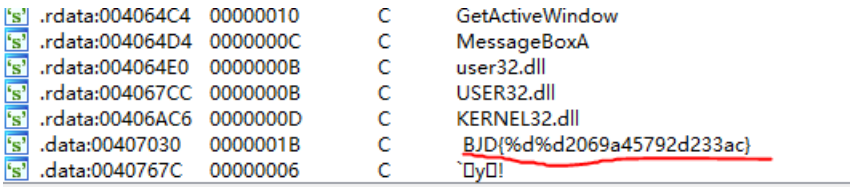
flag{59acc538825054c7de4b26440c0999dd}

16.[BJDCTF2020]JustRE

还是先检查一下有没有壳



没有加壳。
拖到IDA里面看一看。
找到一个比较关键的字符



找到关键函数，反编译一下

```
  if ( a2 != 272 )
  {
    if ( a2 != 273 )
      return 0;
    if ( (_WORD)a3 != 1 && (_WORD)a3 != 2 )
    {
      ++dword_4099F0;
      sprintf(&String, asc_40704C, dword_4099F0);
      if ( dword_4099F0 == 19999 )
      {
        sprintf(&String, aBjdDD2069a4579, 19999, 0);
        SetWindowTextA(hWnd, &String);
        return 0;
      }
      SetWindowTextA(hWnd, &String);
      return 0;
    }
    EndDialog(hWnd, (unsigned __int16)a3);
  }
  return 1;
}
```


输出了aBjdDD2069a4579，19999，0，aBjdDD2069a4579是BJD{%d%d2069a45792d233ac}，19999和0填入到里面的%d位置，得到flag

> flag{1999902069a45792d233ac}

17.[GWCTF 2019]pyre

看到名字的时候就猜到肯定和python有关。
下载文件，果不其然.pyc文件(python代码转换为字节码后的文件)
用python反编译工具：python反编译

打开后得到

```
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',
```


轻轻松松反编译出来
结果我自己数学不好，python写的又烂，还是看了别人的wp才写出来的；
关于异或，我们要知道这样一个技巧：
a ^ 0=a,a ^ a=0,那么可得：a ^ b ^ a=b,a ^ b ^ b = a;
由code[i]=code[i] ^ code[i+1]，i从0取到l-1-1。处理后，code[l-1]没有变，那么要逆向，则令x从l-2取到0，使code[x]=code[x] ^ code[x+1](a ^ b ^ b = a) 。
关于取模，(a%c+b%c)=(a+b)%c,所以num等价于(input[i] +i)%128

解密脚本：

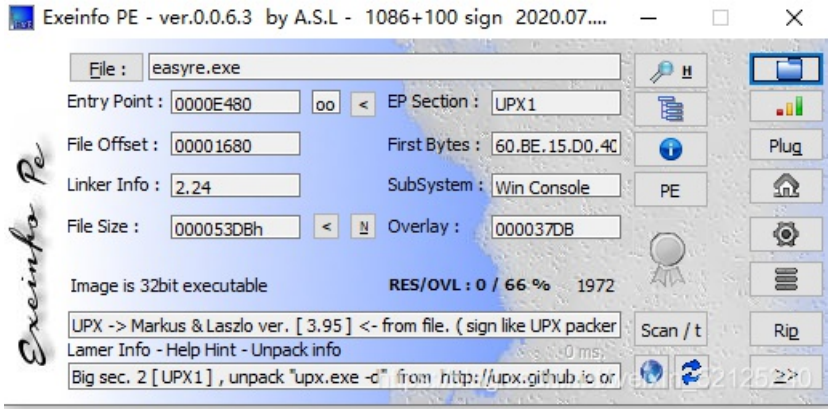
```
code = ['\x1f','\x12','\x1d','(','0','4','\x01','\x06','\x14','4',
        ',','\x1b','U','?','o','6','*',':','\x01','D',';','%','\x13']

l=len(code)
flag=' '
for i in range(l-2,-1,-1):
    code[i]=chr(ord(code[i])^ord(code[i+1]))

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

print(flag)
```

得到flag为GWHT{Just_Re_1s_Ha66y!}

```
flag{Just_Re_1s_Ha66y!}
```

19.[ACTF新生赛2020]easyre

还是先查查壳。



有壳，upx，拖到Linux里面去脱壳。

脱壳后的文件拖到IDA里面反编译一下，点击main()。

得到的伪代码

```
__main();
v4 = 42;
v5 = 70;
v6 = 39;
v7 = 34;
v8 = 78;
v9 = 44;
v10 = 34;
v11 = 40;
v12 = 73;
v13 = 63;
v14 = 43;
v15 = 64;
printf("Please input:");
scanf("%s", &v19);
if ( (_BYTE)v19 == 65 && HIBYTE(v19) == 67 && v20 == 84 && v21 == 70 && v22 == 123 && v26 == 125 )
```

```
{
  v16 = v23;
  v17 = v24;
  v18 = v25;
  for ( i = 0; i <= 11; ++i )
  {
    if ( *(&v4 + i) != _data_start__[*((_BYTE *)&v16 + i) - 1] )
      return 0;
  }
  printf("You are correct!");
  result = 0;
}
else
{
  result = 0;
}
return result;
}
```


代码还算简单，但是我自己理解能力太差，看了好久还是没怎么看明白。
着眼观察for循环就行，从for循环了解到flag长度应该是12，
*(&v4 + i) !=data_start_[*((char *)&v16 + i) - 1]) 的意思是在_data_start__字符串里面寻找一个字符然后 -1 与v4进行对比，注意括号的位置就能知道 -1 是减到索引还是索引值上，所以逆过来就是 +1。

```
__data_start__  db '~'                  ; DATA XREF: _main+ECTr
aZyxwvutsrqponm db '}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>='
                db '<;:9876543210/.-,+*)(',27h,'&%$# !"',0
```

python脚本

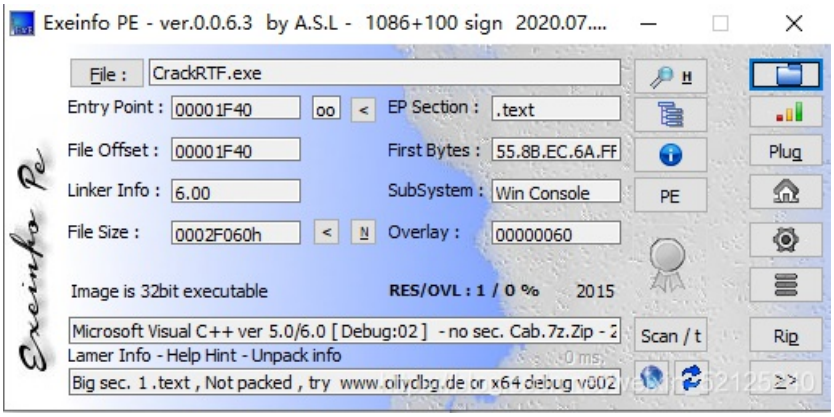
```
key = '~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)(\'&%$# !"' #'一定要加\
encrypt = [42,70,39,34,78,44,34,40,73,63,43,64]
x = []
flag = ''
for i in encrypt:
  x.append(key.find(chr(i))+1)
for i in x:
  flag += chr(i)
print(flag)
```

借鉴一下别人的脚本，自己写的错误百出。
最后flag为：

> flag{U9X_1S_W6@T?}

20.CrackRTF

首先还是看看有没有加壳；



没有加壳，继续分析，拖到IDA里面看看。
这个代码真的是看的我头疼

```
printf("pls input the first passwd(1): ");
scanf("%s", &pbData);
if ( strlen((const char *)&pbData) != 6 )
{
  printf("Must be 6 characters!\n");
  ExitProcess(0);
}
v5 = unknown_libname_1((char *)&pbData);
if ( v5 < 100000 )
  ExitProcess(0);
strcat((char *)&pbData, "@DBApp");
v0 = strlen((const char *)&pbData);
sub_40100A(&pbData, v0, &String1);
if ( !_strcmpi(&String1, "6E32D0943418C2C33385BC35A1470250DD8923A9") )
{
  printf("continue...\n\n");
  printf("pls input the first passwd(2): ");
  memset(&String, 0, 0x104u);
  scanf("%s", &String);
  if ( strlen(&String) != 6 )
  {
    printf("Must be 6 characters!\n");
    ExitProcess(0);
  }
  strcat(&String, (const char *)&pbData);
  memset(&String1, 0, 0x104u);
  v1 = strlen(&String);
  sub_401019((BYTE *)&String, v1, &String1);
  if ( !_strcmpi("27019e688a4e62a649fd99cadaafdb4e", &String1) )
  {
    if ( !(unsigned __int8)sub_40100F(&String) )
    {
      printf("Error!!\n");
      ExitProcess(0);
    }
    printf("bye ~~\n");
```


自己看了好久也没有进展，看了几位大佬的wp；

```
if ( strlen((const char *)&pbData) != 6 )
  {
    printf("Must be 6 characters!\n");
    ExitProcess(0);
  }
  v5 = unknown_libname_1((char *)&pbData);
  if ( v5 < 100000 )
```

从这里可以看出我们的密码是六位的，点击unknown_libname_1函数，得到的是下图的函数

```
__int32 __cdecl unknown_libname_1(char *a1)
{
  return atol(a1);
}
```

atol函数：用来将字符串转换成长整型数，并且转化后的整型要大于100000；

```
strcat((char *)&pbData, "@DBApp");
v0 = strlen((const char *)&pbData);
```

这个代码，第一行将“@DBApp”连接到密码的后面，第二行求出连接之后的密码为12；

```
sub_40100A(&pbData, v0, &String1)
```

sub_40100A是一个加密函数

```
memset(&v4, 0xCCu, 0x68u);
  if ( CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
  {
    if ( CryptCreateHash(phProv, 0x8004u, 0, 0, &phHash) )
    {
      if ( CryptHashData(phHash, pbData, dwDataLen, 0) )
      {
        CryptGetHashParam(phHash, 2u, v7, &pdwDataLen, 0);
        *lpString1 = 0;
        for ( i = 0; i < pdwDataLen; ++i )
        {
          wsprintfA(&String2, "%02X", v7[i]);
          lstrcatA(lpString1, &String2);
        }
        CryptDestroyHash(phHash);
        CryptReleaseContext(phProv, 0);
        result = 1;
      }
      else
      {
        CryptDestroyHash(phHash);
        CryptReleaseContext(phProv, 0);
        result = 0;
      }
    }
    else
    {
      CryptReleaseContext(phProv, 0);
      result = 0;
    }
  }
  else
  {
    result = 0;
  }
  return result;
}
```

这加密函数看着就头疼；

```
CryptCreateHash(phProv, 0x8004u, 0, 0, &phHash)
```

搜索过后发现这个是哈希函数。

代码中的**0x8004**是标识符，并且是SHA1的算法

CALG_SHA1	0x00008004	Same as **CALG_SHA**. This algorithm is supported by the Microsoft Base Cryptographic Provider.

然后自己查了一下hashlib模块使用的方法
里面的16进制str类型的消息摘要，都是小写的英文字母
参考一下别人写的的代码
爆破代码如下：

```
import hashlib
string='@DBApp'
for i in range(100000,999999):
    flag=str(i)+string
    x = hashlib.sha1(flag.encode("utf8"))
    y = x.hexdigest()
    if "6e32d0943418c2c33385bc35a1470250dd8923a9" == y:
            print(flag)
            break
```

得到密码为：**123321@DBApp**
所以我们知道前六位密码是123321

一些代码的讲解：
str通过encode（）转换为bytes（二进制）
在python3中，encode（）和decode（）默认使用UTF-8
x.hexdigest()函数表示获得16进制str类型的消息摘要

第二部分和第一部分相类似，但是没有给我们范围，没办法爆破，只能自己慢慢看函数；
进入函数看到0x8003u，是MD5加密，但是解不出来；

```
memset(&v4, 0xCCu, 0x64u);
if ( CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
{
  if ( CryptCreateHash(phProv, 0x8003u, 0, 0, &phHash) )
  {
    if ( CryptHashData(phHash, pbData, dwDataLen, 0) )
    {
      CryptGetHashParam(phHash, 2u, v7, &pdwDataLen, 0);
      *lpString1 = 0;
      for ( i = 0; i < pdwDataLen; ++i )
      {
        wsprintfA(&String2, "%02X", v7[i]);
        lstrcatA(lpString1, &String2);
      }
      CryptDestroyHash(phHash);
      CryptReleaseContext(phProv, 0);
      result = 1;
    }
    else
    {
      CryptDestroyHash(phHash);
      CryptReleaseContext(phProv, 0);
      result = 0;
    }
  }
```

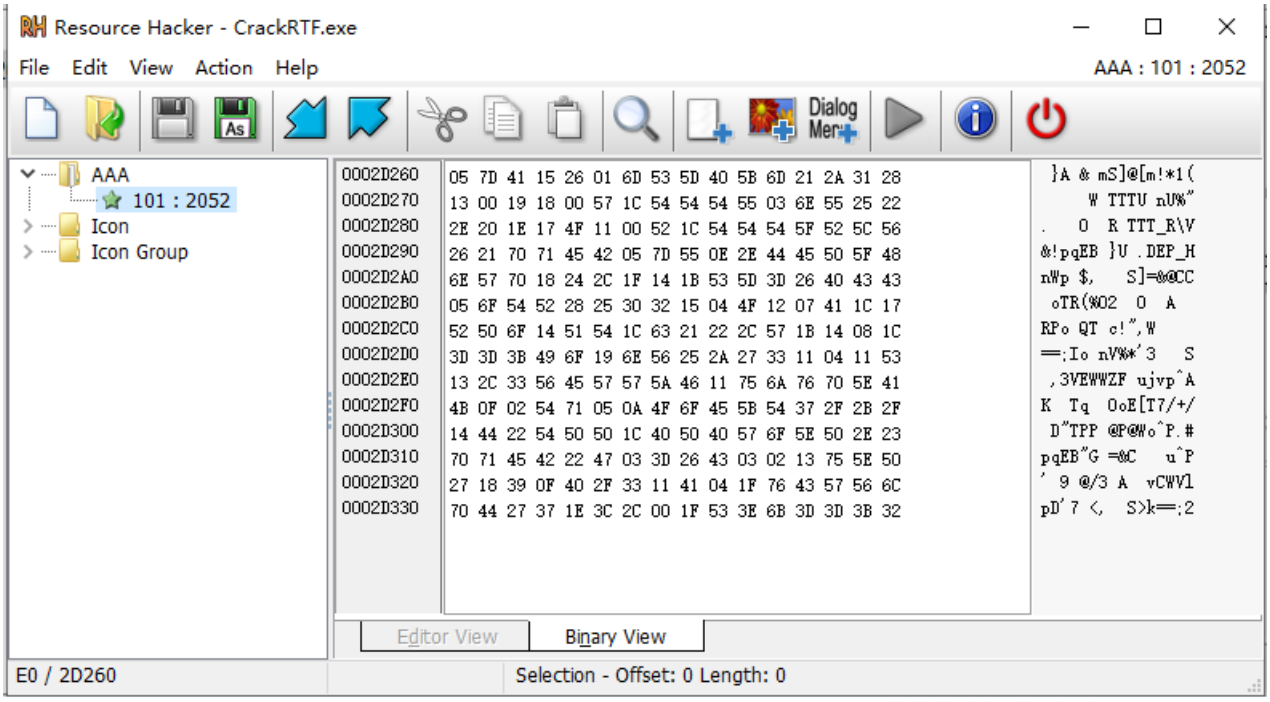

再进入加密函数看一下

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


就没有正常一点的函数吗？？？

这一段代码的含义就是，从AAA文件中查找字符，然后如果没有找到就返回，找到了的话就计算出资源的大小，把资源第一个字符出的指针传给lpBuffer

这里我们可以用一个叫做ResourceHacker的工具来查看资源



但是还是看不懂；

往里面看看还有异或的函数：

```
unsigned int __cdecl sub_401420(LPCSTR lpString, int a2, int a3)
{
  unsigned int result; // eax@2
  char v4; // [sp+Ch] [bp-4Ch]@1
  unsigned int i; // [sp+4Ch] [bp-Ch]@1
  LPCSTR v6; // [sp+50h] [bp-8h]@1
  int v7; // [sp+54h] [bp-4h]@1

  memset(&v4, 0xCCu, 0x4Cu);
  v7 = lstrlenA(lpString);
  v6 = lpString;
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( i >= a3 )
      break;
    *(_BYTE *)(i + a2) ^= v6[i % v7];
  }
  return result;
}
```


a2就是AAA中得到的首部指针，v5是字符串的长度，也就是密码的长度。
整理一下，在进行异或完之后会生成一个RTF文件；
这个时候我们不妨打开一个RTF文件来查看它的头部

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0123456789ABCDEF
0000h: 7B 5C 72 74 66 31 5C 61 6E 73 69 5C 61 6E 73 69   {\rtf1\ansi\ansi
```

得到前六位是{\rtf1
再看资源的前六位是

```
05 7D 41 15 26 01
```

资源的每一位和密码的每一位循环异或
异或结束之后，生成一个rtf文件
接下来这一步是我觉得很骚气 巧妙的一步
思考一下
我们的密码一共是6+12=18位
我们现在想要的是前六位的密码，循环异或的话，那么也就是说，资源的前六位与密码的前六位异或的结果就是rtf文件的前六位
这里的思想是借鉴的别人的

下面的脚本是借鉴别人的；

```
rtf = '{\\rtf1' \\需要注意，\r需要转义，变成\\r
A = [0x05, 0x7D, 0x41, 0x15, 0x26, 0x01]
password=''
for i in range(len(rtf)):
    x = ord(rtf[i]) ^ A[i]
    password+=chr(x)
print(password)
```

得到的结果为~!3a@0
第二段的密码为~!3a@0123321@DBApp
两端密码输入完后，就会在程序所在文件夹中生成一个带有flag的rtf文件，打开就能得到flag
flag为Flag{N0_M0re_Free_Bugs}

flag{N0_M0re_Free_Bugs}