buuctf-re-[ACTF新生赛2020]Oruga

原创

TD.Jia 于 2020-10-22 22:32:16 发布 155 收藏

elf文件，ida分析，定位到main函数

开始判断开头四个字符为actf{

```
__int64 __fastcall main(__int64 a1, char **a2, char **a3)
{
  __int64 result; // rax
  __int64 v4; // [rsp+0h] [rbp-40h]
  char v5; // [rsp+9h] [rbp-37h]
  char s2[4]; // [rsp+Ah] [rbp-36h]
  char s[40]; // [rsp+10h] [rbp-30h]
  unsigned __int64 v8; // [rsp+38h] [rbp-8h]

  v8 = __readfsqword(0x28u);
  memset(s, 0, 0x19uLL);
  printf("Tell me the flag:", 0LL);
  scanf("%s", s);                               // 输入点
  strcpy(s2, "actf{");                          // 前四个字符
  LODWORD(v4) = 0;
  while ( (signed int)v4 <= 4 )
  {
    *((_BYTE *)&v4 + (signed int)v4 + 4) = s[(signed int)v4];// v4[4] = s[0]
                                                // v4[5] = s[1]
                                                // v4[6] = s[2]
                                                // v4[7] = s[3]
    LODWORD(v4) = v4 + 1;
  }
  v5 = 0;
  if ( !strcmp((const char *)&v4 + 4, s2) )     // 确定前四个字符
  {
    if ( (unsigned __int8)sub_78A((__int64)s) ) // 关键算法
      printf("That's True Flag!", s2, v4);
    else
      printf("don't stop trying...", s2, v4);
    result = 0LL;
  }
  else
  {
    printf("Format false!", s2, v4);
    result = 0LL;
  }
  return result;
}
```

进入关键函数分析算法

发现是个迷宫算法

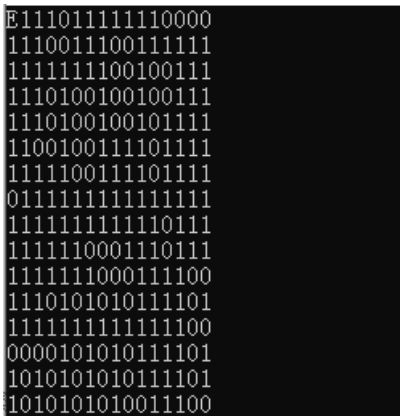
```
_BOOL8 __fastcall sub_78A(__int64 a1)
{
  int v2; // [rsp+Ch] [rbp-Ch]
  signed int v3; // [rsp+10h] [rbp-8h]
  signed int v4; // [rsp+14h] [rbp-4h]

  v2 = 0;
  v3 = 5;
  v4 = 0;
  while ( byte_201020[v2] != 33 )               // 33是出口
  {
    v2 -= v4;                                   // 索引移动
    if ( *(_BYTE *)(v3 + a1) != 'W' || v4 == 0xFFFFFFF0 )// 向上
    {
      if ( *(_BYTE *)(v3 + a1) != 'E' || v4 == 1 )// 向右
      {
        if ( *(_BYTE *)(v3 + a1) != 'M' || v4 == 16 )// 向下
        {
          if ( *(_BYTE *)(v3 + a1) != 'J' || v4 == -1 )// 向左
            return 0LL;
          v4 = -1;
        }
        else
        {
          v4 = 16;
        }
      }
      else
      {
        v4 = 1;
      }
    }
    else
    {
      v4 = -16;
    }
    ++v3;
    while ( !byte_201020[v2] )                  // 不为0   0为路线   遇到0停止
    {
      if ( v4 == -1 && !(v2 & 0xF) )            // 最左一列不能向左移动
        return 0LL;
      if ( v4 == 1 && v2 % 16 == 0xF )          // 最右一列，不能向右移动
        return 0LL;
      if ( v4 == 16 && (unsigned int)(v2 - 0xF0) <= 0xF )// 最下一列，不能向下移动
        return 0LL;
      if ( v4 == -16 && (unsigned int)(v2 + 15) <= 30 )// 最上一列，不能向上移动
        return 0LL;
      v2 += v4;                                 // 遇到0停止
    }
  }
  return *(_BYTE *)(v3 + a1) == '}';
}
```

这个算法之前没遇到过，判断是迷宫其实也容易，提取出opcode来写脚本

```c
#include <stdio.h>
unsigned char maze[] =
{
  0x00, 0x00, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x23, 0x23, 0x23, 0x23, 0x00, 0x00, 0x00, 0x23,
  0x23, 0x00, 0x00, 0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x4F, 0x4F, 0x00, 0x50, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x4C, 0x00, 0x4F, 0x4F, 0x00, 0x4F, 0x4F, 0x00, 0x50,
  0x50, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4C, 0x00, 0x4F,
  0x4F, 0x00, 0x4F, 0x4F, 0x00, 0x50, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x4C, 0x4C, 0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00,
  0x00, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x4F, 0x4F, 0x00, 0x00, 0x00, 0x00, 0x50, 0x00, 0x00,
  0x00, 0x00, 0x23, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x23, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x4D, 0x4D, 0x4D, 0x00, 0x00, 0x00, 0x23, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4D, 0x4D, 0x4D,
  0x00, 0x00, 0x00, 0x00, 0x45, 0x45, 0x00, 0x00, 0x00, 0x30,
  0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00, 0x00, 0x00,
  0x45, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x45, 0x45, 0x54, 0x54,
  0x54, 0x49, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00,
  0x00, 0x00, 0x45, 0x00, 0x00, 0x54, 0x00, 0x49, 0x00, 0x4D,
  0x00, 0x4D, 0x00, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x45, 0x00,
  0x00, 0x54, 0x00, 0x49, 0x00, 0x4D, 0x00, 0x4D, 0x00, 0x4D,
  0x21, 0x00, 0x00, 0x00, 0x45, 0x45
};
int main()
{
 int i,j;
 for(i=0;i<16;i++)
 {
  for(j=0;j<16;j++)
  {
   if(i==0&&j==0)//开始
   {
    printf("E");
   }
   else if(maze[16*i+j]==0)//路线
   {
    printf("1");
   }
   else if(maze[16*i+j==0x21])//终点
   {
    printf("Z");
   }
   else//障碍物
   {
    printf("0");
   }
  }
  printf("\n");
 }
}
```

有一处地方，遇到道路会继续执行，知道遇到墙（0）才停止

```
while ( !byte_201020[v2] )                  // 不为0    0为路线    遇到0停止
{
  if ( v4 == -1 && !(v2 & 0xF) )            // 最左一列不能向左移动
    return 0LL;
  if ( v4 == 1 && v2 % 16 == 0xF )          // 最右一列，不能向右移动
    return 0LL;
  if ( v4 == 16 && (unsigned int)(v2 - 0xF0) <= 0xF )// 最下一列，不能向下移动
    return 0LL;
  if ( v4 == -16 && (unsigned int)(v2 + 15) <= 30 )// 最上一列，不能向上移动
    return 0LL;
  v2 += v4;                                 // 遇到0停止
}
```



根据上下左右分别对应的字符，得到flag

flag{MEWEMEWJMEWJM}