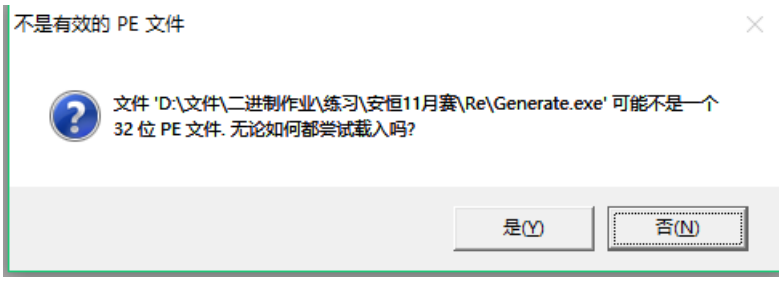
安恒11月赛逆向Generate

原创

一梦不醒 于 2018-11-30 06:33:34 发布 193 收藏

一、运行程序：

程序不能动态调试：



载入ida：

程序是让用户输入一个大整数，然后该整数和程序中数组404020的每一位异或，在异或v5，v5初始为0，后面v5^=byte404020每一位，输入的整数右移一位，再进入下一次循环，满足条件则存入数组408040，

is_begin_with函数测试是否以FLAG{开头：start:内容入下图：

最后等于“}”则输出flag。所以我们可以爆破，但最好的方法是Z3约束求解。

二、了解Z3模块

Z3是什么？Z3由微软开发的一套约束求解器，你可以简单的理解它是解方程的神器。在CTF题目中，我们经常遇到一些给定的条件，或是算法难以逆向、或是涉及到未知的数学技巧又或是爆破时间过长，而在我们学会使用z3后，一类问题便迎刃而解了。想了解更多关于z3知识的，这里有篇专栏：点我
Z3仅支持python2

一个简单的例子给大家介绍一下z3如何使用：

```
>>> from z3 import *
>>> x = Int('x')
>>> y = Int('y')
>>> solve(x+y==4)
[y = 0, x = 4]
```

它为我们提供了一个关于x+y==4的解，可是如果我们想要x=3呢？

```
>>> solve(x==3,x+y==4)
[y = 1, x = 3]
```

当然了，z3能做的肯定不止这么简单的运算，例如：

```
>>> from z3 import *
>>> x = Real('x')
>>> y = Real('y')
>>> solve(x**2 + y**2 == 3, x**3 == 2)
[x = 1.2599210498?, y = -1.1885280594?]
```

OK，大概了解到它是干嘛的我们就开始看一道例题吧。二进制文件可以在这里下载。

这是whctf的一道逆向题，它的核心代码如下：

```
v1 = 0;
gets(flag);
for ( i = 0; i <= 35; ++i )
{
  if ( !flag[i] )
  {
    flag[i] = 1;
    ++v1;
  }
}
if ( v1 != 9 )
  exit(0);
convert(a);
Transposition(a);
Multi(a, b);
for ( j = 0; j <= 5; ++j )
{
  for ( k = 0; k <= 5; ++k )
  {
    if ( c[0][k + 6 * j] != d[0][k + 6 * j] )
      exit(0);
  }
}
printf("congratulations!you have gottern the flag!");
```

其中convert(a)是将flag赋值给a，你可以把a当做一个6*6的矩阵。

Transposition(a)是把a的转置矩阵赋值给b

Multi(a,b)是把a和b的乘积赋值给c

而d就是堆中正确的flag经过上述运算后的结果，也就是说，如果用简单的思路去做，就是想办法爆破27位的flag添加9位1到尾部，然后经过运算结果为d中的值。但未知位数已经达到了20个，常规的爆破思路很难解决，网上的一篇writeup是经过一系列数学运算后逐行爆破，但每行依旧要消耗近10分钟的时间。在实际比赛的过程中，时间始终是最宝贵的，况且如果你对线性代数不太理解，可能会有一些棘手。

首先数学知识当然是必要的，我们应该保持着一个敬畏之心去学习这里的数学原理，但为了节省时间，或许用约束器去做会有意想不到的效果。

以下是我的脚本，注释的很详细就不多说了：

```
#coding:utf-8

'''

    @DateTime:     2017-11-28 10:19:29

    @Version:     1.0

    @Author:     Unname_Bao

'''

from z3 import *

import time

t1 = time.time()

#创建一个解决方案实例

solver = Solver()

#flag长度先设置为36，包括尾部的9个1

flag = [Int('flag%d'%i) for i in range(36)]

#保存flag的矩阵

a = [i for i in flag]

#保存flag的转置矩阵

b = [i for i in range(36)]

#保存a*b的矩阵

c = [0 for i in range(36)]

#堆中正确flag的运算结果
```

```
d = [0x12027,0x0F296,0x0BF0E,0x0D84C,0x91D8,0x297,
    0x0F296,0x0D830,0x0A326,0x0B010,0x7627,0x230,
    0x0BF0E,0x0A326,0x8FEB,0x879D,0x70C3,0x1BD,
    0x0D84C,0x0B010,0x879D,0x0B00D,0x6E4F,0x1F7,
    0x91D8,0x7627,0x70C3,0x6E4F,0x9BDC,0x15C,
    0x297,0x230,0x1BD,0x1F7,0x15C,0x6]
#获得a的转置矩阵
for i in range(6):
    for j in range(6):
        b[i+6*j] = a[6*i+j]
#运算a*b
for i in range(6):
    for j in range(6):
        for k in range(6):
            c[j+6*i] = c[j+6*i] + a[6*i+k]*b[6*k+j]
        #添加约束，正确flag的运算结果
        solver.add(simplify(c[j+6*i]) == d[j+6*i])
#添加约束，除了尾部，flag的字符一定在可见字符范围内
for i in range(6,36-10):
    solver.add(flag[i]>=32)
    solver.add(flag[i]<=127)
#添加约束，由于flag有格式，前6位一定为whctf{
for i in range(6):
    solver.add(flag[i] == ord('whctf{'[i]))
#添加约束，flag的尾部为9个1
for i in range(36-9,36):
    solver.add(flag[i] == 0x1)
#添加约束，flag的最后一个肯定是}
solver.add(flag[-10] == ord('}'))
#这里一定要有，不check的话会报错
```

```
if solver.check() == sat:

    m = solver.model()

    s = []

    #获得结果

    for i in range(36):

        s.append(m[flag[i]].as_long())

    #输出flag

    print(bytes(s))

else:

    print('error')

t2 = time.time()

print(t2-t1)
```

这是最终的运行结果：

```
D:\2017_WEB_Test\ulb_manager\backend\spider>python z3test.py

b'whctf{Y0u_ar3_g00d_a7_m4th}\x01\x01\x01\x01\x01\x01\x01\x01\x01'

4.042840003967285
```

是的，仅仅用了4s就跑出了最终结果、可见z3约束器的强大！

更多信息请看z3的官方GitHub：点我
三、题解脚本
from z3 import *

```
def gen():
    print ('[*]Computing key...')
    a = [0xA4, 0x19, 0x04, 0x82, 0x7E, 0x85, 0x50, 0xA8, 0xD1, 0xEA,
         0xE3, 0xF9, 0xE8, 0xE1, 0x60, 0x3A, 0x1A, 0x0A, 0x87, 0xDD,
         0xE1, 0x61, 0xA0, 0xC0, 0x60, 0xA4, 0x48, 0x28, 0x16, 0x0B,
         0x05, 0x20]
    num = BitVec('x', 64)#num范围不会超过64位

    s = Solver()#定义运算器
    s.add(num >= 2 ** 31)#添加约束条件
    s.add(num < 2 ** 32)
    num2 = 0
    b = 0
    for i in range(32):
        if i < 5:
            s.add((a[num2] ^ (num & 0xff) ^ b) & 0xff == ord("FLAG{"[i]))#约束条件前几位为固定数
        elif 5 <= i < 31:
            s.add(Or(                                                   #Or
                And((a[num2] ^ (num & 0xff) ^ b) & 0xff <= ord("Z"),#范围在“@”“Z”之间 And
                    (a[num2] ^ (num & 0xff) ^ b) & 0xff >= ord("A")),
                (a[num2] ^ (num & 0xff) ^ b) & 0xff == ord("_")))#异或之后为“_”
        else:
            s.add((a[num2] ^ (num & 0xff) ^ b) & 0xff == ord("}"))

        b ^= a[num2]
        b &= 0xff
        num2 += 1
        num >>= 1
    if s.check() == sat:#检测是否有解 有解为sat
        a = s.model()   #将结果赋给a
        print (a)
        return
    print ('[*]No Result ,end')


gen()
```

得到结果：

```
C:\Users\yr1>D:\文件\二进制作业\练习\安恒11月赛\Re\Generate.exe
try to enter a number to generate the flag
3658134498
Congratulations!
flag is:
FLAG{__ZZLOZEZ_Z__AAPHTZIZ____}
C:\Users\yr1>
```