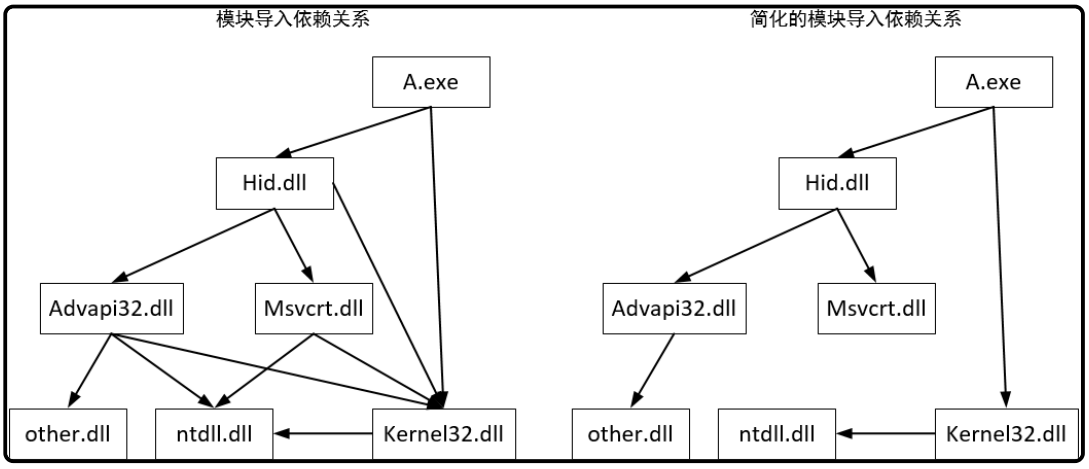
导入表（IAT）高级玩法背景（一）

首先让我们换一个角度观察一下模块的导入关系。假设有一个可执行程序A.exe，运行之后，其进程空间模块有hid.dll、msvcrt.dll、advapi32.dll，技术模块有ntdll.dll、kernel32.dll，代表其他模块的是other.dll，他们的导入关系如图所示：

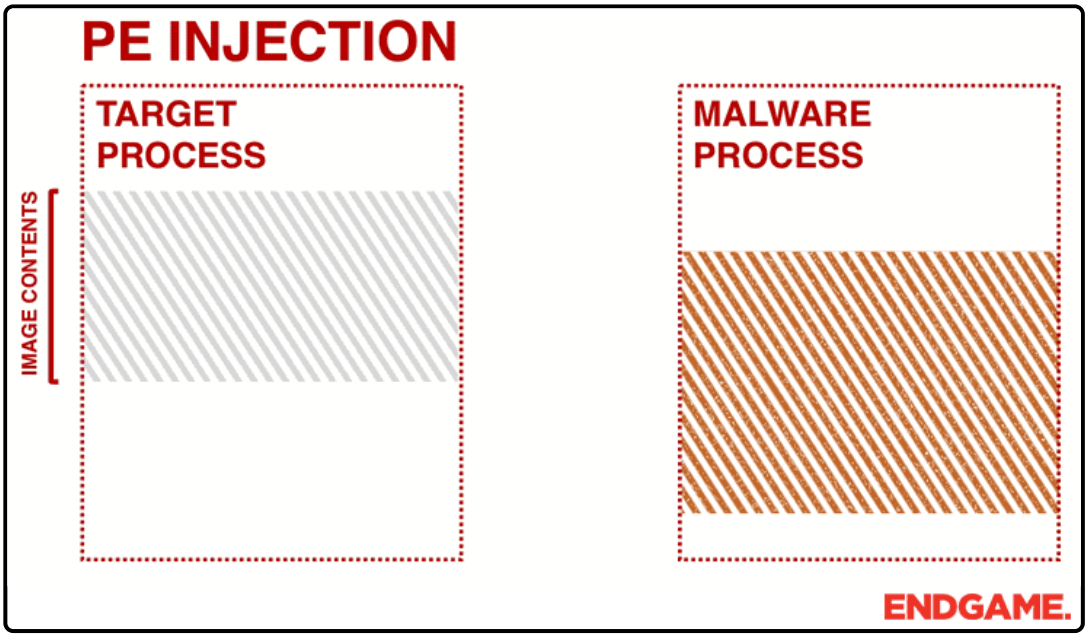


因为一个进程空间对于同一个dll模块只需要加载一个，所以简化一下模块的导入依赖关系可以看到如图所示。

简化后的模块导入依赖关系图看上去比较清晰和简洁，很像一棵树，树枝代表上面输入表逻辑结构中的IMAGE_IMPORT_DESCRIPTIOR结构。Windows的PE加载器就是根据这种依赖关系来加载模块的。

注入是为了让自己的模块可以加载到目标进程空间中，所以根据上述原理，我们可不可以在上面的导入关系中添加一个是我们自己模块的节点，就可以完成注入。

一般情况，输入表的加载都是通过修改进程、输入依赖树上某个模块的输入表来实现的（树上增加节点），这个修改动作是在进程加载该模块之前完成的（即修改模块文件的输入表）。等到待注入模块被加载到进程控件后，就可以将被修改的模块文件改回来，防止文件修改操作被发现。



导入表（IAT）高级玩法实现流程（二）

PE输入表注入的两种方法

静态修改PE文件法：

1.备份原IID结构
2.在原IID区域构造新的OriginalFirstThunk、Name和FirstThunk结构
3.填充新输入表项的IID结构
4.修正PE文件头的信息

进程创建期修改PE输入表法：

1.以挂起方式创建目标进程
2.获取目标进程中的PE结构信息
3.获取原IID大小，增加一项，搜索可用的节空隙
4.构造心的IID及其相关的OriginalFirstThunk、Name和FirstThunk结构
5.修正PE映像头
6.更新目标进程的内存。
7.继续运行主线程

据上述原理介绍，对代码的流程可以如此梳理：

1.写一个我们自己的输入表IID结构
1.新增一个节（块section）来保存旧的输入表IID结构数组和我们新加的输入表IID结构
2.因为新加了节，我们就需要增加一个节表指向我们新加的节
3.因为新加了节和节表就需要更改PE文件头的信息

- 新增节用来存储新导入表
- 复制默认导入表数据到新增节起始位置,并修正IMAGE_OPTIONAL_HEADER目录项中VirtualAddress
- 在默认导入表结构后追加一个导入表并修正其成员.
- Name,开辟一块空间存储DLL名,并修正Name指向(RVA)
- OriginalFirstThunk,指向结构体数组IMAGE_THUNK_DATA32,至少导入一个函数否则系统不会加载此模块
- 开辟8字节内存,前4字节存储函数名(RVA)指向IMAGE_IMPORT_BY_NAME,后4字节设置0即可.
- FirstThunk,同设置INT表相同,开辟对应空间初始化即可

导入表（IAT）高级玩法实现读取文件（三）

```
PVOID FileToMem(IN PCHAR szFilePath, OUT LPDWORD dwFileSize)
{
    //打开文件
    FILE* pFile = fopen(szFilePath, "rb");
    if (!pFile)
    {
        printf("FileToMem fopen Fail \r\n");
        return NULL;
    }

    //获取文件长度
    fseek(pFile, 0, SEEK_END);          //SEEK_END文件结尾
    DWORD Size = ftell(pFile);
    fseek(pFile, 0, SEEK_SET);          //SEEK_SET文件开头

    //申请存储文件数据缓冲区
    PCHAR pFileBuffer = (PCHAR)malloc(Size);
    if (!pFileBuffer)
    {
        printf("FileToMem malloc Fail \r\n");
        fclose(pFile);
        return NULL;
    }

    //读取文件数据
    fread(pFileBuffer, Size, 1, pFile);
```

```
27
28      //判断是否为可执行文件
29      if (*(PSHORT)pFileBuffer != IMAGE_DOS_SIGNATURE)
30      {
31          printf("Error: MZ \r\n");
32          fclose(pFile);
33          free(pFileBuffer);
34          return NULL;
35      }
36
37      if (*(PDWORD)(pFileBuffer + *(PDWORD)(pFileBuffer + 0x3C)) != IMAGE_NT_SIGNATURE)
38      {
39          printf("Error: PE \r\n");
40          fclose(pFile);
41          free(pFileBuffer);
42          return NULL;
43      }
44
45      if (dwFileSize)
46      {
47          *dwFileSize = Size;
48      }
49
50      fclose(pFile);
51
52      return pFileBuffer;
53  }
```

导入表（IAT）高级玩法实现输出文件（四）

```
1   VOID MemToFile(IN PCHAR szFilePath, IN PVOID pFileBuffer, IN DWORD dwFileSize)
2   {
3       //打开文件
4       FILE* pFile = fopen(szFilePath, "wb");
5       if (!pFile)
6       {
7           printf("MemToFile fopen Fail \r\n");
8           return;
9       }
10
11      //输出文件
12      fwrite(pFileBuffer, dwFileSize, 1, pFile);
13
14      fclose(pFile);
15  }
```

导入表（IAT）高级玩法实现输入函数（五）

```
1   VOID ImportInject(PCHAR szModuleName, PCHAR szFunName)
2   {
3       //读取文件二进制数据
4       DWORD dwFileSize = 0;
5       PCHAR pFileBuffer = FileToMem(FILE_PATH_IN, &dwFileSize);
6       if (!pFileBuffer)
7       {
8           return;
9       }
10
11      //新增节存储导入表
12      pFileBuffer = AddNewSection(pFileBuffer, 0x6000, &dwFileSize);
13
14      //定位结构
15      PIMAGE_DOS_HEADER           pDos = (PIMAGE_DOS_HEADER)pFileBuffer;
```

```
    PIMAGE_NT_HEADERS           pNth = (PIMAGE_NT_HEADERS)(pFileBuffer + pDos->e_lfanew);
    PIMAGE_FILE_HEADER          pFil = (PIMAGE_FILE_HEADER)((PUCHAR)pNth + 4);
    PIMAGE_OPTIONAL_HEADER32    pOpo = (PIMAGE_OPTIONAL_HEADER32)((PUCHAR)pFil + IMAGE_SIZEOF_FILE_HEADER);
    PIMAGE_SECTION_HEADER       pSec = (PIMAGE_SECTION_HEADER)((PUCHAR)pOpo + pFil->SizeOfOptionalHeader);

    //判断该PE文件是否有导入表
    if (!pOpo->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress)
    {
        printf("该PE文件不存在导入表 \r\n");
        return;
    }
    PIMAGE_IMPORT_DESCRIPTOR pImp = (PIMAGE_IMPORT_DESCRIPTOR)(pFileBuffer + RvaToFoa(pFileBuffer, pOpo->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress));

    //修复可选PE头中目录项指向
    PUCHAR pCurrent = pFileBuffer + pSec[pFil->NumberOfSections - 1].PointerToRawData;
    pOpo->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress = FoaToRva(pFileBuffer, pCurrent - pFileBuffer);

    //拷贝默认导入表数据
    memcpy(pCurrent, pImp, pOpo->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size - sizeof(IMAGE_IMPORT_DESCRIPTOR));//最后一个导入表后跟随一块同导入表结构大小的0数据.
    pCurrent += pOpo->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].Size - sizeof(IMAGE_IMPORT_DESCRIPTOR);

    //修正新导入表数据
    PIMAGE_IMPORT_DESCRIPTOR pNew = (PIMAGE_IMPORT_DESCRIPTOR)pCurrent;
    pNew->ForwarderChain = -1;
    pNew->TimeDateStamp = 0;

    //新增导入表后填充0数据
    memset(pCurrent + sizeof(IMAGE_IMPORT_DESCRIPTOR), 0, sizeof(IMAGE_IMPORT_DESCRIPTOR));
    pCurrent += sizeof(IMAGE_IMPORT_DESCRIPTOR);

    //INT
    pNew->OriginalFirstThunk = FoaToRva(pFileBuffer, pCurrent - pFileBuffer);
    PDWORD pInt = (PDWORD)pCurrent;
    memset(pInt, 0, 8);
    pCurrent += 8;

    //IAT
    //IAT表地址必须为4字节对齐
    pNew->FirstThunk = FoaToRva(pFileBuffer, pCurrent - pFileBuffer);
    PDWORD pIat = (PDWORD)pCurrent;
    memset(pIat, 0, 8);
    pCurrent += 8;

    //IMAGE_IMPORT_BY_NAME.HINT
    DWORD dwData = (DWORD)pCurrent;
    *(PWORD)pCurrent = 0;
    pCurrent += 2;

    //IMAGE_IMPORT_BY_NAME.Name
    DWORD dwFunNameSize = strlen(szFunName) + 1;
    memcpy(pCurrent, szFunName, dwFunNameSize);
    pCurrent += dwFunNameSize;

    //修复INT IAT指向数据
    *pInt = FoaToRva(pFileBuffer, dwData - (DWORD)pFileBuffer);
    *pIat = FoaToRva(pFileBuffer, dwData - (DWORD)pFileBuffer);

    //拷贝Name数据
    DWORD dwModuleNameSize = strlen(szModuleName) + 1;
    memcpy(pCurrent, szModuleName, dwModuleNameSize);
    //修正结构Name指向
    pNew->Name = FoaToRva(pFileBuffer, pCurrent - pFileBuffer);
    pCurrent += dwModuleNameSize;

    //将二进制数据输出到文件
```

```
81      MemToFile(FILE_PATH_OUT, pFileBuffer, dwFileSize);
82
83  }
```