攻防世界（新手篇）

原创

chan3301 于 2019-06-02 12:37:54 发布 10139 收藏 9

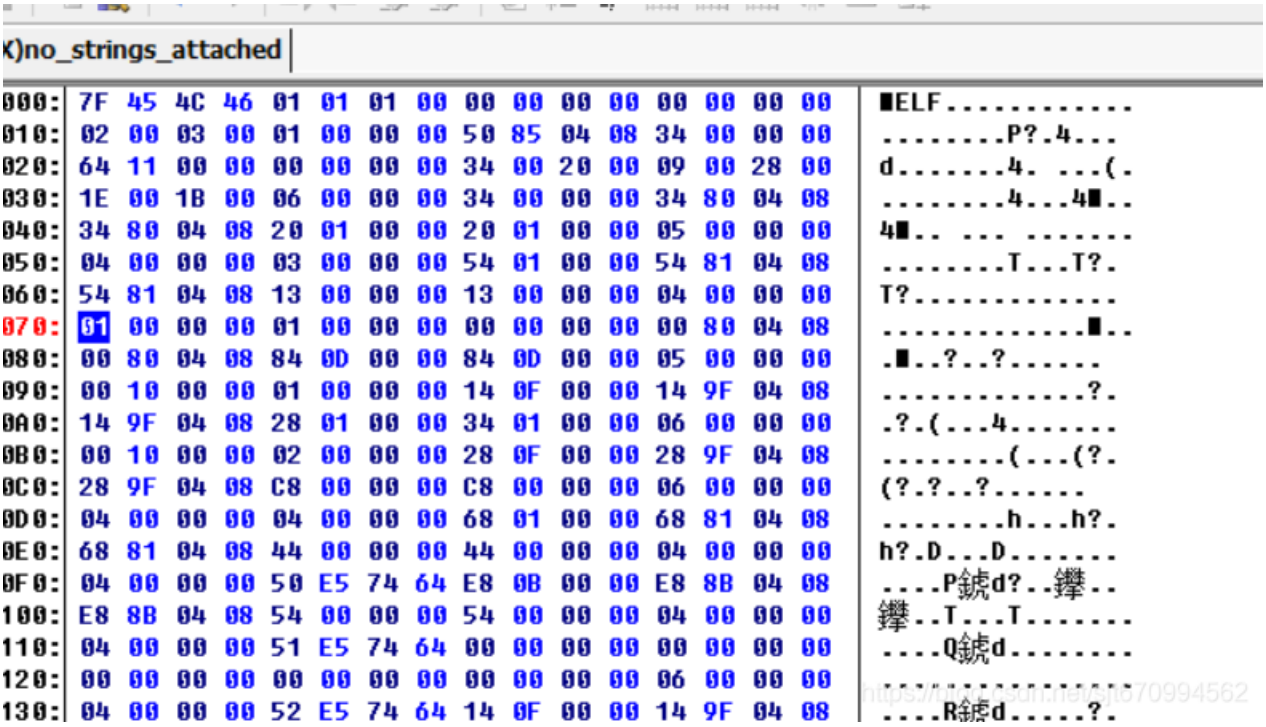
分类专栏： 逆向题目练习 文章标签： 攻防世界 逆向入门

本文链接：https://blog.csdn.net/sjt670994562/article/details/90738133

作为刚刚进入reserve的小白，这几天在攻防世界的一些解题

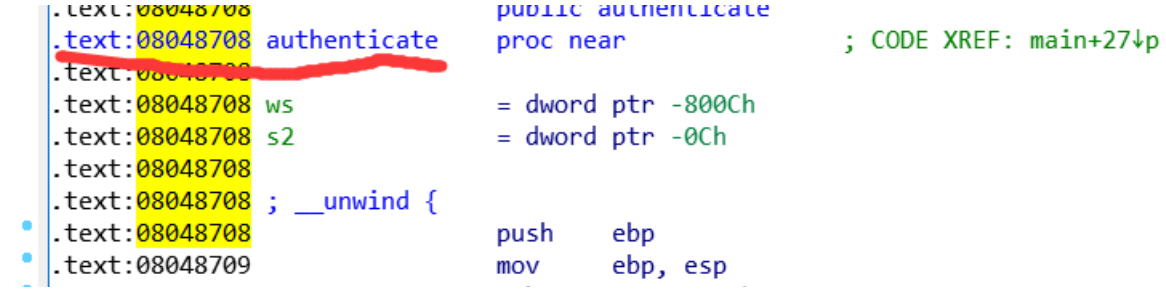
no_strings_attached

一看没有exe，估计是linux的文件了，扔进C32Asm里



果然elf文件

先用扔进ida分析，发现主要函数authenticate，找到他的地址



OK，接下来就是扔到linux的gdb里面分析啦（这里我用的是peda显示的更方便点）

```
root@kali:/mnt/hgfs/vm-share# gdb no_strings_attached
GNU gdb (Debian 8.2-1) 8.2
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from no_strings_attached...(no debugging symbols found)...done.
gdb-peda$
```

用b来下断点，b*0x08048708，然后r运行，n单步往前走

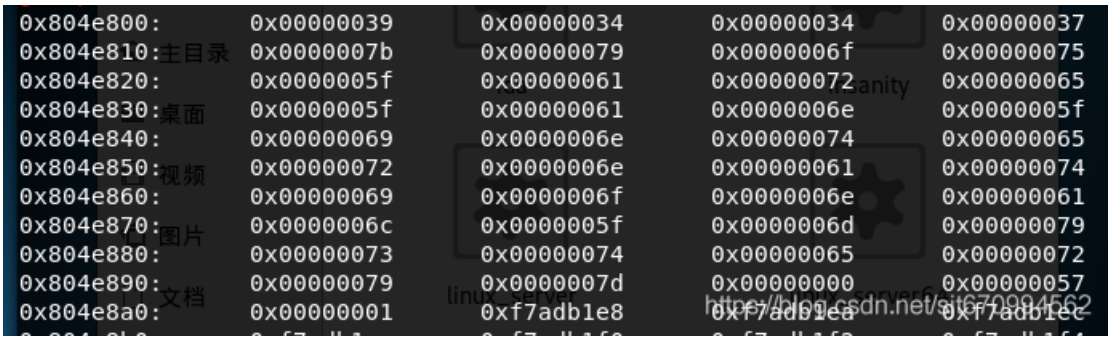
```
[----------------------------------code----------------------------------]
   0x8048705 <decrypt+173>:     pop    ebx
   0x8048706 <decrypt+174>:     pop    ebp
   0x8048707 <decrypt+175>:     ret
=> 0x8048708 <authenticate>:    push   ebp
   0x8048709 <authenticate+1>:  mov    ebp,esp
   0x804870b <authenticate+3>:  sub    esp,0x8028
   0x8048711 <authenticate+9>:  mov    DWORD PTR [esp+0x4],0x8048a90
   0x8048719 <authenticate+17>: mov    DWORD PTR [esp],0x8048aa8
[----------------------------------stack----------------------------------]
0000| 0xffffd23c --> 0x80487d5 (<main+44>:      mov    eax,0x0)
0004| 0xffffd240 --> 0x6
0008| 0xffffd244 --> 0x8048be4 --> 0x0
0012| 0xffffd248 --> 0x80487e9 (<__libc_csu_init+9>:    add    ebx,0x180b)
0016| 0xffffd24c --> 0x0
0020| 0xffffd250 --> 0xf7fa9000 --> 0x1d9d6c
0024| 0xffffd254 --> 0xf7fa9000 --> 0x1d9d6c
0028| 0xffffd258 --> 0x0
[------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x08048708 in authenticate ()
gdb-peda$
```

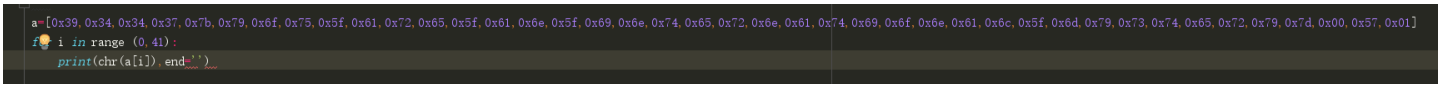
```
gdb-peda$ info reg
eax            0x804e800        0x804e800
ecx            0x1480           0x1480
edx            0x7d             0x7d
ebx            0x0              0x0
esp            0xffff5210       0xffff5210
ebp            0xffffd238       0xffffd238
esi            0xf7fa9000       0xf7fa9000
edi            0xf7fa9000       0xf7fa9000
eip            0x8048725        0x8048725 <authenticate+29>
eflags         0x282            [ SF IF ]
cs             0x23             0x23
ss             0x2b             0x2b
ds             0x2b             0x2b
es             0x2b             0x2b
fs             0x0              0x0
gs             0x63             0x63
```

断点指令reg info查看寄存器

用x指令查看0x282，x/282x &eax



这便是数据，copy出来通过python转码既得flag



csaw2013reversing2

首先终于是一个exe文件，但是我一直打不开，而且不容易动态分析，所以这里就只讲静态分析了

```
memcpy_s(lpMem, MaxCount, &dword_409B10, MaxCount);
if ( sub_40102A() || IsDebuggerPresent() )
```

扔进ida，找到main函数
发现dword409B10提供了数据，猜测应该是源数据

```
| 0BCA0CCBBh
| 0B8BED1DCh
| 0AEBECFCDh
| 82ABC4D2h
| 0B393D9D2h
| 0A993DED4h
| 82B8CBD3h
| 0B9BECBD3h
| 0DDCCD79Ah
```

发现sub_401000是变形函数，就是源数据和提供的一个数据进行抑或
dword_409B38便是提供数据

```
38 dword_409B38     dd 0DDCCAABBh
```

用python写出代码

```
a=["BCA0CCBB","B8BED1DC","AEBECFCD","82ABC4D2","B393D9D2","A993DED4","82B8CBD3","B9BECBD3","DDCCD79A"]
t=("DDCCAABB")
p=[]
i=0
while(i<=8):
    for j in range(3,-1,-1):
        print (chr(int(a[i][j*2:j*2+2],16)^int(t[j*2:j*2+2],16)),end='')
    i=i+1
```

得到flag

```
flag{reversing_is_not_that_hard!}
```

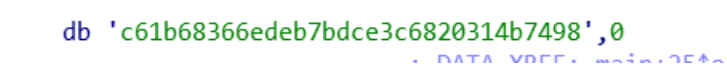
getit

静态分析
放入ida，查看到中心函数

```
while ( (signed int)v5 < strlen(s) )
{
  if ( v5 & 1 )
    v3 = 1;
  else
    v3 = -1;
  *(&t + (signed int)v5 + 10) = s[(signed int)v5] + v3;
  LODWORD(v5) = v5 + 1;
}
```

意思就是根据index值得奇偶来判断该值ord是+1还是-1

s的值也已经给了，如下图



写出python脚本就行

```
a=("c61b68366edeb7bdce3c6820314b7498")
for i in range (0,32):
    if(i%2!=0):
        print(chr(ord(a[i])+1),end='')
    else:
        print(chr(ord(a[i])-1), end='')
```

```
b70c59275fcfa8aebf2d5911223c6589
```

python-trade

这道题比较简单，用py反编译器用一下就行，下载地址http://sourceforge.net/projects/easypythondecompiler/?source=directory
反编译结束后，得到代码

```
# Embedded file name: 1.py
import base64

def encode(message):
    s = ''
    for i in message:
        x = ord(i) ^ 32
        x = x + 16
        s += chr(x)

    return base64.b64encode(s)


correct = 'XlNkVmtUI1MgXWBZXCFeKY+AaXNt'
flag = ''
print ('Input flag:')
flag = raw_input()
if encode(flag) == correct:
    print ('correct')
else:
    print ('wrong')
```


只要倒着推先base64解码，再-16，和32异或即可

```
import base64

buf = base64.b64decode('XlNkVmtUI1MgXWBZXCFeKY+AaXNt')
flag = ''
for i in buf:
    i -= 16
    i ^= 32
    flag += chr(i)

print(flag)
```


这里最好用python软件直接base64解码，不要用外部网站，不然最后结果会差一点点，具体情况为啥我也不知道，正能猜测base64解码的代码不同吧
最后得到flag

```
nctf{d3c0mpil1n9_PyC}
```

maze

这道题在做的时候完全摸不清头脑，主要还是自己的思路太窄了吧，很有趣的一道迷宫题
首先ida静态分析，代码很多，很烦，但是要有耐心，一步一步来
第一是在判断密码位是否24，开头是否为 nctf{ ，结尾是否为 }

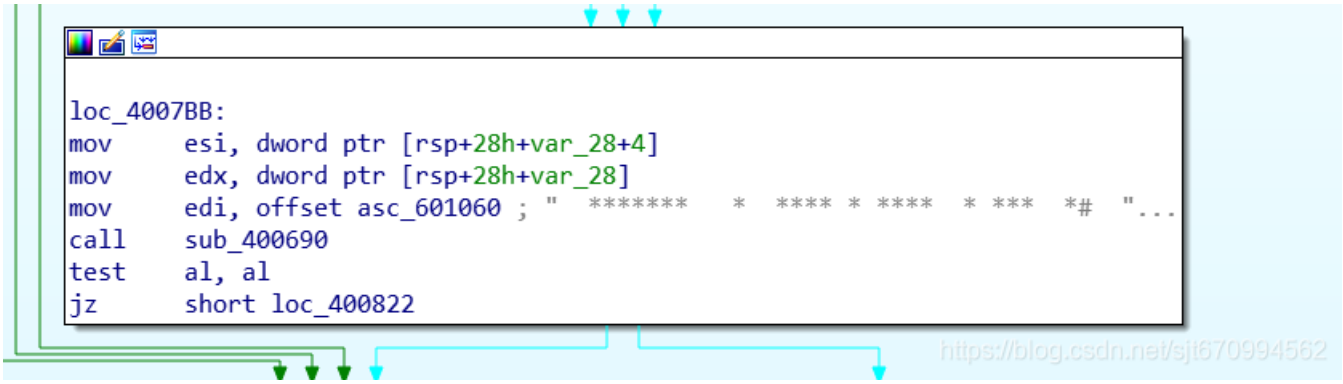
```
scanf("%s", &s1, 0LL);
```

```
if ( strlen(&s1) != 24 || (v3 = "nctf{", strncmp(&s1, "nctf{", 5uLL)) || *(&byte_6010BF + 24) != '}' )
```

这时候，你会发现，有一段代码比较类似，是平行出现的，这就是在迷宫里的上下左右的四个动作

```
  v4 = (unsigned __int8)v4;
  if ( (unsigned __int8)v4 == '0' )
  {
    v6 = sub_400650((_DWORD *)&v9 + 1);
    goto LABEL_14;
  }
  if ( v4 == 'o' )
  {
    v6 = sub_400660((int *)&v9 + 1);
    goto LABEL_14;
  }
}
else
{
  v4 = (unsigned __int8)v4;
  if ( (unsigned __int8)v4 == '.' )
  {
    v6 = sub_400670(&v9);
    goto LABEL_14;
  }
  if ( v4 == '0' )
  {
    v6 = sub_400680((int *)&v9);
_14:
    v5 = v6;
    goto LABEL_15;
  }
```


四个函数sub_400650\sub_400660\sub_400670\sub_400680是+1或-1的动作
v9是一个二维数组，用来控制上下和左右这两种方向的。
两种可能：1.v9代表横向、&v9+1代表竖向2.v9代表竖向、&v9+1代表横向
假设v9代表竖向，则"."上"0"下"o"右"O"左
根据流程图找到要求的迷宫



根据call sub_400690得出迷宫分布规律是8个字符一行

```
movsxd  rcx, edx
movzx   eax, byte ptr [rax+rcx*8]
cmp     eax, ' '
```

然后就是本人的方法了
在Excel里面画出来，简单易懂

根据道路得出代码为：

o0oo00O000oooo…OO

加上外壳flag便得到了：

nctf{o0oo00O000oooo…OO}