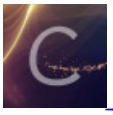
JCTF2014逆向题目小菜两碟writeup

原创

iqiqiya 于 2018-10-25 21:25:53 发布 922 收藏
分类专栏： 我的逆向之路 我的CTF之路 ------汇编学习 我的CTF进阶之路 文章标签： JCTF2014小菜两碟writeup JCTF2014小菜两碟 小菜两碟writeup


我的逆向之路 同时被 3 个专栏收录
108 篇文章 10 订阅
订阅专栏


我的CTF之路
92 篇文章 5 订阅
订阅专栏


------汇编学习
4 篇文章 0 订阅
订阅专栏

拿到文件时 以为和第一题 小菜一碟一样 又是一个.apk文件

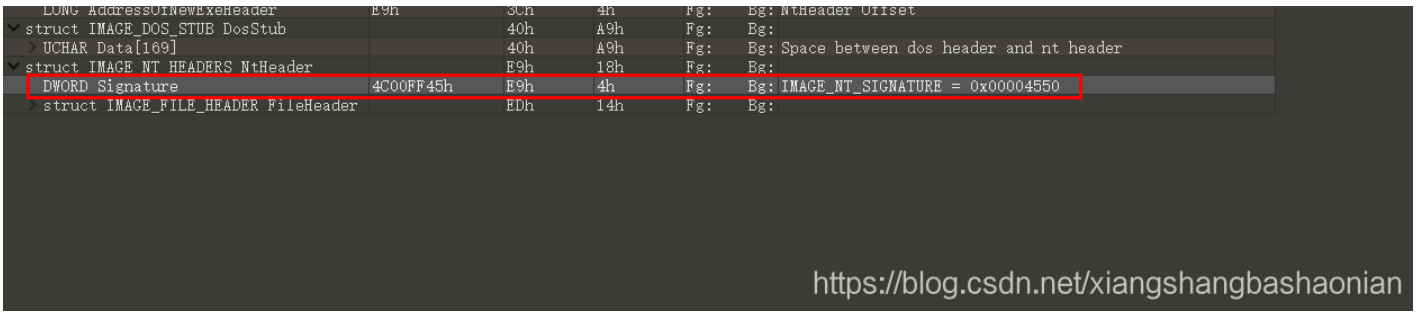
结果安卓模拟器安装不了 发现不对劲 用winhex打开发现是一个PE文件

于是加上.exe后缀 结果无法运行

Exeinfo PE载入发现不是一个PE文件 猜测是做了手脚

用010Editor打开 运行模板 EXE.bt

发现PE头有问题 50 45 FF 00不对啊 正常应该是50 45 00 00才对



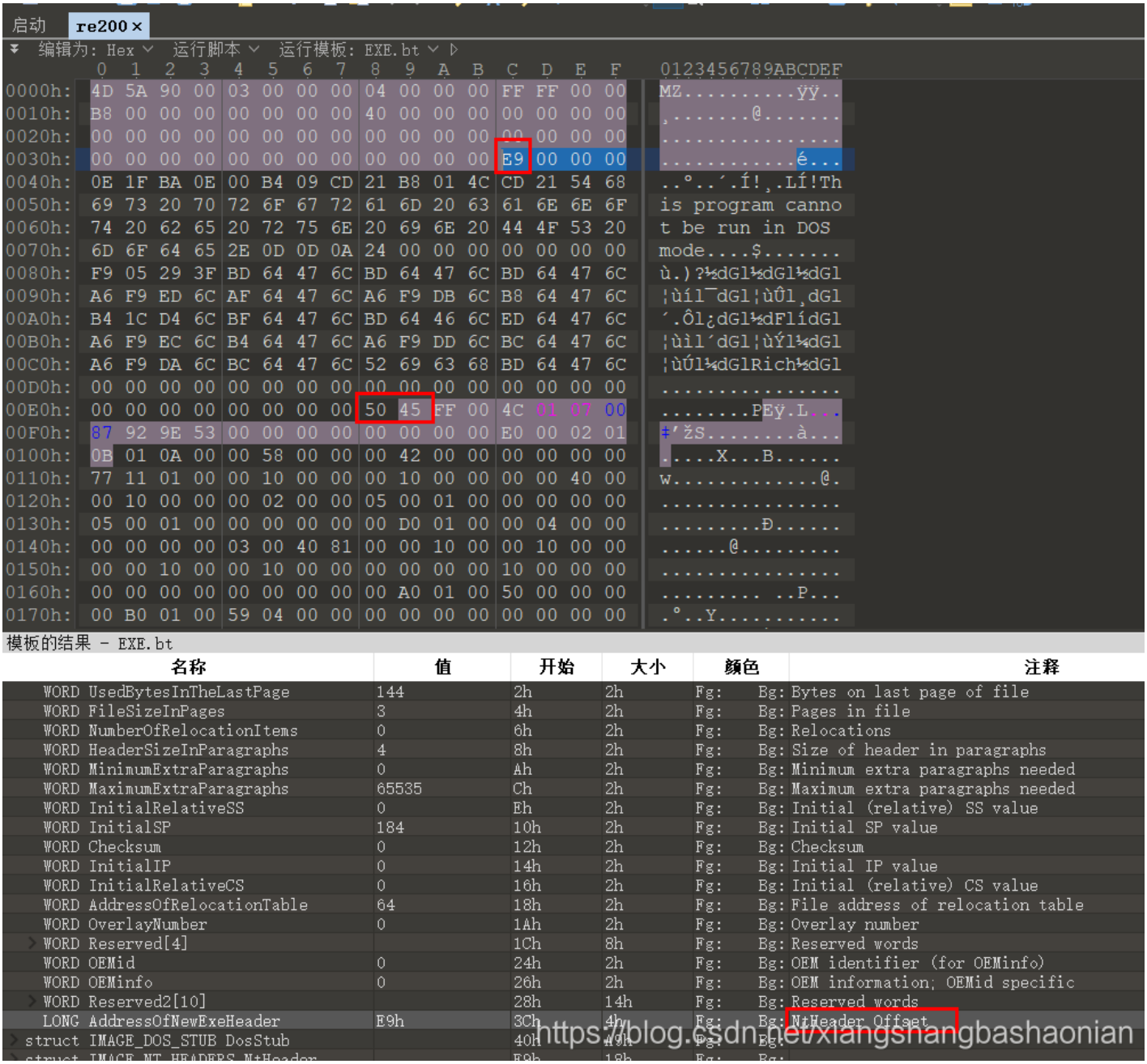
> 原因：struct IMAGE_NT_HEADERS NtHeader结构体中的DWORD Signature就是用来标识该文件是否是PE文件
>
> 该部分占用4字节,也就是“50 45 00 00”
>
> 该标识符在Winnt.h中也有宏定义，如下：

```
#define IMAGE_NT_SIGNATURE                  0x00004550  // PE00
```

那么还有一个问题就是模板 没有正确标识50 45 FF 00这四个字节所在位置 (ps:有底色的部分是被标识到的)

这个是因为DOS头最后一个字段e_lfanew有问题

> e_lfanew : 这个字段保存着PE头[IMAGE_NT_HEADERS]的起始位置偏移
>
> 这个文件中正确起始位置应该是E8h才对



好啦

修改好上述两个位置之后 保存成re200.exe就可以正常运行了

(ps：我的win10系统有毒 还得加上msvcp100d.dll msvcr100d.dll这两个dll才可以运行 2333....)

载入IDA进行分析 可以确定关键代码就在main_0()这个函数中

可以F5反汇编一下

但里边变量有点乱 所以有些我重命名了

根据栈中存放顺序 调整名字 F5重新分析

v20 21 22 23 24 25分别对应以下四个变量

```
-0000005C var_5C          dd ?
-00000058 var_58          dd ?
-00000054 var_54          dd ?
-00000050 var_50          dd ?
-0000004C var_4C          dd ?
-00000048 var_48          dd ?
```

最后得到

```
__int64 main_0()
{
  int v0; // eax
  __int64 v1; // rax
  int v2; // eax
  int v3; // ST08_4
  int v4; // eax
  int v5; // eax
  int v6; // eax
  int v7; // eax
  int v8; // eax
  int v9; // eax
  int v11; // [esp-10h] [ebp-170h]
  int v12; // [esp-Ch] [ebp-16Ch]
  char *v13; // [esp-8h] [ebp-168h]
  int v14; // [esp-4h] [ebp-164h]
  int *v15; // [esp+Ch] [ebp-154h]
  int k; // [esp+D4h] [ebp-8Ch]
  int j; // [esp+E0h] [ebp-80h]
  int i; // [esp+ECh] [ebp-74h]
  char v19; // [esp+FBh] [ebp-65h]
  int v20; // [esp+104h] [ebp-5Ch]
  int v21; // [esp+108h] [ebp-58h]
  int v22; // [esp+10Ch] [ebp-54h]
  int v23; // [esp+110h] [ebp-50h]
  int v24; // [esp+114h] [ebp-4Ch]
  int v25; // [esp+118h] [ebp-48h]
  char v26; // [esp+14Ch] [ebp-14h]
  int v27; // [esp+14Dh] [ebp-13h]
  int v28; // [esp+151h] [ebp-Fh]
  char v29; // [esp+155h] [ebp-Bh]

  v0 = printf(std::cout, "欢迎来到数字游戏 请输入9个数字");
  std::basic_ostream<char,std::char_traits<char>>::operator<<(v0, std::endl);
  v26 = 0;
  v27 = 0;
  v28 = 0;
  v29 = 0;
  v20 = 0;
  j_memset(&v21, 0, 0x3Cu);                     // memset方法为0x3C大小的内存做初始化(ps:全部用0填充)操作，返回值
  for ( i = 0; i < 9; ++i )                     // 从v20 - v28定义九个变量
    std::basic_istream<char,std::char_traits<char>>::operator>>(std::cin, &v20 + i);
  if ( v22 * v21 * v20 / 11 != 106 )            // 逆过来就是 v22*v21*v20 / 11 == 106
    goto LABEL_31;                              // LABEL_31为失败的地方   所以这样的跳转都不能让它成立
  if ( (v21 ^ v20) != v22 - 4 )                 // 逆过来就是(v21 ^ v20) == v22 - 4
```

```
  if ( (v21 ^ v20) != v22 - 4 )                 // 逆过来就是(v21 ^ v20) == v22 - 4
    goto LABEL_31;
  HIDWORD(v1) = (v22 + v21 + v20) % 100;        // (v22 + v21 + v20) % 100 == 34
  if ( HIDWORD(v1) != 34 )
    goto LABEL_31;                              // v23 ==  80
  if ( v23 == 80 )
  {
    for ( j = 0; j < 3; ++j )                   // 这个for循环可以忽略  与下面的if没啥关系
    {
      HIDWORD(v1) = (j + 1) % 3;
      for ( *(&v26 + j) = *((_BYTE *)&v20 + 4 * HIDWORD(v1)) + *(&v20 + j % 3); ; *(&v26 + j) /= 2 )
      {
        while ( *(&v26 + j) < 33 )
        {
          HIDWORD(v1) = j;
          *(&v26 + j) *= 2;
        }
        if ( *(&v26 + j) <= 126 )
          break;
        v1 = *(&v26 + j);
      }
    }
    if ( v24 == 94 && v25 == 98 )               // v24 == 94 并且 v25 == 98
    {
      for ( k = 3; k < 9; ++k )                 // 这个for循环也没什么用  因为与下面的if没啥关系
      {
        for ( *(&v26 + k) = *(&v26 + (k + 1) % 3) + *(&v26 + k % 3); ; *(&v26 + k) /= 2 )
        {
          while ( *(&v26 + k) < 33 )
            *(&v26 + k) *= 2;
          if ( *(&v26 + k) <= 126 )
            break;
        }
      }
      if ( !j_strcmp(&v26, "*&8P^bP^b") )
      {
        v2 = printf(std::cout, "success!");
        std::basic_ostream<char,std::char_traits<char>>::operator<<(v2, std::endl);
        v14 = std::endl;
        v13 = "abc}";
        v12 = v22;
        v11 = v21;
        v3 = v20;
        v15 = &v11;
        v4 = printf(std::cout, "jlflag{");
        v5 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v4, v3);
        v6 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v5, v11);
        v7 = std::basic_ostream<char,std::char_traits<char>>::operator<<(v6, v12);
        v8 = printf(v7, v13);
        std::basic_ostream<char,std::char_traits<char>>::operator<<(v8, v14);
        sub_4112D0(std::cin, &v19);
        goto LABEL_32;
      }
LABEL_31:
      v9 = printf(std::cout, "please try again!");
      std::basic_ostream<char,std::char_traits<char>>::operator<<(v9, std::endl);
      goto LABEL_32;
    }
  }
LABEL_32:
```

```
  v14 = HIDWORD(v1);
  v13 = 0;
  return *(_QWORD *)&v13;
}
```

python脚本跑下可以得到：

```
for v20 in range(100):
        for v21 in range(100):
                v22 = (v20 ^ v21) + 4
                if v20*v21*v22 // 11 == 106 and (v22 + v21 + v20) % 100 == 34:
                        print v20,v21,v22
'''
猜测最大数字不超过100 得到以下6组结果
6 13 15
6 15 13
13 6 15
13 15 6
15 6 13
15 13 6
'''
```

将他们与v23,v34,v25的值进行组合 再任意加上三个数字(比如1,2,3)

> 6 13 15 80 94 98 ? ? ?
> 6 15 13 80 94 98 ? ? ?
> 13 6 15 80 94 98 ? ? ?
> 13 15 6 80 94 98 ? ? ?
> 15 6 13 80 94 98 ? ? ? 一个一个输入 得到这个是正确的
> 15 13 6 80 94 98 ? ? ?

```
欢迎来到数字游戏 请输入9个数字
15
6
13
80
94
98
1
2
3
success!
jlflag{15613abc}
```

下面是这次学到的汇编指令：

cdq指令它大多出现在除法运算之前。它实际的作用只是把EDX的所有位都设成EAX最高位的值

idiv是有符号数除法指令,完成两个有符号数相除

memset是计算机中C/C++语言函数。将s所指向的某一块内存中的前n个 字节的内容全部设置为ch指定的ASCII值， 第一个值为指定的内存地址，块的大小由第三个参数指定，这个函数通常为新申请的内存做初始化工作， 其返回值为指向s的指针。

函数介绍：

void *memset(void *s, int ch, size_t n);

函数解释：将s中前n个字节 （typedef unsigned int size_t ）用 ch 替换并返回 s 。

memset：作用是在一段内存块中填充某个给定的值，它是对较大的结构体或数组进行清零操作的一种最快方法

参考链接：

https://blog.csdn.net/m0_37812124/article/details/76396566

https://blog.csdn.net/z724133545/article/details/52044670