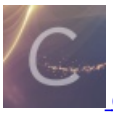
攻防世界Reverse进阶区-easy_Maze-writeup

原创

y4ung 于 2020-12-04 20:44:13 发布 303 收藏 1

分类专栏： ctf 文章标签： ctf



1. 介绍

本题是xctf攻防世界中Reverse的进阶区的题easy_Maze

2. 分析

```
$ file easy_Maze
easy_Maze: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=0cb6db3836551a104de9a42d40517d6430258127, not stripped
$ chmod +x easy_Maze
$ ./easy_Maze
Please help me out!
123
include illegal words.
include illegal words.
include illegal words.
abcd
Oh no!,Please try again~~
```

2.1 Step_2函数

用IDA打开，查看strings window，发现 `Congratulations!` 字符串在Step_2函数中被使用，打印出该字符串的条件是v10等于6，并且v9等于6。

让我们来分析一下Step_2函数。

首先是一个while循环，条件是 `v8 <= 29 && (*a1)[7 * v10 + v9] == 1` 。看到这个 `(*a1)[7 * v10 + v9] == 1` ，就能猜测到，迷宫应该是保存在a1中，其中a1是Step_2函数的参数。并且一行有7个字符，v10用来控制行（上下），v9控制列（左右），也就是每一行的第几个元素。同时，只有当当前字符为1时，才能继续走下去。结合前面分析的打印出 `Congratulations!` 字符串的条件可知，最后必须走到第7行、第7列（v9和v10从0开始，最终的值为6），才能走出迷宫。

在while循环中，用 `w,s,a,d` 分别控制上下左右。

```
  v10 = 0;
  v9 = 0;
  v8 = 0;
  while ( v8 <= 29 && (*a1)[7 * v10 + v9] == 1 )// a1应该就是迷宫，一行有7个字符，
                                                // v10控制行，v9控制列，也就是每一行的第几个元素
  {
    std::operator>><char,std::char_traits<char>>(&std::cin, &v7);// 用户输入
    v1 = v8++;
    v6[v1] = v7;
    if ( v7 == 'd' )
    {
      ++v9;
    }
    else if ( v7 > 100 )
    {
      if ( v7 == 's' )
      {
        ++v10;
      }
      else
      {
        if ( v7 != 'w' )
          goto LABEL_14;
        --v10;
      }
    }
    else if ( v7 == 'a' )
    {
      --v9;
    }
    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 )                    // v10必须为6，v9也必须为6
  {
    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;
  }
```

```
0000170E _Z6Step_2PA7_ii:14 (170E)
```


那现在要分析的就是这个迷宫a1是什么样子的了。我们到调用Step_2函数的地方，也就是main函数里去看一下。（其实看到Step_2这个名字，也能猜到前面至少还有个Step_1哈哈哈）

2.2 main函数

main函数中，定义了3个int数组v5、v7和v9，每个数组中有7个元素。其中，v5和v7都初始化为0，v9初始化为 `[1, 1, -1, 1, -1, 1, -1 ]` 。

```
...
v9[0] = 1;
v9[1] = 1;
v9[2] = -1;
v9[3] = 1;
v9[4] = -1;
v9[5] = 1;
v9[6] = -1;
...
memset(v7, 0, 0xC0uLL);
...
memset(v5, 0, 0xC0uLL);
...
```

接下来首先调用的是Step_0函数：`Step_0((int (*)[7])v9, 7, (int (*)[7])v7);`，然后再调用Step_1函数：`Step_1((int (*)[7])v7, 7, (int (*)[7])v5);`。从参数可以推测出，通过Step_0，对v7做了手脚，然后通过Step_1对v5做了手脚。最后v5就是迷宫，传到Step_2函数中：`Step_2((int (*)[7])v5);`

2.3 Step_0函数

先来看看Step_0函数。用了两层的for循环对a3，也就是main函数里的v7数组进行赋值。其中，i的取值范围是`[0, 6]`，j的取值范围也是`[0, 6]`。因此，v7应该是一个二维数组，每一维都是7个元素，一共有49个元素。

```
1  // v9, 7, v7
2  __int64 __fastcall Step_0(int (*a1)[7], signed int a2, int (*a3)[7])
3  {
4    __int64 result; // rax
5    int j; // [rsp+20h] [rbp-8h]
6    unsigned int i; // [rsp+24h] [rbp-4h]
7
8    for ( i = 0; ; ++i )                          // i: [0, 6]
9    {
10     result = i;
11     if ( (signed int)i >= a2 )
12       break;
13     for ( j = 0; j < a2; ++j )                  // j: [0, 6]
14       (*a3)[7 * i + j] = (*a1)[7 * j + a2 - i - 1];
15   }
16   return result;
17 }
```


当我尝试写脚本算出a3最后的结果时，发现在里层循环中a1获取元素时报错了，报的是数组越界的错。

回过头来看main函数中对v5、v7和v9的定义。可以看到，v9的上面还定义了一些数据：$v10_{v51}$。因此，当取从a1的首地址开始取，如果越界了，就会从v10v51里的数据取。

变量	内存地址
v51	rbp-10h
…	…
v9	rbp-D0h
v8	rbp-E0h
v7	rbp-1A0h
v6	rbp-1B0h
v5	rbp-270h

搞清楚了以后，我们可以把v10~v51的值全放到a1里。写个脚本把a3算出来：

```
tmp = """
v9[0] = 1;
v9[1] = 1;
v9[2] = -1;
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;
"""
tmp = tmp.split("=")
a1 = []
for each in tmp:
    if ";" in each:
        a1.append(int(each.split(";")[0].strip()))

a2 = 7
a3 = [0] * 49  # v7

for i in range(0, 7):
    for j in range(0, 7):
        a3[7*i + j] = a1[7*j + a2 - i - 1]

print("a3:", a3)
```

最终得到a3，也就main函数中的v7的值为：`[-1, 0, -1, 0, 1, 2, 0, 1, -1, 0, -1, 0, -1, 1, -1, 1, 1, 1, 0, 0, -1, 1, 0, 0, 0, 0, -1, 0, -1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, -1, -1, 1, 0, 0, -1, -1, -1, 1]`

2.4 Step_1函数 or 动态调试

到Step_1函数，发现里面调用了getA函数，感觉继续这样逆下去挺麻烦的。

既然我们的目的是要得到v5的内容。那么，直接用动态调试，在调用Step_2函数之前看一下v5往后的49个数值是啥，就知道迷宫的形状了…
再来看看main函数中对v5的定义。v5是int [7]，v6与v5之间隔了192个字节，1个int是4个字节，也就是中间隔了 48 个int，还需要查看从v6开始的4字节。因此，我们只要查看从rbp-270h到rbp-1ACh之间的196个字节的地址内容，即可知道迷宫的样子。

变量	内存地址
v6	rbp-1B0h
v5	rbp-270h

启动gdb，在调用Step_2函数之前下断点。

```
mov     esi, 7          ; int
mov     rdi, rax        ; int (*)[7]
call    _Z6Step_2PA7_ii ; Step_2(int (*)[7],int)
```

```
gef➤  b main
Breakpoint 1 at 0x1869
gef➤  b *0x555555555aa8  # 通过IDA与gdb的地址差值，计算出mov     esi, 7 指令的地址
Breakpoint 2 at 0x555555555aa8
gef➤  continue
```

打印出内存地址的值

```
$r11   : 0x00007ffff71d97b0  →  <fflush+0> test rdi, rdi
$r12   : 0x00005555555550b0  →  <_start+0> xor ebp, ebp
$r13   : 0x00007fffffffe610  →  0x0000000000000001
$r14   : 0x0
$r15   : 0x0
$eflags: [CARRY parity ADJUST zero sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x0033 $ss: 0x002b $ds: 0x0000 $es: 0x0000 $fs: 0x0000 $gs: 0x0000

0x00007fffffffe2c0|+0x0000: 0x0000000000000001   ← $rax, $rsp
0x00007fffffffe2c8|+0x0008: 0x0000000100000000
0x00007fffffffe2d0|+0x0010: 0x0000000100000001
0x00007fffffffe2d8|+0x0018: 0x0000000100000001
0x00007fffffffe2e0|+0x0020: 0x0000000100000000
0x00007fffffffe2e8|+0x0028: 0x0000000000000001
0x00007fffffffe2f0|+0x0030: 0x0000000100000000
0x00007fffffffe2f8|+0x0038: 0x0000000100000001

   0x555555555a9c <main+567>       call   0x555555555060 <_ZNSolsEPFRSoS_E@plt>
   0x555555555aa1 <main+572>       lea    rax, [rbp-0x270]
   0x555555555aa8 <main+579>       mov    esi, 0x7
 → 0x555555555aad <main+584>       mov    rdi, rax
   0x555555555ab0 <main+587>       call   0x5555555556ff <_Z6Step_2PA7_ii>
   0x555555555ab5 <main+592>       lea    rdi, [rip+0x5cc]        # 0x555555556088
   0x555555555abc <main+599>       call   0x555555555030 <system@plt>
   0x555555555ac1 <main+604>       mov    eax, 0x0
   0x555555555ac6 <main+609>       leave

[#0] Id 1, Name: "easy_Maze", stopped 0x555555555aad in main (), reason: BREAKPOINT

[#0] 0x555555555aad → main()

gef➤  p $rax
$1 = 0x7fffffffe2c0
gef➤  x/196xb $rax
0x7fffffffe2c0: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2c8: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2d0: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2d8: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2e0: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2e8: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe2f0: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe2f8: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe300: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe308: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe310: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe318: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe320: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe328: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe330: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe338: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe340: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe348: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe350: 0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe358: 0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe360: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe368: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe370: 0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x7fffffffe378: 0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x7fffffffe380: 0x01    0x00    0x00    0x00
gef➤
```


ok，现在写个脚本把迷宫提取出来即可。

需要注意的是，这里每一个0x01代表一个字节。地址是从低地址到高地址，由于计算机一般都是小端，也就是高位在高地址，低位在低地址。即 `0x01 0x00 0x00 0x00` 其实是 `0x00000001`

```
v5_raw = """
0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x00    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x01    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x01    0x00    0x00    0x00
"""
v5_raw = v5_raw.strip().replace("\n", "    ").split("    ")

v5 = []
tmp_v5 = []

for i in range(0, len(v5_raw), 4):
    curr = int("".join(v5_raw[i:i+4][::-1]).replace("0x", ""), 16)
    tmp_v5.append(curr)

    if len(tmp_v5) == 7:
        v5.append(tmp_v5)
        tmp_v5 = []

v5
```

最终输出为：

```
v5

Out[92]: [[1, 0, 0, 1, 1, 1, 1],
          [1, 0, 1, 1, 0, 0, 1],
          [1, 1, 1, 0, 1, 1, 1],
          [0, 0, 0, 1, 1, 0, 0],
          [1, 1, 1, 1, 0, 0, 0],
          [1, 0, 0, 0, 1, 1, 1],
          [1, 1, 1, 1, 1, 0, 1]]
```

因此，每一步的走向为：sssddwdwdddssaasasaaassddddwdds

运行程序，拿到flag: `UNCTF{ssddwdwdddssaasasaaassddddwdds}`

```
$ ./easy_Maze
Please help me out!
ssddwdwdddssaasasaaassddddwdds
Congratulations!
Thanks! Give you a flag: UNCTF{ssddwdwdddssaasasaaassddddwdds}
```

3. 总结

1. 遇到循环条件是 `7*v10 + v9` 这样的，基本可以猜测v10控制行（上下），v9控制列（左右）
2. 迷宫的形状不一定要写代码或者读代码去逆向，可以直接用动态调试，根据变量在栈中的地址，打印出地址里的内容。不过需要事先判断好迷宫的大小：几行几列，这样才能判断迷宫中的元素。