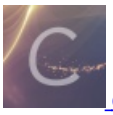
XCTF-Reverse-ExerciseArea-012-writeup

原创

y4ung 于 2020-09-14 23:12:11 发布 226 收藏

分类专栏： ctf 文章标签： ctf



0x00 介绍

本题是xctf攻防世界中Reverse的新手第十二题。题目来源： `NJUPT CTF 2017`

题目描述：菜鸡想要走出菜狗设计的迷宫

这题是一道迷宫类型的题目，可以参考CTF-Wiki里的迷宫问题

0x01 解题过程

Linux下的可执行文件

```
$ file bdb2c015b0fd4f74bc4c3e5a6e54bcf4
bdb2c015b0fd4f74bc4c3e5a6e54bcf4: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, inter
preter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=eda1df76eec45447cd0e1ad208a7eff914e86758
, stripped

$ ./bdb2c015b0fd4f74bc4c3e5a6e54bcf4
Input flag:
123
Wrong flag!
```

拖到IDA里，先通过 `Shift+F12` 打开 Strings Window，找到 `Input flag:` ，双击， `x` 查看引用的地方，是main()函数。ok，现在到main()函数里查看。 `F5` 查看main()函数的伪代码。

首先是读取用户输入到变量s1中，然后进行检查。可以看出，用户的输入长度必须为24，并且以 `nctf{` 开头，以 `}` 结尾。

```
10  v9 = 0LL;
11  puts("Input flag:");
12  scanf("%s", &s1, 0LL);
13  if ( strlen(&s1) != 24 || strncmp(&s1, "nctf{", 5uLL) || *(&byte_6010BF + 24) != '}' )
14  {
15LABEL_22:
16    puts("Wrong flag!");
17    exit(-1);
18  }
```

在while循环中，判断用户输入的每个字符串，可以看到程序中将每个字符和'O'，'o'，'.'，'0'进行对比，如果有相等的，进入相应的if语句块中。在sub_400650，sub_400660，sub_400670，sub_400680函数中，对传进来的参数进行了自增或自减的操作。现在基本可以判定这几个if语句块是用来控制在迷宫中的走向的。具体哪个字符控制什么方向还要进一步分析。

```
bool __fastcall sub_400650(_DWORD *a1)
{
  int v1; // eax

  v1 = (*a1)--;
  return v1 > 0;
}
```

```
bool __fastcall sub_400660(int *a1)
{
  int v1; // eax

  v1 = *a1 + 1;
  *a1 = v1;
  return v1 < 8;
}
```

回到main函数中，看到传入sub_400650，sub_400660，sub_400670，sub_400680函数的参数为 v9 或是 v9+1。说明这两个变量有一个是控制x方向，一个是控制y方向的，其中x控制在二维数组迷宫中的垂直方向，y控制的是在二维数组迷宫中的水平方向（看0x02的总结部分）。具体谁控制x、谁控制y呢？

```
    puts("Wrong flag!");
    exit(-1);
  }
  v3 = 5LL;
  if ( strlen(&s1) - 1 > 5 )
  {
    while ( 1 )
    {
      v4 = *(&s1 + v3);                         // v5是user_input的每个字符，v4是索引
      v5 = 0;
      if ( v4 > 'N' )                           // v5 > 'N'
      {
        v4 = (unsigned __int8)v4;
        if ( (unsigned __int8)v4 == 'O' )
        {
          v6 = sub_400650((_DWORD *)&v9 + 1);   // v9+1 值减1，相当于在迷宫里向左
          goto LABEL_14;
        }
        if ( v4 == 'o' )
        {
          v6 = sub_400660((int *)&v9 + 1);      // v9+1 值加1，相当于在迷宫里向右
          goto LABEL_14;
        }
      }
      else                                      // v5<='N'
      {
        v4 = (unsigned __int8)v4;
        if ( (unsigned __int8)v4 == '.' )       // v9值减1，相当于在迷宫里向上
        {
          v6 = sub_400670(&v9);
          goto LABEL_14;
        }
        if ( v4 == '0' )
        {
          v6 = sub_400680((int *)&v9);          // v9值加1，相当于在迷宫里向下
LABEL_14:
          v5 = v6;
          goto LABEL_15;
        }
      }
LABEL_15:
      if ( !(unsigned __int8)sub_400690((__int64)maze_str, SHIDWORD(v9), v9) )// (*((int32*)&(v9)+1))
        goto LABEL_22;
      if ( ++v3 >= strlen(&s1) - 1 )            // 退出循环的条件
      {
        if ( v5 )
0000075E main:42 (40075E)
```


感觉还是没什么头绪，继续往下看。sub_400690函数是一个比较关键的地方。传入的参数为`(maze_str, v9+1, v9)`。

```
LABEL_15:
      if ( !(unsigned __int8)sub_400690((__int64)maze_str, SHIDWORD(v9), v9) )// (*((int32*)&(v9)+1))
        goto LABEL_22;
      if ( ++v3 >= strlen(&s1) - 1 )            // 退出循环的条件
      {
        if ( v5 )
          break;
```

跟进sub_400690函数，result是在迷宫字符串中取一个字符的操作，也就是判断经过当前的操作以后走到迷宫的什么位置：

```
result = *(unsigned __int8 *)(a1 + a2 + 8LL * a3);
```

这个迷宫字符串是一维数组的形式，既然要通过x和y来取，那么很明显的，x必须要乘以每一行的字符数才行，因此上面代码中的a3，也就是main函数中的v9就是x，v9+1对应的y。回到main函数中，可以判断出'O'，'o'，'.'，'0'分别对应y-1(向左)，y+1(向右)，x-1(向上)，x+1(向下)。

```
// maze_str, SHIDWORD(v9), v9
__int64 __fastcall sub_400690(__int64 a1, int a2, int a3)
{
  __int64 result; // rax

  result = *(unsigned __int8 *)(a1 + a2 + 8LL * a3);// a2是(*((int32*)&(v9)+1))，也就是y，然后a3是x
                                                    // 联想maze_str在内存中是一维数组，因此，
                                                    // 为了取出当前的位置，需要用maze_str的起始地址+y+8*x。
                                                    // 这里的8其实也从侧面说明了这个迷宫是8个字符为一行
  LOBYTE(result) = (_DWORD)result == ' ' || (_DWORD)result == '#';// ' '是迷宫中走路的路径
                                                    // '#'是迷宫的出口
                                                    //
                                                    // 如果当前的位置是' '或'#'，就返回True
  return result;
}
```


其实sub_400690函数就是用来判断走了当前的步骤以后，会不会撞到墙，还是说没有撞到墙或是已经到达终点（' ', '#'）。如果撞到墙则返回False，在main函数中if条件判断为真，跳转到 `Wrong flag!` 。

重新对迷宫字符串按行编排：

```
maze_str = '  *******   *  **** * ****  * ***  *#  *** *** ***     *********'#.replace(" ", "x")
tmp_str = ""
maze_str_len = len(maze_str)
for i in range(0, maze_str_len-1, 8):
    print(maze_str[i:i+8])
```

```
  ******
*   *  *
*** * **
**  * **
*  *#  *
** *** *
**     *
********
```


找出通往出口的路径，最终的flag为： `nctf{o0oo00O000oooo..OO}`

0x02 总结

解决迷宫问题：

- 需要找到迷宫的形状，比如本题中迷宫为一维数组的字符串。在函数 `sub_400690` 中，确定了迷宫的字符串为8个字符一行。
- 除此之外，还需要找到在迷宫里行走时的变量 x(行) 和 y(列)，以及用户输入的字符如何控制 x 和 y 变换。比如本题中是用v9这个变量表示x，v9+1表示y。
- 同时也需要找出在迷宫中行走的条件判断，比如本题中用'O'，'o'，'.'，'0'控制x和y的改变。

```
    y ---> y变大
 x              ******
 |             *   *  *
 |             *** * **
\|/            **  * **
 x变大          *  *#  *
               ** *** *
               **     *
               ********
```