PE文件学习笔记（五）：导入表、IAT、绑定导入表解析

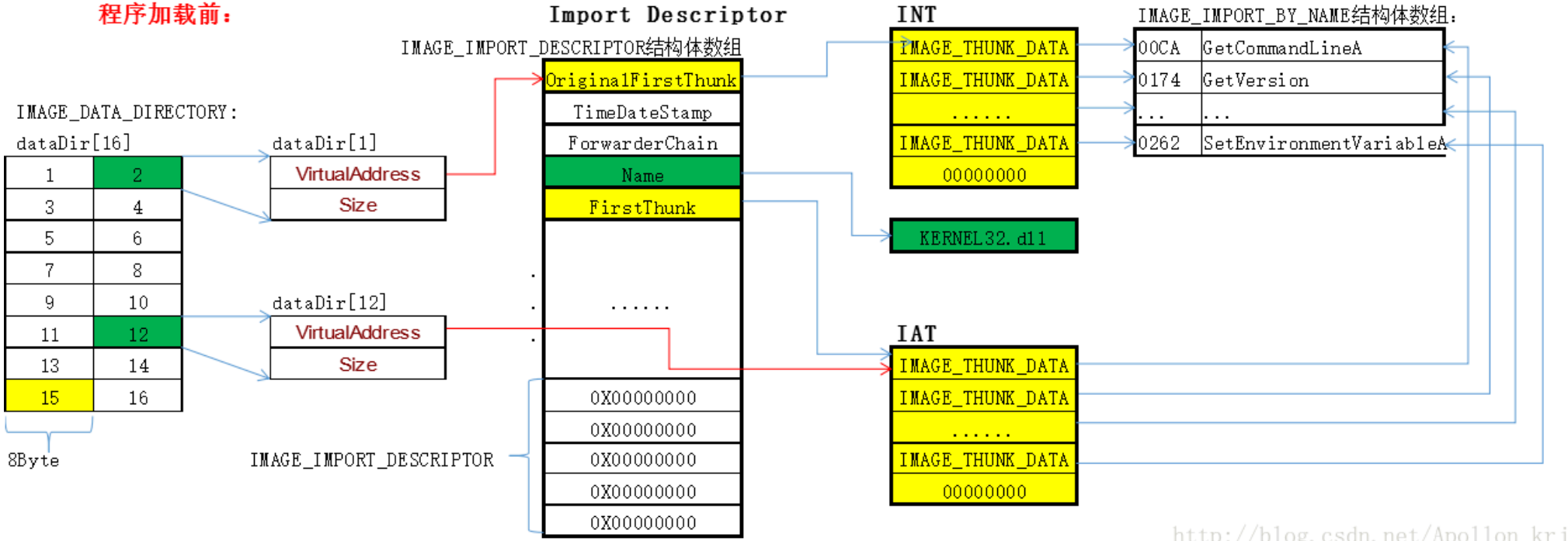
原创 Apollon_krj 于 2017-08-19 23:02:13 发布 阅读量1.3w 收藏 44 点赞数 12
分类专栏： COFF PE/ELF

1、导入表（Import Descriptor）结构解析：

导入表是记录PE文件中用到的动态连接库的集合，一个dll库在导入表中占用一个元素信息的位置，这个元素描述了该导入dll的具体信息。如dll的最新修改时间、dll中函数的名字/序号、dll加载后的函数地址等。而一个元素即一个结构体，一个导入表即该结构体的数组，其结构体如下所示：

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;            //导入表结束标志
        DWORD   OriginalFirstThunk;         //RVA指向一个结构体数组(INT表)
    };
    DWORD   TimeDateStamp;                  //时间戳
    DWORD   ForwarderChain;                 // -1 if no forwarders
    DWORD   Name;                           //RVA指向dll名字，以0结尾
    DWORD   FirstThunk;                     //RVA指向一个结构体数组(IAT表)
} IMAGE_IMPORT_DESCRIPTOR, *PIMAGE_IMPORT_DESCRIPTOR;
```

在程序加载以前，其具体成员的结构关系如下所示：



导入表结构体数组的第一个元素保存了KERNEL32.dll的信息，我们解析并打印其部分信息如下所示：

```
【Name:KERNEL32.dll】 【NameAddr:0003487C】 【OriginalFirstThunk:00034028】   【FirstThunk:000341B4】   【TimeDateStamp:00000000】
        ThunkOffset     ThunkValue      Hint        API Name
        [00034340]      [00034340]      [00CA]      [GetCommandLineA]
        [00034352]      [00034352]      [0174]      [GetVersion]
        [00034360]      [00034360]      [007D]      [ExitProcess]
        [0003436E]      [0003436E]      [029E]      [TerminateProcess]
        [00034382]      [00034382]      [00F7]      [GetCurrentProcess]
        [00034396]      [00034396]      [00FA]      [GetCurrentThreadId]
        [000343AC]      [000343AC]      [02A5]      [TlsSetValue]
        [000343BA]      [000343BA]      [02A2]      [TlsAlloc]
        ......
        [00034850]      [00034850]      [0022]      [CompareStringW]
        [00034862]      [00034862]      [0262]      [SetEnvironmentVariableA]
```

详细解释结构体每个成员的含义（加载前）：

> ①联合体值为0时（一般**用Characteristics判断是否是0**），表示这是导入表结构体数组最后一个元素，除了最后这一个元素，其它每一个结构体都保存了一个dll信息。联合体的值不为0时，**用OriginalFirstThunk（RVA）来索引INT的地址**。这张INT表存放了该dll的导出函数的信息（序号与函数名）。
> ②**TimeDateStamp**：当时间戳值为0时，表示未加载前IAT表与INT表完全相同；当时间戳不为0（为-1）时，表示IAT与INT表不同，IAT存储的是该dll的所有函数的绝对地址，这样在未加载前就直接填充函数地址的方式为函数地址的绑定，其地址是根据绑定导入表来确定的。也就是说当时间戳为-1时绑定导入表才有效，而真正的时间戳存放到绑定导入表中，否则无效。
> ③**ForwarderChain**：一般情况下我们也可以忽略该字段。在老版的绑定中，它引用API的第一个forwarder chain（传递器链表）。
> ④**Name**：RVA指向dll的名字字符串。
> ⑤**FirstThunk**：RVA指向IAT表。

2、IAT（Import Address Table）、INT（import Name Table）结构解析：

关于绑定导入表和IAT表的特殊情况这里先不做研究，我们先来看看IAT和INT结构相同的时情况。加载到内存前我们看到IAT和INT都指向一个结构体数组，这个数组存储了序号和函数名。IAT和INT的元素为**IMAGE_THUNK_DATA**结构，而其指向为**IMAGE_IMPORT_BY_NAME**结构，这两个结构体如下所示：

IMAGE_THUNK_DATA结构体汇总只有一个联合体，一般用四字节的AddressOfData来获取**IMAGE_IMPORT_BY_NAME**的地址。

```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        DWORD ForwarderString;      // PBYTE
        DWORD Function;             // PDWORD
        DWORD Ordinal;
        DWORD AddressOfData;        //RVA 指向_IMAGE_IMPORT_BY_NAME
    } u1;
} IMAGE_THUNK_DATA32;
typedef IMAGE_THUNK_DATA32 * PIMAGE_THUNK_DATA32;
```

IMAGE_IMPORT_BY_NAME里有两个成员一个是序号一个是函数名。

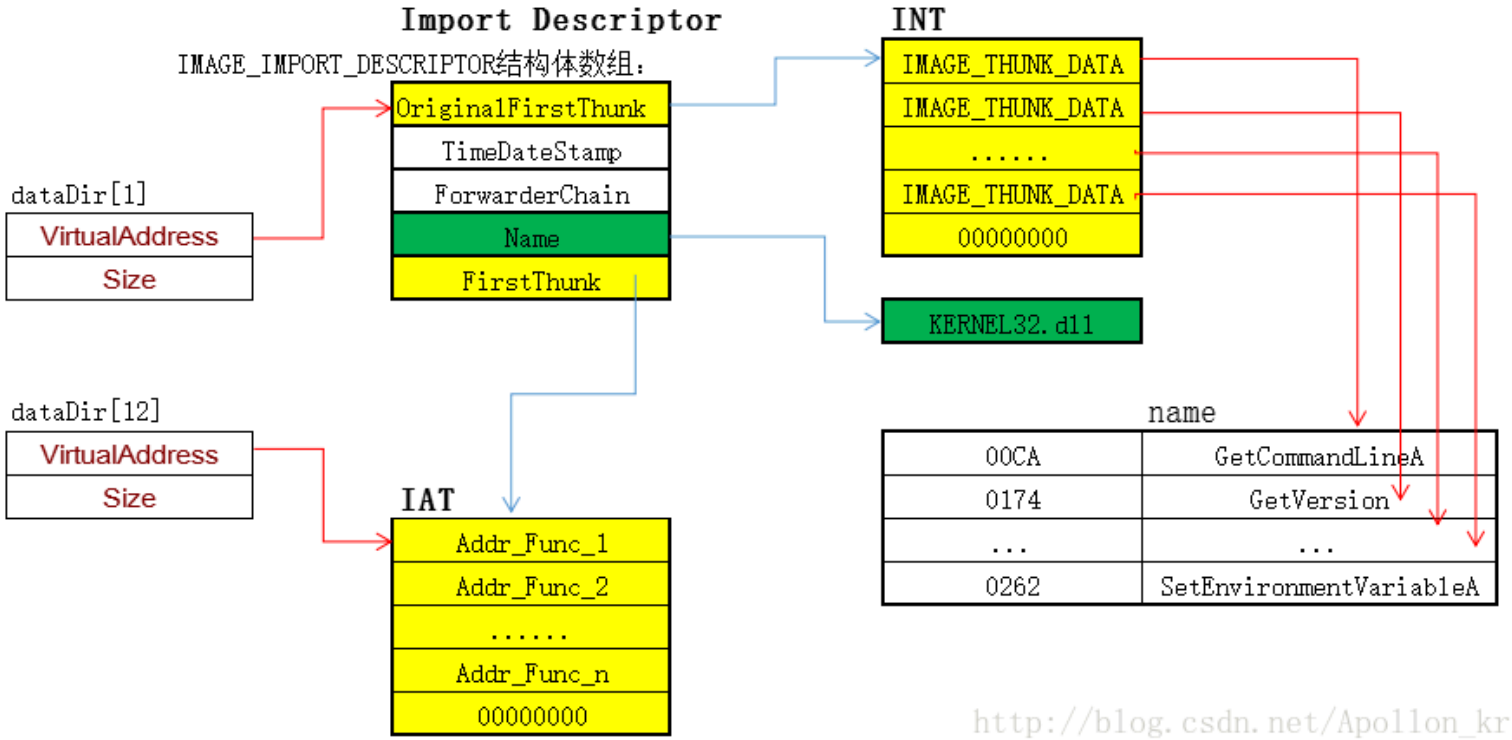
```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;       //可能为0，编译器决定，如果不为0，是函数在导出表中的索引

```

```
    BYTE    Name[1];    //函数名称，以0结尾，由于不知道到底多长，所以干脆只给出第一个字符，找到0结束
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

注意：一个**IMAGE_THUNK_DATA32**结构占用四字节，索引一个函数名/序号，但是索引是有条件的，即四字节的最高位如果为0则这四字节的值为**IMAGE_IMPORT_BY_NAME**的RVA；但是如果四字节的最高位为1，则不需要（不能够）用该值去索引**IMAGE_IMPORT_BY_NAME**，而是直接去掉最高位，剩下31位的值便是dll函数在导出表中的导出序号。如下所示，有最高位为0解析出来的也有最高位为1解析出来的导入表：

```
1  //最高位为0，则根据值索引IMAGE_IMPORT_BY_NAME解析hint和name
2  【Name:WINSPOOL.DRV】【NameAddr:000314EE】【OriginalFirstThunk:00030390】  【tFirstThunk:0002844C】  【TimeDateStamp:00000000】
3     ThunkOffset     ThunkValue      Hint        API Name
4     [000314B8]      [000314B8]      [001B]      [ClosePrinter]
5     [000314C8]      [000314C8]      [0046]      [DocumentPropertiesA]
6     [000314DE]      [000314DE]      [007D]      [OpenPrinterA]
7  【Name:ADVAPI32.dll】【NameAddr:00031590】【OriginalFirstThunk:0002FF44】  【tFirstThunk:00028000】  【TimeDateStamp:00000000】
8     ThunkOffset     ThunkValue      Hint        API Name
9     [0003157E]      [0003157E]      [0204]      [RegSetValueExA]
10    [0003156C]      [0003156C]      [01D1]      [RegCreateKeyExA]
11    [0003155A]      [0003155A]      [01F6]      [RegQueryValueA]
12    [0003154C]      [0003154C]      [01EB]      [RegOpenKeyA]
13    [0003153E]      [0003153E]      [01DD]      [RegEnumKeyA]
14    [0003152E]      [0003152E]      [01D4]      [RegDeleteKeyA]
15    [0003151E]      [0003151E]      [01EC]      [RegOpenKeyExA]
16    [0003150A]      [0003150A]      [01F7]      [RegQueryValueExA]
17    [000314FC]      [000314FC]      [01CB]      [RegCloseKey]
18 【Name:SHLWAPI.dll】  【NameAddr:000315C8】【OriginalFirstThunk:000301E4】  【FirstThunk:000282A0】  【TimeDateStamp:00000000】
19    ThunkOffset     ThunkValue      Hint        API Name
20    [0003159E]      [0003159E]      [002F]      [PathFindExtensionA]
21    [000315B4]      [000315B4]      [0031]      [PathFindFileNameA]
22 //最高位为1，去掉最高位得到函数序号
23 【Name:OLEAUT32.dll】【NameAddr:000315D4】【OriginalFirstThunk:000301D4】  【FirstThunk:00028290】  【TimeDateStamp:00000000】
24    ThunkOffset     ThunkValue      Hint        API Name
25    [00000009]      [00000009]      [--]        函数序号[0009H:9D]
26    [0000000C]      [0000000C]      [--]        函数序号[000CH:12D]
27    [00000008]      [00000008]      [--]        函数序号[0008H:8D]
```

以上是程序加载前的情况，IAT和INT指向同一结构，而加载后INT不变依旧保存dll函数名与函数序号的地址信息。而IAT则根据导入表INT（IAT加载前）的内容和导出表信息，修改为对应的函数的地址信息，如下所示：



3、绑定导入表（Bound Import Descriptor）与IAT：

我们上面分析了加载前，IAT中存储非函数地址的情况，下面我们来分析加载前IAT表中存储函数地址的情况。IAT中存储的函数地址是dll未加载的地址，当PE文件中不存在绑定导入表时，IAT就与INT一样，此时导入表中的时间戳就为0；否则导入表中的时间戳为-1时，dll的真正时间戳存放于绑定导入表中（绑定导入表地址存放在数据目录的第12项，IAT是第13项）。

现在大多数情况，导入表的TimeDateStamp都为0，而Windows早期的自带软件（如WinXP的notepad.exe）基本都采用了TimeDateStamp为-1的情况即包含绑定导入表的情况。PE中包含导入表的优点是程序启动快，但是其缺点也十分明显，当存在dll地址重定位和dll修改更新，则绑定导入表也需要修改更新。

绑定导入表的结构由两个结构体来组成：

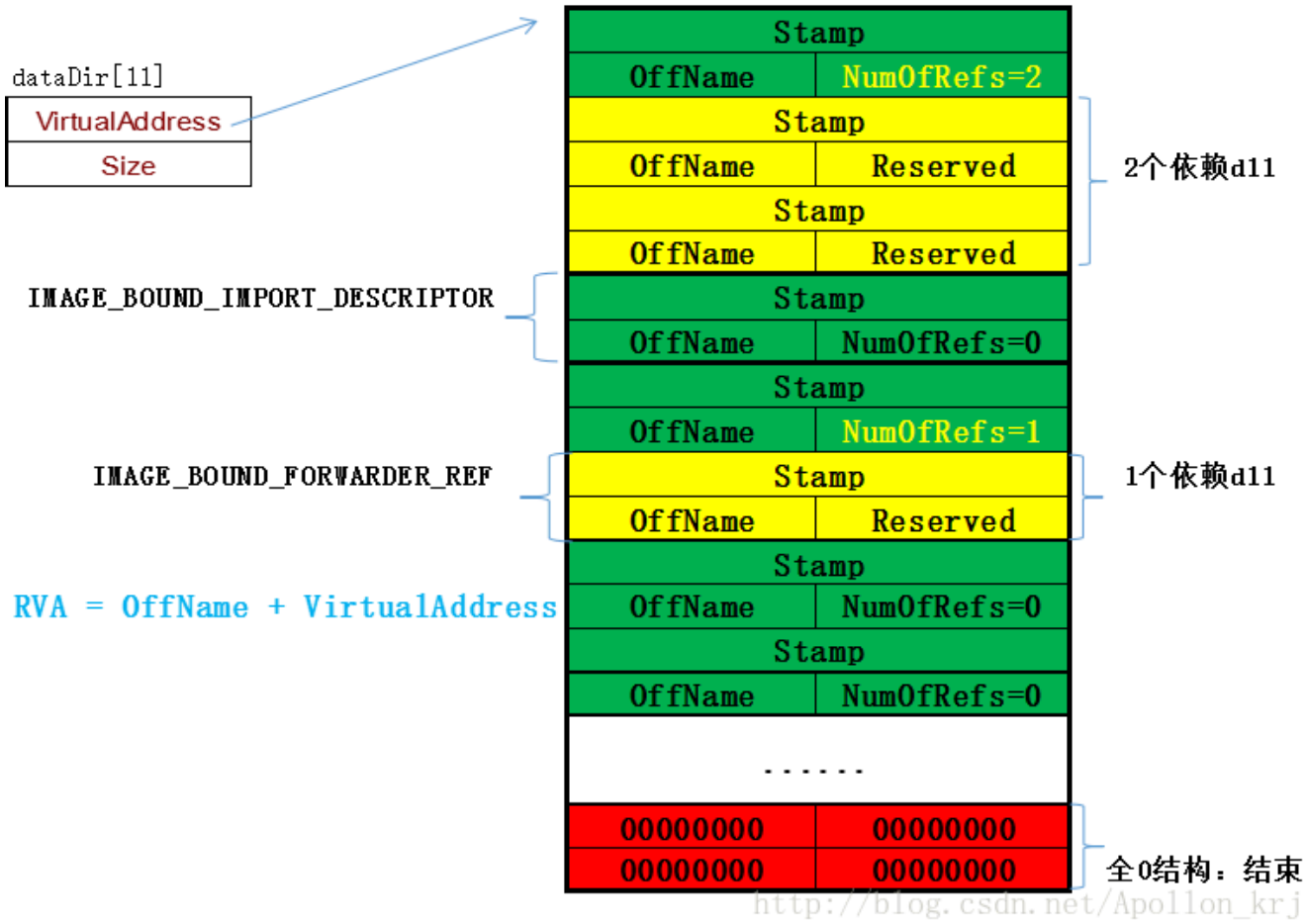
```
//最后一个结构全0表示绑定导入表结束
typedef struct _IMAGE_BOUND_IMPORT_DESCRIPTOR {
    DWORD   TimeDateStamp;      //表示绑定的时间戳，如果和PE头中的TimeDateStamp不同则可能被修改过
    WORD    OffsetModuleName;   //dll名称地址
    WORD    NumberOfModuleForwarderRefs;    //依赖dll个数
// Array of zero or more IMAGE_BOUND_FORWARDER_REF follows
} IMAGE_BOUND_IMPORT_DESCRIPTOR,  *PIMAGE_BOUND_IMPORT_DESCRIPTOR;
```

NumberOfModuleForwarderRefs是指该dll自身依赖的dll的个数。值为n代表该结构后面紧跟了n个**IMAGE_BOUND_FORWARDER_REF**结构。之后才是导入表导入的下一个dll的结构。而**IMAGE_BOUND_FORWARDER_REF**结构体如下所示：

```
typedef struct _IMAGE_BOUND_FORWARDER_REF {
    DWORD   TimeDateStamp;  //时间戳，同样的作用检查更新情况
    WORD    OffsetModuleName;   //dll名称地址
    WORD    Reserved;   //保留
} IMAGE_BOUND_FORWARDER_REF, *PIMAGE_BOUND_FORWARDER_REF;
```

注意：这两个结构体中所有的OffsetModuleName均不是相对于ImageBase的RVA也不是FOA，而是相对于绑定导入表首地址的偏移地址，即：**绑定导入表首地址 + OffsetModuleName= RVA。**

绑定导入表结构图解如下所示：



打印出的WinXP自带notepad.exe的绑定导入表：

```
绑定导入表(Bound Import Descriptor):
    DllName:comdlg32.dll
        TimeDateStamp:[4802BDA2H:1208139170D]
        GMT:2008-04-14 02:12:50
        OffsetModuleName:0058
```

```
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:SHELL32.dll
        TimeDateStamp:[4802BDB6H:1208139190D]
        GMT:2008-04-14 02:13:10
        OffsetModuleName:0065
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:WINSPOOL.DRV
        TimeDateStamp:[4802BDCAH:1208139210D]
        GMT:2008-04-14 02:13:30
        OffsetModuleName:0071
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:COMCTL32.dll
        TimeDateStamp:[4802BD6CH:1208139116D]
        GMT:2008-04-14 02:11:56
        OffsetModuleName:007E
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:msvcrt.dll
        TimeDateStamp:[4802BD6CH:1208139116D]
        GMT:2008-04-14 02:11:56
        OffsetModuleName:008B
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:ADVAPI32.dll
        TimeDateStamp:[4802BD89H:1208139145D]
        GMT:2008-04-14 02:12:25
        OffsetModuleName:0096
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:KERNEL32.dll
        TimeDateStamp:[4802BDC6H:1208139206D]
        GMT:2008-04-14 02:13:26
        OffsetModuleName:00A3
        NumberOfModuleForwarderRefs:[0001H:1D]
        ##############################################
        DllName:NTDLL.DLL
            TimeDateStamp:4802BDC5
            GMT:2008-04-14 02:13:25
            OffsetModuleName:00B0
            Reserved:0000
```

```
    DllName:GDI32.dll
        TimeDateStamp:[4802BD81H:1208139137D]
        GMT:2008-04-14 02:12:17
        OffsetModuleName:00BA
        NumberOfModuleForwarderRefs:[0000H:0D]
    DllName:USER32.dll
        TimeDateStamp:[4802BDBDH:1208139197D]
        GMT:2008-04-14 02:13:17
        OffsetModuleName:00C4
        NumberOfModuleForwarderRefs:[0000H:0D]
```

IAT表部分信息：

```
IAT表(Import Address Table):
    dllName:【comdlg32.dll】:
        Function Addr:[76344906]
        Function Addr:[763385CE]
        Function Addr:[76349D84]
        Function Addr:[7633C3E1]
        Function Addr:[76322306]
        Function Addr:[76337B9D]
        Function Addr:[76338602]
        Function Addr:[76330036]
        Function Addr:[76337C2B]
    dllName:【SHELL32.dll】:
        Function Addr:[7D647C18]
        Function Addr:[7D5E18CE]
        Function Addr:[7D5FB1A9]
        Function Addr:[7D632E6F]
    dllName:【WINSPOOL.DRV】:
        Function Addr:[72F7643C]
        Function Addr:[72F74D40]
        Function Addr:[72F75091]
    dllName:【COMCTL32.dll】:
        Function Addr:[7718D270]
```

```
    dllName:【msvcrt.dll】:
        Function Addr:[4CFB2DAE]
        Function Addr:[4CFB9E9A]
        ......
```

与上面IAT所对应的INT表的部分信息（INT与IAT是一一对应的）：

```
导入表(Import Descriptor):
    【Name:comdlg32.dll】 【NameAddr:00006EAC】 【OriginalFirstThunk:00006D90】    【FirstThunk:000006C4】    【TimeDateStamp:FFFFFFFF】
        ThunkOffset     ThunkValue      Hint        API Name
        [00006E7A]      [00006E7A]      [000F]      [PageSetupDlgW]
        [00006E5E]      [00006E5E]      [0006]      [FindTextW]
        [00006E9E]      [00006E9E]      [0012]      [PrintDlgExW]
        [00006E50]      [00006E50]      [0003]      [ChooseFontW]
        [00006E40]      [00006E40]      [0008]      [GetFileTitleW]
        [00006E8A]      [00006E8A]      [000A]      [GetOpenFileNameW]
        [00006E6A]      [00006E6A]      [0015]      [ReplaceTextW]
        [00006E14]      [00006E14]      [0004]      [CommDlgExtendedError]
        [00006E2C]      [00006E2C]      [000C]      [GetSaveFileNameW]
    【Name:SHELL32.dll】  【NameAddr:00006EFA】 【OriginalFirstThunk:00006C40】    【FirstThunk:00000574】    【TimeDateStamp:FFFFFFFF】
        ThunkOffset     ThunkValue      Hint        API Name
        [00006EC8]      [00006EC8]      [001F]      [DragFinish]
        [00006ED6]      [00006ED6]      [0023]      [DragQueryFileW]
        [00006EE8]      [00006EE8]      [001E]      [DragAcceptFiles]
        [00006EBA]      [00006EBA]      [0103]      [ShellAboutW]
    【Name:WINSPOOL.DRV】 【NameAddr:00006F3A】 【OriginalFirstThunk:00006D80】    【FirstThunk:000006B4】    【TimeDateStamp:FFFFFFFF】
        ThunkOffset     ThunkValue      Hint        API Name
        [00006F16]      [00006F16]      [0078]      [GetPrinterDriverW]
        [00006F06]      [00006F06]      [001B]      [ClosePrinter]
        [00006F2A]      [00006F2A]      [007E]      [OpenPrinterW]
    【Name:COMCTL32.dll】 【NameAddr:00006F5E】 【OriginalFirstThunk:00006AEC】    【FirstThunk:00000420】    【TimeDateStamp:FFFFFFFF】
        ThunkOffset     ThunkValue      Hint        API Name
        [00006F48]      [00006F48]      [0008]      [CreateStatusWindowW]
    【Name:msvcrt.dll】   【NameAddr:00007076】 【OriginalFirstThunk:00006DB8】    【FirstThunk:000006EC】    【TimeDateStamp:FFFFFFFF】
        ThunkOffset     ThunkValue      Hint        API Name
        [00006FDC]      [00006FDC]      [004E]      [_XcptFilter]
```

```
29
30
31          [00006FD4]      [00006FD4]      [00F6]      [_exit]
            ......
```

4、代码解析导入表（INT、IAT）与绑定导入表：

```
    void PETool::print_ImportDescriptor()
{
1       fprintf(fp_peMess, "导入表(Import Descriptor):\n");
2       if(dataDir[1].VirtualAddress == 0){
3           fprintf(fp_peMess, "\t不存在导入表!\n");
4           return;
5       }
6       char str[TIMESTRING] = {0};
7       //导入表为数据目录的第2项,将import指向导入表第一个结构体
8       IMAGE_IMPORT_DESCRIPTOR * import = (IMAGE_IMPORT_DESCRIPTOR *)(pFileBuffer + RVAToFOA(dataDir[1].VirtualAddress));
9       while(true){
10          if(import->Characteristics == 0){
11              break;//最后一个结构体20字节为0则结束(直接判断一个Characteristics即可)
12          }
13          DWORD name = RVAToFOA(import->Name);
14          DWORD original_ft = RVAToFOA(import->OriginalFirstThunk);
15          DWORD ft = RVAToFOA(import->FirstThunk);
16          //打印结构体信息
17          fprintf(fp_peMess, "\t【Name:%s】\t"
18                             "【NameAddr:%08X】\t"
19                             "【OriginalFirstThunk:%08X】\t"
20                             "【FirstThunk:%08X】\t"
21                             "【TimeDateStamp:%08X】\n",
22                  pFileBuffer + name, name, original_ft, ft, import->TimeDateStamp);
23          memset(str, 0, TIMESTRING);
24
25          IMAGE_THUNK_DATA32 * thunk = (IMAGE_THUNK_DATA32 * )(pFileBuffer + original_ft);
26          //打印INT表的详细信息
27          print_INT(thunk);
28          import++;
29
```

```
    }
}
void PETool::print_INT(IMAGE_THUNK_DATA32 * thunk)
{
    fprintf(fp_peMess, "\t\tThunkOffset\t\tThunkValue\t\tHint\t\tAPI Name\n");
    while(true){
        DWORD thunkValue = thunk->u1.AddressOfData;
        if(thunkValue == 0){
            break;//读取完毕
        }
        if(thunkValue >> 31){//最高位为1打印序号
            DWORD rva = thunkValue & 0X7FFFFFFF;//去掉最高位才是实际的值,否则RVAToFOA会出错
            DWORD offset = RVAToFOA(rva);
            fprintf(fp_peMess, "\t\t[%08X]\t\t[%08X]\t\t[--]\t\t函数序号[%04XH:%dD]\n",
                    offset, offset, rva, rva);
        }else{//最高位为0打印名称
            DWORD offset = RVAToFOA(thunkValue);
            //获取IMAGE_IMPORT_BY_NAME的地址
            IMAGE_IMPORT_BY_NAME * byName = (IMAGE_IMPORT_BY_NAME * )(pFileBuffer + offset);
            fprintf(fp_peMess, "\t\t[%08X]\t\t[%08X]\t\t[%04X]\t\t[%s]\n",
                    offset, offset, byName->Hint, byName->Name);
        }
        thunk++;
    }
}

void PETool::print_IAT()
{
    fprintf(fp_peMess, "IAT表(Import Address Table):\n");
    IMAGE_IMPORT_DESCRIPTOR * import = (IMAGE_IMPORT_DESCRIPTOR *)(pFileBuffer + RVAToFOA(dataDir[1].VirtualAddress));
    while(true){
        if(import->Characteristics == 0){
            break;
        }
        DWORD * addr = (DWORD *)(pFileBuffer + RVAToFOA(import->FirstThunk));
        //根据导入表的时间戳判断IAT中存放的是函数地址还是名字结构体的地址
        if(import->TimeDateStamp == -1){//函数地址
```

```
            fprintf(fp_peMess, "\tdllName:【%s】:\n", pFileBuffer + RVAToFOA(import->Name));
            for(int i = 0; addr[i]; i++){
                fprintf(fp_peMess, "\t\tFunction Addr:[%08X]\n", addr[i]);
            }
        }
        else if(import->TimeDateStamp == 0){//等同于INT表
            fprintf(fp_peMess, "\t等同于INT表!\n");
            break;
        }
        import++;
    }
}

void PETool::print_BoundImportDescriptor()
{
    fprintf(fp_peMess, "绑定导入表(Bound Import Descriptor):\n");
    if(dataDir[11].VirtualAddress == 0){
        fprintf(fp_peMess, "\t不存在绑定导入表!\n");
        return;
    }
    DWORD desAddr = dataDir[11].VirtualAddress;//获取第一个Bound Import Descriptor的RVA
    char str[TIMESTRING] = {0};\
    DWORD stamp = 0, off = 0, ref = 0, i = 0;

    IMAGE_BOUND_IMPORT_DESCRIPTOR * bound = (IMAGE_BOUND_IMPORT_DESCRIPTOR * )(pFileBuffer + RVAToFOA(desAddr));
    while(bound->TimeDateStamp != 0 && bound->OffsetModuleName != 0){
        stamp = bound->TimeDateStamp;//获取时间戳
        TimeDateStampToString(stamp, str);//时间戳转时间
        off = bound->OffsetModuleName;//获取名字偏移地址
        ref = bound->NumberOfModuleForwarderRefs;//获取依赖dll数

        fprintf(fp_peMess, "\tDllName:%s\n", pFileBuffer + RVAToFOA(desAddr + off));
        fprintf(fp_peMess, "\t\tTimeDateStamp:[%08XH:%dD]\n", stamp, stamp);
        fprintf(fp_peMess, "\t\tGMT:%s\n", str);
        fprintf(fp_peMess, "\t\tOffsetModuleName:%04X\n", off);

```

```
        fprintf(fp_peMess, "\t\tNumberOfModuleForwarderRefs:[%04XH:%dD]\n", ref, ref);

        IMAGE_BOUND_FORWARDER_REF * boundFor = (IMAGE_BOUND_FORWARDER_REF *)(bound);
        for(boundFor++, i = 0; i < ref; i++, boundFor++){
            memset(str, 0, TIMESTRING);
            off = boundFor->OffsetModuleName;
            stamp = boundFor->TimeDateStamp;
            TimeDateStampToString(stamp, str);

            fprintf(fp_peMess, "\t\t################################################\n");
            fprintf(fp_peMess, "\t\tDllName:%s\n", pFileBuffer + RVAToFOA(desAddr + off));
            fprintf(fp_peMess, "\t\t\tTimeDateStamp:%08X\n",stamp);
            fprintf(fp_peMess, "\t\t\tGMT:%s\n", str);
            fprintf(fp_peMess, "\t\t\tOffsetModuleName:%04X\n", off);
            fprintf(fp_peMess, "\t\t\tReserved:%04X\n", boundFor->Reserved);
        }
        bound = (IMAGE_BOUND_IMPORT_DESCRIPTOR *)(boundFor);//下一个绑定dll
        memset(str, 0, TIMESTRING);
    }
}
```