buu逆向刷题（四）

原创

北风~ 于 2020-10-04 10:14:49 发布

分类专栏： 逆向与保护

本文链接：https://blog.csdn.net/weixin_45055269/article/details/108894222

1.[V&N2020 公开赛]CSRe

CSRe，打开报错，DIE查了一下，啥也没有，只知道.net程序，.net程序有de4dot（脱壳）和dnSpy（反编译），exeinfope查壳，由此可见.net程序在exeinfope查壳，发现Eazfuscator.NET的混淆，去混淆后在dnSpy里打开，在每个类里搜索main，终于在class3里找到main函数，

```
35    // Token: 0x0600000F RID: 15 RVA: 0x00002374 File Offset: 0x00000574
36    private static void Main(string[] args)
37    {
38        if (!Class1.smethod_1())
39        {
40            return;
41        }
42        bool flag = true;
43        Class3 @class = new Class3();
44        string str = Console.ReadLine();
45        if (Class3.smethod_0("3" + str + "9") != "B498BFA2498E21325D1178417BEA459EB2CD28F8")
46        {
47            flag = false;
48        }
49        string text = Console.ReadLine();
50        string string_ = Class3.smethod_0("re" + text);
51        string text2 = @class.method_0(string_, "63143B6F8007B98C53CA2149822777B3566F9241");
52        for (int i = 0; i < text2.Length; i++)
53        {
54            if (text2[i] != '0')
55            {
56                flag = false;
57            }
58        }
59        if (flag)
60        {
61            Console.WriteLine("flag{" + str + text + "}");
62        }
63    }
64 }
```


smethod_0加密两段字符串，最终得到了加密后的结果，第二段异或为0，表示字符串相等，smethod_0函数SHA1，找个解密网站即可得到原文。

2.[ACTF新生赛2020]usualCrypt

主函数打开，sub401080加密函数，下面是一个比较。

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

```
4  int result; // eax
5  int v5; // [esp+8h] [ebp-74h]
6  int v6; // [esp+Ch] [ebp-70h]
7  int v7; // [esp+10h] [ebp-6Ch]
8  __int16 v8; // [esp+14h] [ebp-68h]
9  char v9; // [esp+16h] [ebp-66h]
10 char v10; // [esp+18h] [ebp-64h]
11
12 sub_403CF8(&unk_40E140);
13 scanf(aS, &v10);
14 v5 = 0;
15 v6 = 0;
16 v7 = 0;
17 v8 = 0;
18 v9 = 0;
19 sub_401080((int)&v10, strlen(&v10), (int)&v5);
20 v3 = 0;
21 while ( *((_BYTE *)&v5 + v3) == byte_40E0E4[v3] )
22 {
23   if ( ++v3 > strlen((const char *)&v5) )
24     goto LABEL_6;
25 }
26 sub_403CF8(aError);
27LABEL_6:
28 if ( v3 - 1 == strlen(byte_40E0E4) )
29   result = sub_403CF8(aAreYouHappyYes);
30 else
31   result = sub_403CF8(aAreYouHappyNo);
32 return result;
33}
```


sub_401080仔细看分为三层：

sub_401000对base64变表处理

中间base64加密

sub_401030大小写互换

逻辑清晰，逆向写出脚本。

```
import base64

Str = list("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/")
model = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
for i in range(6,15):
    Str[i],Str[i+10] = Str[i+10],Str[i]
Str = ''.join(Str)
enc = "zMXHz3TIgnxLxJhFAdtZn2fFk3lYCrtPC2l9".swapcase()
dec = ""
for i in range(len(enc)):
    dec += model[Str.find(enc[i])]
print (dec)
print (Str)
print (base64.b64decode(dec))![在这里插入图片描述](https://img-blog.csdnimg.cn/202010011712288.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3dlaXhpbl80NTA1NTI2OQ==,size_16,color_FFFFFF,t_70#pic_center)
```

函数一大串极有可能是某种常见算法。

识别出是一种算法，看其中还包含那些其他函数。

base64的换表操作，在表中的位置不变。

base64换表操作的进一步说明：

dec += model[Str.find(enc[i])]

3.[GWCTF 2019]xxor

```
  v10 = 0LL;
  for ( i = 0; i <= 5; ++i )
  {
    printf("%s", "input: ", (unsigned int)i);
    __isoc99_scanf("%d", (char *)&v6 + 4 * i);
  }
  v11 = 0LL;
  v12 = 0LL;
  v13 = 0LL;
  v14 = 0LL;
  v15 = 0LL;
  for ( j = 0; j <= 4; j += 2 )
  {
    dword_601078 = *((_DWORD *)&v6 + j);
    dword_60107C = *((_DWORD *)&v6 + j + 1);
    sub_400686((unsigned int *)&dword_601078, &unk_601060);
    *((_DWORD *)&v11 + j) = dword_601078;
    *((_DWORD *)&v11 + j + 1) = dword_60107C;
  }
  if ( (unsigned int)sub_400770(&v11) != 1 )
  {
    puts("NO NO NO~ ");
    exit(0);
  }
  puts("Congratulation!\n");
  puts("You seccess half\n");
  puts("Do not forget to change input to hex and combine~\n");
  puts("ByeBye");
  return 0LL;
}
```


主函数一个加密函数。一个比较函数。

比较函数，Z3脚本爆破。

```
from z3 import *
s=Solver()
a1=[0 for i in range(6)]
for i in range(6):
 a1[i]=Int('a1['+str(i)+']')
s.add(a1[2]-a1[3]==0x84A236FF)
s.add(a1[3]+a1[4]==0xFA6CB703)
s.add(a1[2]-a1[4]==0x42D731A8)
s.add(a1[0]==0xDF48EF7E)
s.add(a1[5]==0x84F30420)
s.add(a1[1]==0x20CAACF4)
print(s.check())
answer=s.model()
print(answer)
#a1[0] = 3746099070
#a1[1] = 550153460
#a1[2] = 3774025685
#a1[3] = 1548802262
#a1[4] = 2652626477
#a1[5] = 2230518816
```

加密函数，异或部分看成整体。IDA脚本向C脚本转变。

```
#include<iostream>
using namespace std;
int main()
{
 __int64 result; // rax
 unsigned int v3; // [rsp+1Ch] [rbp-24h]
 unsigned int v4; // [rsp+20h] [rbp-20h]
 int v5; // [rsp+24h] [rbp-1Ch]
 unsigned int i; // [rsp+28h] [rbp-18h]

 __int64 a1[6] = { 3746099070 ,550153460 ,3774025685 ,1548802262 ,2652626477,2230518816 };
 char a2[4] = { 2,2,3,4 };
 for (char j = 0; j <= 4; j += 2)
 {
  v3 = a1[j];
  v4 = a1[j+1];
  v5 = 0x458BCD42 * 64;
  for (i = 0; i <= 63; ++i)
  {
   v4 -= (v3 + v5 + 20) ^ ((v3 << 6) + a2[2]) ^ ((v3 >> 9) + a2[3]) ^ 0x10;
   v3 -= (v4 + v5 + 11) ^ ((v4 << 6) + a2[0]) ^ ((v4 >> 9) + a2[1]) ^ 0x20;
   v5 -= 0x458BCD42;
  }

 a1[j] = v3;
 a1[j+1] = v4;
 }

 for (int i = 0; i <= 5; i++)
 {
  cout << hex << a1[i];
 }

}
//666c61677b72655f69735f6772656174217d
```

16进制到文本字符串

加密或解密字符串长度不可以超过10M

```
1  666c61677b72655f69735f6772656174217d
2
3
```

16进制转字符 | 字符转16进制 | 测试用例 | 清空结果 | 复


```
1  flag{re_is_great!}
```


C 16进制输出 cout << hex << a1[i];

4.[MRCTF2020]Transform

IDA虽然爆红，但整体逻辑还是清楚的，核心操作数组赋值加异或，和已知字符串比较。

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

```
  __int64 v3; // rdx
  __int64 v4; // rdx
  char v6[104]; // [rsp+20h] [rbp-70h]
  int j; // [rsp+88h] [rbp-8h]
  int i; // [rsp+8Ch] [rbp-4h]

  sub_402230(argc, argv, envp);
  sub_40E640(argc, (__int64)argv, v3, (__int64)"Give me your code:\n");
  sub_40E5F0(argc, (__int64)argv, (__int64)v6, (unsigned __int64)"%s");
  if ( strlen(*(const char **)&argc) != 33 )
  {
    sub_40E640(argc, (__int64)argv, v4, (__int64)"Wrong!\n");
    system(*(const char **)&argc);
    exit(argc);
  }
  for ( i = 0; i <= 32; ++i )
  {
    byte_414040[i] = v6[dword_40F040[i]];
    v4 = i;
    byte_414040[i] ^= LOBYTE(dword_40F040[i]);
  }
  for ( j = 0; j <= 32; ++j )
  {
    v4 = j;
    if ( byte_40F0E0[j] != byte_414040[j] )
    {
      sub_40E640(argc, (__int64)argv, j, (__int64)"Wrong!\n");
      system(*(const char **)&argc);
      exit(argc);
    }
```


000009B0 main:29 (4015B0)

逆向写出脚本。

```
r=[0x67, 0x79, 0x7B, 0x7F, 0x75, 0x2B, 0x3C, 0x52, 0x53, 0x79, 0x57, 0x5E, 0x5D, 0x42, 0x7B, 0x2D, 0x2A, 0x66, 0
x42, 0x7E,0x4C, 0x57, 0x79, 0x41, 0x6B, 0x7E, 0x65, 0x3C, 0x5C, 0x45, 0x6F, 0x62, 0x4D]
tmp1=[0x09, 0x0A, 0x0F, 0x17, 0x07, 0x18, 0x0C, 0x06, 0x01, 0x10, 0x03, 0x11, 0x20, 0x1D, 0x0B, 0x1E, 0x1B, 0x16
, 0x04, 0x0D,0x13, 0x14, 0x15, 0x02, 0x19, 0x05, 0x1F, 0x08, 0x12, 0x1A, 0x1C, 0x0E,0x0]
tmp2=[0 for i in range(33)]
flag=[0 for i in range(33)]
for i in range(33):
 tmp2[i]=tmp1[i]^r[i]
for i in range(33):
 flag[tmp1[i]]=tmp2[i]
print(''.join(map(chr,flag)))
#MRCTF{Tr4nsp0sltiON_Clph3r_1s_3z}
```

5.crackMe

输入要求输入姓名和密码，姓名已知是welcomebeijing。

sub_401090函数生成byte_416050数组。

main函数想要跳出死循环，sub_401830和sub_4011A0（loc_4011A0全部选中，按p，创造函数）返回值大于0。

sub_4011A0函数反编译可知只生成成功字符串和失败字符串，返回值必为1，所以重点来到sub_401830。

点击进入sub_401830，从后往前看 return v14 == 43924，找最近的v14调用，上面的函数sub_401470就必需成立，进入之后v17=dbappsec满足条件。其中掺着类似Ntglobalflag的反调试，通过分析反调试代码可知a[5]=s而不是f。

接着向上找v17的调用。

```
      v13 = v11 + v12;
    *(&v17 + v6) = byte_416050[(unsigned __int8)(v8 + v13)] ^ *(&v15 + v5);
    if ( *(_DWORD *)(__readfsdword(0x30u) + 2) & 0xFF )
    {
      v11 = -83;
      v12 = 43;
    }
    sub_401710((int)&v17, (const char *)a1, v6++);
    v5 = v6;
    if ( v6 >= (unsigned int)(&v15 + strlen(&v15) + 1 - &v16) )
      v5 = 0;
  }
  v14 = 0;
  sub_401470(ebx0, &v17, &v14);
  return v14 == 43924;
}
```


sub_401710调用v17，但此函数a3<=v6一直成立，导致下面的if条件根本不会进入，对v17的没有改变。

上面的异或才是真正改变v17的值。找关键函数后，从前往后看。

字符转整型

```
  v4 = 0;
  while ( v7 < strlen(a2) )
  {
    if ( isdigit(a2[v7]) )                    // 判断一个字符是否是十进制数字（看这个字符是不是0-9）
    {
      v9 = a2[v7] - 48;                       // 字符转整型（如果是0-9，就'9'->9）
    }
    else if ( isxdigit(a2[v7]) )              // 检查所传的字符是否是十六进制数字(字符不是0-9，再看是不是a-f)
    {
      if ( *(_DWORD *)(*(_DWORD *)(__readfsdword(0x30u) + 24) + 12) != 2 )// 反调试
        a2[v7] = 34;
      v9 = (a2[v7] | 0x20) - 87;              // 字符转整型（若是a-f,则'f'->f）
    }
    else
    {
      v9 = ((a2[v7] | 0x20) - 97) % 6 + 10;   // 都不是前面的情况，就只可能是字符'g-z',则字符按顺序，6个为一组，分别对应整型10-15
    }
    v10 = v9 + 16 * v10;                      // v7是偶数,原字符经变换后的值不会赋给v15，而是暂存v10，遇到v7是奇数，v10左移4位，
    if ( !((signed int)(v7 + 1) % 2) )        // 变成高位，再加上奇数位的数。
    {
      *(&v15 + v4++) = v10;
      ebx0 = v4;
      v10 = 0;
    }
    ++v7;                                     // 上述的操作就是将一个字符变成对应的整型
  }
  while ( v6 < 8 )
  {
```


复杂数组变化，动调解决

```
  while ( v6 < 8 )
  {
    v11 += byte_416050[++v12];                // 复杂的数组变化
    v13 = byte_416050[v12];
    v8 = byte_416050[v11];
    byte_416050[v11] = v13;
    byte_416050[v12] = v8;
    if ( *(_DWORD *)(__readfsdword(0x30u) + 104) & 0x70 )
      v13 = v11 + v12;
    *(&v17 + v6) = byte_416050[(unsigned __int8)(v8 + v13)] ^ *(&v15 + v5);// 求出v15的值就是flag的值，byte_416050每次异或的值动调可得
    if ( *(_DWORD *)(__readfsdword(0x30u) + 2) & 0xFF )// byte_416050每次异或的值都是数组的复杂运算，所以直接动调
    {
      v11 = -83;
      v12 = 43;
    }
    sub_401710((int)&v17, (const char *)a1, v6++);
    v5 = v6;
    if ( v6 >= (unsigned int)(&v15 + strlen(&v15) + 1 - &v16) )
      v5 = 0;
  }
```


动调可得byte_416050每次异或的值是0xd7,0x92,0xe9,0x53,0xe2,0xc4,0xcd。

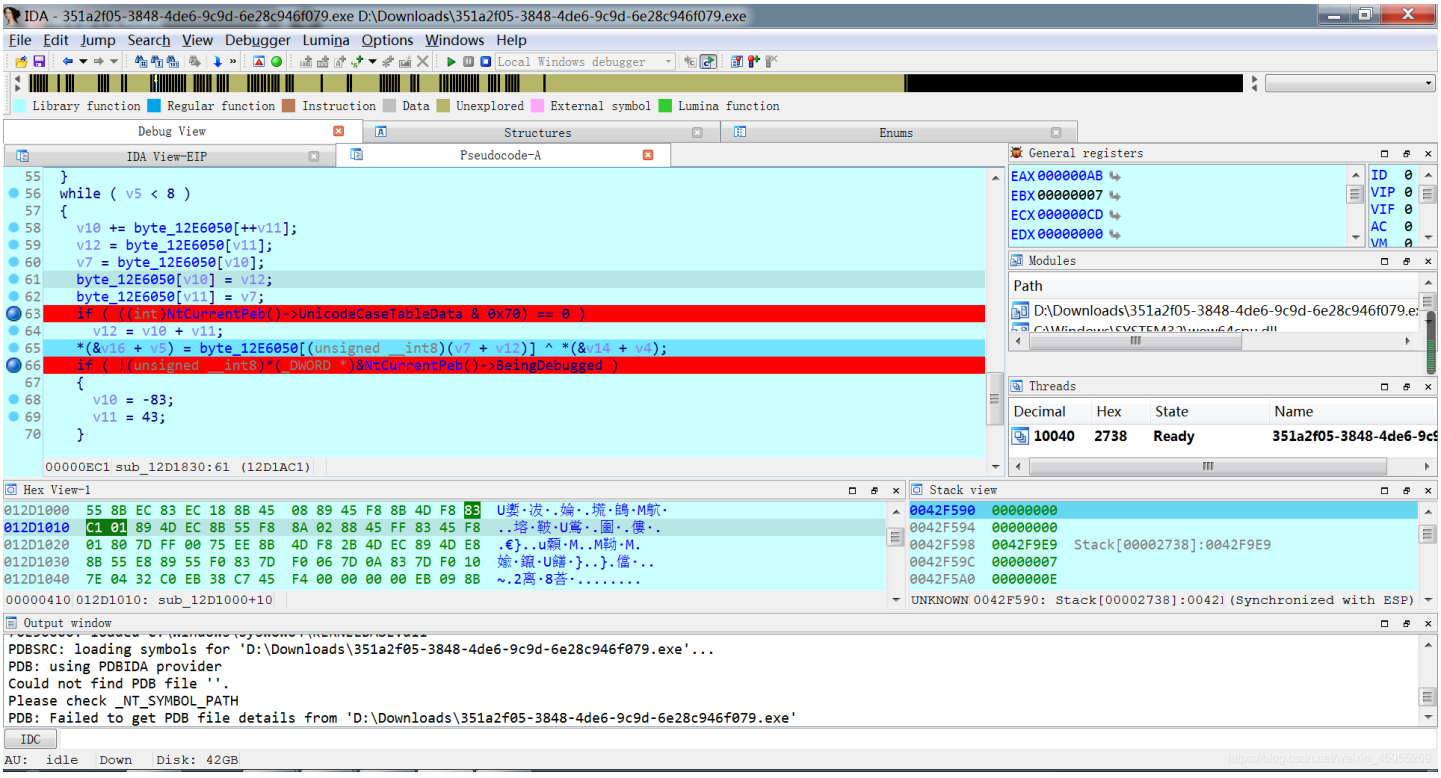
```python
import hashlib

box = [0x2a, 0xd7, 0x92, 0xe9, 0x53, 0xe2, 0xc4, 0xcd]
s = "dbappsec"

secret = []

for i in range(len(s)):
    secret.append(hex(ord(s[i])^box[i]).replace("0x",''))
flag = ''.join(secret)
print(flag)
md = hashlib.md5()
md.update(flag.encode('utf-8'))
print ("flag{"+md.hexdigest()+"}")
```

这里存在的问题，字符转整型，这个过程是不可逆的，所以真正的password是得不到了，还有偶数位和奇数位高低位互换，最后的flag要互换回来吗？。出题没想这么多，只是将异或出来的v15的值作为了flag。
反调试的绕过是本题的重点：两种思路一种是利用插件使程序检测不到异常，不触发反调试，另一种思路是触发反调试改调试的跳转，这个需要精确的找到在哪里跳转。
本题采用第二种反调试方法将关键异或的上下两句反调的jz跳转改为jnz（字节74->75），因为只有上下两个反调影响v12，也就是影响的异或的取值。



本题中还穿插着Ntglobalflag的反调试技巧。

6.[BJDCTF2020]BJD hamburger competition

小汉堡，233。unity游戏，找Assembly-CSharp.dll，再通过搜索flag或dasctf等关键字符串，找到核心函数。

在ButtonSpawnFruit里找到Spawn函数，SHA1加密的字符串，SHA1爆破后，再md5加密即可。

```
        }
        else if (name == "汉堡顶" && Init.spawnCount == 5)
        {
            Init.secret ^= 127;
            string str = Init.secret.ToString();
            if (ButtonSpawnFruit.Sha1(str) == "DD01903921EA24941C26A48F2CEC24E0BB0E8CC7")
            {
                this.result = "BJDCTF{" + ButtonSpawnFruit.Md5(str) + "}";
                Debug.Log(this.result);
            }
        }
        Init.spawnCount++;
        Debug.Log(Init.secret);
```


进入md5加密。

```
    public static string Md5(string str)
    {
        byte[] bytes = Encoding.UTF8.GetBytes(str);
        byte[] array = MD5.Create().ComputeHash(bytes);
        StringBuilder stringBuilder = new StringBuilder();
        foreach (byte b in array)
        {
            stringBuilder.Append(b.ToString("X2"));
        }
        return stringBuilder.ToString().Substring(0, 20);
```


自己定义的md5取前20位大写的md5加密

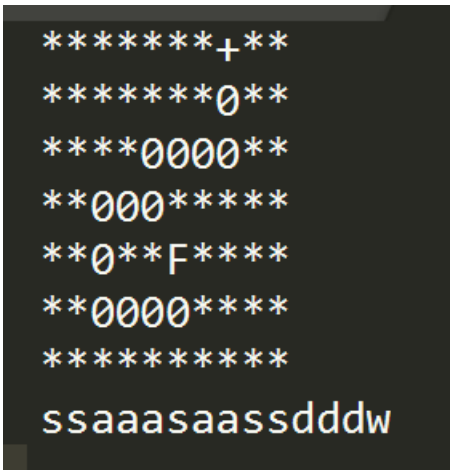
7.[HDCTF2019]Maze

UPX的壳，脱壳后，拖到IDA里。主函数反编译不了，考虑去除花指令。

```
.text:00401019                  push    eax
.text:0040101A                  push    offset a14s     ; "%14s"
.text:0040101F                  call    _scanf
.text:00401024                  add     esp, 8
.text:00401027                  push    eax
.text:00401028                  xor     eax, ecx
.text:0040102A                  cmp     eax, ecx
.text:0040102C                  jnz     short near ptr loc_40102E+1
.text:0040102E
.text:0040102E loc_40102E:                              ; CODE XREF: .text:0040102C↑j
.text:0040102E                  call    near ptr 0EC85D78Bh
.text:0040102E ; ---------------------------------------------------------------------------
.text:00401033                  db 0
.text:00401034                  dd 0EB000000h, 0EC4D8B09h, 8901C183h, 7D83EC4Dh, 6E7F0DECh
.text:00401034                  dd 0FEC558Bh, 0F01544BEh, 8BE84589h, 0E983E84Dh, 0E84D8961h
.text:00401034                  dd 16E87D83h, 458B5277h, 8AD233E8h, 40111290h, 9524FF00h
.text:00401034                  dd 4010FEh, 807C0D8Bh, 0C1830040h, 7C0D8901h, 0EB004080h
.text:00401034                  dd 7C158B2Fh, 83004080h, 158901EAh, 40807Ch, 78A11EEBh
.text:00401034                  dd 83004080h, 78A301E8h, 0EB004080h, 780D8B0Fh, 83004080h
.text:00401034                  dd 0D8901C1h, 408078h, 3D8383EBh, 408078h, 83297505h, 40807C3Dh
```

`jnz short near ptr loc_40102E+1`跳转到下一行指令，花指令去除，nop掉，剩下的字符按C转汇编。红色代码部分按P创建函数。F5得到观察逻辑得到上下左右-wsad，和赛车游戏的上下左右同。

搜字符找到地图，猜测7*10地图。原本是空格我这里用0填充了一下。



IDA指令：按D转数据，按C转代码。

8.[FlareOn4]IgniteMe

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

  NumberOfBytesWritten = 0;
  hFile = GetStdHandle(0xFFFFFFF6);
  dword_403074 = GetStdHandle(0xFFFFFFF5);
  WriteFile(dword_403074, aG1v3M3T3hFl4g, 0x13u, &NumberOfBytesWritten, 0);
  sub_4010F0();                                 // 排除'\n'和'\r'
  if ( sub_401050() )                           // 异或操作，v4的值动调可得
    WriteFile(dword_403074, aG00dJ0b, 0xAu, &NumberOfBytesWritten, 0);
  else
    WriteFile(dword_403074, aN0tT00H0tRWe7r, 0x24u, &NumberOfBytesWritten, 0
  ExitProcess(0);
}
```



```
signed int sub_401050()
{
  int v0; // ST04_4
  int i; // [esp+4h] [ebp-8h]
  unsigned int j; // [esp+4h] [ebp-8h]
  char v4; // [esp+Bh] [ebp-1h]

  v0 = sub_401020((int)byte_403078);
  v4 = sub_401000();
  for ( i = v0 - 1; i >= 0; --i )
  {
    byte_403180[i] = v4 ^ byte_403078[i];       // v4=4
    v4 = byte_403078[i];
  }
  for ( j = 0; j < 39; ++j )
  {
    if ( byte_403180[j] != (unsigned __int8)byte_403000[j] )
      return 0;
  }
  return 1;
}
```


函数少的一批，输入字符串反向的异或操作。

```
tmp=[0xd,0x26,0x49,0x45,0x2a,0x17,0x78,0x44,0x2b,0x6c,
0x5d,0x5e,0x45,0x12,0x2f,0x17,0x2b,0x44,0x6f,0x6e,![在这里插入图片描述](https://img-blog.csdnimg.cn/20201003174551
425.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3dlaX
hpbl80NTA1NTI2OQ==,size_16,color_FFFFFF,t_70#pic_center)

0x56,0x9,0x5f,0x45,0x47,0x73,0x26,0xa,0xd,0x13,
0x17,0x48,0x42,0x1,0x40,0x4d,0xc,0x2,0x69]
v4=4
flag=[0 for i in range(39)]
for i in range(38,-1,-1):
 flag[i]=tmp[i]^v4
 v4=flag[i]
print(''.join(map(chr,flag)))
#R_y0u_H0t_3n0ugH_t0_1gn1t3@flare-on.com
```

9.[WUSTCTF2020]level1

奇偶数的分开操作，数组下标的关系是本题考点。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  FILE *stream; // ST08_8
  signed int i; // [rsp+4h] [rbp-2Ch]
  char ptr[24]; // [rsp+10h] [rbp-20h]
  unsigned __int64 v7; // [rsp+28h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  stream = fopen("flag", "r");
  fread(ptr, 1uLL, 0x14uLL, stream);
  fclose(stream);
  for ( i = 1; i <= 19; ++i )
  {
    if ( i & 1 )
      printf("%ld\n", (unsigned int)(ptr[i] << i));// 奇数
    else
      printf("%ld\n", (unsigned int)(i * ptr[i]));// 偶数
  }
  return 0;
}
```



```
tmp=[198,232,816,200,1536,300,6144,984,51200,570,92160,1200,565248,756,1474560,800,6291456,1782,65536000]
flag=[0 for i in range(19)]
for i in range(1,20):
 if(i&1):
  flag[i-1]=tmp[i-1]>>i
 else:
  flag[i-1]=tmp[i-1]//i
print(''.join(map(chr,flag)))
#ctf2020{d9-dE6-20c}
```

10.[WUSTCTF2020]level2

linux下UPX脱壳，IDA打开，flag映入眼帘。

```
; __unwind {
lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    ecx
sub     esp, 14h
mov     [ebp+var_C], offset flag ; "wctf2020{Just_upx_-d}"
sub     esp, 0Ch
push    offset aWhereIsIt ; "where is it?"
call    puts
add     esp, 10h
mov     eax, 0
mov     ecx, [ebp+var_4]
leave
lea     esp, [ecx-4]
retn
; } // starts at 804887C
main endp
```

11.firmware

固件安全，完整版binwalk+firmware-mod-kit，步骤操作得flag。