汇编语言与逆向技术

第7章 区块表、输入表、输出表

王志
zwang@nankai.edu.cn

2021.11.08

南开大学 网络空间安全学院

2021-2022学年

本章知识点

1. 区块表
2. 输入表
3. 输出表

1. 区块表

区块表

- 保证程序的安全性
  - 把code和data放在同一个内存区块中相互纠缠，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“区块”（Section）的地方

区块表

```
IMAGE_SECTION_HEADER   STRUCT
   Name                        BYTE8 DUP (?)     ;8 字节的块名
   union Misc                                    ;区块尺寸
     PhysicalAddress           DWORD  ?
     VirtualSize               DWORD  ?
   Ends
   VirtualAddress              DWORD  ?          ;区块的 RVA 地址
   SizeOfRawData               DWORD  ?          ;在文件中对齐后的尺寸
   PointerToRawData            DWORD  ?          ;在文件中的偏移
   PointerToRelocations        DWORD  ?          ;在 OBJ 文件中使用，重定位的偏移
   PointerToLinenumbers        DWORD  ?          ;行号表的偏移（供调试用）
   NumberOfRelocations         WORD   ?          ;在 OBJ 文件中使用，重定位项数目
   NumberOfLinenumbers         WORD   ?          ;行号表中行号的数目
   Characteristics             DWORD  ?          ;区块的属性
IMAGE_SECTION_HEADER  ENDS
```

区块表

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

区块表

- Name（8 BYTE）：块名
- VirtualSize（DWORD）：在内存空间中，区块的大小
- VirtualAddress（DWORD）：区块在内存空间中的起始RVA

区块表

- SizeOfRawData（DWORD）：该区块在硬盘中所占的空间
- PointerToRawData（DWORD）：该区块在硬盘中的偏移

区块表

- 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

内存属性

- Characteristics：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

区块内容

- 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

常见的区块

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

常见的区块

- .idata，输入表
- .edata，输出表
- .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

区块的对齐

- 硬盘上的对齐
  - FileAlignment
  - 200h，扇区对齐，区块间隙
- 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
  - 根据区块表的定义加载
- 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

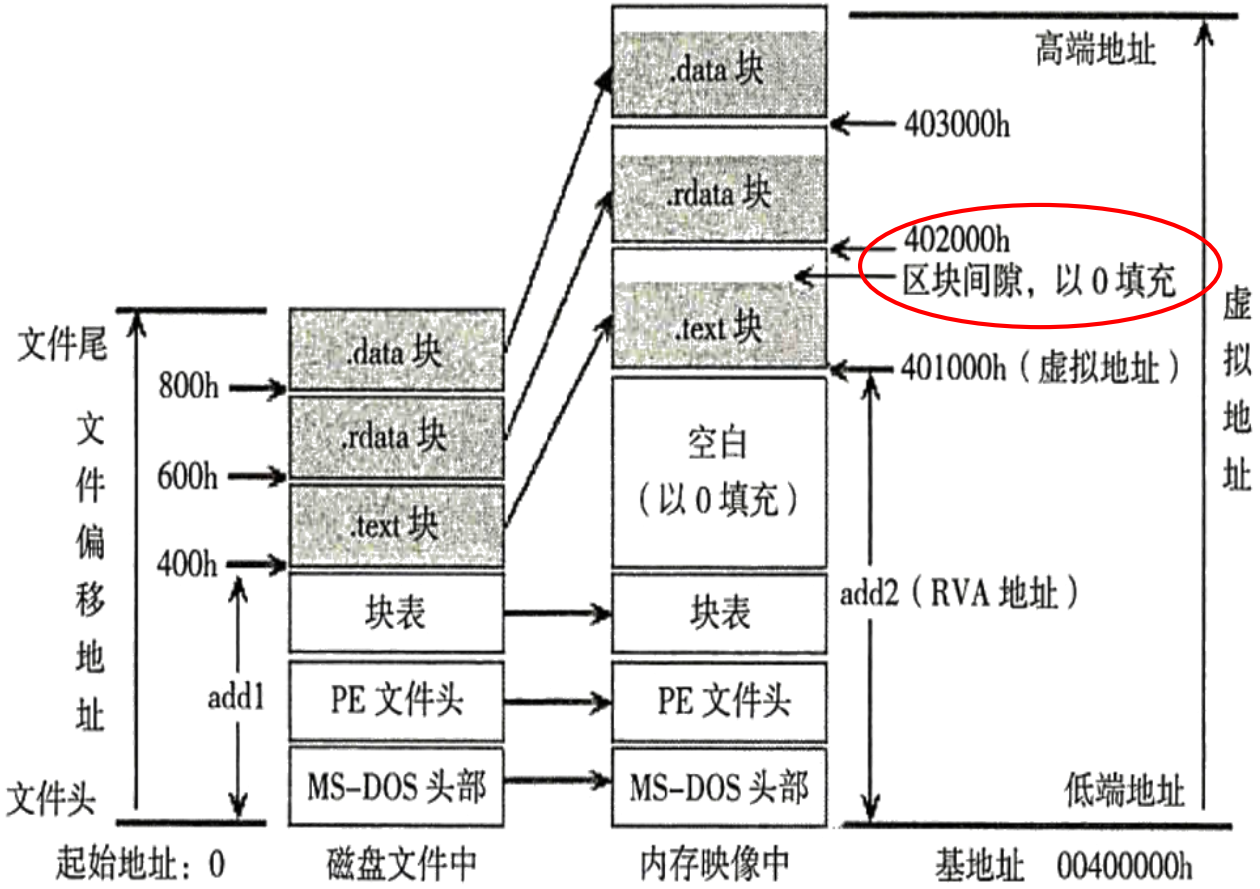


图 11.11 应用程序加载映射示意图

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文件头和区块表的偏移位置与大小均没有变化
- 各区块被映射到内存中后，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

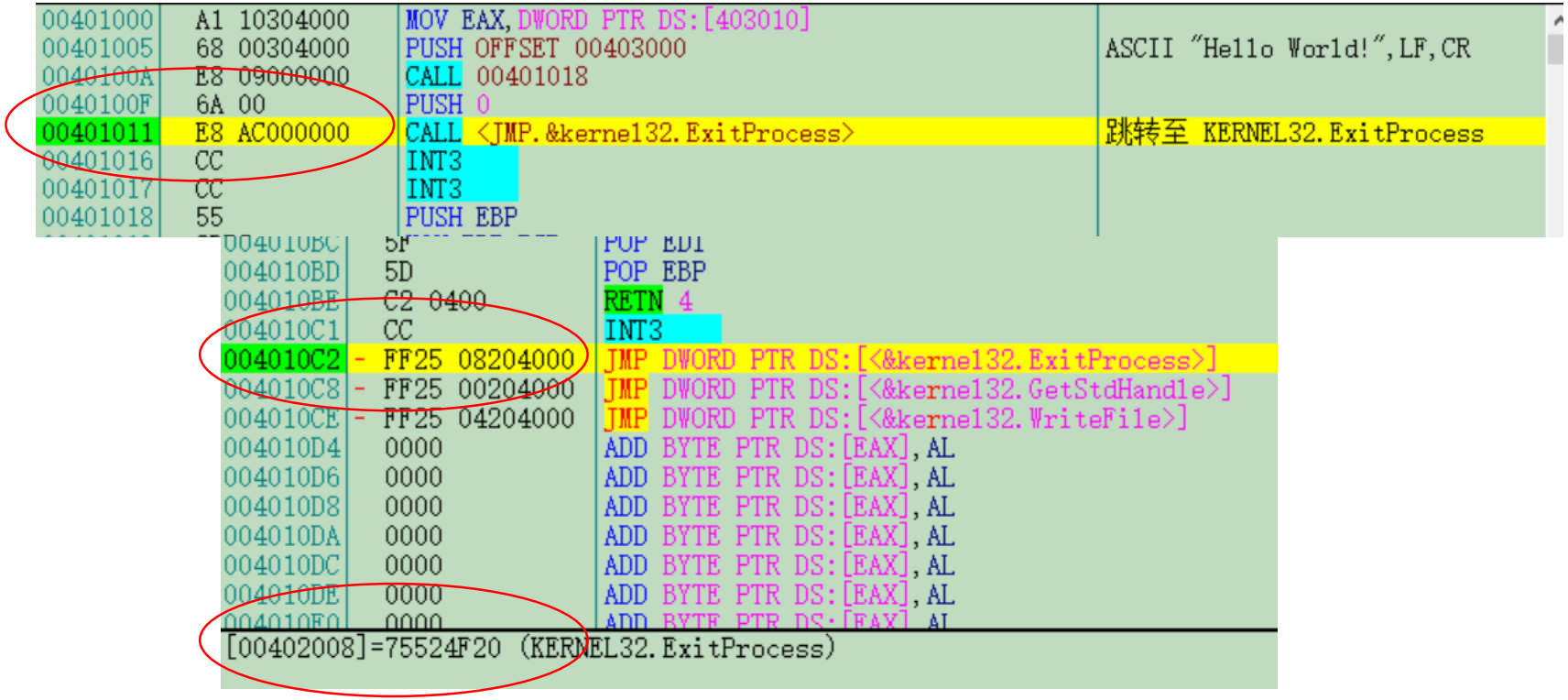
- RVA=2123h在.rdata区块
  - .rdata区块的相对虚拟地址RVA范围是2000h到3000h
- VirtualAddress = 2000h
- PointerToRawData=600h
- RAW=2123h（RVA）- 2000h（VirtualAddress）+600h（PointerToRawData）=**723**h

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. 输入表 IAT

hello.exe



为什么不直接使用call 75524F20调用函数？

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

输入表 IAT

- 输入表 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”

输入表IAT

- IAT是一种表格结构
- 标记程序需要使用哪些库中的哪些函数

IMAGE_IMPORT_DESCRIPTOR

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

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
    union                                   ;00h
        Characteristics         DWORD ?
        OriginalFirstThunk      DWORD ?
    ends
    TimeDateStamp               DWORD ? ;04h
    ForwarderChain              DWORD ? ;08h
    Name                        DWORD ? ;0Ch
    FirstThunk                  DWORD ? ;10h
IMAGE_IMPORT_DESCRIPTOR ENDS
```

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

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

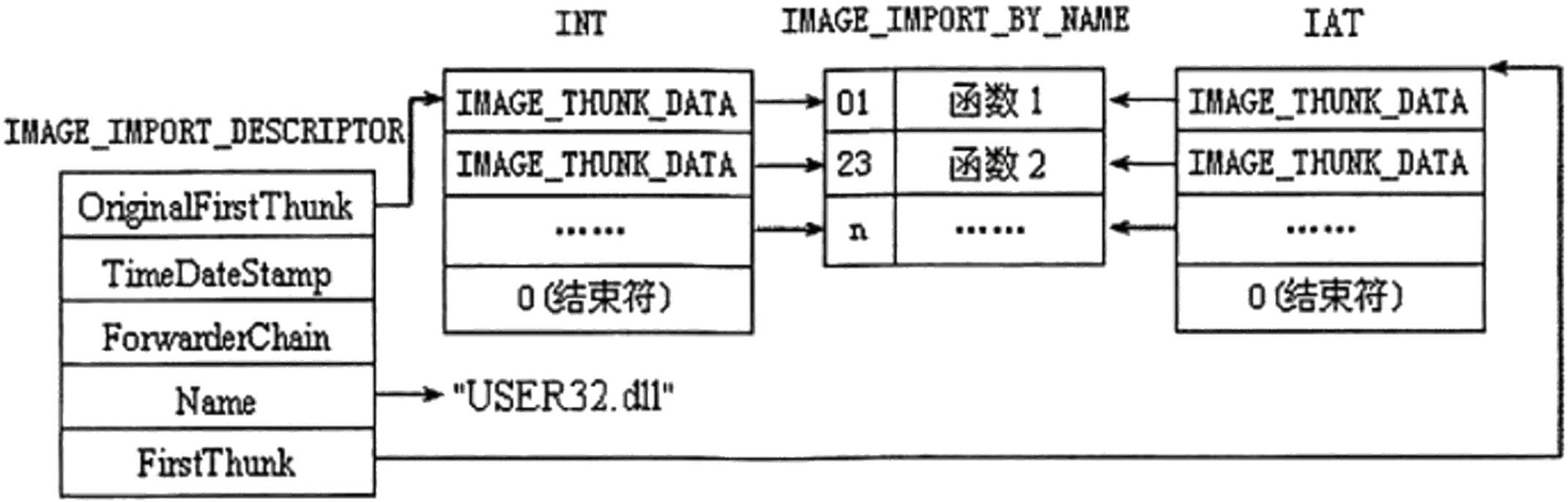


图 11.13　两个并行的指针数组

IMAGE_IMPORT_BY_NAME

```
IMAGE_IMPORT_BY_NAME STRUCT
    Hint                WORD ?
    Name                BYTE ?
IMAGE_IMPORT_BY_NAME ENDS
```

- 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，指导INT结束

PE文件加载后的IAT

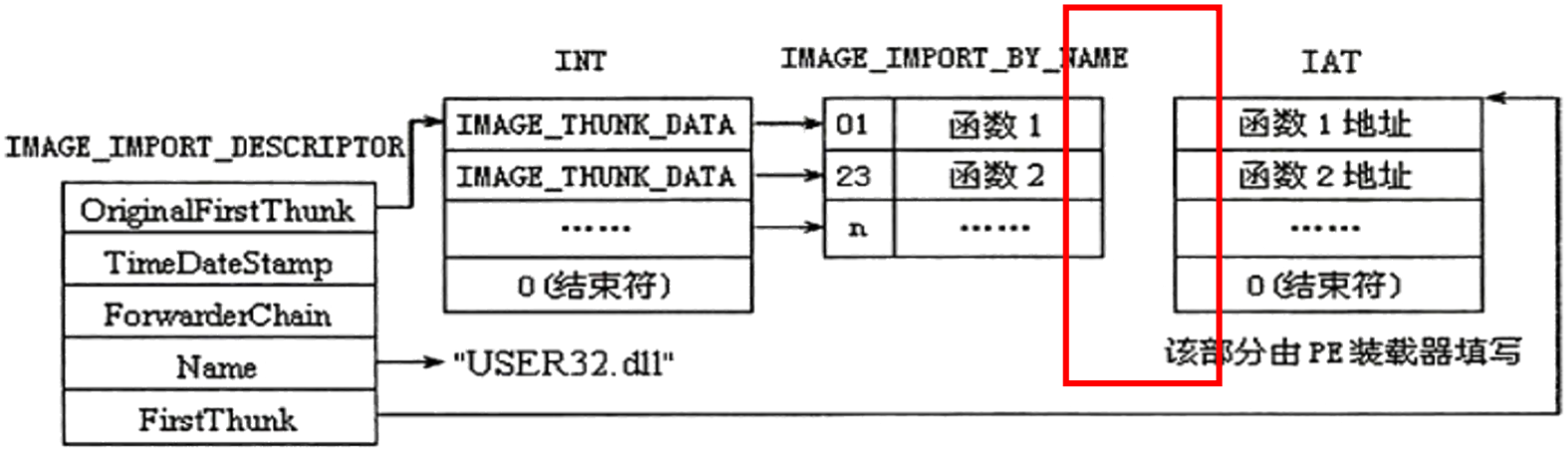


图 11.14　PE 文件加载后的 IAT

3. 输出表 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

- EAT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

```
IMAGE_EXPORT_DIRECTORY STRUCT
    Characteristics             DWORD   ?   ;未使用，总是为 0
    TimeDateStamp               DWORD   ?   ;文件生成时间
    MajorVersion                WORD    ?   ;主版本号，一般为 0
    MinorVersion                WORD    ?   ;次版本号，一般为 0
    Name                        DWORD   ?   ;模块的真实名称
    Base                        DWORD   ?   ;基数，序数减这个基数就是函数地址数组的索引值
    NumberOfFunctions           DWORD   ?   ;AddressOfFunctions 阵列中的元素个数
    NumberOfNames               DWORD   ?   ;AddressOfNames 阵列中的元素个数
    AddressOfFunctions          DWORD   ?   ;指向函数地址数组
    AddressOfNames              DWORD   ?   ;函数名字的指针地址
    AddressOfNameOrdinals       DWORD   ?   ;指向输出序列号数组
IMAGE_EXPORT_DIRECTORY ENDS
```

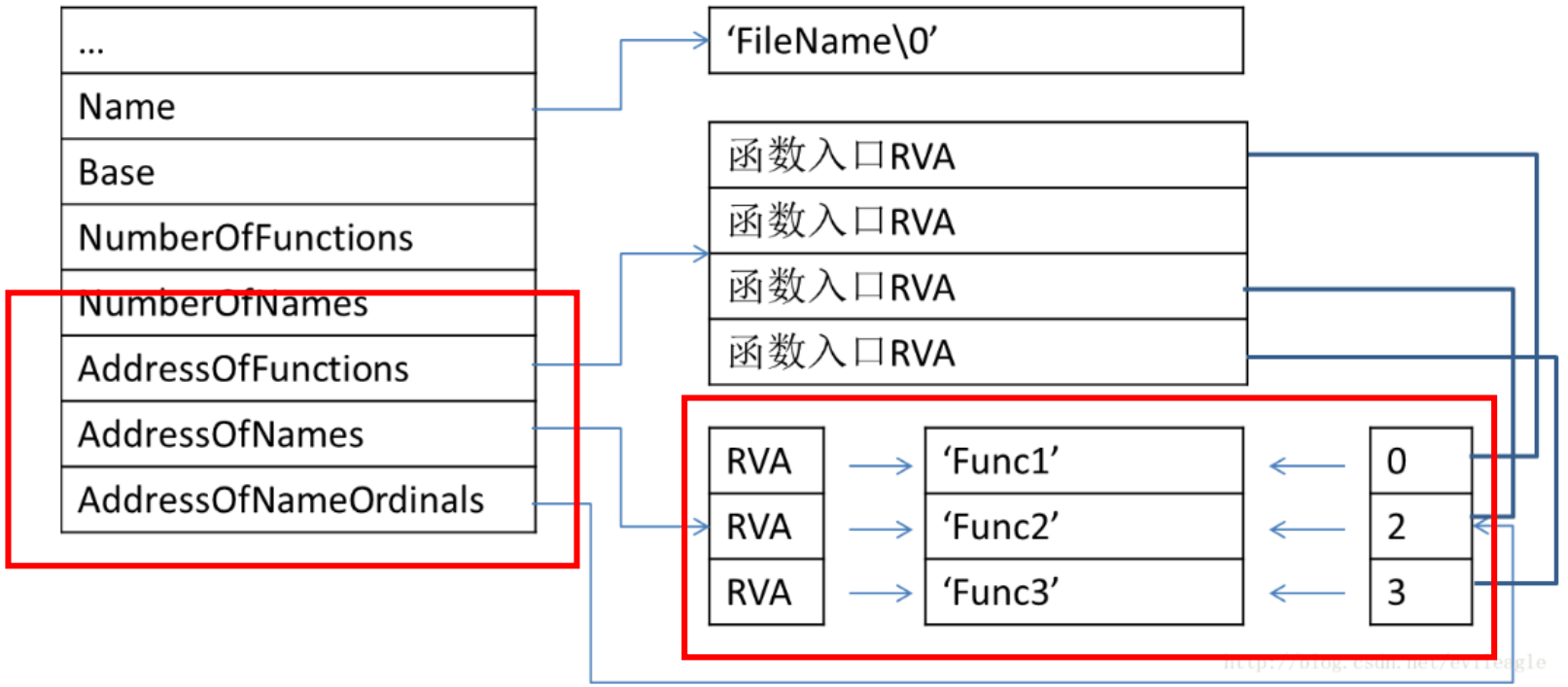
IMAGE_EXPORT_DIRECTORY

- 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

EAT


...
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

EAT

- 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 区块表、输入表、输出表

王志
zwang@nankai.edu.cn

2021.11.08

南开大学 网络空间安全学院

2021-2022学年