StrangeLanguage WriteUp（第三届“第五空间”网络安全大赛 reverse wp）

原创

drawlone 于 2021-09-18 11:52:07 发布 972 收藏

文章标签： 网络安全 python wp

本文链接：https://blog.csdn.net/Drawlonely/article/details/120364514

StrangeLanguage WriteUp

题目给了`main.exe`，看大小和图标判断是使用pyinstaller将**py**文件打包成的可执行文件，简单运行，让你输入（猜测输入flag），然后会有提示"nonono"

```
(base) PS F:\ctf\5space\StrangeLanguage\pyinstxtractor> .\main.exe
aaaaa
nonono
```

提取py

使用工具PyInstaller Extractor可直接提出**pyc**，再使用Uncompyle6反编译就能得到`main.py`

```
python .\pyinstxtractor.py main.exe
cd .\main.exe_extracted\
uncompyle6 main.pyc > main.py
```

```
# main.py
import brainfuck
brainfuck.main_check()
```

可以看到`main.py`里只是调用brainfuck模块，在提取的文件中里刚好有个brainfuck模块**brainfuck.cp38-win_amd64.pyd**，因此程序的主要逻辑大概率在该模块中

pyd逆向

.pyd是一种python动态模块，本质上是**dll**文件，故拿出ida分析一波，直接从入口点分析难度太大，先看Strings Window，如果了解过brainfuck(bf)代码形式，很容易就能发现有一串bf，idc脚本将其搞出来，先看看这串代码干什么用的
使用brainfuck解释器运行这串代码发现运行结果和`main.exe`的运行结果完全一致，遂猜测这就是程序的主要逻辑，bf模块只是个解释器，负责解释**bf代码**，到这里调用逻辑很清楚了`main->bf.pyd->bf代码`，因此只要弄清楚**bf代码**实现的功能就可以拿到flag

bf代码逆向

这部分是整个逆向过程中最难的了，因为bf代码的基本操作非常简单，相当于最原始的图灵机，而该bf代码使用基本运算实现了复杂操作，鉴于并没有很好bf逆向工具，因此只能硬来，这里主要使用*大胆猜测*和*动态调试*拿到最终flag
先使用工具bf2any将bf代码转成py，分别查找下flag{}这几个字符串的ascii码，发现都存在，似乎胜利就在眼前。
然后可以看到有两处输出"nonono"的地方，*大胆猜测*第一处是判断输入长度，第二处应该是和已知的串比较。
*动态调试*看下输入长度，我是在字符串`{`附近下的断点，可以发现有一处赋值比较可疑

```
317     while m[p+44] :  # 判断长度
318         m[p+45] = 0
319         m[p+44] = 37
```

```
        while m[p+44] :
```

大胆猜测 `37-len('flag{')=32` 就是flag长度，加flag{}共38个字符，尝试输入此种形式发现程序不会再进入第一处"nonono"，猜测正确

接着寻找可疑的地方（就在第一处"nonono"后边），发现一串赋值语句，赋值的都是比较特殊的值，而非0或1

```
    while m[p+44] :
        m[p+41] = 1
        m[p+40] = 5
        m[p+45] = 83 # S
        m[p+46] = 15
        m[p+47] = 90
        m[p+48] = 84
        m[p+49] = 80
        m[p+50] = 85
        m[p+51] = 3
```


拿出这32个值[83, 15, 90, 84, 80, 85, 3, 2, 0, 7, 86, 7, 7, 91, 9, 0, 80, 5, 2, 3, 93, 92, 80, 81, 82, 84, 90, 95, 2, 87, 7, 52, 5]

根据经验*大胆猜测*输入经过某种运算后只要等于这里的值就可以了，既然要运算，运算后的值必然再内存中，且前缀包含 `flag{` ，于是开启对内存的监视，或者在第二处"nonono"输出一下m的部分值(我这里打印的m[200:400])。

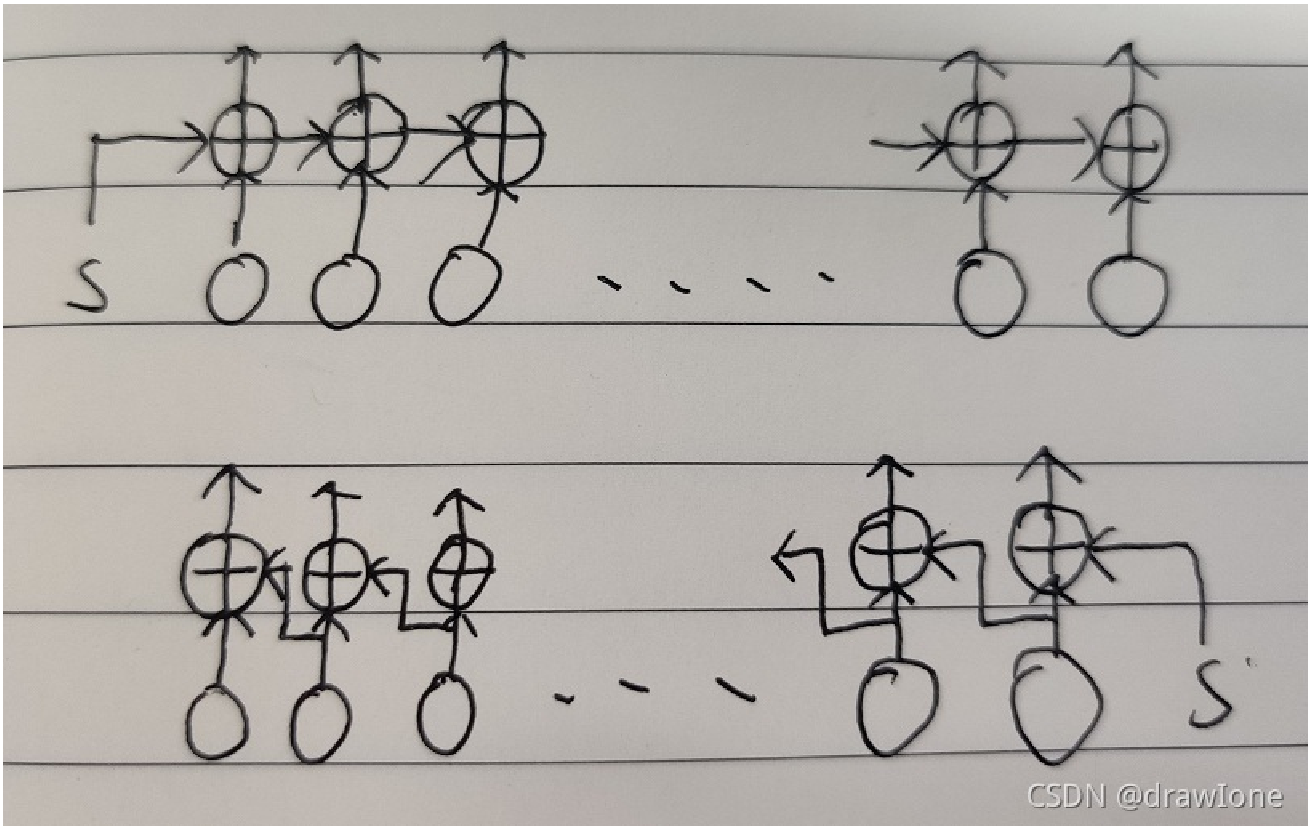
先观察下输入输出

输入flag{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}

输出flag{0000000000000000000000000000000a}

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 38, 102, 108, 97, 103, 123, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 97, 125, 32, 0, 1, 0, 1, 83, 15, 90, 84, 80, 85, 3, 2, 0, 7, 86, 7, 7, 91, 9, 0, 80, 5, 2, 3,
93, 92, 80, 81, 82, 84, 90, 95, 2, 87, 7, 52, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

这显然是**异或**后的结果，可还原的异或结构有好几种，我这里画出两种



s代表某个已知值，经过试验和*大胆猜测*这里的s=0，使用的是图中下面的结构进行计算，到此就可以写解密脚本了，附上flag获取脚本

~~求脚本~~

```
# 解密脚本
res = [83, 15, 90, 84, 80, 85, 3, 2, 0, 7, 86, 7, 7, 91, 9, 0, 80, 5, 2, 3, 93, 92, 80, 81, 82, 84, 90, 95, 2, 8
7, 7, 52, 0]
res.reverse()
flag = ""

for i in range(0, len(res)-1):
 a = res[i]
 b = res[i+1]
 x = a^b
 res[i+1] = x
 flag += chr(x)

test = flag[::-1]
print(test)
```

附上**bf代码**的运算逻辑

```
# bf计算逻辑
test  =  '12345678901234567890123456789012'

out  =  ''
for  i  in  range(0, len(test)-1):
 x  =  ord(test[i])^ord(test[i+1])
 print(x, end=',')
print(ord(test[-1])^0)
```