攻防世界 easy_Maze

原创

Bxb0 于 2020-05-04 14:08:41 发布 759 收藏 3

文章标签： c语言 指针

本文链接：https://blog.csdn.net/Bsecure/article/details/105916981

easy_Maze

这道题还是挺好玩的.

由题目我们能知道是关于迷宫的题. 下载下来是elf文件, 查壳后无壳, 在linux下运行看看大概流程.

载入IDA, 先分析了接受我们输入的字符的函数. 知道在外面一层的主函数是先生成一个每行7个元素的数组. 然后通过我们输入的字符按控制在矩阵迷宫中走向, 最后按照规定的到达一个目的地.

```
__int64 __fastcall Step_2(int (*a1)[7])
{
  int v1; // eax
  __int64 v2; // rax
  __int64 v3; // rax
  __int64 result; // rax
  __int64 v5; // rax
  char v6[35]; // [rsp+10h] [rbp-30h]
  char v7; // [rsp+33h] [rbp-Dh]
  int v8; // [rsp+34h] [rbp-Ch]
  int v9; // [rsp+38h] [rbp-8h]
  int v10; // [rsp+3Ch] [rbp-4h]

  v10 = 0;
  v9 = 0;
  v8 = 0;
  while ( v8 <= 29 && (*a1)[7 * v10 + v9] == 1 )// 我感觉这里是应该是 a1[v10][v9]   数组指针啊 只不过这样也可以都是先得到首地址
  {                                             // 到这里, 初步知道了是个矩阵, 通过下面的4个操作, 根据最后的结果, 走规定的路线.
    std::operator>><char,std::char_traits<char>>(&std::cin, &v7);
    v1 = v8++;
    v6[v1] = v7;
    if ( v7 == 100 )                            // 输入字符 d  的话  v9++   向右
    {
      ++v9;
    }
    else if ( v7 > 100 )
    {
      if ( v7 == 115 )                          // 输入字符 s 的话 v10++   向下
      {
        ++v10;
      }
      else
      {
        if ( v7 != 119 )                        // 输入字符 w 的话 v10--   向上
          goto LABEL_14;
        --v10;
      }
    }
    else if ( v7 == 97 )                        // 输入字符 a 的话 v9--   向左
    {
      --v9;
    }                                           // 就是 我们打游戏熟悉的  WASD  啊, 哈哈
    else
    {
LABEL_14:
      v2 = std::operator<<<std::char_traits<char>>(&_bss_start, "include illegal words.");
      std::ostream::operator<<(v2, &std::endl<char,std::char_traits<char>>);
    }
  }
  if ( v10 != 6 || v9 != 6 )                    // 由上一层生成矩阵的函数,  我们的矩阵是 7*7 , 那这里就表示终点是最后一个元素
  {
    v5 = std::operator<<<std::char_traits<char>>(&_bss_start, "Oh no!,Please try again~~");
    std::ostream::operator<<(v5, &std::endl<char,std::char_traits<char>>);
    result = 0LL;
```

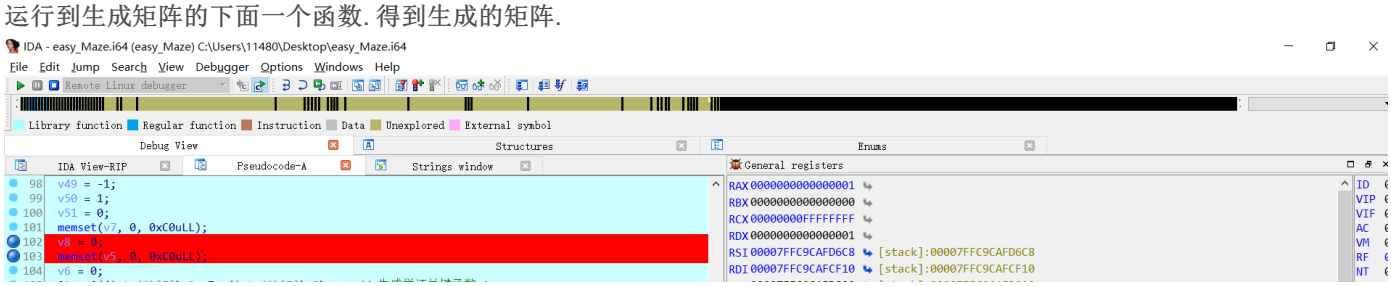
```
  }
  else
  {
    v3 = std::operator<<<std::char_traits<char>>(&_bss_start, "Congratulations!");
    std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
    output(v6, v8);
    result = 1LL;
  }
  return result;
}
```

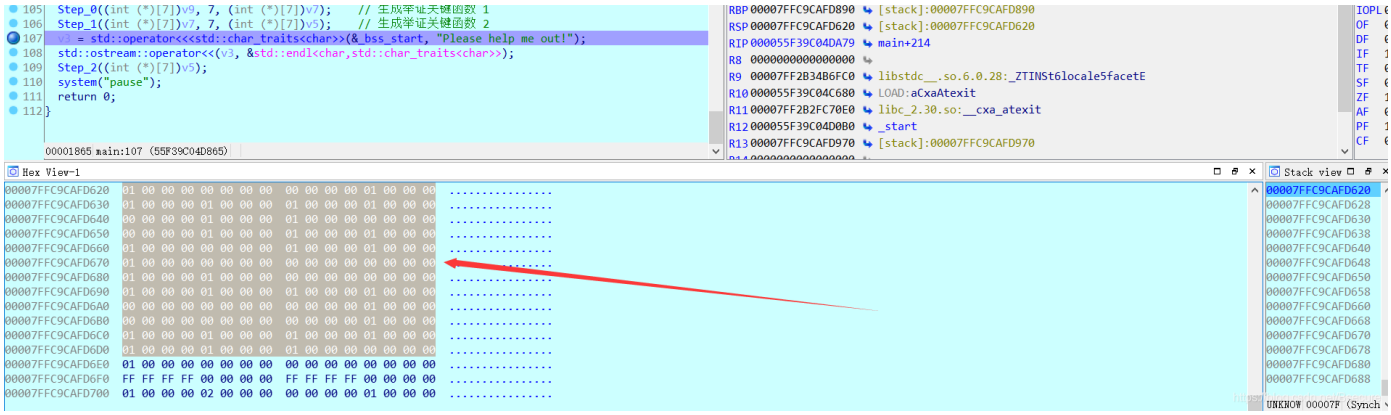

接下来就是找生成那个矩阵了. 通过上面的49个元素知道且下面函数的参数7, 知道规模 7*7.

```
  v9[3] = 1;
  v9[4] = -1;
  v9[5] = 1;
  v9[6] = -1;
  v10 = 0;
  v11 = 0;
  v12 = 0;
  v13 = 0;
  v14 = 1;
  v15 = -1;
  v16 = 0;
  v17 = 0;
  v18 = 1;
  v19 = 0;
  v20 = 0;
  v21 = 1;
  v22 = 0;
  v23 = -1;
  v24 = -1;
  v25 = 0;
  v26 = 1;
  v27 = 0;
  v28 = 1;
  v29 = -1;
  v30 = 0;
  v31 = -1;
  v32 = 0;
  v33 = 0;
  v34 = 0;
  v35 = 0;
  v36 = 0;
  v37 = 1;
  v38 = -1;
  v39 = -1;
  v40 = 1;
  v41 = -1;
  v42 = 0;
  v43 = -1;
  v44 = 2;
  v45 = 1;
  v46 = -1;
  v47 = 0;
  v48 = 0;
  v49 = -1;
  v50 = 1;
  v51 = 0;
  memset(v7, 0, 0xC0uLL);
  v8 = 0;
  memset(v5, 0, 0xC0uLL);
  v6 = 0;
  Step_0((int (*)[7])v9, 7, (int (*)[7])v7);    // 生成举证关键函数 1
  Step_1((int (*)[7])v7, 7, (int (*)[7])v5);    // 生成举证关键函数 2
  v3 = std::operator<<<std::char_traits<char>>(&_bss_start, "Please help me out!");
  std::ostream::operator<<(v3, &std::endl<char,std::char_traits<char>>);
  Step_2((int (*)[7])v5);
  system("pause");
  return 0;
}
```


看了 **Step_0** 函数, 还是简单, 但是 **step_1** 中嵌套了2个函数且很复杂. 开始我就把生成矩阵的所有函数及数据都复制到**VC** 再根据栈的特点改下数据的顺序, 然后运行打印出矩阵. 问题是运行后什么也没有打印, 调试发现一个函数中的内存空间和另外一个冲突, 相互覆盖值. 这个函数太多, 改起来也麻烦.

然后转向GDB调试, 但是不熟练. 又转向ida动态调试. 先下断点, 找到储存矩阵的空间的地址, 把这个地址转到数据窗口跟随. 运行到生成矩阵的下面一个函数. 得到生成的矩阵.

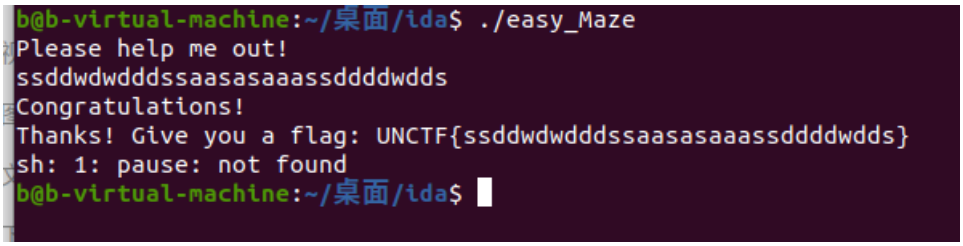




把数据复制下来, 用C语言打印出来号观察, 注意每个数据4个字节(小端).

```
#include <stdio.h>

int main(void)
{
 union
 {
  unsigned char ida_chars[196];
  int a[49];
 }A = {
      1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      0,    0,    1,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    1,    0,    0,    0,    1,    0,
      0,    0,    0,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      0,    0,    1,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    1,    0,    0,    0,    0,    0,
      0,    0,    1,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      0,    0,    0,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      0,    0,    1,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    1,    0,    0,    0,    1,    0,
      0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      1,    0,    0,    0,    0,    0,    0,    0,    0,    0,
      0,    0,    0,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    1,    0,    0,    0,    1,    0,
      0,    0,    1,    0,    0,    0,    1,    0,    0,    0,
      1,    0,    0,    0,    1,    0,    0,    0,    0,    0,
      0,    0
 };
 int i = 0, j = 0;
 int (*p)[7] = (int (*)[7])A.a;

 for(i = 0; i < 7; i++)
 {
  for(j = 0; j < 7; j++)
  {
   printf("%d ", p[i][j]);
  }
  putchar(10);
 }

 return 0;

}
```

- 根据我们打游戏的熟悉 aswd ,朝着1走到最后. **ssddwdwdddsssaasasaaassddddwdds** 最后在linux下输入.



- **总结: 更加熟悉了linux动态调试的运用, 加深了数组指针的理解.**