实验二 PE文件结构分析

# 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： AddInProcess.exe
3. 使用LordPE“PE编辑器”打开exe文件，确定可选头部大小是：00E0（00F0则是64位文件，请返回第一步重新选择exe文件），点击“目录”，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 = 00 00 6D E8

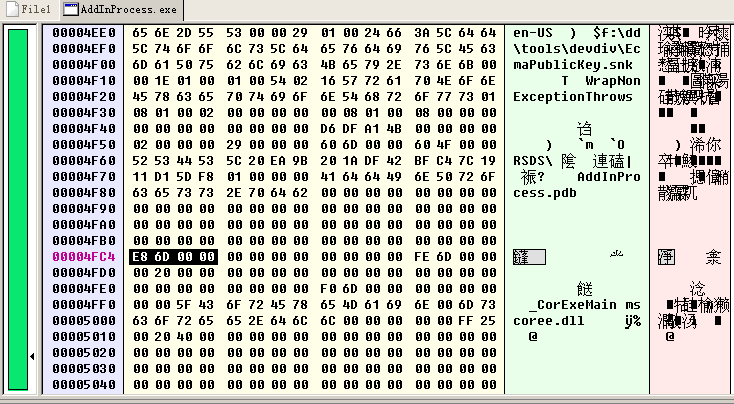
TimeDateStamp = 00 00 00 00

ForwarderChain = 00 00 00 00

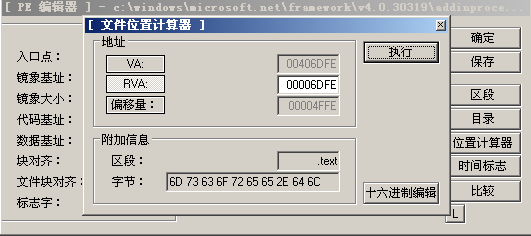
Name = 00 00 6D FE

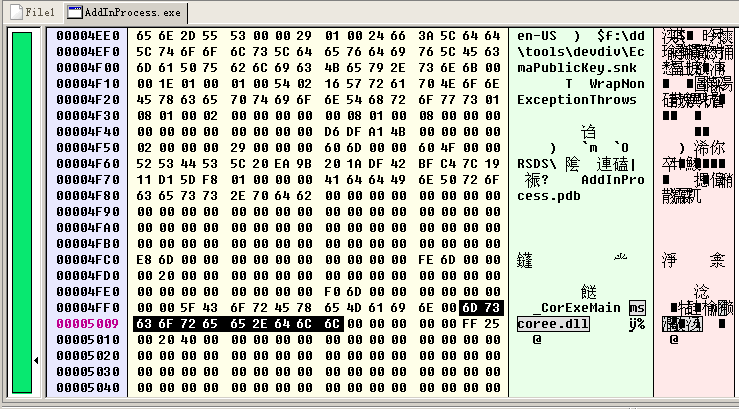
FirstThunk = 00 00 20 00

}

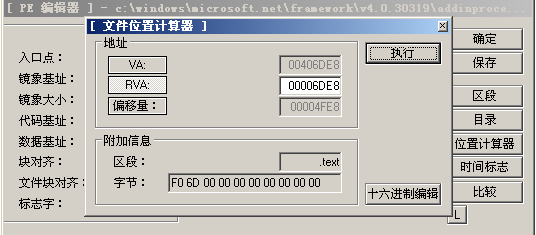


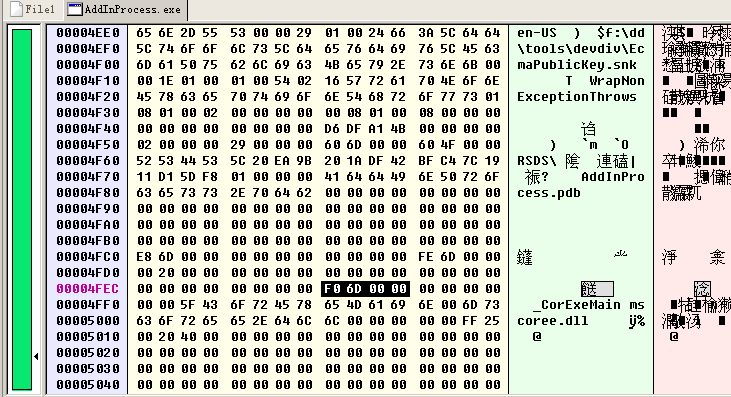
1. 关注OriginalFirstThunk和Name两个成员，其中Name是一个RVA，用步骤**(3)**的方法得到其文件偏移值为 00 00 4F FE ，在16进制编辑工具转到这个偏移地址，可见输入表的第一个DLL名为 mscoree.dll ，截图如下（图3）：



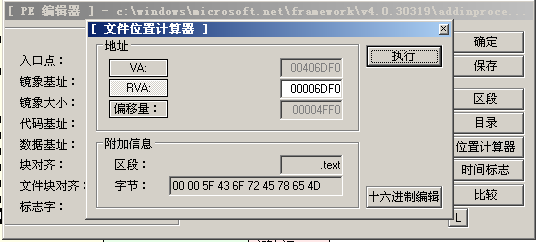


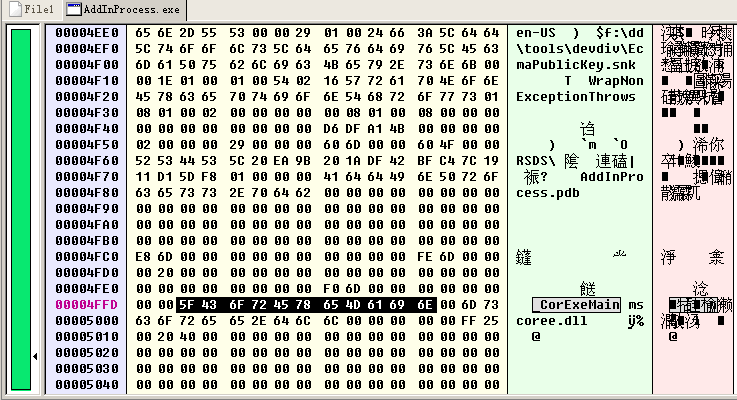
1. 分析一下OriginalFirstThunk，它指向一个类型为IMAGE\_THUNK\_DATA的数组，上面已经分析出了它的值为 00 00 6D E8 ，这是一个RVA，用步骤**(3)**的方法得到文件偏移地址 00 00 4F E8 。在16进制编辑工具转到这个偏移地址，其中前面4个字节的数据为 00 00 6D F0 ，截图如下（图4）：



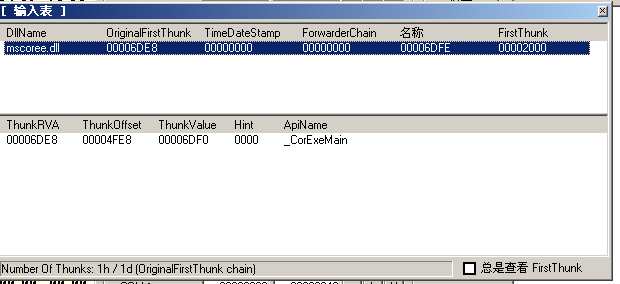


1. 可以看出，这是 以名字 （填“以名字”或“以序号”）的方式输入函数；
2. 以名字的方式则用与步骤**(3)**相同的方式在16进制编辑工具中对应IMAGE\_IMPORT\_BY\_NAME结构的数据，可以看到函数的输入序号为 0 ，函数名为 \_CorExeMain ，截图如下（图5）：





1. ~~以序号的方式则可知道其调用的是系统内 号函数（转十进制）~~
2. 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/序号值是否正确。截图如下（图6）：

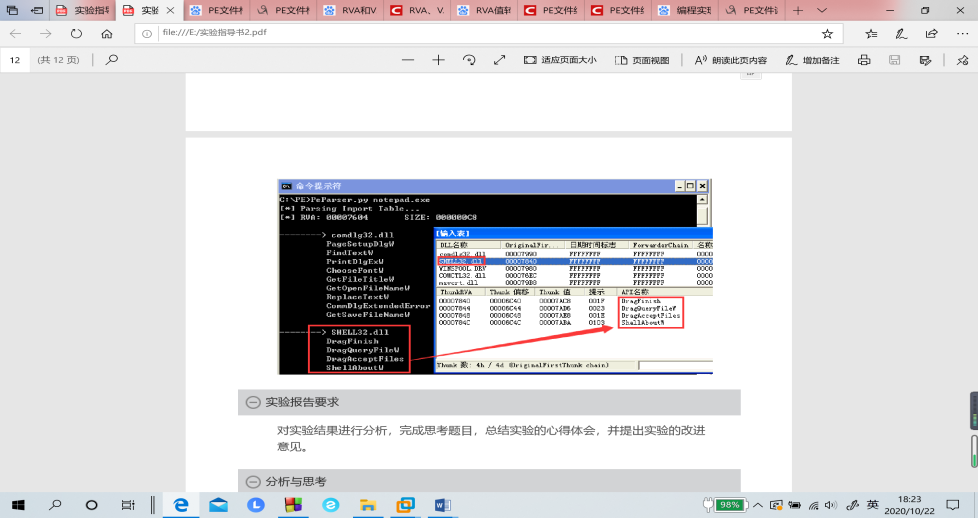


1. **第二步：编程实现输入表的解析**

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：

# -\*-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):  
 e\_magic = self.data[0:2]  
  
 if e\_magic != self.MZSIG:  
 return False  
 else:  
 e\_lfanew = self.get\_dword(self.data[60:64])  
 Machine = self.data[e\_lfanew:e\_lfanew + 4]  
  
 if Machine != self.PESIG:  
 return False  
 else:  
 return True  
  
 # RVA转偏移地址  
 def rva\_to\_offset(self, rva):  
 # 取PE头地址RVA  
 PE\_header\_RVA = self.get\_dword(self.data[60:64])  
 # 取可选头大小，用于跳过可选头找到节表  
 SizeOfOptionalHeader = self.get\_dword(self.data[PE\_header\_RVA + 0x14:PE\_header\_RVA + 0x14 + 2] + b"\x00\x00")  
 IMAGE\_SECTION\_HEADER = PE\_header\_RVA + SizeOfOptionalHeader + 0x18  
 NumberOfSections = self.get\_dword(self.data[PE\_header\_RVA + 2:PE\_header\_RVA + 4] + b"\x00\x00")  
  
 # section\_ptr用于遍历所有节，找到RVA位于哪个节  
 section\_ptr = IMAGE\_SECTION\_HEADER  
 for i in range(NumberOfSections + 1):  
 # 若RVA位于当前节内 FOA = PointerToRawData + (VRA- VirtualAddress)  
 if rva <= self.get\_dword(self.data[section\_ptr + 0x10:section\_ptr + 0x14]) + self.get\_dword(  
 self.data[section\_ptr + 0xC: section\_ptr + 0x10]) and rva >= self.get\_dword(  
 self.data[section\_ptr + 0xc:section\_ptr + 0x10]):  
 return rva + self.get\_dword(self.data[section\_ptr + 0x14:section\_ptr + 0x18]) - self.get\_dword(  
 self.data[section\_ptr + 0xc:section\_ptr + 0x10])  
 section\_ptr = section\_ptr + 0x28  
 return 0  
  
 # 输入表结构解析  
 def parse\_import\_table(self):  
  
 PE\_header\_RVA = self.get\_dword(self.data[60:64])  
 import\_descriper\_RVA = self.get\_dword(self.data[PE\_header\_RVA + 0x80:PE\_header\_RVA + 0x80 + 4])  
 IID\_list\_ptr = self.rva\_to\_offset(import\_descriper\_RVA)  
  
 import\_size = self.get\_dword(self.data[PE\_header\_RVA + 0x80 + 4:PE\_header\_RVA + 0x80 + 8])  
 print("RVA:0x%x" % import\_descriper\_RVA + "\t" + str(import\_descriper\_RVA))  
 print("Size:0x%x" % import\_size + "\t" + str(import\_size))  
  
 IID\_list = [] # 用于保存IID  
 # 遍历直到IID 对应 INT\_list == 0  
 while True:  
 IID\_struct = [] # IID数组结构: [DDL\_name,INT\_RVA]  
  
 # 处理 name  
 name\_RVA = self.get\_dword(self.data[IID\_list\_ptr + 12:IID\_list\_ptr + 16])  
 name\_str = self.rva\_to\_offset(name\_RVA)  
 name = self.get\_string(name\_str)  
 IID\_struct.append(name)  
  
 # 处理INT RVA  
 INT\_list = self.get\_dword(self.data[IID\_list\_ptr:IID\_list\_ptr + 4])  
  
 if INT\_list == 0:  
 break  
  
 IID\_struct.append(INT\_list)  
 # 添加入IID LIST  
 IID\_list.append(IID\_struct)  
 IID\_list\_ptr += 20  
  
 # 遍历解析每个IID的INT数组  
 for i in range(len(IID\_list)):  
 print((str(IID\_list[i][0], encoding='UTF-8')))  
 self.parse\_iid\_int(IID\_list[i][1])  
  
 # 解析每个IID对应的IMAGE\_THUNK\_DATA类型的INT数组  
 def parse\_iid\_int(self, ptr):  
 INT = self.rva\_to\_offset(ptr)  
 while True:  
 INT\_name = self.get\_dword(self.data[INT: INT + 4])  
 if INT\_name == 0:  
 break  
 print("\t" + str(self.get\_string(self.rva\_to\_offset(INT\_name) + 2), encoding='UTF-8'))  
 INT += 4  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) == 2:  
 p = PeParser(sys.argv[1])  
 p.parse()

结果截图：

