导出表

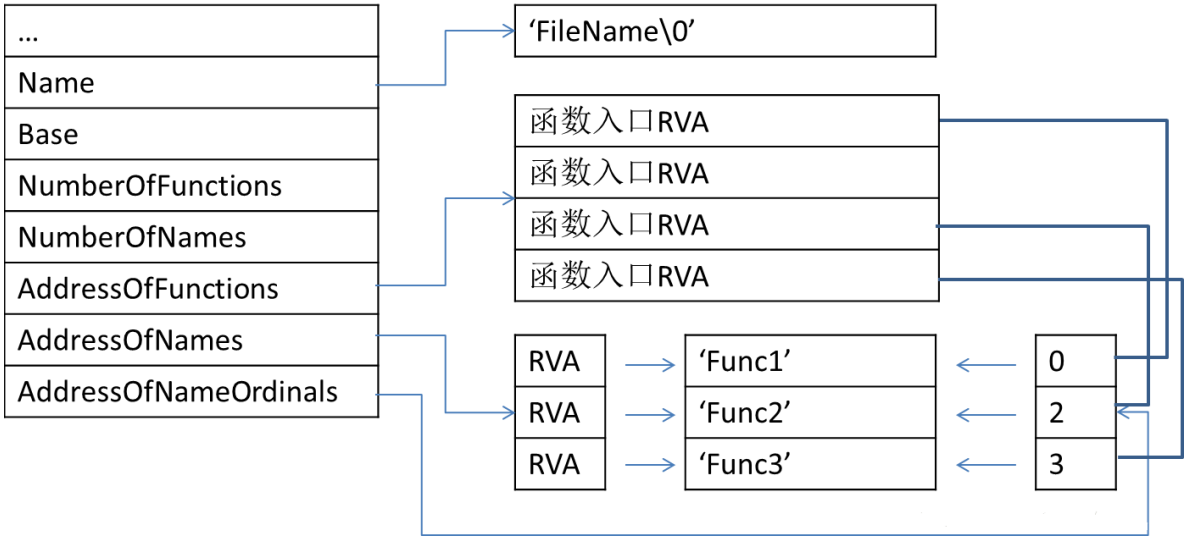
在`IMAGE_DIRECTORY_ENTRY_EXPORT`的导出表目录中的大小是有实际意义的，如实填写，如需修改

IMAGE_EXPORT_DIRECTORY

```
#define IMAGE_DIRECTORY_ENTRY_EXPORT    0

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;
```

- `Characteristics`
  一般为0，不做引用和参考
- `TimeDateStamp`
  导出表生成时的时间戳，不做引用和参考
- `MajorVersion`、`MinorVersion`
  版本号一般为0，不做引用和参考
- `Name`
  模块名的相对虚拟地址
- `Base`
  序号的基数（即序号的最小值），按序号导出函数的序号值从`Base`开始递增，即`序号 - base = index`
- `NumberOfFunctions`
  导出函数的数量，计算的是导出序号的差值
- `NumberOfNames`
  按名字导数的函数的数量
- `AddressOfFunctions`
  函数地址表，每一项是一个相对虚拟地址(大小为**DWORD**)，表中存储函数的相对虚拟地址，顺序与导出序号相同
- `AddressOfNames`
  函数名表，每一项是一个相对虚拟地址(大小为**DWORD**)，表中储存函数名的指针
- `AddressOfNameOrdinals`
  每一项(大小为**WORD**)与`AddressOfNames`对应，表示该名字在`AddressOfFunctions`中的序号



GetProcAddress

`lpProcName`高16位为0则是序号，否正是名称

`GetProcAddress`以序号获得地址：

1. `序号 - base = index < NumberOfFunctions`判断是否越界
2. 遍历`AddressOfFunctions`取得`AddressOfFunctions[index]`，即函数的RVA，不存在的函数是0

`GetProcAddress`以名称获得地址：

1. 取得`AddressOfNames`函数名称表，即名称所在的RVA
2. 遍历对比函数名（操作系统用的折半查找），得到`index`
3. 取得`AddressOfNameOrdinals[index]`去找`AddressOfFunctions`，即函数的RVA

实现GetProcAddress

```
void * MyGetProcAddress(HMODULE hModule, LPCSTR lpProcName)
{
    // 获取dos头
    PIMAGE_DOS_HEADER dos_header = (PIMAGE_DOS_HEADER)hModule;
    // 获取nt头
    PIMAGE_NT_HEADERS nt_headers = (PIMAGE_NT_HEADERS)((BYTE *)hModule +
dos_header->e_lfanew);

    // 定位到导出表位置
    PIMAGE_EXPORT_DIRECTORY export_directory = (PIMAGE_EXPORT_DIRECTORY)((BYTE
*)hModule +
        nt_headers-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

    if((DWORD)lpProcName & 0xffff0000) {
```

```
        // 是名称
        // 获取函数名称表
        PDWORD name_addr_table = (PDWORD)((BYTE *)hModule + export_directory->AddressOfNames);

        // 遍历名称
        bool is_found = false;
        DWORD index = 0;
        for(DWORD i = 0; i < export_directory->NumberOfNames; i++) {
            // 获取到名称
            char *api_name = (char *)((BYTE *)hModule + name_addr_table[i]);

            // 对比名称
            const char *str1 = api_name;
            const char *str2 = lpProcName;
            while(*str1 && *str2) {
                if(*str1 != *str2) {
                    break;
                }
                str1++, str2++;
            }
            if(!*str1 && !*str2) {
                is_found = true;
                index = i;
                break;
            }
        }

        if(is_found) {
            // 找到名称
            // 获取名称序号表
            PWORD name_ordinals_table = (PWORD)((BYTE *)hModule + export_directory->AddressOfNameOrdinals);
            // 获取函数索引
            WORD fun_index = name_ordinals_table[index];
            // 获取函数索引表
            PDWORD fun_index_table = (PDWORD)((BYTE *)hModule + export_directory->AddressOfFunctions);
            // 拿到函数地址并返回
            return (void *)((BYTE *)hModule + fun_index_table[fun_index]);
        }
    } else {
        // 是序号
        DWORD index = (DWORD)lpProcName - export_directory->Base;
        if(index < export_directory->NumberOfFunctions) {
            // 没有越界
            // 获取函数索引表
            PDWORD fun_index_table = (PDWORD)((BYTE *)hModule + export_directory->AddressOfFunctions);
            // 拿到函数地址并返回
            return (void *)((BYTE *)hModule + fun_index_table[index]);
        }
    }

    return NULL;
}
```

注意事项

在拿到函数的RVA后，先检查此地址是否在导出表的范围内（所以导出表的大小是有实际意义的），不在范围内直接返回其VA

若是，代表此函数是转发到别的库（例如**NTDLL.xxxxxxx**），此时读取此字符串左边`LoadLibrary`，递归调用`MyGetProcAddress`获取函数地址（以`0x2e`分割的两部分）

即获取的地址在导出表范围内的，统统是别的库的转发

与导入表的联合应用

脱壳后修复导入表

- 定位到目标进程的IAT位置，读取IAT表里面的函数地址
- 遍历目标进程的模块，确定该IAT表属于哪个模块
- 根据IAT表中的地址，反查出的函数名
- 重新构建导入表（添加新节，存放导入描述符表，IAT不动）

对于转发的函数，通过地址可以获得库名，反查可以获得函数名，以`.`进行拼接接着到该IAT所属于的库中查找导出表范围内的地址，拿到范围内的地址与之前拼接的名称做对比，取得函数名