实验二 PE文件结构分析

1180300829 余涛

# 实验目的

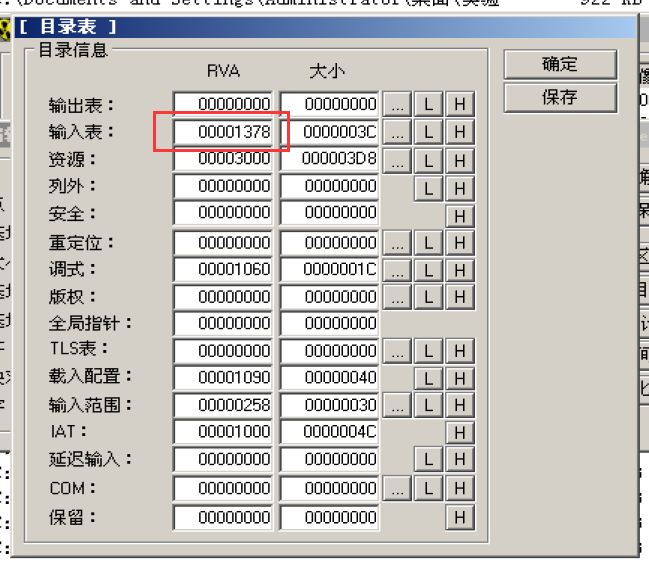
1．了解PE文件的输入表结构；

2．手工解析PE文件的输入表；

3．编程实现PE文件输入表的解析。

# 实验内容

1. **第一步：手动解析输入表结构**
2. 使用工具箱中的工具everything,寻找当前系统中任意一个exe文件，文件名称是： actmovie.exe
3. 使用LordPE“PE编辑器”打开exe文件，确定输入表的RVA，截图如下（图1）：



1. 点击PE编辑器右侧的“位置计算器”，得到文件偏移值，截图如下（图2）：



1. 使用16进制编辑工具，跳转到相应的输入文件偏移地址，输入表是每个IID对应一个DLL，根据IID大小，这里取20字节的数据进行分析，将输入表第一个IID结构的数据与IID结构体的成员一一对应，具体如下所示：

IMAGE\_IMPORT\_DESCRIPTOR {

OriginalFirstThunk = 000013C0

TimeDateStamp = FFFFFFFF

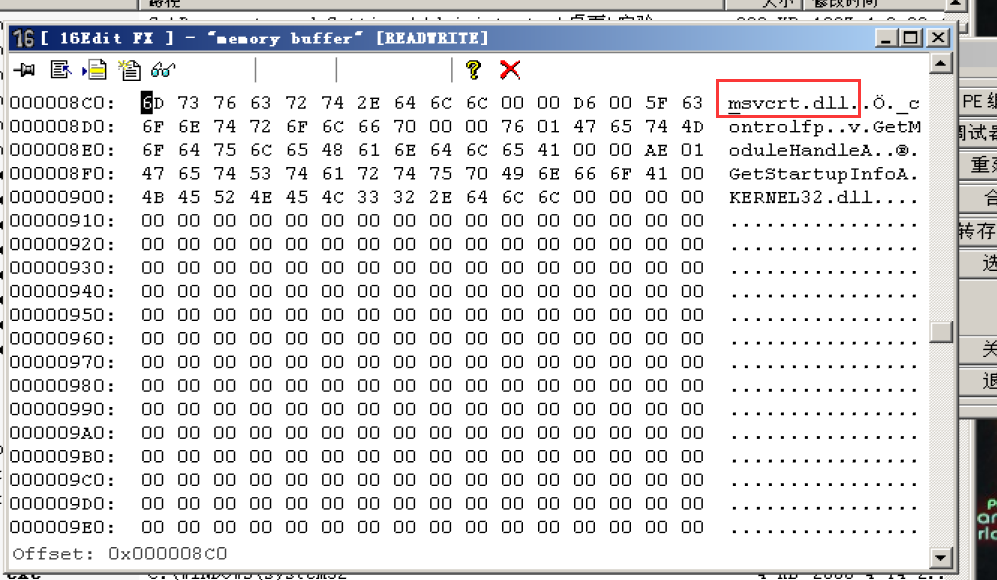
ForwarderChain = FFFFFFFF

Name = 000014C0

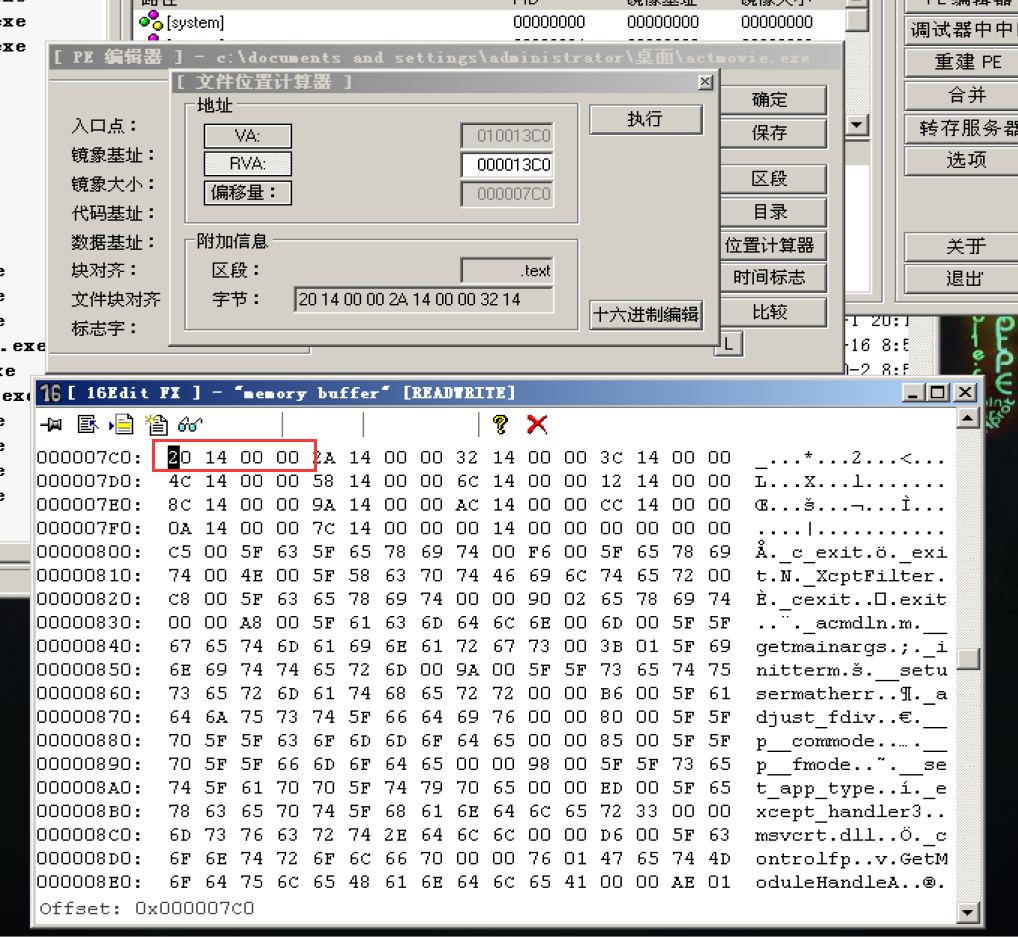
FirstThunk = 0000100C

}

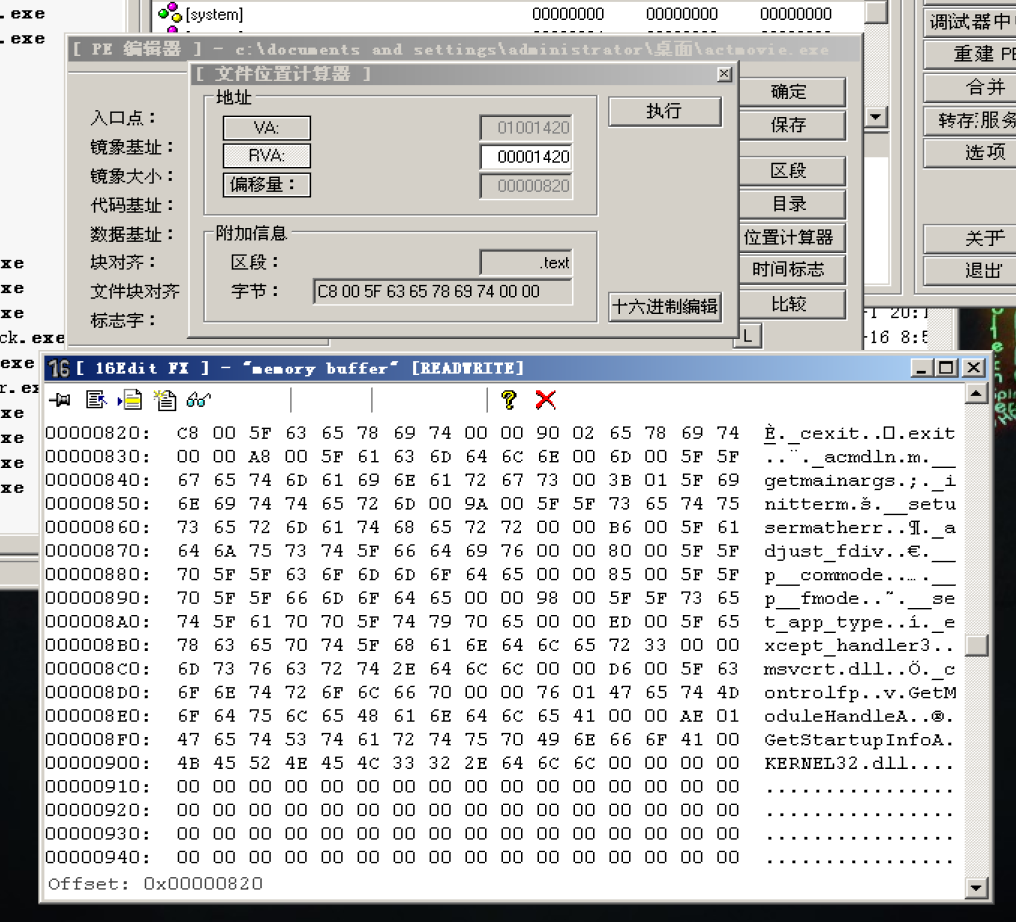
1. 关注OriginalFirstThunk和Name两个成员，其中Name是一个RVA，用步骤**(3)**的方法得到其文件偏移值为 000008C0 ，在16进制编辑工具转到这个偏移地址，可见输入表的第一个DLL名为 msvcrt.dll ，截图如下（图3）：



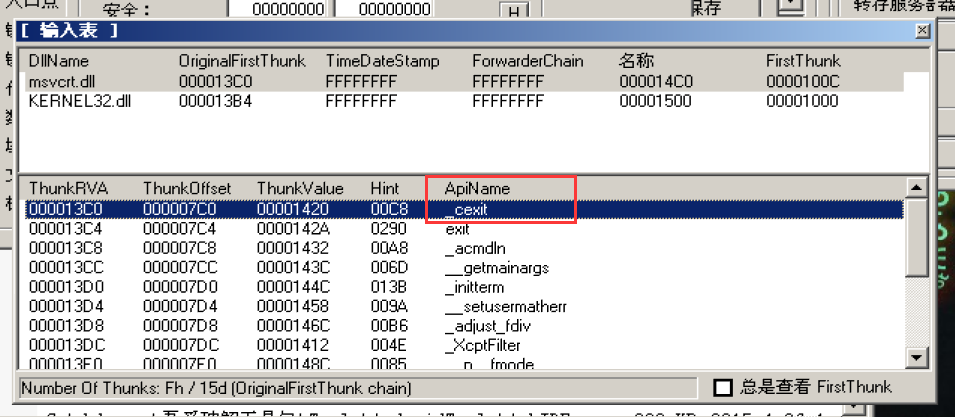
1. 分析一下OriginalFirstThunk，它指向一个类型为IMAGE\_THUNK\_DATA的数组，上面已经分析出了它的值为 000013C0 ，这是一个RVA，用步骤**(3)**的方法得到文件偏移地址 00007C0 。在16进制编辑工具转到这个偏移地址，其中前面4个字节的数据为 63 5F 00 C8 ，截图如下（图4）：



1. 可以看出，这是 以序号 （填“以名字”或“以序号”）的方式输入函数；用与步骤**(3)**相同的方式在16进制编辑工具中对应IMAGE\_IMPORT\_BY\_NAME结构的数据，可以看到函数的输入序号为 20 ，函数名为 cexit ，截图如下（图5）：



1. 验证：使用LordPE单击“目录表”界面中输入表右侧的“…按钮”，打开输入表对话框，可以验证获取的DLL名和函数名是否正确。截图如下（图6）：

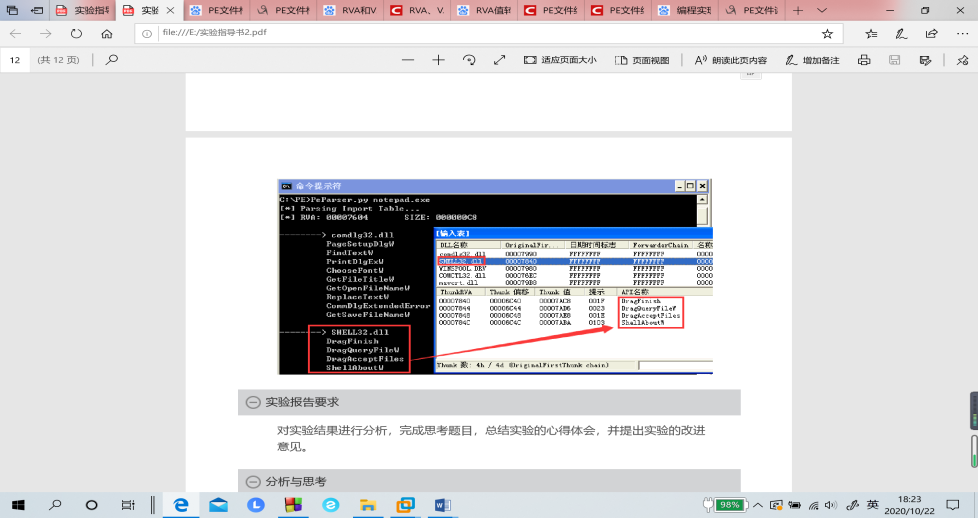


1. **第二步：编程实现输入表的解析**

程序要求：

调用程序解析PE文件输入表，输出**输入表大小，RVA，以及调用的每个dll的名称和相应的调用的函数名称**。

1. 从数据目录表的第二项读取输入表的RVA以及大小，找到第一个IID的文件偏移位置，获取IID的数据，获取IID中Name成员的RVA值和OriginalFirstThunk的RVA值，循环直到得到一个空的IID，表明这是最后一个IID，结束解析循环；
2. 将步骤(1)中获得的Name的RVA转换为文件偏移值，并读取DLL的名字
3. 解析IID对应的INT数组。将步骤(1)中OriginalFirstThunk的RVA值转为文件偏移值，指向一个类型为IMAGE\_THUNK\_DATA的数组，判断输入函数方式。循环获取IID对应的IMAGE\_THUNK\_DATA结构，等于0,表示是最后一个IMAGE\_THUNK\_DATA，结束循环。
4. 运行程序即可看到输入表的解析结果，与LordPE的解析结果是否一致，输出参考与验证截图示例如下：



完成以下函数使程序运行成功，并给出结果截图

Is\_valid\_pe:检查文件合法性并读取数据，主要检查MZ标志和PE标志来校验合法性，随后读取数据目录表到self.data\_dirs，读取节表头到self.sec\_hdrs

parse\_import\_table：输入表结构解析。

rva\_to\_offset：RVA转偏移地址

parse\_iid\_int：解析每个IID对应的IMAGE\_THUNK\_DATA类型的INT数组

参考代码：

# -\*-coding:utf-8-\*-

import sys

import struct

class PeParser:

def \_\_init\_\_(self,file\_path):

self.MZSIG = b'MZ'

self.PESIG = b'PE\0\0'

self.path = file\_path

#将十六进制数据转换为小端格式的数值

def get\_dword(self, data):

return struct.unpack('<L', data)[0]

#提取ASCII字符串

def get\_string(self, ptr):

beg = ptr

while ptr < len(self.data) and self.data[ptr] != 0:

ptr += 1

return self.data[beg:ptr]

def parse(self):

self.read\_data()

if not self.is\_valid\_pe():

print("[Error] Invalid PE file")

self.parse\_import\_table()

#读取文件数据

def read\_data(self):

fd = open(self.path, "rb")

self.data = fd.read()

fd.close()

#检查文件合法性并读取数据

def is\_valid\_pe(self)：

#RVA转偏移地址

def rva\_to\_offset(self,rva):

#输入表结构解析

def parse\_import\_table(self):

# 解析每个IID对应的IMAGE\_THUNK\_DATA类型的INT数组

def parse\_iid\_int(self,ptr):

if \_\_name\_\_ == "\_\_main\_\_":

if len(sys.argv) == 2:

p = PeParser(sys.argv[1])

p.parse()

实验代码：

**import** sys  
**import** struct  
  
  
**class** PeParser:  
  
 **def** \_\_init\_\_(self, file\_path):  
 self.MZSIG = **b'MZ'** self.PESIG = **b'PE\0\0'** self.path = file\_path  
  
 *# 将十六进制数据转换为小端格式的数值* **def** get\_dword(self, data):  
 **return** struct.unpack(**'<L'**, data)[0]  
  
 *# 提取ASCII字符串* **def** get\_string(self, ptr):  
 beg = ptr  
 **while** ptr < len(self.data) **and** self.data[ptr] != 0:  
 ptr += 1  
 **return** self.data[beg:ptr]  
  
 **def** parse(self):  
 self.read\_data()  
 **if not** self.is\_valid\_pe():  
 print(**"[Error] Invalid PE file"**)  
 self.parse\_import\_table()  
  
 *# 读取文件数据* **def** read\_data(self):  
 fd = open(self.path, **"rb"**)  
 self.data = fd.read()  
 fd.close()  
  
 *# 检查文件合法性并读取数据* **def** is\_valid\_pe(self):  
 temp\_ptr = self.get\_dword(self.data[0x3c:0x40]) *#3C表示是指偏移量,e\_lfanew* **if** self.PESIG == self.data[temp\_ptr:temp\_ptr + 4]:  
 **return True  
 else**:  
 **return False** *# RVA转偏移地址* **def** rva\_to\_offset(self, rva):  
 h32\_size\_ptr = self.get\_dword(self.data[0x3c:0x40]) + 0x14 *#F4* h32\_size = self.get\_dword(self.data[h32\_size\_ptr:h32\_size\_ptr + 2] + **b'\x00\x00'**) *#IMAGE\_OPTIONAL\_HEADER E0* temp\_rva = self.get\_dword(self.data[0x3c:0x40]) + 0x18 + h32\_size *#1D8* **while True**:  
 **if** self.get\_dword(self.data[temp\_rva + 0xc:temp\_rva + 0x10]) + self.get\_dword( self.data[temp\_rva + 0x10:temp\_rva + 0x14]) > rva **and** self.get\_dword( self.data[temp\_rva + 0xc:temp\_rva + 0x10]) <= rva:  
 **return** rva + self.get\_dword(self.data[temp\_rva + 20:temp\_rva + 24]) - self.get\_dword( self.data[temp\_rva + 12:temp\_rva + 16])  
 temp\_rva += 40  
  
  
  
 *# 输入表结构解析* **def** parse\_import\_table(self):  
 self.pe\_rva = self.get\_dword(self.data[0x3c:0x40]) + 0x80  
 self.import\_table\_rva = self.get\_dword(self.data[self.pe\_rva:self.pe\_rva + 4]) *#000012D0* print(**"%x"**%self.import\_table\_rva)  
 self.import\_table\_size = self.get\_dword(self.data[self.pe\_rva + 4:self.pe\_rva + 8]) *#00000078* print(**"%x"**%self.import\_table\_size)  
 self.import\_table\_offset = self.get\_dword(self.data[self.pe\_rva + 8:self.pe\_rva + 12]) *#00000000* print(**"%x"**%self.import\_table\_offset)  
 print(**"rva:\t%d"** % self.import\_table\_rva)  
 print(**"size:\t%d"** % self.import\_table\_size)  
 print()  
  
 self.iid\_list = []  
 ptr\_temp = self.rva\_to\_offset(self.import\_table\_rva)  
 print(**"%x"**%ptr\_temp)  
 **while True**:  
 iid\_list\_temp = []  
 iid\_temp = self.get\_dword(self.data[ptr\_temp:ptr\_temp + 4])  
 **if** iid\_temp == 0:  
 **break** iid\_list\_temp.append(iid\_temp)  
 temp\_name=self.get\_string(self.rva\_to\_offset(self.get\_dword(self.data[ptr\_temp+12:ptr\_temp+16])))*#获得Name* iid\_list\_temp.append(temp\_name)  
 self.iid\_list.append(iid\_list\_temp)  
 ptr\_temp += 20  
 **for** i **in** range(len(self.iid\_list)):  
 print(str(self.iid\_list[i][1], encoding=**"UTF-8"**))  
 self.parse\_iid\_int(self.iid\_list[i][0])  
  
 *# 解析每个IID对应的IMAGE\_THUNK\_DATA类型的INT数组* **def** parse\_iid\_int(self, ptr):  
 *#处理FirstThunk及其之后的函数* ptr\_temp = self.rva\_to\_offset(ptr)  
 **while True**:  
 name\_temp = self.get\_dword(self.data[ptr\_temp:ptr\_temp + 4])  
 **if** name\_temp == 0:  
 **break** print(**"\t"** + str(self.get\_string(self.rva\_to\_offset(name\_temp) + 2), encoding=**"UTF-8"**))  
 ptr\_temp += 4  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **if** len(sys.argv) == 2:  
 p = PeParser(sys.argv[1])  
 p.parse()

结果截图：

