

实验二 PE 文件结构分析

汤添凝 1170300728

一. 实验目的

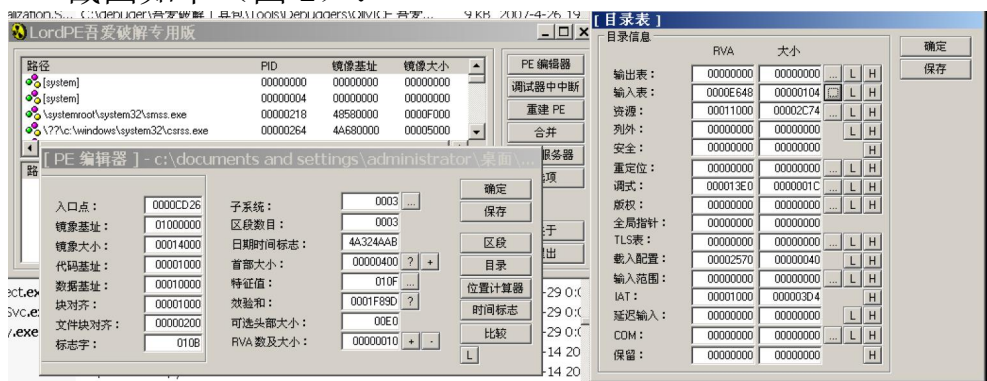
1. 了解 PE 文件的输入表结构；
2. 手工解析 PE 文件的输入表；
3. 编程实现 PE 文件输入表的解析。

二. 实验内容

1. 第一步：手动解析输入表结构

(1) 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: tsshutdn.exe

(2) 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):



(3) 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):



- (4) 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 = 0000EA48
    TimeDateStamp = 00000000
    ForwarderChain = 00000000
    Name = 0000ED50
    FirstThunk = 000012FC
}
```

- (5) 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 0000E150，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）：

| tsshutdn.exe | | tntsess.exe | | | | | | | | | | | | | | | | |
|--------------|--|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| Offset | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
| 0000E120 | | 80 | 00 | 5F | 5F | 70 | 5F | 5F | 63 | 6F | 6D | 6D | 6F | 64 | 65 | 00 | 00 | € __p_commode |
| 0000E130 | | 85 | 00 | 5F | 5F | 70 | 5F | 5F | 66 | 6D | 6F | 64 | 65 | 00 | 00 | 98 | 00 | ... __p_fmode " |
| 0000E140 | | 5F | 5F | 73 | 65 | 74 | 5F | 61 | 70 | 70 | 5F | 74 | 79 | 70 | 65 | 00 | 00 | __set_app_type |
| 0000E150 | | 6D | 73 | 76 | 63 | 72 | 74 | 2E | 64 | 6C | 6C | 00 | 00 | D6 | 00 | 5F | 63 | msvcrt.dll Ö_c |
| 0000E160 | | 6F | 6E | 74 | 72 | 6F | 6C | 66 | 70 | 00 | 00 | E1 | 00 | 46 | 72 | 65 | 65 | ontrolfp á Free |
| 0000E170 | | 53 | 69 | 64 | 00 | CB | 01 | 52 | 65 | 67 | 43 | 6C | 6F | 73 | 65 | 4B | 65 | Sid E RegCloseKe |

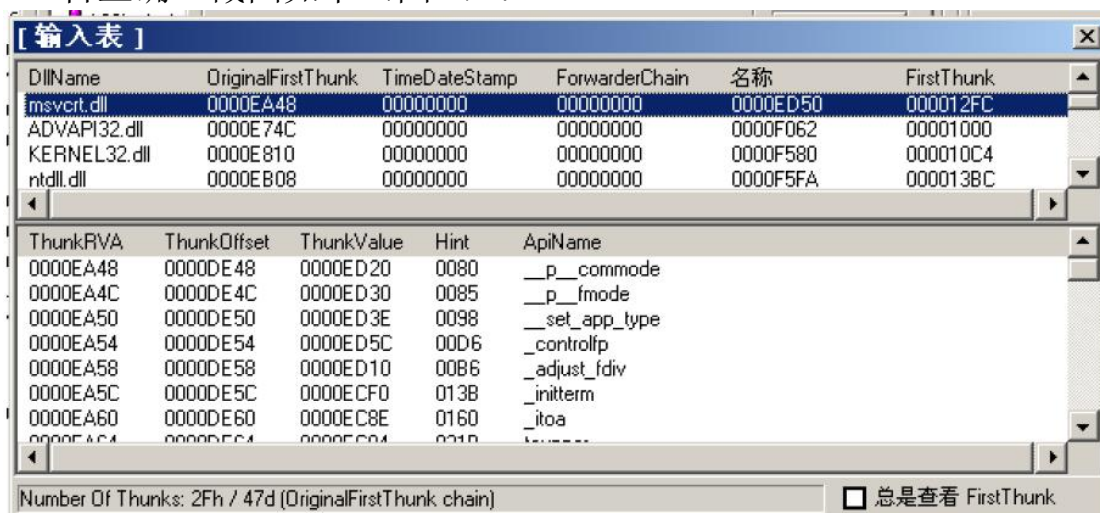
- (6) 分析一下 OriginalFirstThunk，它指向一个类型为 IMAGE_THUNK_DATA 的数组，上面已经分析出了它的值为 0000EA48，这是一个 RVA，用步骤 (3) 的方法得到文件偏移地址 0000DE48。在 16 进制编辑工具转到这个偏移地址，其中前面 4 个字节的数据为 0000ED20，截图如下（图 4）：

| | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|
| 0000DE20 | 33 | 00 | 00 | 80 | 09 | 00 | 00 | 80 | 37 | 00 | 00 | 80 | 08 | 00 | 00 | 80 | 3 | € | €7 | € | € |
| 0000DE30 | 0C | 00 | 00 | 80 | 34 | 00 | 00 | 80 | 39 | 00 | 00 | 80 | 6F | 00 | 00 | 80 | €4 | €9 | €0 | € | |
| 0000DE40 | 0B | 00 | 00 | 80 | 00 | 00 | 00 | 00 | 20 | ED | 00 | 00 | 30 | ED | 00 | 00 | € | i | Öi | | |
| 0000DE50 | 3E | ED | 00 | 00 | 5C | ED | 00 | 00 | 10 | ED | 00 | 00 | FO | EC | 00 | 00 | >i | \i | i | 8i | |
| 0000DE60 | 8E | EC | 00 | 00 | 84 | EC | 00 | 00 | 78 | EC | 00 | 00 | 6E | EC | 00 | 00 | Ži | „i | xi | ni | |
| 0000DE70 | 64 | EC | 00 | 00 | 58 | EC | 00 | 00 | 2A | EB | 00 | 00 | 32 | EB | 00 | 00 | di | Yi | *# | 2# | |

- (7) 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 80（十六进制），函数名为 __p_commode，截图如下（图 5）：

| | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------|
| 0000E110 | B6 | 00 | 5F | 61 | 64 | 6A | 75 | 73 | 74 | 5F | 66 | 64 | 69 | 76 | 00 | 00 | ¶ __adjust_fdiv |
| 0000E120 | 80 | 00 | 5F | 5F | 70 | 5F | 5F | 63 | 6F | 6D | 6D | 6F | 64 | 65 | 00 | 00 | € __p_commode |
| 0000E130 | 85 | 00 | 5F | 5F | 70 | 5F | 5F | 66 | 6D | 6F | 64 | 65 | 00 | 00 | 98 | 00 | ... __p_fmode " |
| 0000E140 | 5F | 5F | 73 | 65 | 74 | 5F | 61 | 70 | 70 | 5F | 74 | 79 | 70 | 65 | 00 | 00 | __set_app_type |
| 0000E150 | 6D | 73 | 76 | 63 | 72 | 74 | 2E | 64 | 6C | 6C | 00 | 00 | D6 | 00 | 5F | 63 | msvcrt.dll Ö_c |

- (8) 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）：

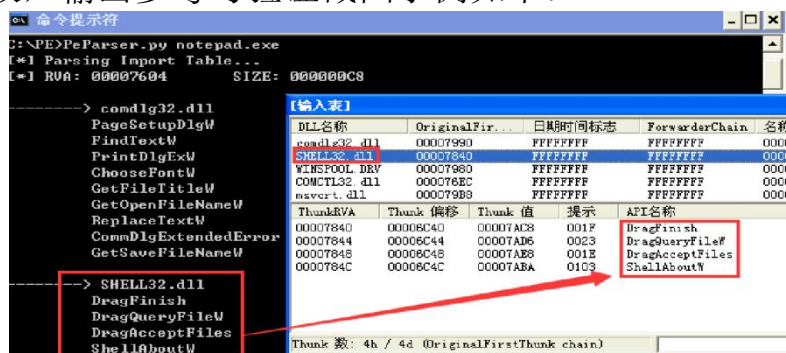


2. 第二步：编程实现输入表的解析

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：

```
# PEfile_Path = "D:\ttn\逆向分析\实验二\tlntsess.exe"

# -*-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):

        temp_ptr = self.get_dword(self.data[0x3c:0x40])

        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

    h32_size = self.get_dword(self.data[h32_size_ptr:h32_size_ptr+2] + b'\x00\x00')

    temp_rva = self.get_dword(self.data[0x3c:0x40]) + 0x18 + h32_size

    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

    return -1
```

```
#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])

    self.import_table_size = self.get_dword(self.data[self.pe_rva+4:self.pe_rva+8])

    self.import_table_offset = self.get_dword(self.data[self.pe_rva+8:self.pe_rva+12])

    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)

    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)

        iid_list_temp.append(self.get_string(self.rva_to_offset(self.get_dword(self.data[ptr_temp+12:ptr_temp+16]))))

        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):

    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：

```
PS D:\ttn\逆向分析\实验二\code> & D:/APP/miniconda3/python.exe d:/ttn/逆向分析/实验二/code/test.py D:\ttn\逆向分析\实验二\tlntsess.exe
rva: 58952
size: 260

msvcrt.dll
__p_commode
__p_fmode
__set_app_type
__controlfp
__adjust_fdiv
__initterm
__itoa
toupper
tolower
wcsncat
wcschr
__wcsnicmp
free
strncpy
strtoul
strchr
__getmainargs
__initenv
exit
_cexit
_XcptFilter
_exit
??3@YAXPAX@Z
__setusermatherr
sprintf
strnicmp
```

| [输入表] | | | | | |
|--------------|--------------------|---------------|----------------|------------------|------------|
| DllName | OriginalFirstThunk | TimeDateStamp | ForwarderChain | 名称 | FirstThunk |
| msvcrt.dll | 0000EA48 | 00000000 | 00000000 | 0000ED50 | 000012FC |
| ADVAPI32.dll | 0000E74C | 00000000 | 00000000 | 0000F062 | 00001000 |
| KERNEL32.dll | 0000E810 | 00000000 | 00000000 | 0000F580 | 000010C4 |
| ntdll.dll | 0000EB08 | 00000000 | 00000000 | 0000F5FA | 0000138C |
| | | | | | |
| ThunkRVA | ThunkOffset | ThunkValue | Hint | ApiName | |
| 0000EA48 | 0000E48 | 0000ED20 | 0080 | __p_commode | |
| 0000EA4C | 0000E4C | 0000ED30 | 0085 | __p_fmode | |
| 0000EA50 | 0000E50 | 0000ED3E | 0098 | __set_app_type | |
| 0000EA54 | 0000E54 | 0000ED5C | 00D6 | __controlfp | |
| 0000EA58 | 0000E58 | 0000ED10 | 00B6 | __adjust_fdiv | |
| 0000EA5C | 0000E5C | 0000ECF0 | 0138 | __initterm | |
| 0000EA60 | 0000E60 | 0000EC8E | 0160 | __itoa | |
| 0000EA64 | 0000E64 | 0000EC64 | 0110 | __setusermatherr | |

Number Of Thunks: 2Fh / 47d (OriginalFirstThunk chain) ☐ 总是查看 FirstThunk