# 汇编语言与逆向技术

## 第7章 节表、导入表、导出表

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2022-2023学年

# 本章知识点

- 节表（Section Table，ST）
  - 难点：文件偏移RAW到内存地址RVA的转换方法
- 导入表（Import Table，IT）
  - 难点：INT（Import Name Table）与IAT（Import Address Table）
- 导出表（Export Table，ET）
  - 难点：AddressOfNames, AddressOfNameOrdinals，AddressOfFunctions

# 1. 节表

# 节表

- 保证程序的安全性
  - 把code和data放在同一个内存节中相互纠缠，很容易引发安全问题
  - code有可能被data覆盖，导致崩溃
  - PE文件格式将内存属性相同的数据统一保存在一个被称为“节”（Section）的地方

# IMAGE_SECTION_HEADER(winnt.h)

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-image_section_header

# 节表

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A① | B | C | D | E | F | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000001A0 | 00 | 00 | 00 | 00 | 00③ | 00 | 00 | 00 | 2E | 74 | 65 | 78 | 74 | 00⑤ | 01 | 59 | .....text..Y |
| 000001B0② | 9A | 01 | 00 | 00 | 00 | 10 | 00 | 00④ | 00 | 02 | 00 | 00 | 00 | 04 | 00 | 00 | ?..... |
| 000001C0⑥ | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 20 | 00 | 00 | 60 | ..... ..` |
| 000001D0 | 2E | 72 | 64 | 61 | 74⑦ | 61 | 00 | 00 | C2⑧ | 01 | 00⑨ | 00 | 00 | 20⑩ | 00 | 00 | .rdata..?... .. |
| 000001E0 | 00 | 02 | 00 | 00 | 00 | 06 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ..... |
| 000001F0 | 00 | 00 | 00 | 00 | 40 | 00 | 00 | 40 | 2E | 64 | 61 | 74 | 61 | 00 | 00 | 00 | ....@..@.data... |
| 00000200 | 38 | 00 | 00 | 00 | 00 | 30 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | 08 | 00 | 00 | 8....0..... |
| 00000210 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 40 | 00 | 00 | C0 | .....@..@ |

图 11.9　十六进制工具中的块表

# 节表

- Name（8 BYTE）：块名
  - An 8-byte, null-padded UTF-8 string.
  - There is no terminating null character if the string is exactly eight characters long.
- VirtualSize（DWORD）：在内存空间中，节的大小
  - The total size of the section when loaded into memory, in bytes.
- VirtualAddress（DWORD）：节在内存空间中的起始RVA
  - The address of the first byte of the section when loaded into memory, relative to the image base.

# 节表

- SizeOfRawData（DWORD）：该节在硬盘中所占的空间
  - The size of the initialized data on disk, in bytes.
- PointerToRawData（DWORD）：该节在硬盘中的偏移
  - A file pointer to the first page within the COFF file.
  - This value must be a multiple of the FileAlignment member of the IMAGE_OPTIONAL_HEADER structure.

# 节表

- PointerToReLocations （DWORD）：在EXE文件中无意义
- PointerToLinenumbers （DWORD）：行号表在文件中的偏移量
- NumberOfReLocations （WORD）：在EXE文件中无意义
- NumberOfLinenumbers （WORD）：该块在行号表中的行号数目

# 节的内存属性

- Characteristics（DWORD）：块属性
  - IMAGE_SCN_MEM_EXECUTE
    - 20000000h，可执行
  - IMAGE_SCN_MEM_READ
    - 40000000h，可读
  - IMAGE_SCN_MEM_WRITE
    - 80000000h，可写

# 节的内容

- IMAGE_SCN_CNT_CODE
  - 00000020h，包含可执行代码
- IMAGE_SCN_CNT_INITIALIZED_DATA
  - 00000040h，包含已初始化数据
- IMAGE_SCN_CNT_UNINITIALIZED_DATA
  - 00000080h，包含未初始化数据

# 常见的节

- .text，代码节，链接器把所有目标文件的.text节连接成一个大的.text节
- .data，读、写数据节，全局标量
- .rdata，只读数据节，调试目录、字符串等

# 常见的节

- .idata，导入表
- .edata，导出表
- .rsrc，资源数据，菜单、图标、位图等
- .bss，未初始化数据，被.data取代，增加VirtualSize到足够放下未初始化数据

# 节的对齐

- 硬盘上的对齐
  - FileAlignment
  - 200h，扇区对齐，节间隙
- 内存上的对齐
  - SectionAlignment
  - 1000h，内存页对齐（64位系统，8KB内存页）

# 文件偏移与虚拟内存地址转换

- Image（映像）
- PE文件加载到内存时，不会原封不动地加载
  - 根据节表的定义加载
- PE文件与内存中的Image具有不同的形态

# 文件偏移与虚拟内存地址转换

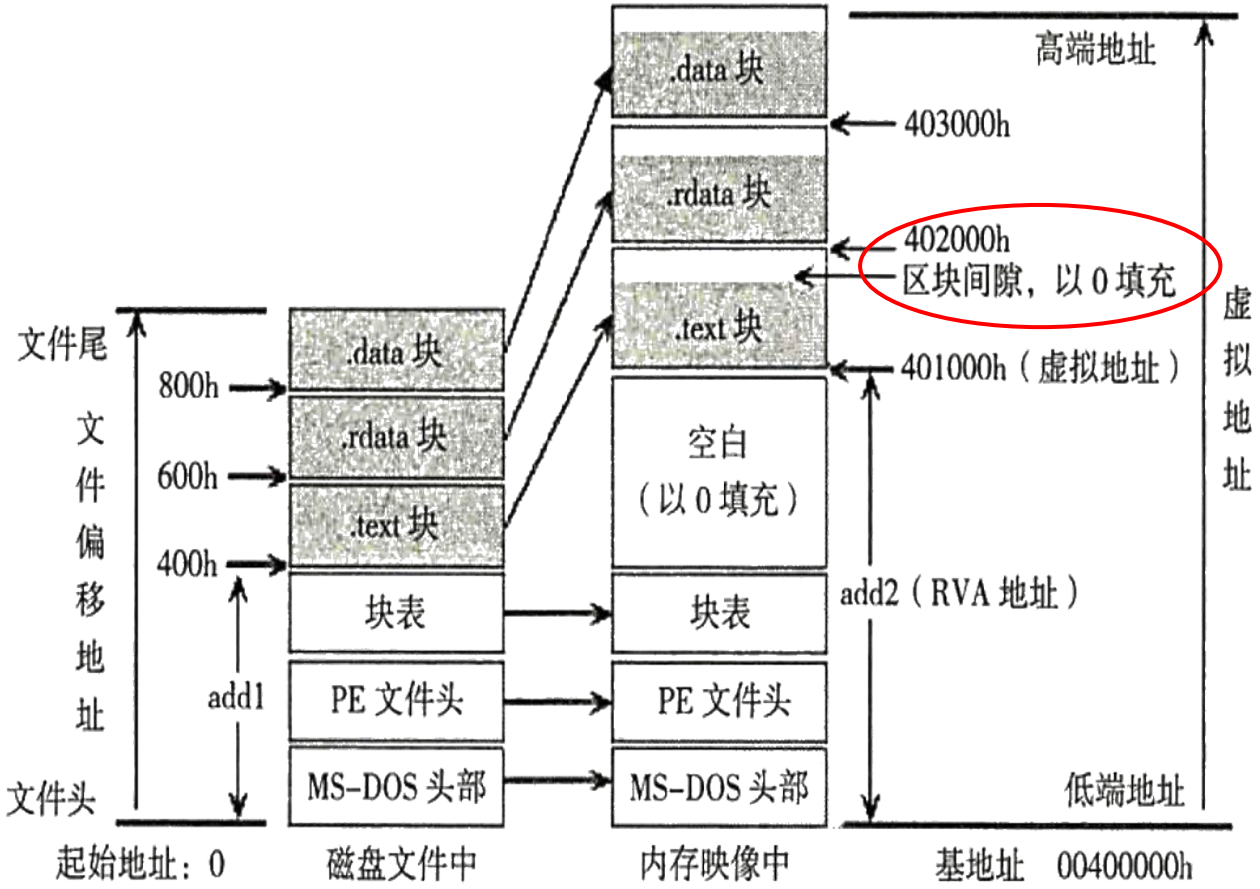



图 11.11 应用程序加载映射示意图

# 文件偏移与虚拟内存地址转换

- 文件被映射到内存中时，MS-DOS头部、PE文件头和节表的偏移位置与大小均没有变化
- 各节被映射到内存中后，其偏移位置就发生了变化

# 文件偏移与虚拟内存地址转换

- RVA to RAW
  - RAW – PointerToRawData = RVA – VirtualAddress
  - RAW = RVA – VirtualAddress + PointerToRawData

# RVA to RAW

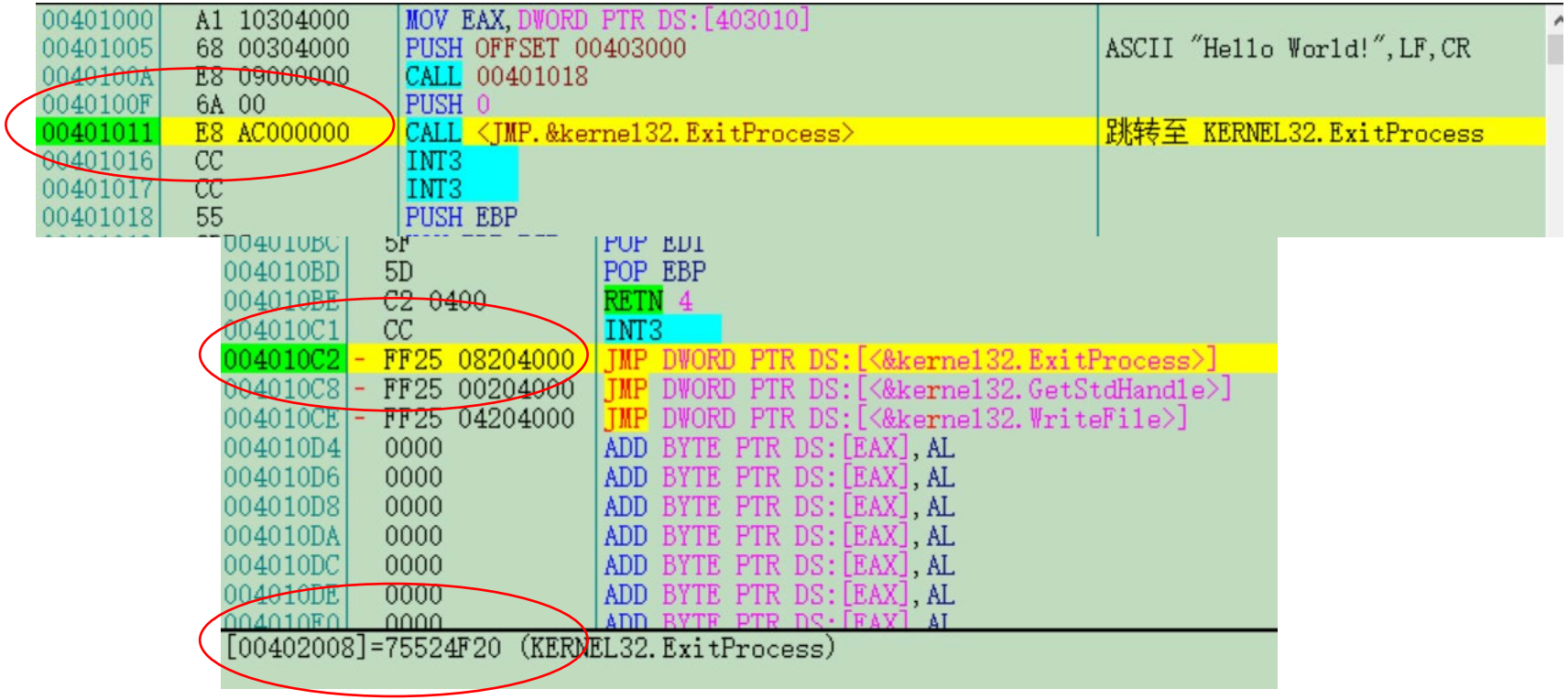
- RVA=2123h在.rdata节
  - .rdata节的相对内存地址范围是2000h到3000h
- VirtualAddress = 2000h
- PointerToRawData=600h
- RAW=2123h（RVA）- 2000h（VirtualAddress）+600h（PointerToRawData）=723h

# RVA to RAW

- RVA与RAW（文件偏移）间的相互变换时PE头的最基本内容，需要熟悉并掌握。

# 2. 导入表 IAT

# hello.exe




为什么不直接使用call 75524F20调用函数？

# JMP-jump

- **近跳转（Near jump）**
  - A jump to an instruction within the current code segment (the segment currently pointed to by the CS register)
  - E9 → JMP rel32/rel16 （长度5字节）
- **短跳转 （Short jump）**
  - A near jump where the jump range is limited to –128 to +127 from the current EIP value.
  - EB → JMP rel8 （长度2字节）
- **远跳转（Far jump）**
  - A jump to an instruction located in a different segment than the current code segment but at the same privilege level, sometimes referred to as an intersegment jump.
  - FF 25 → JMP m16:32 （长度6字节）
    JMP DWORD PTR **DS**:[<&kernel32.ExitProcess>]

# 导入表IAT

- 操作系统的版本不同
- kernel32.dll的版本不同
- DLL重定位
  - ImageBase值在不同程序内存空间中是不一样的

# 导入表 IAT

- 导入表 Import Address Table（IAT）
- 学习PE文件结构，最难过的一关就是IAT
- IAT中的内容与Windows操作系统的核心进程、内存、DLL结构等有关
  - “理解了IAT，就掌握了Windows操作系统的根基”

# 导入表IAT

- IAT是一种表结构
- 标记程序需要使用哪些库中的哪些函数
- IMAGE_IMPORT_DESCRIPTOR（IID）
- IMAGE_IMPORT_BY_NAME

# IMAGE_IMPORT_DESCRIPTOR（IID）

IMAGE_IMPORT_DESCRIPTOR结构体中记录PE文件要导入哪些库文件

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;
        DWORD   OriginalFirstThunk;
    } DUMMYUNIONNAME;
    DWORD   TimeDateStamp;
    DWORD   ForwarderChain;
    DWORD   Name;
    DWORD   FirstThunk;
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED *PIMAGE_IMPORT_DESCRIPTOR;
```

# IMAGE_IMPORT_DESCRIPTOR

- PE程序往往需要导入多个库
- 导入多少个库，就存在多少个IMAGE_IMPORT_DESCRIPTOR结构体
- 多个结构体组成数组，以NULL结构体结束

# IMAGE_IMPORT_DESCRIPTOR

- Name （DWORD），库文件名字符串的地址（RVA）
- OriginalFirstThunk（DWORD），INT(Import Name Table)的地址（RVA）
- FirstThunk （DWORD），IAT(Import Address Table)的地址（RVA）

# IMAGE_IMPORT_DESCRIPTOR

- Table
  - 在PE头中，Table即指数组
- INT与IAT是DWORD数组，以NULL结束
- INT与IAT的各元素指向相同地址

# INT（Import Name Table），IAT（Import Address Table）

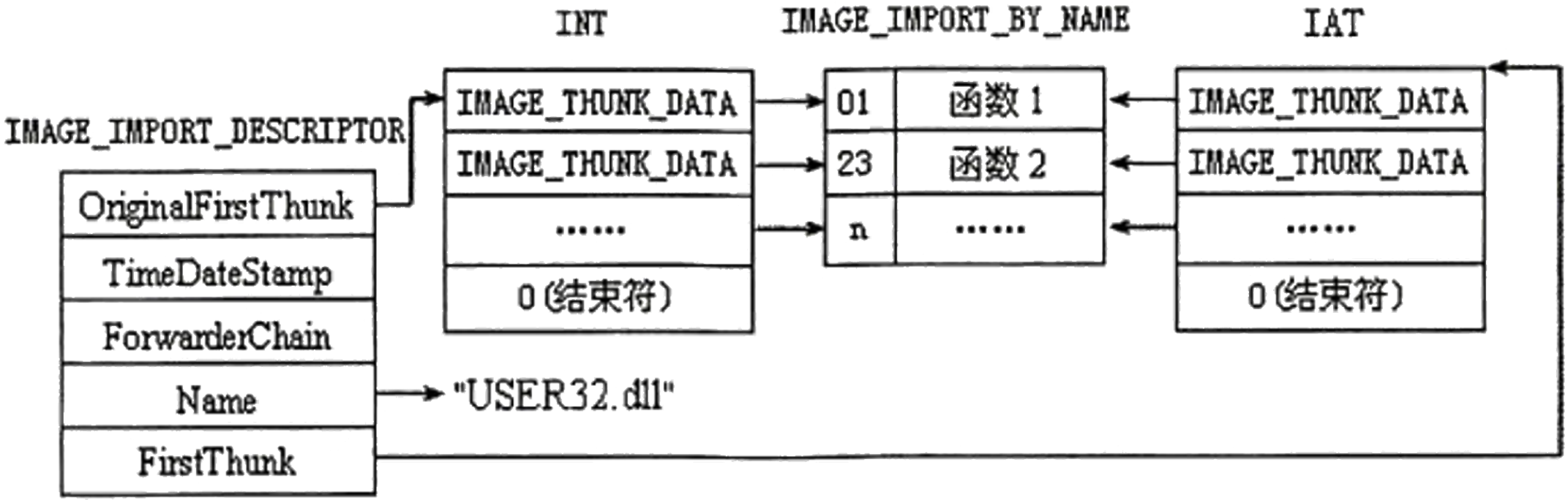



图 11.13　两个并行的指针数组

# IMAGE_IMPORT_BY_NAME

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    CHAR   Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

- Hint 是函数在DLL引出表中的索引号。
  - PE装载器用Hint在DLL的引出表里快速查询函数
  - 该值不是必须的，一些连接器将此值设为0。
- Name 引入函数的函数名
  - 函数名是一个ASCII字符串。
  - **是可变尺寸域**，以NULL结尾

# PE装载器

- 1. 读取IID的name成员，获取库名称字符串，例如“kernel32.dll”
- 2. 装载相应的库，类似LoadLibrary（“kernel32.dll”）

# PE装载器

- 3. 读取IID的OriginalFirstThunk成员，获取INT地址
- 4. 读取INT，逐一获得IMAGE_IMPORT_BY_NAME的地址（RVA）

# PE装载器

- 5. 使用IMAGE_IMPORT_BY_NAME的Hint或者Name项，获得函数的起始地址
  - 类似GetProcAddress（“ExitProcess”）
- 6. 读取IID的FirstThunk成员，获得IAT地址

# PE装载器

- 7. 将第5步获得的函数地址写入IAT数组相应位置
- 8. 重复步骤4到7，直到INT结束

# PE文件加载后的IAT

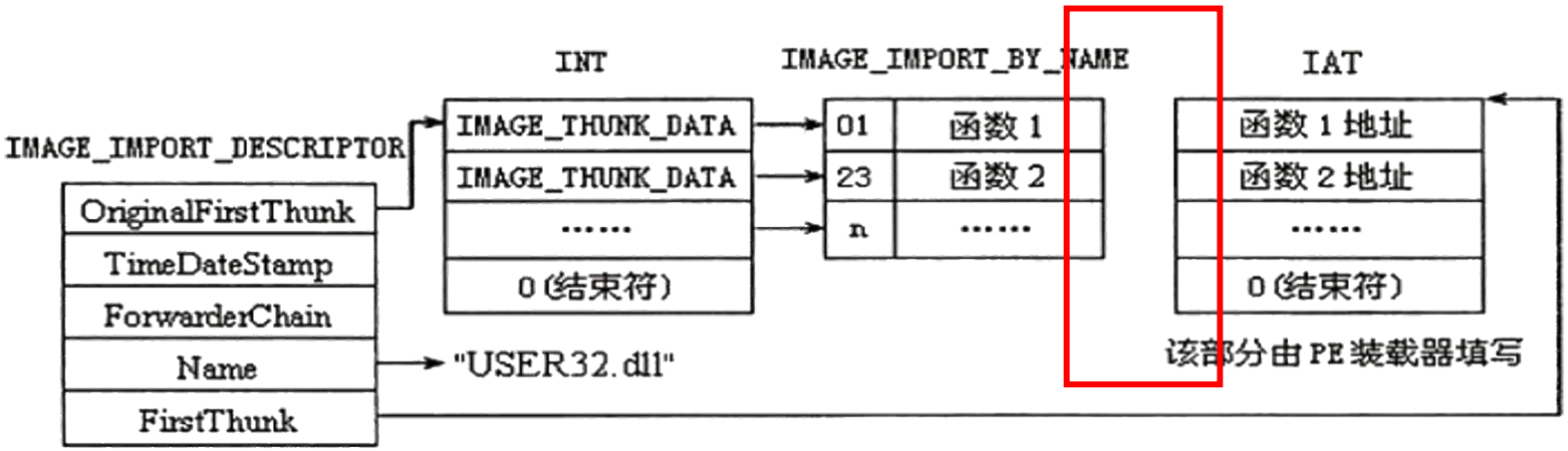



图 11.14　PE 文件加载后的 IAT

# 主观题 10分

设置

导入表有哪些安全问题？如何进行安全加固？

正常使用主观题需2.0以上版本雨课堂

作答

南開大學 Nankai University

# 3. 导出表 EAT

# DLL

- Windows操作系统提供了数量庞大的库函数
  - 进程、内存、窗口、消息、文件、网络等
- 同时运行多个程序时，每个进程都包含相同的库，严重浪费内存
- Dynamic Link Libary(DLL)，内存映射

# IMAGE_EXPORT_DIRECTORY

- 导出表是DLL的核心机制
  - 不同程序可以调用库文件中提供的函数
  - 通过EAT，得到库文件导出函数的入口地址

# IMAGE_EXPORT_DIRECTORY

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   Name;
    DWORD   Base;
    DWORD   NumberOfFunctions;
    DWORD   NumberOfNames;
    DWORD   AddressOfFunctions;     // RVA from base of image
    DWORD   AddressOfNames;         // RVA from base of image
    DWORD   AddressOfNameOrdinals;  // RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

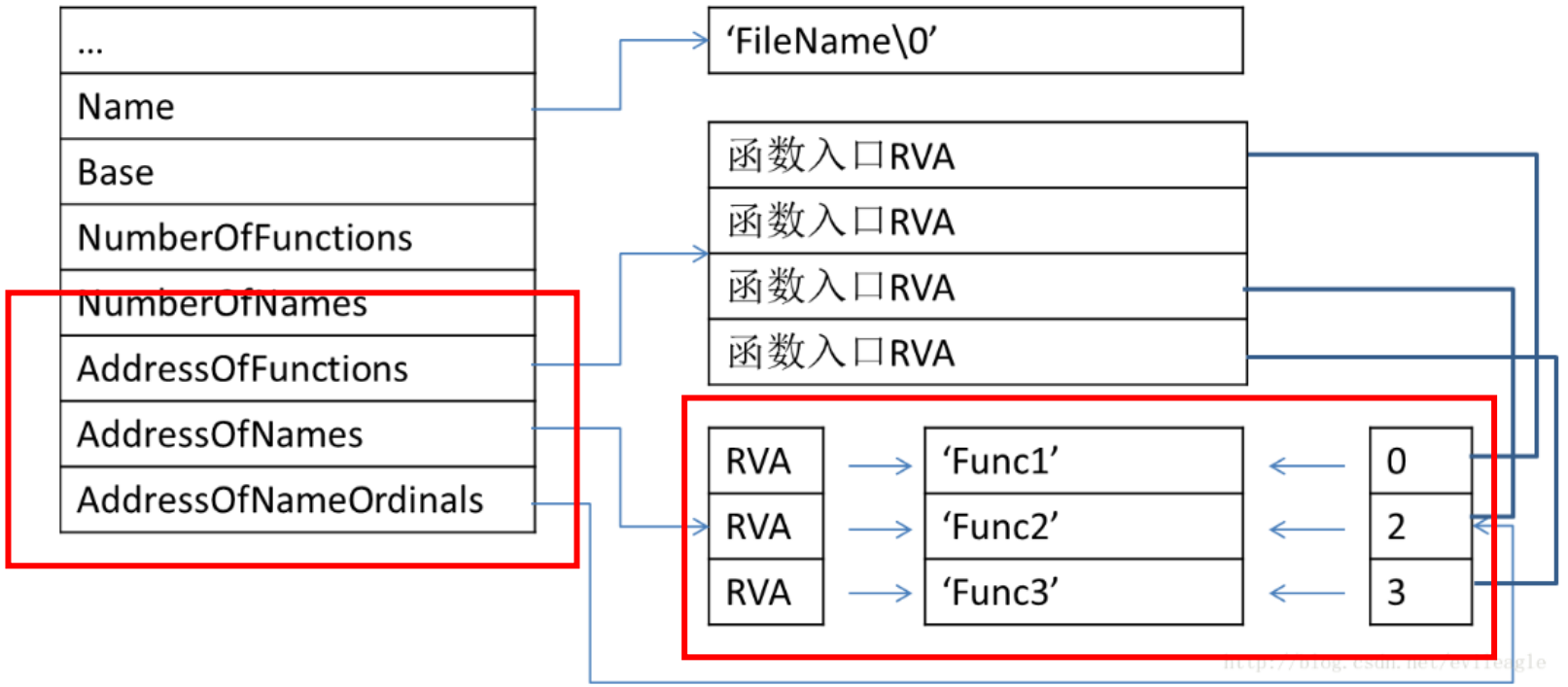
# IMAGE_EXPORT_DIRECTORY

- Name
  - 库文件名字符串地址
- NumberOfFunctions
  - 实际Export函数的个数
- NumberOfNames
  - Export函数中具有名字的函数个数

# IMAGE_EXPORT_DIRECTORY

- AddressOfFunctions
  - Export函数地址数组
- AddressOfNames
  - 函数名称地址数组
- AddressOfNameOrdinals
  - Ordinal地址数组

# 导出表


...
Name
Base
NumberOfFunctions
NumberOfNames
AddressOfFunctions
AddressOfNames
AddressOfNameOrdinals
'FileName\0'
函数入口RVA
函数入口RVA
函数入口RVA
函数入口RVA
RVA
RVA
RVA
'Func1'
'Func2'
'Func3'
0
2
3

# GetProcAddress()操作原理

- 从库中获得函数地址的API为GetProcAddress()函数
  - 如何通过EAT获得函数地址？

# GetProcAddress()操作原理

- 1. AddressOfName定位“函数名称数组”
- 2. 在“函数名称数组”中，通过比较字符串（strcmp），查找指定的函数名称
  - 此时的数组索引称为name_index
- 3. 利用AddressOfNameOridinals成员，定位ordinal数组

# GetProcAddress()操作原理

- 4. 在ordinal数组中，通过name_index查找相应的ordinal值
- 5. AddressOfFunctions，定位“函数地址数组”（EAT）
- 在“函数地址数组”中，利用ordinal值作为索引，获得指定函数的起始地址

# 导出表

- 如果函数是以序号导出的，那么查找的时候直接用序号减去Base，得到的值就是函数在AddressOfFunctions中的下标

# 汇编语言与逆向技术

## 第7章 节表、导入表、导出表

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2022-2023学年