2018tjctf:validator writeup及gdb调试相关知识点

原创

s0i1 于 2018-08-26 23:59:48 发布
分类专栏： 逆向工程

这个题当时没做出来，因为我当时只熟悉windows下ida和od的使用，不会linux下gdb的逆向相关的使用

看来**多学些东西**还是有些用处的：

思路一：

放进ida找到mian函数一个F5,出来类c代码：

```
int __cdecl main(int argc, const char **input, const char **envp)
{
  char s1[4]; // [esp+10h] [ebp-38h]  ！！！！！关键就在这个4，一个长度为43的字符串赋值给一个长度为4的数组，空间不够
  int v5; // [esp+20h] [ebp-28h]
  int v6; // [esp+24h] [ebp-24h]
  int v7; // [esp+28h] [ebp-20h]
  unsigned int v8; // [esp+3Ch] [ebp-Ch]

  v8 = __readgsdword(0x14u);
  strcpy(s1, "tjctf{ju57_c4ll_m3_r3v3r53_60d_fr0m_n0w_0n}");
  if ( argc == 2 )
  {
    if ( strlen(input[1]) == 43 )
    {
      HIBYTE(v5) = 51;
      LOWORD(v7) = 29235;
      LOBYTE(v6) = 53;
      *(_WORD *)((char *)&v6 + 1) = 13170;
      HIBYTE(v6) = 118;
      if ( !strcmp(s1, input[1]) )
        puts("Valid flag.");
      else
        puts("Invalid flag.");
    }
    else
    {
      puts("Invalid flag.");
    }
  }
  else
  {
    printf("Usage: %s <flag>\n", *input);
  }
  return 0;
}
```

很明显，flag长度是43

迷惑人的地方就在于这个strcpy(s1, "tjctf{ju57_c4ll_m3_r3v3r53_60d_fr0m_n0w_0n}");

这不是拿着我们的输入跟着后边的一串比的吗?**但是，不对！！！**

于是我又想，是不是有哪个操作让咱们输进去的串改变了，

一个简单的想法:　翻转后结果：

tjctf{n0_w0n_m0rf_d06_35r3v3r_3m_ll4c_75uj}

很明显也不是这个

于是我由向上去寻找别的对我们输入的字符串可能有操作的地方，但，很遗憾，没找到。

然后又放到linux下发现竟然跑不了，然后。。

果断放弃，去玩了。

思路二：

后来又看到这道题的解答都是在dbg下的操作才得到结果：

我心想去再看看吧，发现的一个问题：

./validator　　结果出错：No such file or directory

后来看了其他大佬的解决方案，写了一个自己的操作过程：

在这：###https://mp.csdn.net/postedit/82086035###

等可以执行32位程序了：

gdb -q validator

进去定位到main函数中的strcmp指令（断点2）处，和那个把eax赋值为0x2b（43，断点1）的地方下两个断点：

layout regs　看着寄存器

layout asm　看着汇编指令

执行到　　断点1，由于43个字符太难输入了，直接改eax值

set $eax=43

直接比较正确，往下走，到断点2，前一条指令就是一个push eax 肯定所要的值在eax中，

x/43bc 0xffffd190　//一共43个内存单元，以字节为单位,并以字符显示 取出来这个在eax中strcmp的参数的地址的内容

```
0xffffd190:    116 't' 106 'j' 99 'c'  116 't' 102 'f' 123 '{' 106 'j' 117 'u'
0xffffd198:    53 '5'  55 '7'  95 '_'  99 'c'  52 '4'  108 'l' 108 'l' 95 '_'
0xffffd1a0:    109 'm' 51 '3'  95 '_'  51 '3'  53 '5'  114 'r' 51 '3'  118 'v'
0xffffd1a8:    51 '3'  114 'r' 95 '_'  54 '6'  48 '0'  100 'd' 95 '_'  102 'f'
0xffffd1b0:    114 'r' 48 '0'  109 'm' 95 '_'  110 'n' 48 '0'  119 'w' 95 '_'
0xffffd1b8:    48 '0'  110 'n' 125 '}'
```

化简后flag:

tjctf{ju57_c4ll_m3_35r3v3r_60d_fr0m_n0w_0n}

总结：

这道题真坑：ida中f反汇编之后的结果是是作者故意修改掉的数据，图谋不轨，不过确实学到了很多

思路三：

听大佬做法是改一下 call puts //反正我是不知道怎么做，111

##

知识点：

知识：

1

IN AL,21H；表示从21H端口读取一字节数据到AL

IN AX,21H；表示从端口地址21H读取1字节数据到AL，从端口地址22H读取1字节到AH

MOV DX,379H

IN AL,DX ；从端口379H读取1字节到AL

OUT 21H,AL；将AL的值写入21H端口

OUT 21H,AX；将AX的值写入端口地址21H开始的连续两个字节。（port[21H]=AL,port[22h]=AH）

MOV DX,378H

OUT DX,AX ；将AH和AL分别写入端口379H和378H

2

字符串翻转：

result = s[::-1]

或者：

l = list(s)

result = "".join(l.reverse())

汇编：

movl

mov long ： 字长传送 ： 32位

movw

mov word：字传送 ：16位

movb

mov byte：字节传送 ：8位

md5和sha1加密：是不可逆的，就好像2+3=5，但是你不知道5是有1+4所得，还是3+2所得，SHA1更安全。

test eax,eax 基本上和 and eax,eax 是一样的，不同的是test 不改变eax的结果

如果100h地址的值是5

mov eax 100h

mov eax [eax] //将100h中的值5赋给eax，即eax = 5

另一种：

mov eax 100h

mov [eax] 1234 //将100h地址处的值赋为1234

push offset s //将s所占大小的偏移量压入堆栈，用于访问字符串

3 Manual operand 手工操作数，在ida中可以修改汇编指令的操作数

4 gdb调试附上地址 ###https://blog.csdn.net/changer_WE/article/details/82086182######

##