# REVERSE-PRACTICE-BUUCTF-23

原创

P1umH0 于 2021-02-27 00:14:28 发布 156 收藏
分类专栏： Reverse-BUUCTF
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/weixin_45582916/article/details/114155927
版权

Reverse-BUUCTF 专栏收录该内容
32 篇文章 3 订阅
订阅专栏

### REVERSE-PRACTICE-BUUCTF-23


- [2019红帽杯]Snake
- [BSidesSF2019]blink
- [De1CTF2019]Re_Sign
- [ACTF新生赛2020]Splendid_MineCraft


## [2019红帽杯]Snake

unity游戏，dnSpy打开Snake\Snake_Data\Managed\Assembly-CSharp.dll
发现要载入Interface这个dll

```
Interface
1  using System;
2  using System.Runtime.InteropServices;
3
4  // Token: 0x02000004 RID: 4
5  public class Interface
6  {
7      // Token: 0x0600000F RID: 15
8      [DllImport("Interface", CallingConvention = CallingConvention.Cdecl)]
9      public static extern int Transform(int x, int y);
10
11     // Token: 0x06000010 RID: 16
12     [DllImport("Interface", CallingConvention = CallingConvention.Cdecl)]
13     public static extern int Transform3D(int x, int y, int z);
14
15     // Token: 0x06000011 RID: 17
16     [DllImport("Interface", CallingConvention = CallingConvention.Cdecl)]
17     public static extern int GameObject(int x, int y);
18
19     // Token: 0x06000012 RID: 18
20     [DllImport("Interface", CallingConvention = CallingConvention.Cdecl)]
21     public static extern int GameObject3D(int x, int y, int z);
22
23     // Token: 0x06000013 RID: 19
24     [DllImport("Interface", CallingConvention = CallingConvention.Cdecl)]
```

https://blog.csdn.net/weixin_45582916

ida打开Snake\Snake_Data\Plugins\Interface.dll
交叉引用字符串"You win! flag is "来到GameObject函数
分析GameObject函数，该函数只有一个参数a1，当a1∈[0,99]时，会执行if语句包络的代码，其中就有输出flag的语句

```
1 signed __int64 __fastcall GameObject(int a1)
2 {
3   char v1; // di
4   __int64 *v2; // rbx
5   __int64 *v3; // rax
6   int v4; // er8
7   int v5; // er9
```

```
80   v50 = 0;
81   v1 = 0;
82   if ( a1 >= 0 )
83   {
84     if ( (unsigned int)(a1 - 2) <= 97 )
85     {
86       LOBYTE(Memory) = 0;
87       _mm_storeu_si128((__m128i *)&v48, _mm_load_si128((c
88       sub_180006D10(
89         &Memory,
```

https://blog.csdn.net/weixin_45582916

于是考虑爆破，写C代码，载入Interface.dll，调用GameObject函数，传入从0到99之间的整数，爆破成功即可得到flag，从flag可以知道是RSA加密算法

```cpp
#include<iostream>
#include<Windows.h>
#include"ida_defs.h"
//函数指针
typedef signed __int64(*Dllfunc)(int);
using namespace std;
int main()
{
 Dllfunc GameObject;//GameObject是dll中想要调用的函数名称
 HINSTANCE hdll = NULL;
 hdll = LoadLibrary("Interface.dll");//加载dll
 if (hdll == NULL)
 {
  cout << "加载动态库失败\n";
 }
 else
 {
  GameObject = (Dllfunc)GetProcAddress(hdll, "GameObject");//到dll中定位函数
  if (GameObject == NULL)
  {
   cout << "加载动态库方法失败\n";
  }
  else
  {
   for (int i = 0; i <= 99; i++)
   {
    signed __int64 res = GameObject(i);
   }
  }
 }
 FreeLibrary(hdll);//释放dll
 return 0;
}
/*
You win! flag is
flag{Ch4rp_W1th_R$@}
*/
```

## [BSidesSF2019]blink

apk文件，jadx-gui打开，com.example.blink.MainActivity什么都没有

在com.example.blink.r2d2中，将一段字符串解base64，分割成byte数组，填入一个图像文件中

```
package com.example.blink;

import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Base64;
import android.widget.ImageView;

16 public class r2d2 extends AppCompatActivity {
       /* access modifiers changed from: protected */
17     public void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView((int) R.layout.activity_r2d2);
20         byte[] imageBytes = Base64.decode(" data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAMCAgICAgMCAgIDAwMDBAYEBAQEBAgGBgUGCQg
21         ((ImageView) findViewById(R.id.imageView)).setImageBitmap(BitmapFactory.decodeByteArray(imageBytes, 0, imageBytes.length));
24     }
26 }
```

https://blog.csdn.net/weixin_45582916

apk在模拟器中双击运行会直接退出

按照jeb调试apk smali的方法，在启动apk activity时，将MainActivity改成r2d2，即adb shell am start -D -n com.example.blink/com.example.blink.r2d2，jeb附加后运行即可得到flag

CTF{PUCKMAN}

## [De1CTF2019]Re_Sign

exe程序，运行后输入，有upx壳，ESP定律脱壳后ida分析

交叉引用字符串"Success"来到sub_401000函数，程序脱壳后不能运行，ida和x32dbg配合分析

调试发现，sub_402BA0函数读取输入，sub_402E40函数和sub_sub_402F80函数都是简单的赋值，调试的输入为"abcdefg"，经过sub_401233函数变换为"GD9MH6SeHd=="，而正常的base64编码结果为"YWJjZGVmZw=="，猜测sub_401233函数是变表base64，重要的是找到变表

```
12  v8 = 0;
13  v7 = 0;
14  input = sub_402BA0(1, 0, 0, 0);              // 读取输入
15  v8 = input;
16  v0 = (char *)input;                          // input赋给v0
17  if ( !input )
18    v0 = &byte_41E300;
19  HIDWORD(v6) = sub_402E40(1, v0, 0, -2147483644);// v0赋给v6
20  LODWORD(v6) = sub_401233((char *)&v6 + 4);   // 对v6变换，"abcdefg"->"GD9MH6SeHd=="，变表base64
21  if ( HIDWORD(v6) )
22    sub_402258(SHIDWORD(v6));
23  v1 = (char *)LODWORD(v6);                   // v6赋给v1
24  if ( !LODWORD(v6) )
25    v1 = &byte_41E300;
26  v5 = sub_402F80(1, *(double *)&v1, -2147483644);// v1赋给v5
27  if ( LODWORD(v6) )
28    sub_402258(SLODWORD(v6));
29  if ( v7 )
30    sub_402258(v7);
31  v7 = v5;                                     // v5赋给v7
```

```
32  if ( sub_401F0A(&v7) )                    // 对v7进行验证，返回1验证成功
33    sub_403220(2, 0, 0, 0, (unsigned int)"Success");
34  else
35    sub_403220(2, 0, 0, 0, (unsigned int)"Fail");
36  v2 = sub_402BA0(1, 0, 0, 0);
37  if ( v2 )
38    sub_402258(v2);
39  if ( v8 )
40    sub_402258(v8);
41  if ( v7 )
42    sub_402258(v7);
43  return 0;
44}
```

https://blog.csdn.net/weixin_45582916

F7单步步入sub_401233函数，在这个地方找到变表

```
00401720   E8 450B0000     call re sign.40226A
00401725   83C4 04         add esp,4
00401728   03D8            add ebx,eax
           断点未设置       8B5D E0         mov ebx,dword ptr ss:[ebp-20]
                           E8 DEFAFFFF     call re sign.401213
EIP  00401735   53              push ebx
00401736   51              push ecx
00401737   8B5D E4         mov ebx,dword ptr ss:[ebp-1C]
0040173A   E8 D4FAFFFF     call re sign.401213
0040173F   B8 00000000     mov eax,0
00401744   3BC1            cmp eax,ecx
00401746   7C 0D           jl re sign.401755
00401748   68 01000000     push 1
0040174D   E8 180B0000     call re sign.40226A
00401752   83C4 04         add esp,4
00401755   03D8            add ebx,eax
00401757   895D D4         mov dword ptr ss:[ebp-2C],ebx
0040175A   68 01030080     push 80000301
0040175F   6A 00           push 0
00401761   68 02000000     push 2
00401766   8B5D D4         mov ebx,dword ptr ss:[ebp-2C]
00401769   8A03            mov al,byte ptr ds:[ebx]
0040176B   25 FF000000     and eax,FF
00401770   68 01030080     push 80000301
00401775   6A 00           push 0
00401777   50              push eax
00401778   68 02000000     push 2

ebx=007CAC48

UPX0:00401735 re sign.exe:$1735 #0
```

内存 1 | 内存 2 | 内存 3 | 内存 4 | 内存 5 | 监视 1 | 局部变量 | 结构体

```
地址      十六进制                                          ASCII
007CAC48  30 31 32 33 34 35 36 37 38 39 51 57 45 52 54 5  0123456789QWERTY
007CAC58  55 49 4F 50 41 53 44 46 47 48 4A 4B 4C 5A 58 4  UIOPASDFGHJKLZXC
007CAC68  56 42 4E 4D 71 77 65 72 74 79 75 69 6F 70 61 7  VBNMqwertyuiopas
007CAC78  64 66 67 68 6A 6B 6C 7A 78 63 76 62 6E 6D 2B 2  dfghjklzxcvbnm+/
007CAC88  AB AB AB AB AB AB AB AB 00 00 00 00 00 00 00 0  ««««««««........
```

https://blog.csdn.net/weixin_45582916

验证一下，sub_401233函数的变表就是找到的这个表

加密解密小玩具 Ver0.2 by Lucky_789 【bbs.chinapyg.com】

RSA | AES | Base64 | Base32 | SHA | MD5 | RC4

字串(T) 0123456789QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm+/ 恢复标准串

明文(M) abcdefg

去空格

GD9MH6SeHd==

分析sub_401F0A函数，输入经变表base64编码后，在while循环体中，一个字节一个字节拷贝到v30，v30赋给v37，v37传入sub_2160函数，返回值与v25(ebx)指向的int数据比较

```
 92    if ( v18 >= v27 )
 93      v18 = sub_40226A(1, v32, v33);
 94    v30 = sub_4034E0(1, *(_BYTE *)(v18 + v19));// 输入一个字节一个字节拷贝到v30
 95    if ( v37 )
 96      sub_402258(v37);
 97    v37 = v30;                              // v30赋给v37
 98    v22 = (int)v39;
 99    sub_401213(v21, v20);
100    v31 = v22;
101    v28 = v23;
102    v24 = v38 - 1;
103    if ( v38 - 1 < 0 )
104      v24 = sub_40226A(4, v23, v22);
105    v25 = v31;
106    if ( v24 >= v28 )
107      v24 = sub_40226A(1, v32, v33);
108    if ( *(_DWORD *)(4 * v24 + v25) != sub_402160((char **)&v37) )// v37作为参数传入sub_402160，返回的是v37在常规base64表中的位置
109                                            // v25是要去比较的值(ebx指向的int数据)，可知也是某字符在常规base64表中的位置
110      break;
111    v13 = v32;
112    v15 = v35;
113    v14 = v33 + 1;
```

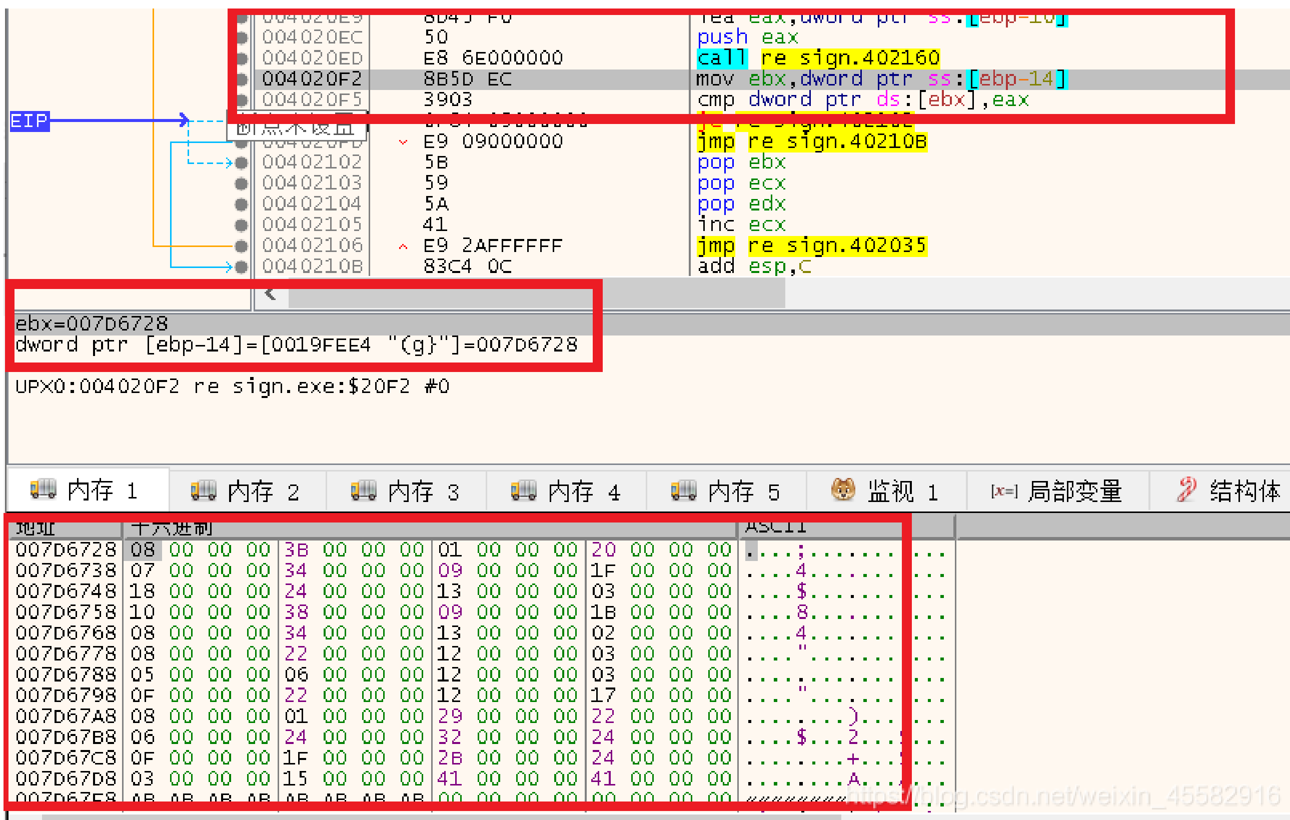
https://blog.csdn.net/weixin_45582916

sub_402160函数中使用到了常规base64表，调试输入为"abcdefg"，变表base64后为"GD9MH6SeHd=="，第一个字节"G"传入sub_402160函数，返回值为7，也就是"G"在常规base64表中的位置(下标从1开始)，于是可知要去比较的数据也是某字符在常规base64表中的位置(下标从1开始)

```
 1 int __stdcall sub_402160(char **a1)
 2 {
 3   char *v1; // eax
 4
 5   v1 = *a1;
 6   if ( !*a1 )
 7     v1 = &byte_41E300;
 8   return sub_403500(
 9            4,
10            "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=",
11            0,
12            -2147483644,
13            v1,
14            0,
15            -2147483644,
16            0,
17            0,
18            0,
19            0);
20 }
```

https://blog.csdn.net/weixin_45582916

x32dbg调试得到要去比较的数据




写逆运算脚本即可得到flag

```python
#coding:utf-8
import base64
table="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/="
table_changed="0123456789QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm+/="
data=[0x08,0x3b,0x01,0x20,0x07,0x34,0x09,0x1f,0x18,0x24,0x13,0x03,0x10,0x38,0x09,0x1b,
      0x08,0x34,0x13,0x02,0x08,0x22,0x12,0x03,0x05,0x06,0x12,0x03,0x0f,0x22,0x12,0x17,
      0x08,0x01,0x29,0x22,0x06,0x24,0x32,0x24,0x0f,0x1f,0x2b,0x24,0x03,0x15,0x41,0x41]
s=""
ss=""
for i in data:
    s+=table[i-1]       #s赋完值其实是输入经变表base64后的结果
for c in s:
    ss+=table[table_changed.find(c)]     #由变表和常规表的映射得到常规base64编码的结果
print(base64.b64decode(ss))
#de1ctf{E_L4nguag3_1s_K3KeK3_N4Ji4}
```

## [ACTF新生赛2020]Splendid_MineCraft

exe程序，运行后输入，无壳，ida分析

交叉引用字符串"Welcome to ACTF_Splendid_MineCraft!“来到sub_401080函数

读取输入，验证输入的长度是否为26，strtok函数用字符'_'将输入分割成三个部分，于是可知输入的格式为"ACTF{xxxxxx_yyyyyy_xxxxxx}"，v12-v13(v14-v15)为{}中的第一部分"xxxxxx"，v8-v9为{}中的第二部分"yyyyyy"，v10-v11为{}中的第三部分"xxxxxx"

```c
18
19  sub_401020((const char *)&unk_404118, (unsigned int)"Welcome to ACTF_Splendid_MineCraft!");
20  sub_401050((const char *)&unk_40411C, (unsigned int)&input);
21  if ( strlen(&input) == 26 )                    // 输入的长度为26
22  {
23    if ( !strncmp(&input, "ACTF{", 5u) && v7 == 125 )// 输入由ACTF{}包络，{}内的字符串长度为20
24    {
25      v7 = 0;
26      v0 = strtok(&input, "_");                  // 用字符'_'分隔input，返回的是input从开始到到一个'_'之间的字符串
27      v12 = *(_DWORD *)(v0 + 5);                 // 跳过前5个字符"ACTF{"，读6个字节，也就是{}内的前6个字符
```

```
28      v13 = *(_WORD *)(v0 + 9);
29      v14 = *(_DWORD *)(v0 + 5);
30      v15 = *(_WORD *)(v0 + 9);
31      v1 = strtok(0, "_");                        // 返回的是input中从第1个'_'到第2个'_'之间的字符串
32      v8 = *(_DWORD *)v1;                         // 在v1中读6个字符
33      v9 = *((_WORD *)v1 + 2);
34      v2 = strtok(0, "_");                        // 返回的是第2个'_'到input结尾之间的字符串
35      *(_DWORD *)v10 = *(_DWORD *)v2;             // 在v2中读6个字节
36      v11 = *((_WORD *)v2 + 2);
37      dword_403354 = (int)dword_4051D8;
38      if ( ((int (__cdecl *)(int *))dword_4051D8[0])(&v12) )// SMC后验证v12的6个字符
39      {
40        v5 = SBYTE2(v14) ^ SHIBYTE(v15) ^ (char)v14 ^ SHIBYTE(v14) ^ SBYTE1(v14) ^ (char)v15;
41        for ( i = 256; i < 496; ++i )             // SMC
42          byte_405018[i] ^= v5;                   // v5与v12的6个字节相关
43        JUMPOUT(__CS__, &byte_405018[256]);
44      }
45      sub_401020("Wrong\n");
46    }
47    else
48    {
49      sub_401020("Wrong\n");
50    }
51  }
```

https://blog.csdn.net/weixin_45582916

v12-v13，即{}中的第一部分要进入dword_4051D8，经SMC后，验证{}中的第一部分

来到data段dword_4051D8，按d转成字节形式的数据，按c转成代码，可看到SMC的代码

```
.data:004051D4                 align 8
.data:004051D8
.data:004051D8 loc_4051D8:                             ; DATA XREF: sub_401080+15A↑o
.data:004051D8                 call    $+5
.data:004051DD                 pop     esi
.data:004051DE                 push    edi
.data:004051DF                 xor     edi, edi
.data:004051E1
.data:004051E1 loc_4051E1:                             ; CODE XREF: .data:004051F5↓j
.data:004051E1                 cmp     edi, 151h
.data:004051E7                 jg      short loc_4051FB
.data:004051E9                 mov     bl, [esi+edi+1Fh]
.data:004051ED                 xor     bl, 72h
.data:004051F0                 mov     [esi+edi+1Fh], bl      SMC
.data:004051F4                 inc     edi
.data:004051F5                 jmp     short loc_4051E1
.data:004051F5 ; ---------------------------------------------------------------
.data:004051F7                 db  48h ; H
.data:004051F8                 db  65h ; e
.data:004051F9                 db  79h ; y
.data:004051FA                 db  21h ; !
.data:004051FB ; ---------------------------------------------------------------
.data:004051FB
.data:004051FB loc_4051FB:                             ; CODE XREF: .data:004051E7↑j
.data:004051FB                 pop     edi
```

https://blog.csdn.net/weixin_45582916

调试，SMC执行完成后是这个样子

```
.data:00DC51FC loc_DC51FC:                             ; DATA X
.data:00DC51FC                                         ; sub_DC
.data:00DC51FC push    ebp
.data:00DC51FD mov     ebp, esp
.data:00DC51FF sub     esp, 2Ch
.data:00DC5202 mov     dword ptr [ebp-8], 0
.data:00DC5209 mov     eax, 1
.data:00DC520E imul    ecx, eax, 0
.data:00DC5211 mov     byte ptr [ebp+ecx-18h], '3'
.data:00DC5216 mov     edx, 1
.data:00DC521B shl     edx, 0
```

```
.data:00DC521E mov     byte ptr [ebp+edx-18h], '@'
.data:00DC5223 mov     eax, 1
.data:00DC5228 shl     eax, 1
.data:00DC522A mov     byte ptr [ebp+eax-18h], '1'
.data:00DC522F mov     ecx, 1
.data:00DC5234 imul    edx, ecx, 3
.data:00DC5237 mov     byte ptr [ebp+edx-18h], 'b'
.data:00DC523C mov     eax, 1
.data:00DC5241 shl     eax, 2
.data:00DC5244 mov     byte ptr [ebp+eax-18h], ';'
.data:00DC5249 mov     ecx, 1
.data:00DC524E imul    edx, ecx, 5
.data:00DC5251 mov     byte ptr [ebp+edx-18h], 'b'
000023FC 00DC51FC: sub_DC51D8:loc_DC51FC (Synchronized with EIP)
```

https://blog.csdn.net/weixin_45582916

往下走，来到验证{}内的第一部分的代码，“3@1b;b"和"elcome"两个字符串异或，再加0x23，结果与传入的{}内的第一部分比较，可知正确的第一部分的6个字符为"yOu0y*”

```
.data:00DC52EF loc_DC52EF:                           ; CODE XREF: .data:00DC52E4↑j
.data:00DC52EF cmp     dword ptr [ebp-4], 6
.data:00DC52F3 jge     short loc_DC5331
.data:00DC52F5 mov     ecx, [ebp-4]
.data:00DC52F8 movsx   edx, byte ptr [ebp+ecx-18h]   ; "3@1b;b"
.data:00DC52FD mov     eax, [ebp-4]
.data:00DC5300 movsx   ecx, byte ptr [ebp+eax-2Bh]   ; "elcome"
.data:00DC5305 xor     edx, ecx                      ; 两个字符串异或
.data:00DC5307 add     edx, 23h                      ; 异或的结果加0x23，存到edx
.data:00DC530A mov     eax, [ebp-4]
.data:00DC530D mov     [ebp+eax-20h], dl
.data:00DC5311 mov     ecx, [ebp-4]
.data:00DC5314 movsx   edx, byte ptr [ebp+ecx-20h]
.data:00DC5319 mov     eax, [ebp+8]
.data:00DC531C add     eax, [ebp-4]
.data:00DC531F movsx   ecx, byte ptr [eax]           ; ecx是传入的{}中的第一部分的6个字符
.data:00DC5322 cmp     edx, ecx                      ; edx与ecx比较
.data:00DC5324 jnz     short loc_DC532F
.data:00DC5326 mov     edx, [ebp-8]
.data:00DC5329 add     edx, 1
.data:00DC532C mov     [ebp-8], edx
```

https://blog.csdn.net/weixin_45582916

由于v5与{}内的第一部分的6个字符相关，更新输入后调试，可知v5==0x20
执行完SMC后，来到验证{}内第二部分的代码，eax指向的是数组byte_DC5018，于是这段代码的验证思路就是，cl=byte_DC5018[input[i]^(i+0x83)]，写脚本可知第二部分为"knowo3"

```
.data:00DC5158 cmp     edi, 6                   ; edi是下标
.data:00DC515B jge     short loc_DC5198
.data:00DC515D xor     ecx, ecx
.data:00DC515F mov     cl, [esi+edi]            ; {}中的第二部分，放入cl
.data:00DC5162 and     cl, 0FFh                 ; cl&0xff
.data:00DC5165 sub     eax, 100h
.data:00DC516A xor     ebx, ebx
.data:00DC516C mov     bl, cl                   ; cl放入bl
.data:00DC516E mov     ecx, edi
.data:00DC5170 add     ecx, 83h
.data:00DC5176 xor     ebx, ecx                 ; bl^(edi+0x83)
.data:00DC5178 mov     bl, [eax+ebx]            ; 从一个数组中以ebx为下标取值到bl
.data:00DC517B jmp     short loc_DC5185         ; 这里的cl是要比较的值
.data:00DC517B ; ---------------------------------------------------------------------
.data:00DC517D db    0
.data:00DC517E db  30h ; 0
.data:00DC517F db    4
```

```
.data:00DC5180 db    4
.data:00DC5181 db    3
.data:00DC5182 db  30h ; 0
.data:00DC5183 db  63h ; c
.data:00DC5184 db  90h
.data:00DC5185 ; ---------------------------------------------------------------------------
.data:00DC5185
.data:00DC5185 loc_DC5185:                             ; CODE XREF: .data:00DC517B↑j
.data:00DC5185 mov     cl, [eax+edi+166h]              ; 这里的cl是要比较的值
.data:00DC518C cmp     bl, cl
.data:00DC518E jnz     short loc_DC51A4
.data:00DC5190 inc     edi
```

https://blog.csdn.net/weixin_45582916

往下走（期间要修改几次ZF的值），来到验证第三部分的代码，这里直接比较第三部分和已知字符串"5mcsM<"

```
.text:00DC12C9 loc_DC12C9:                             ; CODE XREF: sub_DC1080+238↑j
.text:00DC12C9 push    6                               ; MaxCount
.text:00DC12CB push    offset a5mcsm                   ; "5mcsM<"
.text:00DC12D0 lea     ecx, [ebp+Str1]                 ; Str1是第三部分
.text:00DC12D3 push    ecx                             ; Str1
.text:00DC12D4 call    ds:strncmp                      ; 直接比较第三部分和"5mcsM<"
.text:00DC12DA add     esp, 0Ch
.text:00DC12DD test    eax, eax
.text:00DC12DF jz      short loc_DC12F0
.text:00DC12E1 push    offset aWrong                   ; "Wrong\n"
.text:00DC12E6 call    sub_DC1020
.text:00DC12EB add     esp, 4
.text:00DC12EE jmp     short loc_DC1305
.text:00DC12F0 ; ---------------------------------------------------------------------------
```

https://blog.csdn.net/weixin_45582916

于是三个部分的脚本放在一起就是，输入flag，验证成功

```python
#coding:utf-8
flag="ACTF{"
#第一部分
s1="3@1b;b"
s2="elcome"
data=[]
for i in range(len(s1)):
    data.append(ord(s1[i])^ord(s2[i]))
for i in range(len(data)):
    data[i]+=0x23
flag+=''.join(chr(i) for i in data)
flag+='_'
#第二部分
data=[0xF6, 0xA3, 0x5B, 0x9D, 0xE0, 0x95, 0x98, 0x68, 0x8C, 0x65,
  0xBB, 0x76, 0x89, 0xD4, 0x09, 0xFD, 0xF3, 0x5C, 0x3C, 0x4C,
  0x36, 0x8E, 0x4D, 0xC4, 0x80, 0x44, 0xD6, 0xA9, 0x01, 0x32,
  0x77, 0x29, 0x90, 0xBC, 0xC0, 0xA8, 0xD8, 0xF9, 0xE1, 0x1D,
  0xE4, 0x67, 0x7D, 0x2A, 0x2C, 0x59, 0x9E, 0x3D, 0x7A, 0x34,
  0x11, 0x43, 0x74, 0xD1, 0x62, 0x60, 0x02, 0x4B, 0xAE, 0x99,
  0x57, 0xC6, 0x73, 0xB0, 0x33, 0x18, 0x2B, 0xFE, 0xB9, 0x85,
  0xB6, 0xD9, 0xDE, 0x7B, 0xCF, 0x4F, 0xB3, 0xD5, 0x08, 0x7C,
  0x0A, 0x71, 0x12, 0x06, 0x37, 0xFF, 0x7F, 0xB7, 0x46, 0x42,
  0x25, 0xC9, 0xD0, 0x50, 0x52, 0xCE, 0xBD, 0x6C, 0xE5, 0x6F,
  0xA5, 0x15, 0xED, 0x64, 0xF0, 0x23, 0x35, 0xE7, 0x0C, 0x61,
  0xA4, 0xD7, 0x51, 0x75, 0x9A, 0xF2, 0x1E, 0xEB, 0x58, 0xF1,
  0x94, 0xC3, 0x2F, 0x56, 0xF7, 0xE6, 0x86, 0x47, 0xFB, 0x83,
  0x5E, 0xCC, 0x21, 0x4A, 0x24, 0x07, 0x1C, 0x8A, 0x5A, 0x17,
  0x1B, 0xDA, 0xEC, 0x38, 0x0E, 0x7E, 0xB4, 0x48, 0x88, 0xF4,
  0xB8, 0x27, 0x91, 0x00, 0x13, 0x97, 0xBE, 0x53, 0xC2, 0xE8,
  0xEA, 0x1A, 0xE9, 0x2D, 0x14, 0x0B, 0xBF, 0xB5, 0x40, 0x79,
  0xD2, 0x3E, 0x19, 0x5D, 0xF8, 0x69, 0x39, 0x5F, 0xDB, 0xFA,
  0xB2, 0x8B, 0x6E, 0xA2, 0xDF, 0x16, 0xE2, 0x63, 0xB1, 0x20,
  0xCB, 0xBA, 0xEE, 0x8D, 0xAA, 0xC8, 0xC7, 0xC5, 0x05, 0x66,
  0x6D, 0x3A, 0x45, 0x72, 0x0D, 0xCA, 0x84, 0x4E, 0xF5, 0x31,
  0x6B, 0x92, 0xDC, 0xDD, 0x9C, 0x3F, 0x55, 0x96, 0xA1, 0x9F,
  0xCD, 0x9B, 0xE3, 0xA0, 0xA7, 0xFC, 0xC1, 0x78, 0x10, 0x2E,
  0x82, 0x8F, 0x30, 0x54, 0x04, 0xAC, 0x41, 0x93, 0xD3, 0x3B,
  0xEF, 0x03, 0x81, 0x70, 0xA6, 0x1F, 0x22, 0x26, 0x28, 0x6A,
  0xAB, 0x87, 0xAD, 0x49, 0x0F, 0xAF]
res=[0x30,0x4,0x4,0x3,0x30,0x63]
for i in range(len(res)):
    for j in range(len(data)):
        if data[j]==res[i]:
            flag+=chr(j^(i+0x83))
flag+='_'
#第三部分
flag+="5mcsM<"
flag+='}'
print(flag)
#ACTF{yOu0y*_knowo3_5mcsM<}
```