怎样建立你自己的 MASM 导入库

作者：Akkuman
原文链接：https://ld246.com/article/1518195524410
来源网站：链滴

by Iczelion （翻译：花心萝卜yqzq@163.net) 9.5.2000

这篇短文是讲述关于建立MASM导入库（import libraries）技巧，我假设你已经知道什么是导入库在下面，我将集中讲述建立MASM导入库的方法。

<!--more-->

MASM导入库的格式：

MASM和VC++可以使用相同的导入库，MS导入库使用不同于TASM的OMF格式的变更的COFF文件式，这就是为什么TASM和MASM的导入库不能互用的原因，我将不详细介绍有关MS导入库的格式可以这样说，每一个MS导入库都包含某个DLL中函数的信息（你将要用这些信息来调用DLL中的函数，这些信息包括函数名和它所有参数的尺寸。如果你用一个文本编辑器打开kernel32.lib，你回发现些如下格式的信息：

```
_ExitProcess@4
_CreateProcessA@40
```

函数名被装饰上了一个“_”，在“@”之后的数字表示了该函数所有参数的尺寸（字节为单位），ExiProcess 函数只有一个DWORD的参数，所以后面的数字是4。 LIB中为什么要包含这些参数尺寸的信呢？当你用INVOKE调用函数时，这些信息被用来检测传递给函数的参数是否正确。如果你使用“手”将参数压入堆栈，并通过“CALL”来调用函数的话，MASM将无法检测参数是否正确。这将导致们几乎没有办法建立一个DLL的导入库，因为DLL并不包含清楚的关于参数尺寸的信息。

从DLL建立MASM导入库

如果你很乐意用“手动”（CALL）的方法去调用函数的话，你可以象下面这样为任何一个DLL建立MSM的导入库：

使用dumpbin.exe,它可以导出DLL 输出（EXPORT）函数的名字。

```
Dumpbin /EXPORTS  blah.dll  >  output.txt
```

在你获得了函数名列表之后，通过他们建立一个模块定义文件（.DEF）。 举个例子：如果DLL只包含个函数：GetSomeLine 在一个文本文件中输入如下内容：

```
LIBRARY blah
EXPORTS
GetSomeLine
```

并将其保存为“blah.def

象这样，运行lib.exe，通过模块定义文件建立一个导入库：

```
lib /DEF:blah.def
```

就是它了！你将获得blah.lib,只要你不使用INVOKE调用函数的话，你就可以在MASM中使用它。

建立通过INVOKE调用函数的MASM导入库:

我并不反对你使用上面的方法，但INVOKE确实是一个调用函数的好途径。这也是我较TASM更喜欢MSM的原因之一。但就象我早先强调的，我们几乎不可能从一个DLL建立一个能100%工作的MASM导库。如果你使用INVOKE，你将不能用上面的方法建立一个MASM导入库。举个例子，你可以想象如你在.DEF文件中修改了函数的“@XX”部分，导入库将仍然正常建立，但请相信我，他不会工作的。建立一个可以使用INVOKE的导入库的一个简单的方法是使用MASM。如果你写过DLL的代码，你会现你不仅的到了一个DLL，而且还得到了一个导入库，没错，它就是我们要得！ 我们的策略是：

1. 获得函数名和所有参数的尺寸
2. 建立一个包含正确个数和尺寸的DLL源代码
3. 建立一个描述ASM源代码中相应函数的模块定义文件（.DEF）
4. 将源代码按DLL汇编

你将获得一个功能完全的MASM导入库，上面的步骤应做更多的说明

获得函数名和所有参数尺寸

这是我们处理过程中最困难的部分了。如果你仅仅只有DLL，你将经历无意义的冒险。下面是我所能出的方法，不过没有一个能100%工作。

使用交互式反编译工具（Interactive Disassembler (IDA)）反编译DLL，通过这个奇妙的工具，你可获得函数参数的大概尺寸，但这些信息是不完全的，IDA是一个功能强大的工具，不过有时必须靠我自己判断什么是什么。你将不得不仔细分析反编译后的结果。

观察堆栈指针在调用函数之前和之后的值。方法如下：

1. 通过GetProcAddress获得函数的地址。
2. 调用想要测试的每一个函数，但请注意，调用这些函数时，不要给他们传递任何的参数。调用前请意ESP的值。
3. 当函数返回后，比较调用函数前、后ESP的值。基本原理是：stdcall参数调用协定规定，函数自己责恢复堆栈，现在知道为什么我们要不传递任何参数了吧，我们没传递参数，而函数却自作聪明“恢”了ESP指针，所以ESP的变化值就是我们要得参数尺寸了。

不过，上面的方法并不是万无一失的，下面的这些情况将会导致失败：

- 如果DLL中的函数使用了不同于stdcall的别的参数传递协定。
- 如果函数在恢复堆栈时失败，我们将无法得到ESP的正确值。
- 如果这个函数的作用是去做一些危险的事情，比如硬盘格式化，那我们即使得到了ESP，恐怕代价了点

研究现有的使用DLL的程序，你可以通过调试/反编译这些程序去获得函数参数的个数和尺寸。不论如，只要有函数在DLL中，而又没有任何程序调用过它，你可以用上面的两个方法。

建立我们自己的DLL

在你获得了函数的名字和参数尺寸后，你可以建立一个DLL框架并在框架中添加和其他DLL、文件中相同名称的函数。举个例子，如果DLL只含有一个函数：GetSomeLine.它有16BYTES的参数。在AS文件中，你可以这样写：

```
.386
.model flat,stdcall
.code
GetSomeLine proc param1:DWORD, param2:DWORD, param3:DWORD, param4:DWORD
GetSomeline endp
end
```

你可能要问，“这是什么？”。一个没有处理部分的程序？请记住：一个导入库并没有记录一个函数如何实现的，它只是记录函数名和参数尺寸而已，它的任务就是提供函数的名称和尺寸。所以我们不要添加函数的处理部分。当我们建立DLL时，MASM会帮我们完成它的导入库的建立。 MASM在建导入库时并不关心每个具体参数的尺寸，它总是象下面这样：

```
.386
.model flat,stdcall
.code
GetSomeLine proc param1:BYTE, param2:BYTE, param3:BYTE, param4:BYTE
GetSomeline endp
end
```

然后MASM将在导入库中建立_GetSomeLine@16(它会把每一个参数看作DWORD)，而并不管它的数是4个BYTE还是DWORD或是其他什么

建立匹配的模块定义文件（.DEF）

这是一个简单的工作，你需要这个文件来指导MASM去建立正确的DLL和与之匹配的导入库。一个模定义文件模板如下：

```
LIBRARY  <The name of the DLL>
EXPORTS
<The names of the functions>
```

你仅仅需要填入DLL的名字，然后在EXPORTS下添入函数的名字。每个函数名一行。保存文件，你将得一个模块定义文件。

汇编DLL源代码

最后一步也是最简单的一步，仅仅需要ML.EXE和LINK.EXE

```
ml /c /coff /Cp blah.asm
link /DLL /NOENTRY /def:blah.def /subsystem:windows blah.obj
```

好了，查看一下你的项目目录，你会发现你想要的导入库和DLL。

转自http://blog.csdn.net/taowen2002/article/details/15837