实验二 PE文件结构分析

# 实验目的

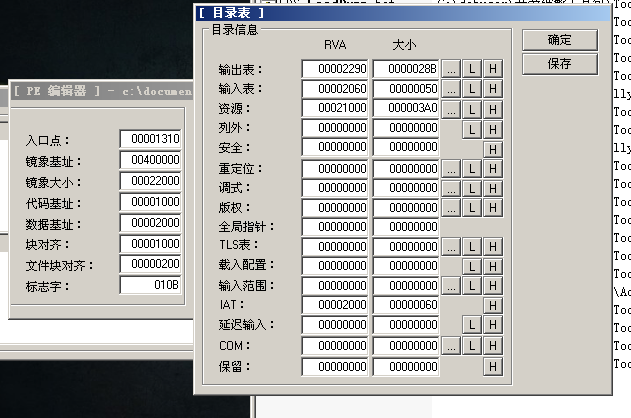
1．了解PE文件的输入表结构；

2．手工解析PE文件的输入表；

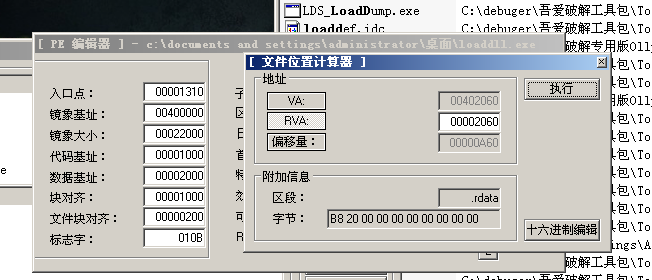
3．编程实现PE文件输入表的解析。

# 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： Loaddll.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 = 000020B8

TimeDateStamp = 00000000

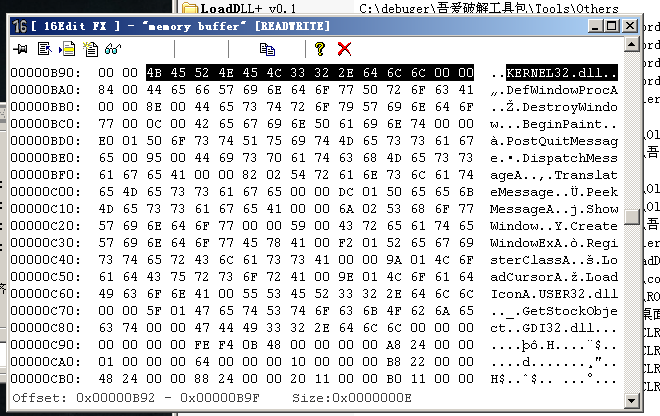
ForwarderChain = 00000000

Name = 00002192

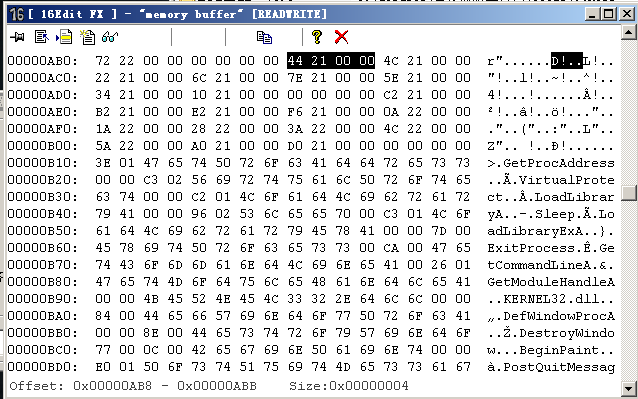
FirstThunk = 00002008

}

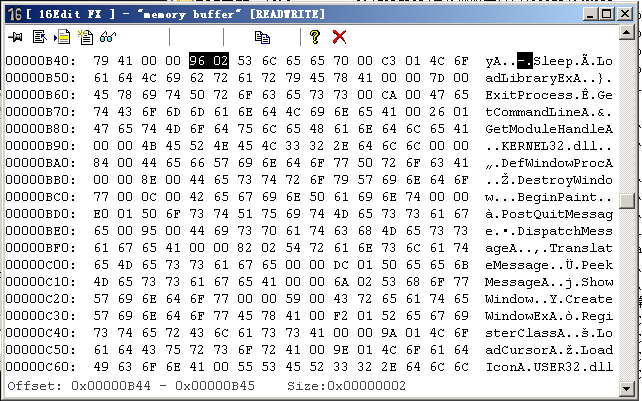
1. 关注OriginalFirstThunk和Name两个成员，其中Name是一个RVA，用步骤**(3)**的方法得到其文件偏移值为 000000B92 ，在16进制编辑工具转到这个偏移地址，可见输入表的第一个DLL名为 kenel32.dll ，截图如下（图3）：



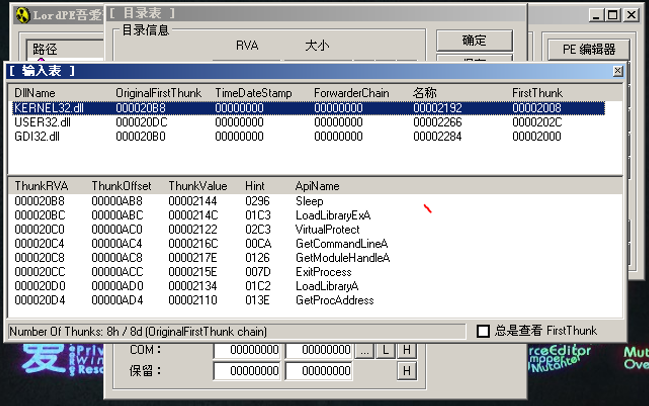
1. 分析一下OriginalFirstThunk，它指向一个类型为IMAGE\_THUNK\_DATA的数组，上面已经分析出了它的值为 000020B8 ，这是一个RVA，用步骤**(3)**的方法得到文件偏移地址 00000AB8 。在16进制编辑工具转到这个偏移地址，其中前面4个字节的数据为 00002144 ，截图如下（图4）：



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；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 0296 ，函数名为 Sleep ，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）：

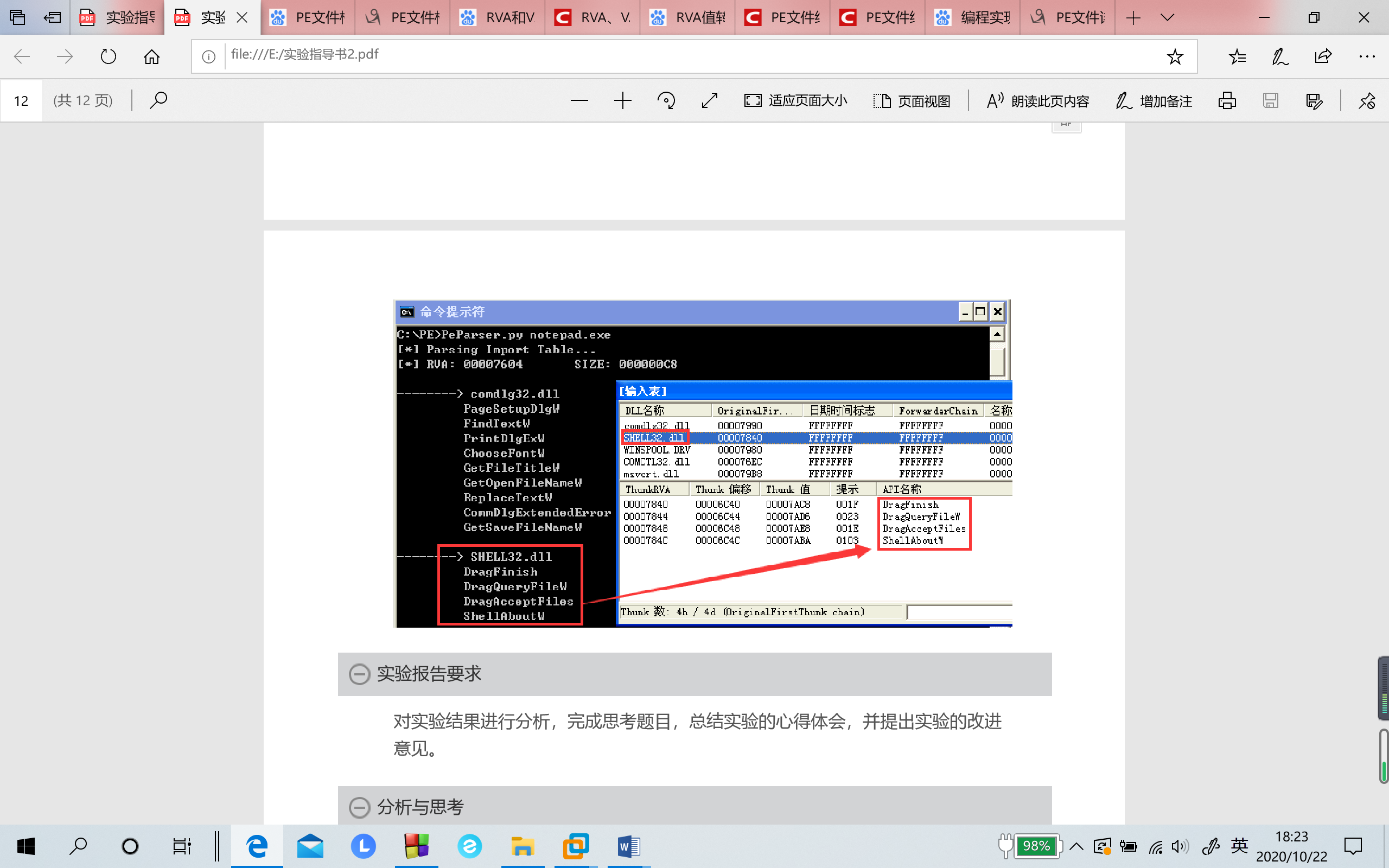


1. **第二步：编程实现输入表的解析**

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:

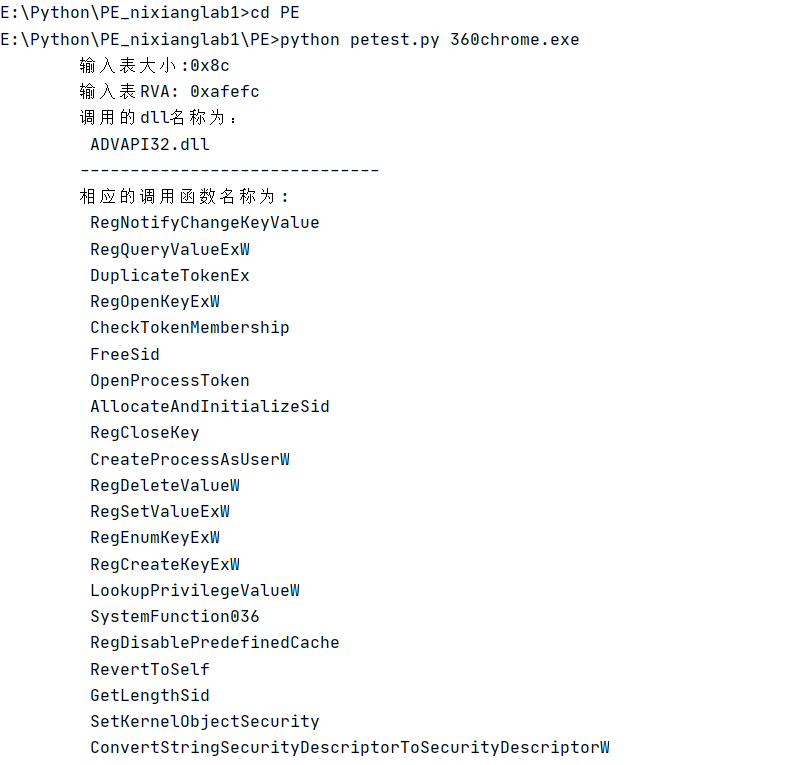
p = PeParser(sys.argv[1])

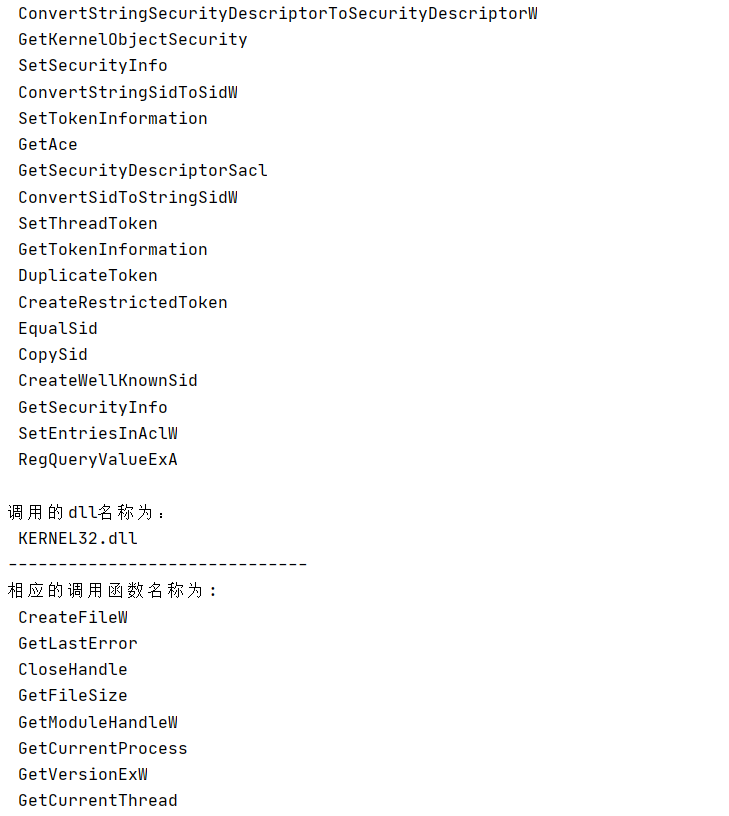
p.parse()

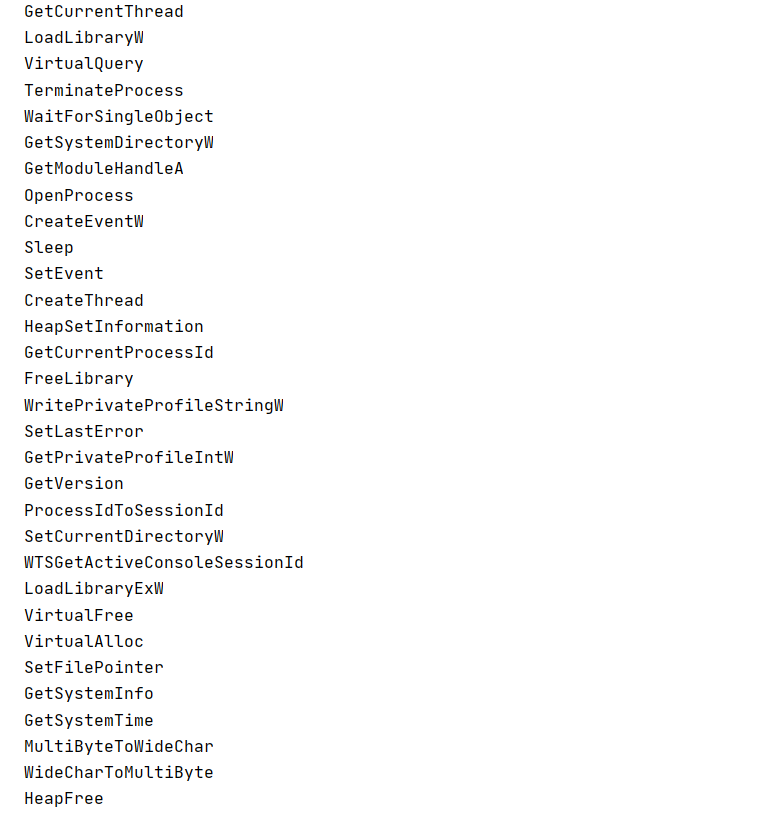
实验代码：

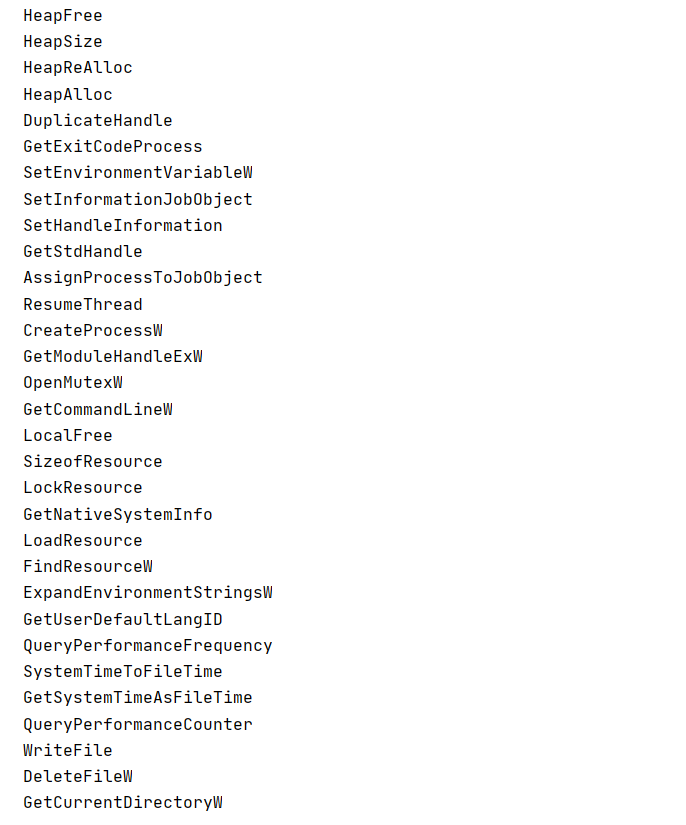
*# -\*-coding:utf-8-\*-*import sys  
import struct  
class PeParser:  
 def \_\_init\_\_(self, file\_path):  
 self.MZSIG = b'MZ'  
 self.PESIG = b'PE\0\0'  
 self.path = file\_path  
  
 *# 将十六进制数据转换为小端格式的数值* def get\_dword(self, data):  
 return struct.unpack('<L', data)[0]  
  
 *# 提取ASCII字符串* def 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):  
 dos\_header\_data = self.data[:2]  
 if not dos\_header\_data == self.MZSIG:  
 print("")  
 return False  
 self.dos\_nt\_rva = self.get\_dword(self.data[60:64])  
 dos\_nt\_signature = self.data[self.dos\_nt\_rva:self.dos\_nt\_rva + 4]  
 self.numberofsections = struct.unpack('<h', self.data[self.dos\_nt\_rva + 6:self.dos\_nt\_rva + 8])[0]  
 self.sizeofoptionheader = struct.unpack('<h', self.data[self.dos\_nt\_rva + 0x14:self.dos\_nt\_rva + 0x16])[  
 0] *# header大小* self.numberofRvaAndSizes = self.get\_dword(self.data[self.dos\_nt\_rva + 0x74:self.dos\_nt\_rva + 0x78])  
 self.sectionrva = self.dos\_nt\_rva + self.sizeofoptionheader + 0x18  
 self.import\_RVA = self.get\_dword(self.data[self.dos\_nt\_rva + 0x80:self.dos\_nt\_rva + 0x84])  
 self.import\_Size = self.get\_dword(self.data[self.dos\_nt\_rva + 0x84:self.dos\_nt\_rva + 0x88])  
 self.solvesection()  
 if not dos\_nt\_signature == self.PESIG:  
 return False  
 return True  
  
 *# RVA转偏移地址* def rva\_to\_offset(self, rva):  
 for i in range(self.numberofsections - 1):  
 col1 = self.section[i]  
 col2 = self.section[i + 1]  
 if rva < col2[1] and rva > col1[1]:  
 return rva - col1[1] + col1[3]  
  
 def solvesection(self):  
 self.section = []  
 ptr = self.sectionrva  
 *# print(hex(ptr))* for i in range(self.numberofsections):  
 col = []  
 col.append(self.data[ptr + 40 \* i:ptr + 8 + 40 \* i])  
 col.append(self.get\_dword(self.data[ptr + 12 + 40 \* i:ptr + 16 + 40 \* i])) *# rva* col.append(self.get\_dword(self.data[ptr + 16 + 40 \* i:ptr + 20 + 40 \* i])) *# 大小* col.append(self.get\_dword(self.data[ptr + 20 + 40 \* i:ptr + 24 + 40 \* i])) *# 区块偏移* self.section.append(col)  
  
 *# 输入表结构解析* def parse\_import\_table(self):  
 print("\t输入表大小:" + str(hex(self.import\_Size)))  
 print("\t输入表RVA: " + hex(self.import\_RVA))  
 i = 0  
 offect = self.rva\_to\_offset(self.import\_RVA)  
 ptr = self.data[offect:offect + 0x14 + 1]  
  
 import\_data = []  
  
 while not ptr == b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00':  
  
 i = i + 1  
 col = []  
 col.append(self.get\_dword(ptr[0:4]))  
 col.append(self.get\_dword(ptr[4:8]))  
 col.append(self.get\_dword(ptr[8:12]))  
 col.append(self.get\_dword(ptr[12:16]))  
 col.append(self.get\_dword(ptr[16:20]))  
 import\_data.append(col)  
  
 ptr = self.data[offect + 0x14 \* i:offect + 0x14 \* i + 0x14]  
 for j in range(i):  
 col = import\_data[j]  
  
 print("\t调用的dll名称为：")  
 print("\t " + self.get\_string(self.rva\_to\_offset(col[3])).decode('utf-8'))  
 print('\t------------------------------')  
 print("\t相应的调用函数名称为:")  
 self.parse\_iid\_int(self.rva\_to\_offset(col[0]))  
 print('')  
  
 *# 解析每个IID对应的IMAGE\_THUNK\_DATA类型的INT数组* def parse\_iid\_int(self, ptr):  
 i = 0  
  
 Drva = self.get\_dword(self.data[ptr:ptr + 4])  
  
  
 while not Drva == 0:  
 i = i + 1  
 rva = int(self.rva\_to\_offset(Drva))  
 print("\t " + self.get\_string(rva+2).decode('utf-8'))  
 Drva = self.get\_dword(self.data[ptr + i \* 4:ptr + 4 \* i + 4])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 if len(sys.argv) == 2:  
 p = PeParser(sys.argv[1])  
 p.parse()

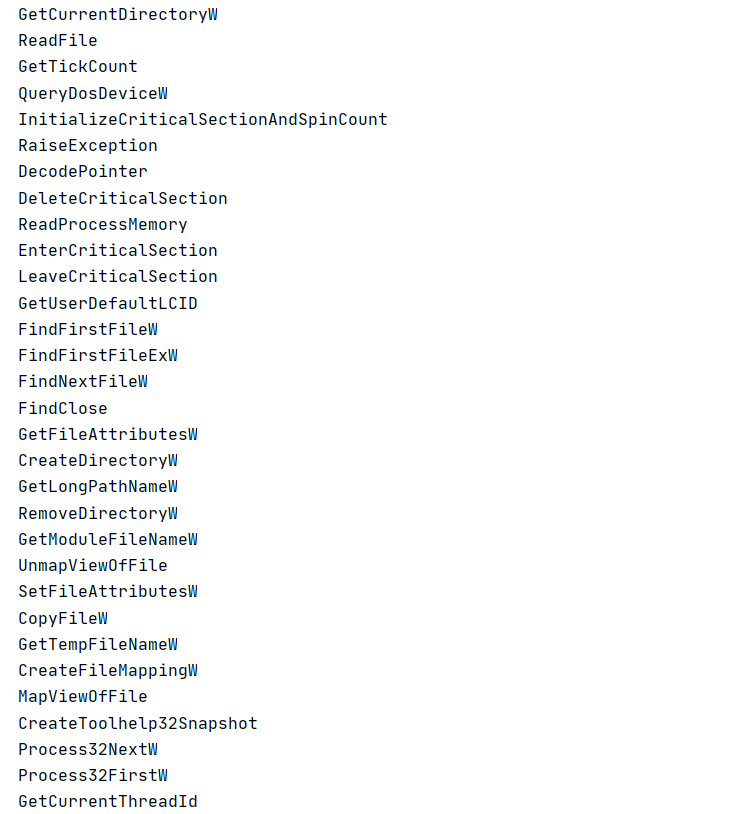
结果截图：



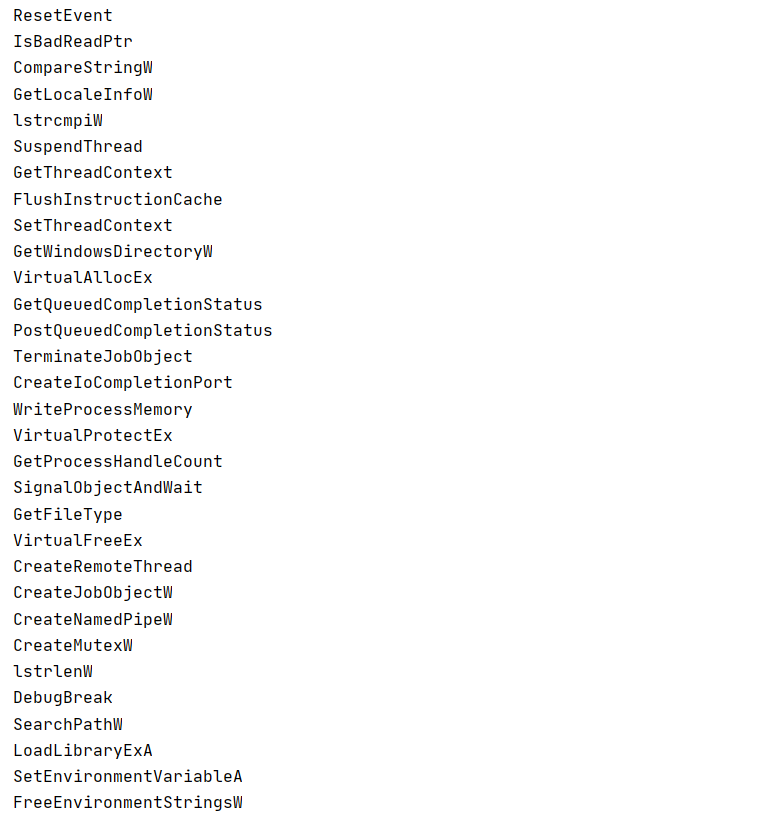


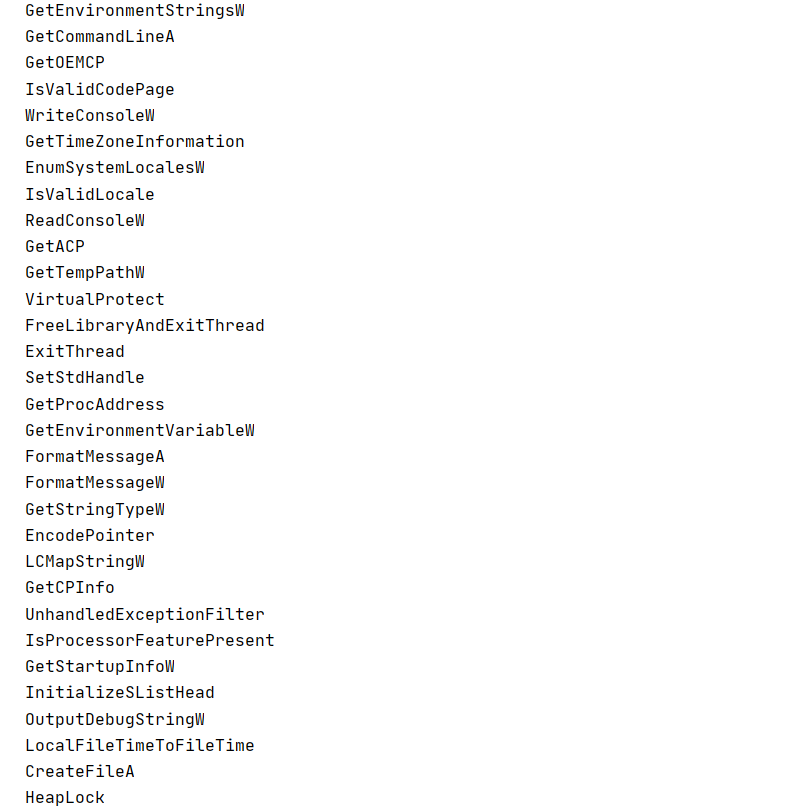


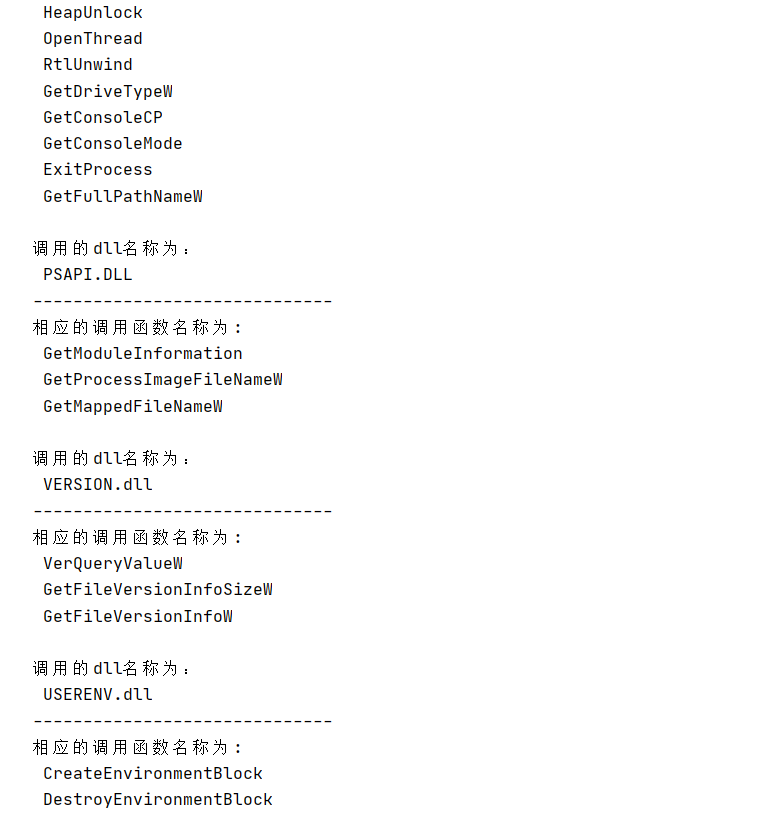


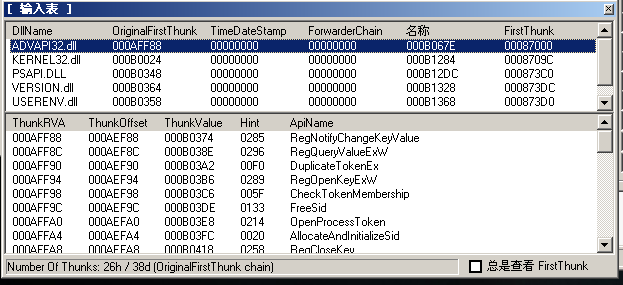


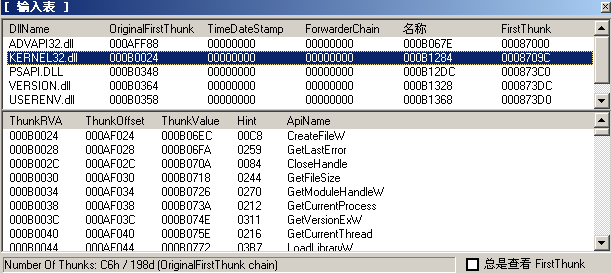


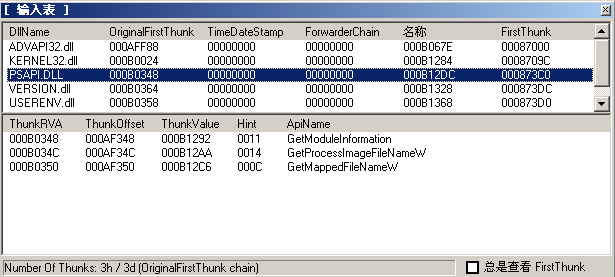


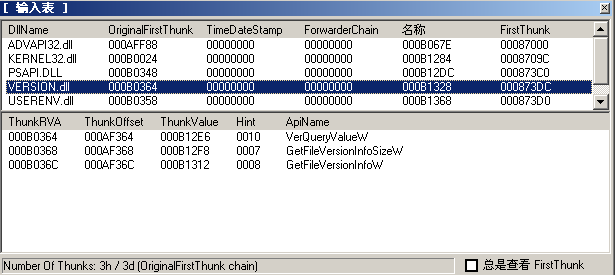


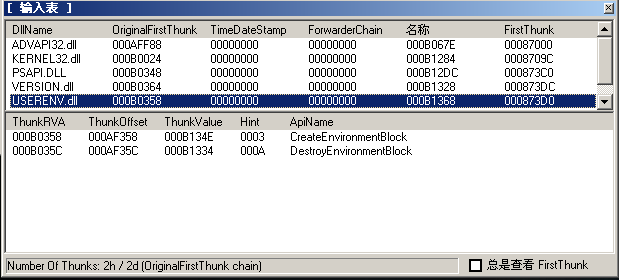


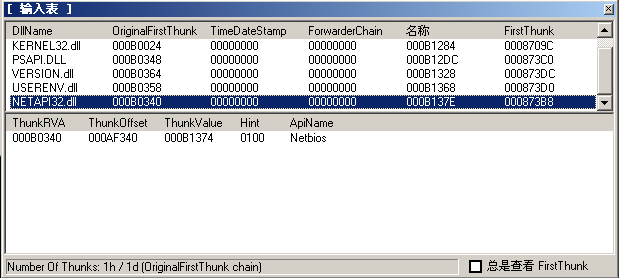












经比对，验证一致。