逆向 Writeup：西湖论剑2020逆向第一题Cellular

原创

Niatruc 于 2020-10-14 21:24:54 发布 571 收藏 1

分类专栏： 逆向 CTF 文章标签： 信息安全 经验分享 反汇编

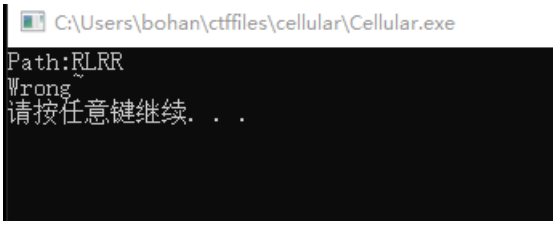
本文链接：https://blog.csdn.net/Niatruc/article/details/109082173

摘要

1. 题目为“蜜蜂如何筑巢”，问题的实质是`寻找路径`。应该将程序视为黑盒。刚开始解题思路不对，一直试图理清代码中的逻辑。
2. 解答的方法是尝试`将各种由‘L’和‘R'组合的长为25的字符串作为输入`，直到打印信息“Congratulations”。可采用遍历法，或逐步试错法。本篇采用后者。
3. 本篇方法概括：
   3.1 在OD中修改源码，改变关键函数CheckFlag的返回值为其中的大循环的循环次数（通过该次数可得知当前前行方向正确与否，从而调整路径方向）；修改打印部分代码，使其打印的信息能反映刚刚说的循环次数。
   3.2 写python脚本，生成输入字符串，借由其subprocess模块将字符串传给Cellular程序进行判断。通过逐步试错找到正确路径。
4. 如果有好的测试工具的话也许就不用自己写脚本这么麻烦了。但目前没找着。

题目概括

程序界面如下。需要输入’L’和’R’组成的字符串。



使用IDA解析，其主函数如下图。

关键函数CheckFlag的逻辑如下图。

在用OD调试时看到其中有类似链表的数据结构。

对于CheckFlag函数其实只要大概明白其意思即可，无需深究。其中for循环的i值会在解题时用到。

解题过程

修改原始程序

将程序拖入OD，找到CheckFlag函数结尾返回部分，修改其返回值为for循环的i值。

修改数据段，添加27个可打印的字符。

修改主函数中打印”Congratulations“信息的代码，使用CheckFlag返回的数值作为索引，从上一步添加的数据中按索引找一个字符并打印。这个字符将在下一步起判断作用。

编写搜索正确字符串的python脚本

```
# 定义一个函数,用于运行Cellular程序、接受输入、获取程序在标准输出流打印的信息.
def test_path(ans):
    path = "C:/Users/bohan/ctffiles/cellular/Cellular_pad2.exe"
    cellular = subprocess.Popen([path], stdin=subprocess.PIPE, stdout=subprocess.PIPE)
    output = cellular.communicate(ans.encode())[0]
    correct_len = output[output.index(b'Path') + 5] - ord('a')
    return correct_len
```

用于搜索路径的算法如下:

```
arr_len = 25
bit_arr = [0 for _ in range(arr_len)]
bit_arr
cur_index = 0
def select_path(path_is_adapted):
 '''path_is_adapted为True时则继续前进，为False则需改当前cur_index所指处的方向'''
    global cur_index
    if cur_index < arr_len and path_is_adapted:
        cur_index += 1
    elif cur_index > -1 and not path_is_adapted:
        bit_arr[cur_index] += 1  # 更改方向
        while bit_arr[cur_index] > 1 and cur_index > -1:  # 如果左右两个方向都试过了，则要向前移动，改前面的方向
#             print("cur_index ", cur_index)
            bit_arr[cur_index] = 0
            cur_index -= 1  # 向前移动指针
            bit_arr[cur_index] += 1  # 然后更改方向

    return cur_index

def get_cur_path():
    arr = bit_arr[0:cur_index + 1]
    lr = ['L', 'R']
    path_str = ''.join([lr[i] for i in arr])
    return arr, path_str
```

测试部分:

```
correct_len = 0
while correct_len != -49:  # correct_len为-49（字符'a'和字符'0'的ascii码值相减所得）时表明打印了字符零，也即找到了正确路径
    bit_path, path_str = get_cur_path()
    print(bit_path, path_str)
    try:
        correct_len = test_path(path_str)
        print(correct_len)
        if correct_len == len(path_str) and correct_len != 25:
            print("T")
            select_path(True)
        else:
            print("F")
            select_path(False)
    except:
        print("发生异常")
        select_path(False)

    if not path_str:
        print("无路径")
        break
```

运行后打印的情况如下（下图为打印结果的最后几行）：
从上图知最终路径为：'RLRLLRLLLRRLLRLLRLRLLRLLR'。当然最后还得用md5计算其散列值，才得到最终flag。

附一个搜索过程中列出的所有路径，方便理解上述路径搜索算法：

```
LL
LR
R
RL
RLL
RLLL
RLLR
RLLRL
RLLRLL
RLLRLLL
RLLRLLR
RLLRLLRL
RLLRLLRR
RLLRLLRRL
RLLRLLRRR
RLLRLR
RLLRR
RLLRRL
RLLRRLL
RLLRRLLL
RLLRRLLLL
RLLRRLLLLL
RLLRRLLLLR
RLLRRLLLLRL
RLLRRLLLLRR
RLLRRLLLLRRL
RLLRRLLLLRRR
RLLRRLLLR
RLLRRLLR
RLLRRLLRL
RLLRRLLRR
RLLRRLLRRL
```

RLLRRLLRRR
RLLRRLR
RLLRRR
RLLRRRL
RLLRRRLL
RLLRRRLLL
RLLRRRLLLL
RLLRRRLLLLL
RLLRRRLLLLLL
RLLRRRLLLLLR
RLLRRRLLLLLRL
RLLRRRLLLLLRR
RLLRRRLLLLLRRL
RLLRRRLLLLLRRR
RLLRRRLLLLR
RLLRRRLLLR
RLLRRRLLLRL
RLLRRRLLLRR
RLLRRRLLLRRL
RLLRRRLLLRRR
RLLRRRLLR
RLLRRRLR
RLLRRRR
RLR
RLRL
RLRLL
RLRLLL
RLRLLLL
RLRLLLLL
RLRLLLLR
RLRLLLLRL
RLRLLLLRLL
RLRLLLLRLR
RLRLLLLRLRL
RLRLLLLRLRR
RLRLLLLRLRRL
RLRLLLLRLRRR
RLRLLLLRR
RLRLLLLRRL
RLRLLLLRRLL
RLRLLLLRRLR
RLRLLLLRRR
RLRLLLLRRRL
RLRLLLLRRRLL
RLRLLLLRRRLLL
RLRLLLLRRRLLR
RLRLLLLRRRLR
RLRLLLLRRRR
RLRLLLR
RLRLLLRL
RLRLLLRLL
RLRLLLRLR
RLRLLLRLRL
RLRLLLRLRR
RLRLLLRLRRL
RLRLLLRLRRR
RLRLLLRR
RLRLLLRRL
RLRLLLRRLL
RLRLLLRRLR

RLRLLLRRR
RLRLLLRRRL
RLRLLLRRRLL
RLRLLLRRRLLL
RLRLLLRRRLLR
RLRLLLRRRLR
RLRLLLRRRR
RLRLLR
RLRLLRL
RLRLLRLL
RLRLLRLLL
RLRLLRLLLL
RLRLLRLLLLL
RLRLLRLLLLLL
RLRLLRLLLLLR
RLRLLRLLLLLRL
RLRLLRLLLLLRR
RLRLLRLLLLLRRL
RLRLLRLLLLLRRLL
RLRLLRLLLLLRRLLL
RLRLLRLLLLLRRLLR
RLRLLRLLLLLRRLLRL
RLRLLRLLLLLRRLLRLL
RLRLLRLLLLLRRLLRLLL
RLRLLRLLLLLRRLLRLLR
RLRLLRLLLLLRRLLRLLRL
RLRLLRLLLLLRRLLRLLRLL
RLRLLRLLLLLRRLLRLLRLR
RLRLLRLLLLLRRLLRLLRLRL
RLRLLRLLLLLRRLLRLLRLRLL
RLRLLRLLLLLRRLLRLLRLRLLL
RLRLLRLLLLLRRLLRLLRLRLLR
RLRLLRLLLLLRRLLRLLRLRLLRL
RLRLLRLLLLLRRLLRLLRLRLLRR
RLRLLRLLLLLRRLLRLLRLRLR
RLRLLRLLLLLRRLLRLLRLRR
RLRLLRLLLLLRRLLRLLRR
RLRLLRLLLLLRRLLRLR
RLRLLRLLLLLRRLLRR
RLRLLRLLLLR
RLRLLRLLLLRL
RLRLLRLLLLRR
RLRLLRLLLLRRL
RLRLLRLLLLRRLL
RLRLLRLLLLRRLLL
RLRLLRLLLLRRLLR
RLRLLRLLLLRRLLRL
RLRLLRLLLLRRLLRLL
RLRLLRLLLLRRLLRLLL
RLRLLRLLLLRRLLRLLR
RLRLLRLLLLRRLLRLLRL
RLRLLRLLLLRRLLRLLRLL
RLRLLRLLLLRRLLRLLRLR
RLRLLRLLLLRRLLRLLRLRL
RLRLLRLLLLRRLLRLLRLRLL
RLRLLRLLLLRRLLRLLRLRLLL
RLRLLRLLLLRRLLRLLRLRLLR

RLRLLRLLLLRRLLRLLRLRLLRL
RLRLLRLLLLRRLLRLLRLRLLRLL
RLRLLRLLLLRRLLRLLRLRLLRLR
RLRLLRLLLLRRLLRLLRLRLLRRR
RLRLLRLLLLRRLLRLLRLRLR
RLRLLRLLLLRRLLRLLRLRR
RLRLLRLLLLRRLLRLLRR
RLRLLRLLLLRRLLRLR
RLRLLRLLLLRRLLRR
RLRLLRLLLLRRLR
RLRLLRLLLLRRR
RLRLLRLLLR
RLRLLRLLLRL
RLRLLRLLLRR
RLRLLRLLLRRL
RLRLLRLLLRRLL
RLRLLRLLLRRLLL
RLRLLRLLLRRLLR
RLRLLRLLLRRLLRL
RLRLLRLLLRRLLRLL
RLRLLRLLLRRLLRLLL
RLRLLRLLLRRLLRLLR
RLRLLRLLLRRLLRLLRL
RLRLLRLLLRRLLRLLRLL
RLRLLRLLLRRLLRLLRLR
RLRLLRLLLRRLLRLLRLRL
RLRLLRLLLRRLLRLLRLRLL
RLRLLRLLLRRLLRLLRLRLLL
RLRLLRLLLRRLLRLLRLRLLR
RLRLLRLLLRRLLRLLRLRLLRL
RLRLLRLLLRRLLRLLRLRLLRLL
RLRLLRLLLRRLLRLLRLRLLRLLL
RLRLLRLLLRRLLRLLRLRLLRLLR