

170720 逆向-misc-cctf(1)

原创

奈沙夜影 于 2017-07-21 19:16:04 发布 805 收藏

分类专栏: [CTF](#)

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/whklhjh/article/details/75670371>

版权



[CTF 专栏收录该内容](#)

163 篇文章 4 订阅

订阅专栏

1625-5 王子昂 总结《2017年7月20日》【连续第291天总结】

A. cctf writeup(1)

B.

Reverse

我也不知道哈哈

拖入IDA发现是64位的elf文件, 直接反编译源码, 很简单, 看到如下判断语句

```
if ( (unsigned int)sub_4007BD((__int64)&v5, (__int64)"flag{PbkD7j4X|8Wz;~;z_01}") )
    puts("[*] You Got It!");
```

直接提交flag发现错误，说明sub_4007BD还有处理功能\ (_ ^) ^也对，flag里面一般不带~这种乱七八糟的符号
sub_4007BD:

```
1  int64 __fastcall sub_4007BD(__int64 a1, __int64 a2)
2  {
3      size_t v2; // rbx@1
4      char *s; // [sp+8h] [bp-28h]@8
5      signed int v5; // [sp+18h] [bp-18h]@1
6      signed int i; // [sp+1Ch] [bp-14h]@8
7      signed int j; // [sp+1Ch] [bp-14h]@13
8
9      v5 = 0;
10     v2 = strlen((const char *)a1);
11     if ( v2 != strlen((const char *)a2) )
12         return 0LL;
13     if ( *(_BYTE *)a1 == 102
14         && *(_BYTE *)a1 + 1 == 108
15         && *(_BYTE *)a1 + 2 == 97
16         && *(_BYTE *)a1 + 3 == 103
17         && *(_BYTE *)a1 + 4 == 123 )
18     {
19         s = (char *)a1 + 5;
20         for ( i = 0; i <= 7; ++i )
21         {
22             if ( ((unsigned __int8)s[i] ^ 7) != *(_BYTE *)a2 + 5LL )
23                 return 0LL;
24         }
25         for ( j = 8; j <= 15; ++j )
26         {
27             if ( ((unsigned __int8)s[j] ^ 8) != *(_BYTE *)a2 + 5LL )
28                 return 0LL;
29         }
30         if ( __PAIR__(*_BYTE *)a1 + 22, *_BYTE *)a1 + 21) != __PAIR__(*_BYTE *)a2 + 22, *_BYTE *)a2 + 21)
31             || *_BYTE *)a1 + 23) != *_BYTE *)a2 + 23) )
32         {
33             return 0LL;
34         }
35         if ( *(_BYTE *)a1 + 24 == 125 )
36             v5 = 1;
37     }
38     return (unsigned int)v5;
39 }
```

<http://blog.csdn.net/whk1h1111>

相当清晰，首先判断长度，然后前5个字符直接对应ASCII，后7个字符为a2+5处的字符与7异或对应，即上文出现的flag中的字符对应与7异或；再后7个字符同理与8异或；最后第21、22、23个字符是相同的，最后一个字符为ASCII为125对应的字符，写一个脚本求出即可：

```
a="flag{Pbk07j4x{8Wz;~;z_01}"
f=""
i=[102,108,97,103,123]
for k in i:
    f=f+chr(k)
for i in range(8):
    f=f+(chr(ord(a[i+5])^7))
for i in range(8):
    f=f+(chr(ord(a[i+13])^8))
for i in range(3):
    f=f+(a[i+21])
f=f+(chr(125))
print(f)
```

密码

解压zip得到一个exe，用ExeInfo查壳，发现无壳、C#.NET编写的
那么掏出方便的ILSpy，反编译得到按钮的代码：

```

private void btn_ok_Click(object sender, EventArgs e)
{
    if (this.textBox_Pass.Text == "")
    {
        MessageBox.Show("请输入密码!");
        return;
    }
    char[] array = this.textBox_Pass.Text.Replace(".", "-").Replace("7", "t").Replace("4", "a")
    //将输入文本中的字符置换
    Array.Reverse(array);
    //逆序
    string text = new string(array);
    text.ToUpper();
    //转大写 (逆向过程中未发现有啥用、(‘_’))
    char[] array2 = Convert.ToBase64String(Encoding.GetEncoding("UTF-8").GetBytes(text)).ToCharArray()
    //b64加密一次
    Array.Reverse(array2);
    //逆序
    string s = new string(array2);
    char[] value = Convert.ToBase64String(Encoding.GetEncoding("UTF-8").GetBytes(s)).ToCharArray()
    //再b64加密一次
    string a = new string(value);
    if (a == "PTBpVGxSM1lqTwtVaE4yU0pOM1J2Qnph")
    //检查是否与给定字符串相等
    {
        MessageBox.Show("密码正确! 密码就是Key!", "成功");
        return;
    }
    MessageBox.Show("密码错误!", "失败");
}
}

```

按照过程反向操作即可得到flag，b64用的网页版，就没写脚本了~

值得一提的是replace部分，原过程中为先将数字0转成小写o，再将大写O转成数字0；但是这一步使用脚本直接复制的时候忘记交换顺序了，导致变换逆序，即先将小写o转成数字0，再将数字0转成大写O了，导致出错好久(:3/ <)

又是个RE

等了好久再又出来一个RE，迫不及待的拖入IDA，这次代码更少，只有一个main函数了：

```
push    ebp
mov     ebp, esp
push    edi
and     esp, 0FFFFFF0h
sub     esp, 30h
mov     dword ptr [esp+2Ch], 0
mov     dword ptr [esp], 18h ; size
call   _malloc
mov     [esp+2Ch], eax
mov     dword ptr [esp+8], 18h ; n
mov     dword ptr [esp+4], 0 ; c
mov     eax, [esp+2Ch]
mov     [esp], eax ; s
call   _memset
mov     eax, [esp+2Ch]
mov     dword ptr [eax], 'GALF'
mov     dword ptr [eax+4], '904-'
mov     word ptr [eax+8], '2'
mov     eax, [esp+2Ch]
mov     dword ptr [esp+1Ch], 0FFFFFFFh
mov     edx, eax
mov     eax, 0
mov     ecx, [esp+1Ch]
mov     edi, edx
repne scasb
mov     eax, ecx
not     eax
lea     edx, [eax-1]
mov     eax, [esp+2Ch]
add     eax, edx
mov     dword ptr [eax], 'u948'
mov     dword ptr [eax+4], 'j2oi'
mov     word ptr [eax+8], 'f'
mov     dword ptr [esp], offset s ; "Loading..."
call   _puts
mov     eax, [esp+2Ch]
mov     dword ptr [esp+1Ch], 0FFFFFFFh
mov     edx, eax
mov     eax, 0
mov     ecx, [esp+1Ch]
mov     edi, edx
repne scasb
mov     eax, ecx
not     eax
```

<http://blog.csdn.net/whklhxxx>

将数字按r以ASCII字符形式显示，说什么where is flag，很明显，上面的GALF就是FLAG的逆序了，因此flag就是那一堆字符咯，记得由于小端序要逆序：flag-4092849uio2jf

然后这题格式也没提示，自己试吧：flag{4092849uio2jf}

MISC

kungfu

zip打开不了，用winhex查看发现是png文件头，遂转换格式，发现是秘籍的图片，用binwalk也没检查出来什么东西。再用winhex查看，拖到最后发现了好东西：KEY IS VF95c0s5XzVyaGtfX3VGTXR9M0Vse251QEUg 直接提交，不好使；猜测是b64，解码试试，得到T_ysK9_5rhk__uFMt}3E{nu@E

出现了{}，估计是栅栏，用len知长度为27，遂以3x9重组，得到Th3_kEy_Is_{Kun9Fu_M@5tEr}

同样也要注意flag提交格式

你知道么？这是什么

下载下来是无格式文件，winhex看到PK文件头，添加zip后缀名，解压出hidden.png

用binwalk查看，发现其中藏了一个zip文件

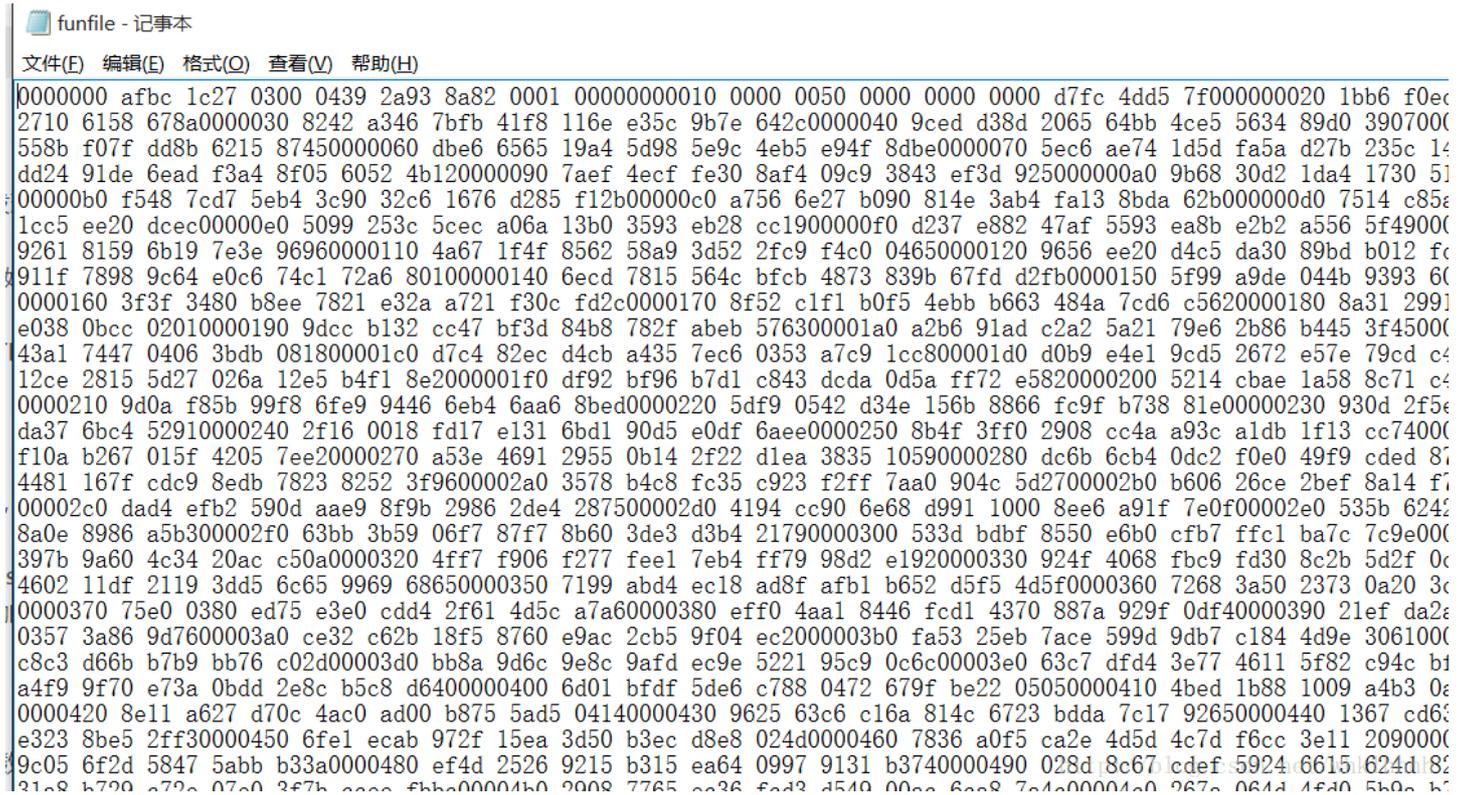
直接将后缀名改成zip，或者根据binwalk得到的偏移位置用c32Asm扒下来都可以

打开发现hidden2.png，但是加密了。没有提示，所以应该是伪加密，将对应位的01修改为02，解压出hidden2.png，打开看到

flag

Have Fun! !

funfile是无格式文件，先用记事本查看，发现是分隔好的二进制数据：



```
funfile - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)
00000000 afbc 1c27 0300 0439 2a93 8a82 0001 00000000010 0000 0050 0000 0000 0000 d7fc 4dd5 7f000000020 1bb6 f0ec
2710 6158 678a0000030 8242 a346 7bfb 41f8 116e e35c 9b7e 642c0000040 9ced d38d 2065 64bb 4ce5 5634 89d0 390700
558b f07f dd8b 6215 87450000060 dbe6 6565 19a4 5d98 5e9c 4eb5 e94f 8dbe0000070 5ec6 ae74 1d5d fa5a d27b 235c 14
dd24 91de 6ead f3a4 8f05 6052 4b120000090 7aef 4ecf fe30 8af4 09c9 3843 ef3d 925000000a0 9b68 30d2 1da4 1730 51
00000b0 f548 7cd7 5eb4 3c90 32c6 1676 d285 f12b00000c0 a756 6e27 b090 814e 3ab4 fa13 8bda 62b000000d0 7514 c85e
1cc5 ee20 dcec00000e0 5099 253c 5cec a06a 13b0 3593 eb28 cc1900000f0 d237 e882 47af 5593 ea8b e2b2 a556 5f4900
9261 8159 6b19 7e3e 96960000110 4a67 1f4f 8562 58a9 3d52 2fc9 f4c0 04650000120 9656 ee20 d4c5 da30 89bd b012 fc
911f 7898 9c64 e0c6 74c1 72a6 80100000140 6ecd 7815 564c bfc9 4873 839b 67fd d2fb0000150 5f99 a9de 044b 9393 60
0000160 3f3f 3480 b8ee 7821 e32a a721 f30c fd2c0000170 8f52 c1f1 b0f5 4ebb b663 484a 7cd6 c5620000180 8a31 2991
e038 0bcc 02010000190 9dcc b132 cc47 bf3d 84b8 782f abeb 576300001a0 a2b6 91ad c2a2 5a21 79e6 2b86 b445 3f4500
43a1 7447 0406 3bdb 081800001c0 d7c4 82ec d4cb a435 7ec6 0353 a7c9 1cc800001d0 d0b9 e4e1 9cd5 2672 e57e 79cd c4
12ce 2815 5d27 026a 12e5 b4f1 8e2000001f0 df92 bf96 b7d1 c843 dcda 0d5a ff72 e5820000200 5214 cbae 1a58 8c71 c4
0000210 9d0a f85b 99f8 6fe9 9446 6eb4 6aa6 8bed0000220 5df9 0542 d34e 156b 8866 fc9f b738 81e00000230 930d 2f5e
da37 6bc4 52910000240 2f16 0018 fd17 e131 6bd1 90d5 e0df 6aee0000250 8b4f 3ff0 2908 cc4a a93c a1db 1f13 cc7400
f10a b267 015f 4205 7ee20000270 a53e 4691 2955 0b14 2f22 d1ea 3835 10590000280 dc6b 6cb4 0dc2 f0e0 49f9 cded 87
4481 167f cdc9 8edb 7823 8252 3f9600002a0 3578 b4c8 fc35 c923 f2ff 7aa0 904c 5d2700002b0 b606 26ce 2bef 8a14 f7
00002c0 dad4 efb2 590d aae9 8f9b 2986 2de4 287500002d0 4194 cc90 6e68 d991 1000 8ee6 a91f 7e0f00002e0 535b 6242
8a0e 8986 a5b300002f0 63bb 3b59 06f7 87f7 8b60 3de3 d3b4 21790000300 533d bdbf 8550 e6b0 cfb7 ffc1 ba7c 7c9e00
397b 9a60 4c34 20ac c50a0000320 4ff7 f906 f277 fee1 7eb4 ff79 98d2 e1920000330 924f 4068 fbc9 fd30 8c2b 5d2f 0c
4602 11df 2119 3dd5 6c65 9969 68650000350 7199 abd4 ec18 ad8f afb1 b652 d5f5 4d5f0000360 7268 3a50 2373 0a20 3c
0000370 75e0 0380 ed75 e3e0 cdd4 2f61 4d5c a7a60000380 eff0 4aa1 8446 fcd1 4370 887a 929f 0df40000390 21ef da2e
0357 3a86 9d7600003a0 ce32 c62b 18f5 8760 e9ac 2cb5 9f04 ec2000003b0 fa53 25eb 7ace 599d 9db7 c184 4d9e 306100
c8c3 d66b b7b9 bb76 c02d00003d0 bb8a 9d6c 9e8c 9afd ec9e 5221 95c9 0c6c00003e0 63c7 dfd4 3e77 4611 5f82 c94c b1
a4f9 9f70 e73a 0bdd 2e8c b5c8 d6400000400 6d01 bdfd 5de6 c788 0472 679f be22 05050000410 4bed 1b88 1009 a4b3 0a
0000420 8e11 a627 d70c 4ac0 ad00 b875 5ad5 04140000430 9625 63c6 c16a 814c 6723 bdda 7c17 92650000440 1367 cd63
e323 8be5 2ff30000450 6fef ecab 972f 15ea 3d50 b3ec d8e8 024d0000460 7836 a0f5 ca2e 4d5d 4c7d f6cc 3e11 209000
9c05 6f2d 5847 5abb b33a0000480 ef4d 2526 9215 b315 ea64 0997 9131 b3740000490 028d 8c67 cdef 5924 6615 624d 82
1318 b790 a79c 07c0 3f7b cccc fbba00004b0 9008 7765 cc36 fcd3 d540 00c8 6cc8 7c4a00004c0 967c 084d 4f40 5b0c b5
```

用UltraEdit选择第9列到最后，将标识地址的00000000排除，然后在c32Asm中新建十六进制文件，特殊粘贴-ASCII-HEX，查询AF BC 1C 27文件头，发现近似的只有7z的37 7a BC AF 27 1C文件头，说明要以小端序即每个字节内逆序处理，再添加37 7a来修复

先存储，然后用python以二进制形式打开，逆序的问题费了我一番脑筋，因为bytes类型不怎么会处理最后以数组形式取出每个数字，然后每两个倒叙一下得到数据流，记得格式化宽度：

```
f=open('F:/ctf/ctf/havefun.zip','rb')
n=f.read()
print(n)
flag=[]
for i in range(50552):
    #print(type(flag))
    flag.append(hex(n[2*i+1]))
    flag.append(hex(n[2*i]))
print(flag)
for i in flag:
    l=i[2:]
    if(len(l)!=2):
        l='0'+l
    print(l,end='')
```

复制输出，然后粘贴到c32Asm中保存为十六进制文件，再添加37 7a的文件头终于成功打开压缩包—又是一张图片：阿狸.jpg
用binwalk查看，未发现任何隐藏信息

以记事本形式打开，突然发现有一句眼熟的东西：`eval(echo(Q1RG{RIVOT11fUDFDVfVSMw==})`
将括号内的字符进行b64解码，终于得到flag

C. 明日计划

继续cctf