IMAGE\_IMPORT\_DESCRIPTOR

在 PE 文件中，IMAGE\_IMPORT\_DESCRIPTOR 通常以数组形式出现在 .rdata或 .idata节区中，固定为20字节。记录PE文件需要导入那些库文件，执行程序时导入多少库就存在几个IMAGE\_IMPORT\_DESCRIPTOR 结构体。

typedef struct \_IMAGE\_IMPORT\_DESCRIPTOR {

DWORD OriginalFirstThunk; *// 指向 INT（Import Name Table）*

DWORD TimeDateStamp; *// 时间戳（未绑定=0）*

DWORD ForwarderChain; *// 转发链索引（通常=0）*

DWORD Name; *// 指向 DLL 名称字符串（RVA）*

DWORD FirstThunk; *// 指向 IAT（Import Address Table）*

} IMAGE\_IMPORT\_DESCRIPTOR;

判断 IMAGE\_IMPORT\_DESCRIPTOR 结构体是在 .rdata 还是在 .idata中需要根据NT头的 IMAGE\_IMPORT\_DESCRIPTORY（数据目录表，在NT头末尾，通常为128字节，具体条目具体分析，这里分析导入表，位于第二个条目）判断。（其实一般都在.idata节区，这一段计算过程可以忽略）

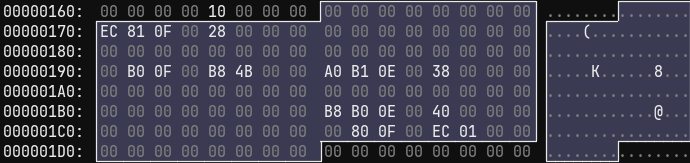


图 1 数据目录表

每个条目占8字节。这里找 IMAGE\_IMPORT\_DESCRIPTOR 需要查找条目1导入表，再根据IMAGE\_DATA\_DIRECTORY 结构内容查找 RVA，图中可看到 RVA = 0x0F81EC,RVA 位于落在哪个地址范围就说明位于哪个节区。

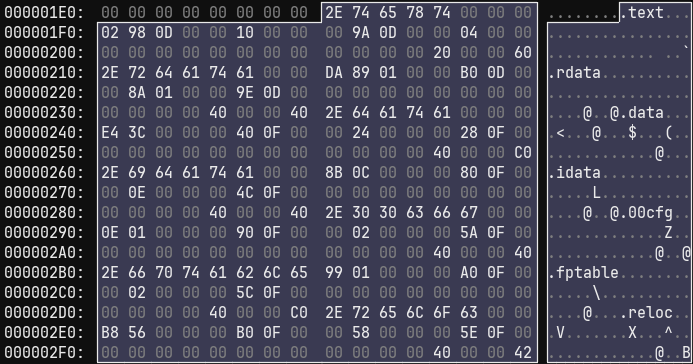


图 2 节区头

以 .rdata 段为例，RVA = 0x018A00，VirtualSize = 0x0D8000，.rdata节区的范围为 0x018A00 ~ 0x0F0A00（ 0x018A00 + 0x0DB000 ），同理可计算.idata节区范围为0x0F8000 ~ 0x0F8E00，和上面的RVA（0x0F81EC）做比较，可以得到结构位于.idata节区。

关注.idata节区，PointerToRawData = 0x0F4C00 ,VirtualAddress = 0x0F8000 ,可以计算RAW = 0x0F81EC - 0x0F8000 + 0x0F4C00 = 0x0F4DEC。跳转，先看第一个 IMAGE\_IMPORT\_DESCRIPTOR 结构体。



图 3 IMAGE\_IMPORT\_DESCRIPTOR 结构体

OriginalFirstThunk（RVA）= 0x0F8214

Name（RVA）= 0x0F8650

FirstThunk（RVA）= 0x0F8000

以Name为例计算RAW = 0x0F8650 - 0x0F8000 + 0x0F4C00 = 0x0F5250，跳转，可以看到库文件名称的字符串。同理OriginalFirstThunk（RAW）= 0x0F4E14 ,FirstThunk（RAW）= 0x0F4C00。

图 4 Name

Name 是当前导入的DLL名称。INT（OriginalFirstThunk）告诉你这个DLL里有哪些函数（按名称或序号）。IAT（FirstThunk）最初的内容和INT一致，加载后会被替换为函数地址，供程序调用。

INT和IAT

INT是一个包含DLL函数信息的结构体指针数组，每个元素是IMAGE\_THUNK\_DATA ，存储函数导入信息。IMAGE\_THUNK\_DATA 是一个RVA ，指向 IMAGE\_IMPORT\_BY\_NAME 结构（按名称导入）。

typedef struct \_IMAGE\_IMPORT\_BY\_NAME {

WORD Hint; *// 可选序号（不一定准确）*

CHAR Name[]; *// 以null结尾的函数名字符串*

} IMAGE\_IMPORT\_BY\_NAME;

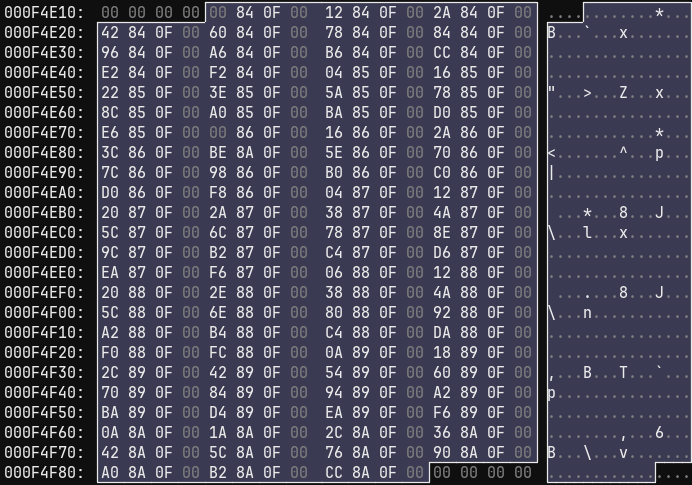


图 5 OriginalFirstThunk

以第一个值为例，RVA = 0x0F8442 ,计算RAW = 0x0F5042。跳转，可以看到INT的原始内容。



图 6 INT的一部分

根据IMAGE\_IMPORT\_BY\_NAME结构，函数的Hint = 0x038D , Name 为0x0F5045到0x0F505F的部分，每个函数名称以“\0”结尾。

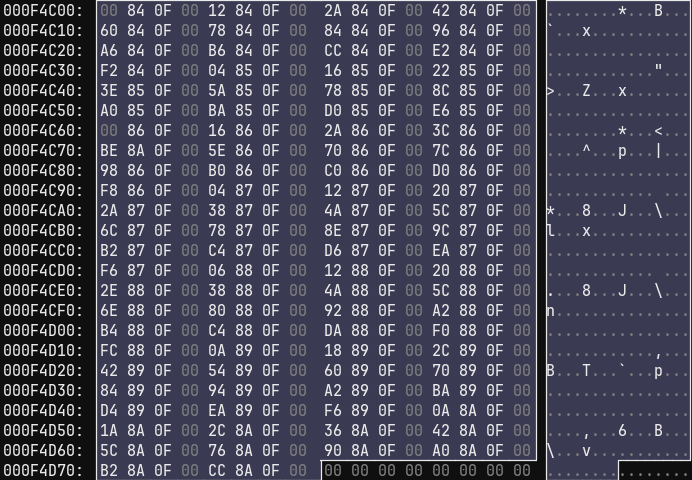


图 7 FirstThunk

FirstThunk内容和OriginalFirstThunk在未加载时（磁盘里）通常是一样的，当加载入内存中时会有所不同，加载后，FirstThunk储存函数地址，而OriginalFirstThunk仍保留原始函数名称。

IMAGE\_EXPORT\_DIRECTORY

固定长度40字节，与导入表不同，每个PE文件仅有一个IMAGE\_EXPORT\_DIRECTORY。

typedef struct \_IMAGE\_EXPORT\_DIRECTORY {

DWORD Characteristics; *// 保留字段，通常为0*

DWORD TimeDateStamp; *// 导出表创建的时间戳*

WORD MajorVersion; *// 主版本号（通常为0）*

WORD MinorVersion; *// 次版本号（通常为0）*

DWORD Name; *// 指向DLL名称的RVA（ASCII字符串）*

DWORD Base; *// 导出函数的起始序号基数*

DWORD NumberOfFunctions; *// 导出函数的总数量*

DWORD NumberOfNames; *// 按名称导出的函数数量*

DWORD AddressOfFunctions; *// 导出函数地址数组的RVA*

DWORD AddressOfNames; *// 导出函数名称数组的RVA*

DWORD AddressOfNameOrdinals;*// 导出函数序号数组的RVA*

} IMAGE\_EXPORT\_DIRECTORY, \*PIMAGE\_EXPORT\_DIRECTORY;

根据NT头数据目录表第一个条目查询RVA，结果为0时说明不导出任何函数。