MazeGarden 出题人WriteUp by 圆周率

选手视角

处理花指令

拿到可执行文件，先放到IDA里，发现程序加入了花指令干扰反编译，还有控制流平坦化，下面先处理花指令

简单手动patch几处后，发现花指令实在太多了，这里就需要我们写一个脚本去patch花指令，下面是花指令的几种类型：

- 直接跳转：jmp + junk
- 互补跳转：jz + jnz + junk
- 清零跳转：xor A, A + jz + junk
- 比较跳转：cmp A, A + jz + junk
- 重叠跳转：jmp short near ptr $+1 (EB FF)

其中直接跳转可以被IDA识别跳过，我们主要处理后面这几类：

```
import ida_bytes
import ida_ida
import ida_idaapi

def patch_jzjnz(addr):
    A = ida_bytes.get_byte(addr+1)
    if ida_bytes.get_byte(addr+3) != A-2:
        return
    for i in range(addr, addr+A+2):
        ida_bytes.patch_byte(i, 0x90)

RRMR = [0xC0, 0xC9, 0xD2, 0xDB, 0xE4, 0xED, 0xF6, 0xFF]

def patch_xorjz(addr):
    A = ida_bytes.get_byte(addr+3)
    if ida_bytes.get_byte(addr+1) not in RRMR:
        return
    for i in range(addr, addr+A+4):
        ida_bytes.patch_byte(i, 0x90)

def patch_cmpjz(addr):
    A = ida_bytes.get_byte(addr+3)
    if ida_bytes.get_byte(addr+1) not in RRMR:
        return
    for i in range(addr, addr+A+4):
        ida_bytes.patch_byte(i, 0x90)

def patch_EBFF(addr):
    if (ida_bytes.get_byte(addr+2) & 0xF0) == 0xC0:
        ida_bytes.patch_byte(addr, 0x90)
```

```
def patch(start, end=None):
    PATS = [
        ("74 ?? 75", patch_jzjnz),
        ("72 ?? 73", patch_jzjnz),
        ("70 ?? 71", patch_jzjnz),
        ("7C ?? 7D", patch_jzjnz),
        ("7F ?? 7E", patch_jzjnz),
        ("77 ?? 76", patch_jzjnz),
        ("78 ?? 79", patch_jzjnz),
        ("31 ?? 74", patch_xorjz),
        ("39 ?? 73", patch_xorjz),
        ("39 ?? 74", patch_xorjz),
        ("39 ?? 76", patch_xorjz),
        ("EB FF", patch_EBFF)
    ]

    def search(x):
        out = ida_bytes.compiled_binpat_vec_t()
        ida_bytes.parse_binpat_str(out, start, x[0], 16)
        return (ida_bytes.bin_search(start, end or ida_ida.inf_get_max_ea(), out,
ida_bytes.BIN_SEARCH_FORWARD), x[1])

    pats = list(filter(lambda x: x[0] != ida_idaapi.BADADDR, map(search, PATS)))
    if not pats:
        return
    (addr, _), func = min(pats, key=lambda x: x[0])
    if end is not None and addr > end:
        return
    func(addr)
    patch(addr+1, end)

patch(0x140001000, 0x140003F49)
patch(0x140004160, 0x14000F043)
```

在IDA中运行patch脚本之后就可以把所有花去除，对加花的两个函数全部执行Undefine（U），然后再在函数开头按P定义重新定义函数，此时F5功能就可以正常使用

如果你的IDA加载了hrtng插件，那么就会自动去平坦化，可以看到一个比较清晰的逻辑，但由于计算混淆的存在，还是有一部分运算不够清楚，可以尝试使用D810插件（较慢），可以恢复一部分，该插件也可以去平坦化

分析setup逻辑

这里有两个核心函数：main和sub_140004160，其中sub_140004160为初始化函数（下称setup函数），其中包含我们需要的地图，但是由于其逻辑较为复杂，我们可以考虑动调

先分析看看，其中大量执行了sub_1400040E0函数，其返回值为unsigned int，进入，发现该函数为一个LCG随机数生成器，初始seed为0x12345678，反馈为0xDEADBEEF * x + 0×CAFEBABE，然后该函数返回值进行一同运算后作为复制字符串的目的偏移，其中部分运算被D810正确复原，还有少量形式如下的运算：

```
((~((_BYTE)v6 - 25) & 0x29) * ((v6 - 1219732249) & 0xFFFFFFD6) + (((_BYTE)v6 - 25) & 0x29)
```

```
* ((v6 - 1219732249) | 0x29) + v1515)
```

其中v6为lcg返回值，v1515为存储字符串之处，交给D老师简化一下：

化简以下表达式：
((~((_BYTE)v6 - 25) & 0x29) * ((v6 - 1219732249) & 0xFFFFFFD6)
+ (((_BYTE)v6 - 25) & 0x29) * ((v6 - 1219732249) | 0x29)
+ v1515)

已思考（用时 115 秒）

化简结果为：

$$41 \times (v6 - 1219732249) + v1515$$

或写成：

$$0x29 \times (v6 - 1219732249) + v1515$$

因为：

- $0x29 = 41$
- $1219732249 = 0x48B3A319$，其低字节是 25
- $0xFFFFFFD6 = \sim 0x29$（32 位下）

所以原式可化简为：

c 复制 下载

```
41 * (v6 - 1219732249) + v1515
```

这与其他表达式形式一致，可以采信，所以我们只需要提取`v??? - [OFFSET]`即可还原为`41*(lcg()-[OFFSET]) + 字符串地址`，这一点可以通过设计Regex表达式，逐行对伪代码匹配实现恢复迷宫文本

由于过于复杂，下面采用动调的方式提取

首先，注意到导入表内含有`IsDebuggerPresent`函数，说明极有可能含有反调试代码，经过交叉应用，发现`setup`函数中含有一个根据该值的分支语句，接下来的策略是：

1. 将跳转指令NOP掉/设为强制跳转，或在该处下断点，根据实时反馈的跳转方向进行`Set IP`
2. 先正常运行程序，待程序运行至I/O时，初始化已完成，此时使用IDA attach程序进行调试

随后就可以提取得到迷宫

分析主逻辑

主逻辑的部分运算也被混淆了，可以通过D810还原

```
// The function seems has been flattened
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v3; // ebx
  int v4; // edi
  int v5; // esi
  __int64 v6; // r8
  int v7; // edi
  _BYTE v10[31]; // [rsp+3A0h] [rbp-488h] BYREF
  char n97; // [rsp+3BFh] [rbp-469h]
  _QWORD v12[2]; // [rsp+3C0h] [rbp-468h] BYREF
  __int64 v13; // [rsp+3D0h] [rbp-458h]
  __int64 v14; // [rsp+3D8h] [rbp-450h]
  __int64 v15; // [rsp+3E0h] [rbp-448h]
  __int64 v16; // [rsp+3E8h] [rbp-440h]
  __int64 v17; // [rsp+3F0h] [rbp-438h]
  _BYTE v18[1004]; // [rsp+3F8h] [rbp-430h] BYREF
  int v19; // [rsp+7E4h] [rbp-44h]

  v19 = 0;
  sub_140004160(v12);
  while ( v14 != v12[0] - 2LL || v15 != v14 || v16 != v12[1] - 1LL )
  {
    n97 = fgetchar();
    if ( n97 < 107 )
    {
      if ( n97 < 100 )
      {
        if ( n97 == 97 )
        {
          --v14;
          v6 = v12[0] & v15;
          if ( *(_BYTE *)(v13 + v12[0] * v15 + v12[0] * v16 * v12[0] + v14) == 35 )
          {
            ++v14;
          }
          else if ( v17 && v18[v17 - 1] == 100 )
          {
            --v17;
          }
          else
          {
            v18[v17++] = n97;
          }
        }
      }
      else if ( n97 < 105 )
      {
        if ( n97 == 100 )
        {
          --v3;
          ++v14;
          --v5;
          v6 = v12[0] * v15;
          if ( *(_BYTE *)(v13 + v14 + v12[0] * v15 + v12[0] * v12[0] * v16) == 35 )
          {
            --v14;
          }
          else
          {
            v7 = v4 + 1;
            if ( v17 && v18[v17 - 1] == 97 )
            {
              --v17;
            }
            else
            {
              v18[v17] = n97;
              v17 = 2 * (v17 & 1) + (v17 ^ 1);
            }
            v4 = v7 + 1;
          }
          --v4;
        }
      }
      else if ( n97 == 105 )
      {
        --v3;
        v6 = v12[0];
        if ( *(_BYTE *)(v13 + v14 + v12[0] * v16 * v12[0] + v12[0] * v15) == 94
          || (v6 = v12[0], *(_BYTE *)(v14 + v12[0] * v15 + v12[0] * v12[0] * v16 + v13) == 88) )
        {
          --v16;
          if ( v17 && v18[v17 - 1] == 107 )
            v17 = 2 * v17 + ~v17;
          else
            v18[v17++] = n97;
        }
      }
    }
    else if ( n97 < 115 )
    {
      if ( n97 == 107 )
      {
        v6 = v14;
        if ( *(_BYTE *)(v13 + v14 + v12[0] * v16 * v12[0] + v12[0] * v15) == 118
          || (v6 = v12[0], *(_BYTE *)(v13 + v15 * v12[0] + v12[0] * v16 * v12[0] + v14) == 88) )
        {
          v16 = (v16 | 1) + (v16 & 1);
          if ( v17 && v18[v17 - 1] == 105 )
            --v17;
          else
            v18[v17++] = n97;
        }
      }
    }
    else if ( n97 < 119 )
    {
      if ( n97 == 115 )
      {
        ++v15;
        ++v4;
        v6 = v12[0];
        if ( *(_BYTE *)(v13 + v14 + v12[0] * v16 * v12[0] + v12[0] * v15) == 35 )
        {
          --v15;
        }
        else if ( v17 && v18[v17 - 1] == 119 )
        {
          --v17;
        }
        else
        {
          v18[v17++] = n97;
        }
      }
    }
    else if ( n97 == 119 )
    {
      v6 = v12[0] & --v15;
      if ( *(_BYTE *)(v13 + v14 + v12[0] * v12[0] * v16 + v12[0] * v15) == 35 )
      {
        ++v15;
      }
```

```
    else if ( v17 && v18[v17 - 1] == 115 )
    {
      --v4;
      --v17;
    }
    else
    {
      v18[v17++] = n97;
    }
  }
  --v3;
}
v18[v17] = 0;
sub_140010B00(v18, v10, v6);
sub_140003F50(
  "Congratulations! The flag is: TSCTF-J{%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x}",
  v10[0],
  v10[1],
  v10[2],
  v10[3],
  v10[4],
  v10[5],
  v10[6],
  v10[7],
  v10[8],
  v10[9],
  v10[10],
  v10[11],
  v10[12],
  v10[13],
  v10[14],
  v10[15]);
return 0;
}
```

先把部分数字转为ASCII，接下来就对逻辑进行一个分析

首先是while的条件，很明显是终点条件，由此可以判断要求相同的两项为x/y，另一个为z，要求上有n、l，结合下方代码可以分辨x、y，对这些变量进行重命名

最后的输出是记录的路径，由此可以得到路径的变量和偏移量变量，命名为step、i

此外经常访问的一个变量肯定是地图的指针，命名为map

etc.

命名后代码如下（手动删除无关变量）：

```
// The function seems has been flattened
int __fastcall main(int argc, const char **argv, const char **envp)
{
  _BYTE out[31]; // [rsp+3A0h] [rbp-488h] BYREF
  char input; // [rsp+3BFh] [rbp-469h]
  __int64 n; // [rsp+3C0h] [rbp-468h] BYREF
  __int64 l; // [rsp+3C8h] [rbp-460h]
  char *map; // [rsp+3D0h] [rbp-458h]
  __int64 x; // [rsp+3D8h] [rbp-450h]
  __int64 y; // [rsp+3E0h] [rbp-448h]
  __int64 z; // [rsp+3E8h] [rbp-440h]
  __int64 i; // [rsp+3F0h] [rbp-438h]
  _BYTE step[1004]; // [rsp+3F8h] [rbp-430h] BYREF

  sub_140004160(&n);
  while ( x != n - 2 || y != x || z != l - 1 )
  {
    input = fgetchar();
    if ( input < 'k' )
    {
      if ( input < 'd' )
      {
        if ( input == 'a' )
        {
          --x;
          if ( map[n * y + n * z * n + x] == '#' )
```

```
            {
              ++x;
            }
            else if ( i && step[i - 1] == 'd' )
            {
              --i;
            }
            else
            {
              step[i++] = input;
            }
          }
        }
        else if ( input < 'i' )
        {
          if ( input == 'd' )
          {
            ++x;
            if ( map[x + n * y + n * n * z] == '#' )
            {
              --x;
            }
            else
            {
              if ( i && step[i - 1] == 'a' )
              {
                --i;
              }
              else
              {
                step[i] = input;
                i = 2 * (i & 1) + (i ^ 1);  // ++i;
              }
            }
          }
        }
        else if ( input == 'i' )
        {
          if ( map[x + n * z * n + n * y] == '^' || map[x + n * y + n * n * z] == 'X' )
          {
            --z;
            if ( i && step[i - 1] == 'k' )
              i = 2 * i + ~i;  // --i;
            else
              step[i++] = input;
          }
        }
      }
      else if ( input < 's' )
      {
        if ( input == 'k' )
        {
          if ( map[x + n * z * n + n * y] == 'v' || map[y * n + n * z * n + x] == 'X' )
          {
```

```
            z = (z | 1) + (z & 1);  // ++z;
            if ( i && step[i - 1] == 'i' )
              --i;
            else
              step[i++] = input;
          }
        }
      }
      else if ( input < 'w' )
      {
        if ( input == 's' )
        {
          ++y;
          if ( map[x + n * z * n + n * y] == '#' )
          {
            --y;
          }
          else if ( i && step[i - 1] == 'w' )
          {
            --i;
          }
          else
          {
            step[i++] = input;
          }
        }
      }
      else if ( input == 'w' )
      {
        if ( map[x + n * n * z + n * y] == '#' )
        {
          ++y;
        }
        else if ( i && step[i - 1] == 's' )
        {
          --i;
        }
        else
        {
          step[i++] = input;
        }
      }
    }
    step[i] = 0;
    sub_140010B00(step, out);
    sub_140003F50(
      "Congratulations! The flag is: TSCTF-
J{%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x}",
      out[0],
      out[1],
      out[2],
      out[3],
      out[4],
      out[5],
```

```
      out[6],
      out[7],
      out[8],
      out[9],
      out[10],
      out[11],
      out[12],
      out[13],
      out[14],
      out[15]);
    return 0;
  }
```

接下来逻辑就非常清晰了，可以很清晰看出 $pos2index(x, y, z) = x + n * y + n^2 * z$ ，墙为#，^上，v下，X上下，移动为

按键	方向
a	x-
d	x+
w	y-
s	y+
i	z-
k	z+

这里还有一个sub_140010B00需要分析，点进去随便看看，发现一串常数：

```
1  __int64 __fastcall sub_14000F0F0(__int64 a1)
2  {
3    __int64 result; // rax
4
5    *(_QWORD *)a1 = 0LL;
6    *(_DWORD *)(a1 + 8) = 0x67452301;
7    *(_DWORD *)(a1 + 12) = 0xEFCDAB89;
8    *(_DWORD *)(a1 + 16) = 0x98BADCFE;
9    result = a1 + 8;
10   *(_DWORD *)(a1 + 20) = 0x10325476;
11   return result;
12 }
```

无疑，这是一个MD5算法

最后就是编写算法，得到迷宫走法

解迷宫

接下来就写一个算法求解提取出来的迷宫：

```
maze = open('maze.txt').read().strip()

N = 41
L = 5

pos2index = lambda x, y, z: x + N*y + N*N*z

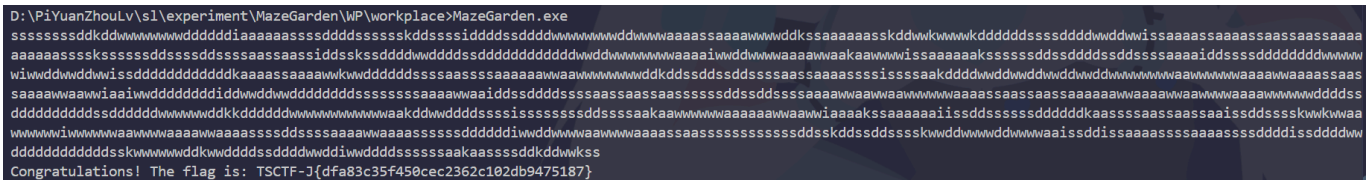
visited = {(1, 1, 0)}
queue = [((1, 1, 0), '')]

def add(x, y, z, p):
    if (x, y, z) in visited:
        return
    visited.add((x, y, z))
    queue.append(((x, y, z), p))
    if (x, y, z) == (N-2, N-2, L-1):
        print('Found path:', p)
        print(f'Congratulations! The flag is: TSCTF-
J{{{__import__('hashlib').md5(p.encode()).hexdigest()}}}')
        exit()

while queue:
    pos, path = queue.pop(0)
    (x, y, z) = pos
    if maze[pos2index(*pos)] == 'X':
        add(x, y, z+1, path+'k')
        add(x, y, z-1, path+'i')
    elif maze[pos2index(*pos)] == 'v':
        add(x, y, z+1, path+'k')
    elif maze[pos2index(*pos)] == '^':
        add(x, y, z-1, path+'i')
    if maze[pos2index(x+1, y, z)] != '#':
        add(x+1, y, z, path+'d')
    if maze[pos2index(x-1, y, z)] != '#':
        add(x-1, y, z, path+'a')
    if maze[pos2index(x, y+1, z)] != '#':
        add(x, y+1, z, path+'s')
    if maze[pos2index(x, y-1, z)] != '#':
        add(x, y-1, z, path+'w')
```

就可以得到flag了，如果担心MD5遭到了魔改，可以把答案路径输入到程序中，程序就会输出正确flag（所以其实不分析这个函数也是可以做的）

```
D:\PiYuanZhouLv\sl\experiment\MazeGarden\WP\workplace>D:\python314\python.exe d:/PiYuanZhouLv/sl/experiment/MazeGarden/WP/workplace/crack.py
Found path: sssssssddkddwwwwwwwwddddddiaaaaaaaaaaddddddssssssskddddssssiddddddssddddwwwwwwwwddwwwwaaaaaaaaaawwwwddkssaaaaaaaaskddwwkwwwwkddddddddssssddddwwddwwiissaaaaaaaaaa
```



出题人视角

首先这道题主要考察花指令和控制流平坦化的去除

因为本人对控制流平坦化的去除不太熟悉，所以就出的比较简单，重点在花指令

花指令是让D老师根据我总结的写了4大类56种花，同时我写了一段代码用来向源码的*每一行*加入一个花指令，所以花指令的量比较大，需要写脚本patch

```
import sys
if len(sys.argv) < 2:
    exit(-1)
import re
import random
random.seed(114514)
lines = []
depth = 0
switch = False
for line in open(sys.argv[1], encoding='utf-8'):
    lines.append(line)
    depth += line.count('{') - line.count('}')
    if 'switch' in line:
        switch = True
    elif 'case' in line:
        switch = False
    if not switch and depth >= 1 and random.random() <= 1:
        lines.append('    '*depth+'FLOWER();\n')
content = ''.join(lines)
flowers = list(map(lambda x: x.split('\n'), open('flowers', encoding='utf-
8').read().split('\n\n')))
to_add = [random.choice(fs) for fs in flowers][:content.count("FLOWER();")]
if len(to_add) < content.count("FLOWER();"):
    to_add.extend(random.choice(random.choice(flowers)) for _ in
range(content.count("FLOWER();")-len(to_add)))
random.shuffle(to_add)
i = -1
def add_flower(*_):
    global i
    i += 1
    return to_add[i].replace("[JUNK]", ', '.join(hex(random.randint(0, 0xff)) for _ in
range(random.randint(2, 8))))
open(sys.argv[1].removesuffix('.c')+'.flowered.c', 'w', encoding='utf-
8').write(re.sub(r'FLOWER\(\);', add_flower, content))
```

```
00000001…  0x1           DC                                      90
00000001…  0x1           EB                                      90
00000001…  0xc           78 0A 79 08 EA 93 48 78 DA DF 50 17      90 90 90 90 90 90 90 90 90 90 90 90
00000001…  0xa           72 08 73 06 CA B5 FC A5 0B 3D           90 90 90 90 90 90 90 90 90 90
00000001…  0x8           31 DB 74 04 D8 2C 55 99                 90 90 90 90 90 90 90 90
00000001…  0x7           31 C9 74 03 8E C2 3D                    90 90 90 90 90 90 90
00000001…  0x9           39 C9 74 05 69 3F 6A F4 3B              90 90 90 90 90 90 90 90 90
00000001…  0x1           EB                                      90
Line 1 of 491
```

~~（也就将近500朵吧）~~

ollvm使用的是ollvm-adaplite/ollvm-clang，不过这个似乎被魔改了，我只启用了控制流平坦化、表达式混淆和一个不知道有没有用的基本块分割

迷宫本来是想写一个与BouncingMaze相匹配的算法来着，但是由于这个出题程序含有bug，偶发多解现象，但是代码又是史山几托，就懒得改了，顺手写了一个3D迷宫，下面是核心逻辑的源码，以供参考：

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "md5.h"
#include "gameType.h"

#define getIndex(game) \
    (game.z * game.maze.n * game.maze.n + game.y * game.maze.n + game.x)

#define endGame(game) \
    (game.x == game.maze.n - 2 && game.y == game.x && game.z == game.maze.l - 1)

extern void setupGame(Game*);

int main() {
    Game game;
    setupGame(&game);
    while (!endGame(game)) {
        char d = getchar();
        switch (d) {
            case 'a':
                game.x--;
                if (game.maze.map[getIndex(game)] == '#') {
                    game.x++;
                } else {
                    if (game.i > 0 && game.step[game.i - 1] == 'd') {
                        game.i--;
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;
            case 'd':
                game.x++;
                if (game.maze.map[getIndex(game)] == '#') {
```

```
                    game.x--;
                } else {
                    if (game.i > 0 && game.step[game.i - 1] == 'a') {
                        game.i--;
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;
            case 'w':
                game.y--;
                if (game.maze.map[getIndex(game)] == '#') {
                    game.y++;
                } else {
                    if (game.i > 0 && game.step[game.i - 1] == 's') {
                        game.i--;
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;
            case 's':
                game.y++;
                if (game.maze.map[getIndex(game)] == '#') {
                    game.y--;
                } else {
                    if (game.i > 0 && game.step[game.i - 1] == 'w') {
                        game.i--;
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;
            case 'i':
                if (game.maze.map[getIndex(game)] == '^' || game.maze.map[getIndex(game)]
== 'X') {
                    game.z--;
                    if (game.i > 0 && game.step[game.i - 1] == 'k') {
                        game.i--;
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;
            case 'k':
                if (game.maze.map[getIndex(game)] == 'v' || game.maze.map[getIndex(game)]
== 'X') {
                    game.z++;
                    if (game.i > 0 && game.step[game.i - 1] == 'i') {
                        game.i--;
```

```
                    } else {
                        game.step[game.i] = d;
                        game.i++;
                    }
                }
                break;

            default:
                break;
        }
    }
    game.step[game.i] = 0;
    uint8_t out[16];
    md5String(game.step, out);
    printf("Congratulations! The flag is: TSCTF-
J{%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x%02x}", out[0], out[1],
out[2], out[3], out[4], out[5], out[6], out[7], out[8], out[9], out[10], out[11], out[12],
out[13], out[14], out[15]);  // 希望计导老师看到这行代码不会来砍我（x
    return 0;
}
```