BUUCTF--[ACTF新生赛2020]Oruga

原创

Hk_Mayfly 于 2020-06-06 00:36:00 发布 39 收藏
文章标签： java python 算法 vue 动态规划

本文链接：https://blog.csdn.net/qq_39542714/article/details/118691243

测试文件：https://lanzous.com/iPyvcddmqsh

代码分析

```
1  __int64 __fastcall main(__int64 a1, char **a2, char **a3)
2  {
3    __int64 result; // rax
4    __int64 v4; // [rsp+0h] [rbp-40h]
5    char v5; // [rsp+9h] [rbp-37h]
6    char s2[4]; // [rsp+Ah] [rbp-36h]
7    char s[40]; // [rsp+10h] [rbp-30h]
8    unsigned __int64 v8; // [rsp+38h] [rbp-8h]
9
10   v8 = __readfsqword(0x28u);
11   memset(s, 0, 0x19uLL);
12   printf("Tell me the flag:", 0LL);
13   scanf("%s", s);
14   strcpy(s2, "actf{");
15   LODWORD(v4) = 0;
16   while ( (signed int)v4 <= 4 )
17   {
18     *((_BYTE *)&v4 + (signed int)v4 + 4) = s[(signed int)v4];
19     LODWORD(v4) = v4 + 1;
20   }
21   v5 = 0;
22   if ( !strcmp((const char *)&v4 + 4, s2) )
23   {
24     if ( (unsigned __int8)sub_78A((__int64)s) )
25       printf("That's True Flag!", s2, v4);
26     else
27       printf("don't stop trying...", s2, v4);
28     result = 0LL;
29   }
30   else
31   {
32     printf("Format false!", s2, v4);
33     result = 0LL;
34   }
35   return result;
36 }
```

前22行代码实际就是告诉我们输入的前5个字符为"actf{"

主要分析sub_78A函数

```
_BOOL8 __fastcall sub_78A(__int64 a1)
{
  int v2; // [rsp+Ch] [rbp-Ch]
  signed int v3; // [rsp+10h] [rbp-8h]
  signed int v4; // [rsp+14h] [rbp-4h]

  v2 = 0;
  v3 = 5;
  v4 = 0;
  while ( byte_201020[v2] != 33 )
  {
    v2 -= v4;
    if ( *(_BYTE *)(v3 + a1) != 'W' || v4 == -16 )
    {
      if ( *(_BYTE *)(v3 + a1) != 'E' || v4 == 1 )
      {
        if ( *(_BYTE *)(v3 + a1) != 'M' || v4 == 16 )
        {
          if ( *(_BYTE *)(v3 + a1) != 'J' || v4 == -1 )
            return 0LL;
          v4 = -1;
        }
        else
        {
          v4 = 16;
        }
      }
      else
      {
        v4 = 1;
      }
    }
    else
    {
      v4 = -16;
    }
    ++v3;
    while ( !byte_201020[v2] )
    {
      if ( v4 == -1 && !(v2 & 0xF) )
        return 0LL;
      if ( v4 == 1 && v2 % 16 == 15 )
        return 0LL;
      if ( v4 == 16 && (unsigned int)(v2 - 240) <= 0xF )
        return 0LL;
      if ( v4 == -16 && (unsigned int)(v2 + 15) <= 0x1E )
        return 0LL;
      v2 += v4;
    }
  }
```

byte_201020为：

> 00 00 00 00 23 00 00 00 00 00 00 00 23 23 23 2300 00 00 23 23 00 00 00 4F 4F 00 00 00 00 00 0000 00 00 00 00 00 00 00 4F 4F 00 50 50 00 00 0000 00 00 4C 00 4F 4F 00 4F 4F 00 50 50 00 00 0000 00 00 4C 00 4F 4F 00 4F 4F 00 50 00 00 00 0000 00 4C 4C 00 4F 4F 00 00 00 00 50 00 00 00 0000 00 00 00 00 4F 4F 00 00 00 00 50 00 00 00 0023 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0000 00 00 00 00 00 00 00 00 00 00 00 23 00 00 0000 00 00 00 00 00 4D 4D 4D 00 00 00 23 00 00 0000 00 00 00 00 00 00 4D 4D 4D 00 00 00 00 45 4500 00 00 30 00 4D 00 4D 00 4D 00 00 00 00 45 0000 00 00 00 00 00 00 00 00 00 00 00 00 00 45 4554 54 54 49 00 4D 00 4D 00 4D 00 00 00 00 45 0000 54 00 49 00 4D 00 4D 00 4D 00 00 00 00 45 0000 54 00 49 00 4D 00 4D 00 4D 21 00 00 00 45 45

这实际也是个迷宫题

外层while循环，就是到0x21时停止，v2为当前位置，v3从输入的第六个字符开始遍历数组，v4实际是对v2位置的移动。

> W 向上移动
>
> E 向右移动
>
> M 向下移动
>
> J 向左移动

第二处，用了个循环，if条件判断的是位置是否到达边界，越界退出。

整个迷宫就是用WEMJ控制移动，0可以移动(一直移动)，到达非0位置停止，返回上一位置。左上角为起点，0x21为终点。手动解迷宫

MEWEMEWJMEWJM

get flag！

> **flag{MEWEMEWJMEWJM}**