20180CTF g0g0g0 writeup

原创

Flying_Fatty 于 2018-04-24 23:22:47 发布 1827 收藏
分类专栏： CTF之旅 reverse

一道很好的题目，从逆向到数学密码到工具

贴几个writeup吧，比较难找：

https://www.jianshu.com/p/d906619a01b7

https://github.com/xiaohuajiao/my-ctf/tree/master/2018/0ctf/g0g0g0

http://westerns.tokyo/writeups/0ctf2018quals.html#g0g0g0

上面的链接，把solve.py改成solve.sage，linux下装个sage，直接跑就可以得到结果

当得到了数学表达式后，可以从这个网站上直接得到结果：

https://math.stackexchange.com/questions/402537/find-integer-in-the-form-fracabc-fracbca-fraccab/409450

首先是go语言的逆向

重点是如何得到数据的处理流程以及函数的逻辑

```
Entering main.main at /tmp/gogo.go:172:6.
.0:
  t0 = new string (sa)
  t1 = new string (sb)
  t2 = new string (sc)
  t3 = new [1]interface{} (varargs)
  t4 = &t3[0:int]
  t5 = make interface{} <- string ("Input 3 numbers":string)
  *t4 = t5
  t6 = slice t3[:]
  t7 = fmt.Println(t6...)


  t8 = new [1]interface{} (varargs)
  t9 = &t8[0:int]
```

```
  t10 = make interface{} <- *string (t0)
  *t9 = t10
  t11 = slice t8[:]
  t12 = fmt.Scanf("%s":string, t11...)

  t13 = new [1]interface{} (varargs)
  t14 = &t13[0:int]
  t15 = make interface{} <- *string (t1)
  *t14 = t15
  t16 = slice t13[:]
  t17 = fmt.Scanf("%s":string, t16...)

  t18 = new [1]interface{} (varargs)
  t19 = &t18[0:int]
  t20 = make interface{} <- *string (t2)
  *t19 = t20
  t21 = slice t18[:]
  t22 = fmt.Scanf("%s":string, t21...)


  t23 = *t0
  t24 = func6(t23)

  t25 = *t1
  t26 = func6(t25)

  t27 = *t2
  t28 = func6(t27)


  t29 = len(t24)
  t30 = t29 == 0:int
  if t30 goto 1 else 4
.4:
  t43 = len(t26)
  t44 = t43 == 0:int
  if t44 goto 1 else 3
.3:
  t41 = len(t28)
  t42 = t41 == 0:int
  if t42 goto 1 else 2
.2:
  t36 = new [1]int (slicelit)
  t37 = &t36[0:int]
  *t37 = 0:int
  t38 = slice t36[:]
  t39 = func1(t24, t38)


  t40 = t39 <= 0:int
  if t40 goto 5 else 8
.8:
  t74 = new [1]int (slicelit)
  t75 = &t74[0:int]
  *t75 = 0:int
  t76 = slice t74[:]
  t77 = func1(t26, t76)
```

```
  t78 = t77 <= 0:int
  if t78 goto 5 else 7
.7:
  t69 = new [1]int (slicelit)
  t70 = &t69[0:int]
  *t70 = 0:int
  t71 = slice t69[:]
  t72 = func1(t28, t71)


  t73 = t72 <= 0:int
  if t73 goto 5 else 6
.6:
  t50 = func2(t24, t26)
  t51 = func2(t24, t28)
  t52 = func2(t26, t28)
  t53 = func4(t50, t51)
  t54 = func4(t53, t24)
  t55 = func4(t50, t52)
  t56 = func4(t55, t26)
  t57 = func4(t51, t52)
  t58 = func4(t57, t28)
  t59 = func2(t56, t58)
  t60 = func2(t54, t59)
  t61 = new [1]int (slicelit)
  t62 = &t61[0:int]
  *t62 = 10:int
  t63 = slice t61[:]
  t64 = func4(t51, t52)
  t65 = func4(t50, t64)
  t66 = func4(t63, t65)
  t67 = func1(t60, t66)
  t68 = t67 == 0:int
   if t68 goto 9 else 11

.11:
  t84 = new [1]interface{} (varargs)
  t85 = &t84[0:int]
  t86 = make interface{} <- string ("Wrong! Try again!!":string)
  *t85 = t86
  t87 = slice t84[:]
  t88 = fmt.Println(t87...)

  jump 10
.10:
  return
Leaving main.main.
```

下面这个.11的函数块是判断某个数位是否大于等于10的：

```
.11:
  t33 = &t3[t31]
  t34 = *t33
  t35 = t34 >= 10:int
  if t35 goto 13 else 10
```

下面这个.5的函数块是得到某个字符串的长度的

```
.5:
  t9 = len(t3)
  t10 = t9 - 1:int
  t11 = slice t3[:t10]
  t12 = len(t11)
  jump 10
```

t31是当前位置，t36是t31之后的一个位置，t37和t38是取值，看到/10和%10，结合这个写法，可以猜测是大整数的进位

```
.13:
  t36 = t31 + 1:int
  t37 = &t3[t36]
  t38 = &t3[t31]
  t39 = *t38
  t40 = t39 / 10:int
  t41 = *t37
  t42 = t41 + t40
  *t37 = t42
  t43 = &t3[t31]
  t44 = *t43
  t45 = t44 % 10:int
  *t43 = t45
  jump 10
```

在程序里看到了这个，但是发现，t12，t17，t22在最后的验证过程中并没有使用到

```
16103 :   t12 = fmt.Scanf("%s":string, t11...)
19526 :   t17 = fmt.Scanf("%s":string, t16...)
21608 :   t22 = fmt.Scanf("%s":string, t21...)
```

可以写一个很简单的小工具，对main中的所有函数调用截取出来分析：

```
numberid = 0

f = open('trace.log')
for lines in f:
 numberid += 1
 if 'func1(' in lines:
  print numberid,':',lines
```

得到这些数据：

```
29452 :   t2 = func0(t0, t1)
24374 :   t39 = func1(t24, t38)
24430 :   t77 = func1(t26, t76)
24486 :   t72 = func1(t28, t71)
32573 :   t67 = func1(t60, t66)
24538 :   t50 = func2(t24, t26)
24722 :   t51 = func2(t24, t28)
24907 :   t52 = func2(t26, t28)
29045 :   t59 = func2(t56, t58)
29447 :   t60 = func2(t54, t59)
25026 :   t53 = func4(t50, t51)
25822 :   t54 = func4(t53, t24)
26991 :   t55 = func4(t50, t52)
27497 :   t56 = func4(t55, t26)
27908 :   t57 = func4(t51, t52)
28442 :   t58 = func4(t57, t28)
29997 :   t64 = func4(t51, t52)
30531 :   t65 = func4(t50, t64)
31741 :   t66 = func4(t63, t65)
24133 :   t24 = func6(t23)
24250 :   t26 = func6(t25)
24292 :   t28 = func6(t27)
```

根据数值的传递关系，t23，t25，t27，t63为四个未知数，func6函数块应该为大数的初始化

```
Leaving main.func1, resuming main.main at /tmp/gogo.go:236:13.
  t68 = t67 == 0:int
  if t68 goto 9 else 11
.11:
  t84 = new [1]interface{} (varargs)
  t85 = &t84[0:int]
  t86 = make interface{} <- string ("Wrong! Try again!!":string)
  *t85 = t86
  t87 = slice t84[:]
  t88 = fmt.Println(t87...)
```

这一段说明，t67应该等于0，那么就是func1(t60，t66)=0，容易猜测是比较函数

分析某一个func2函数块：

```
Leaving main.func0, resuming main.func2 at /tmp/gogo.go:52:18.
  t3 = t2 + 1:int
  t4 = make []int t3 t3
  t5 = len(t4)
  jump 1
.1:
  t6 = phi [0: 0:int, 9: t22] #carry
  t7 = phi [0: -1:int, 9: t8]
  t8 = t7 + 1:int
  t9 = t8 < t5
  if t9 goto 2 else 3
.2:
  t10 = len(a)
  t11 = t8 < t10
  if t11 goto 4 else 5
.4:
  t12 = &a[t8]
  t13 = *t12
  jump 5
.5:
  t14 = phi [2: 0:int, 4: t13] #a_i
  t15 = len(b)
  t16 = t8 < t15
  if t16 goto 6 else 7
.6:
  t17 = &b[t8]
  t18 = *t17
  jump 7
.7:
  t19 = phi [5: 0:int, 6: t18] #b_i
  t20 = t14 + t19
  t21 = t20 + t6
  t22 = t21 / 10:int
  t23 = t21 >= 10:int
  if t23 goto 8 else 9
.8:
  t24 = t21 % 10:int
  jump 9
.9:
  t25 = phi [7: t21, 8: t24] #tmp
  t26 = &t4[t8]
  *t26 = t25
  jump 1
```

会发现之后都是重复的东西，重点关注的应该是.7这个小部分，明显是加法！

```
Entering main.func4 at /tmp/gogo.go:104:6.
.0:
  t0 = len(a)
  t1 = len(b)
  t2 = t0 + t1
  t3 = make []int t2 t2
  t4 = len(t3)
  jump 1
.1:
  t5 = phi [0: -1:int, 2: t6]
  t6 = t5 + 1:int
  t7 = t6 < t4
  if t7 goto 2 else 3
```

这里的.0已经知道了，是大整数乘法
所以，我们知道了func2是大整数加法，func1是大整数比较，func4是大整数乘法

然后得到表达式：

a/(b+c)+b/(c+a)+c/(a+b)=10

这里可以参考知乎：史上最贱的数学题，看上去简单的式子其实本质上是个椭圆曲线

题目总结：

把写程序想象成一棵代码树，编译器的地方是树根，会先找到main，然后main作为子树的树根，继续调用其它函数

这个题的逆向思路：把树前序遍历一下，就可以理解了，就是程序往哪里走，下一条指令就是什么。把树状的框架变成了线性的