

2021/01/13_32位汇编_第4课_代码注入

笔记本: 32位汇编
创建时间: 2021/1/13 星期三 10:47
作者: ileemi

- [编译器如何获取API地址](#)

汇编（二进制）注入代码

使用过程函数计算函数所占的字节数，会生成一些额外的代码（伪指令生成的），使用标号计算标号内部代码所占的字节数就不会生成额外的代码。

ollydbg 目录下的 UDD文件夹 会保存上次调试程序的配置信息

编译器如何获取API地址

API 调用的流程

程序中使用了API（MessageBoxA），在编译阶段编译器不知道API的地址，程序就无法通过编译（MessageBoxA函数地址不固定），MessageBoxA 的地址需要操作系统加载 USER32.dll 之后在知道其地址（也就是MessageBoxA的地址需要程序运行起来才知道）。

编译器在编译阶段获取API地址方法：

- 定义一个未初始化的全局变量（编译器负责构造，一般在主模块中定义）
- 将全部遍历的地址当作 MessageBoxA 的地址
- 等到程序运行起来的时候（成功加载 USER32.dll 之后），将全局变量的地址替换成 MessageBoxA 的地址（API运行之前由操作系统完成这一步骤），主要用来解决未初始化地址的编译问题

```
.data?  
    MY_MSG dd ?  
.code  
START  
    xxx  
    call dword ptr [MY_MSG]  
    xxx  
end START
```

通过ollydbg调试程序：

call <jmp.&user32.MessageBoxA>：双击（call 00401156）后，按 "enter"

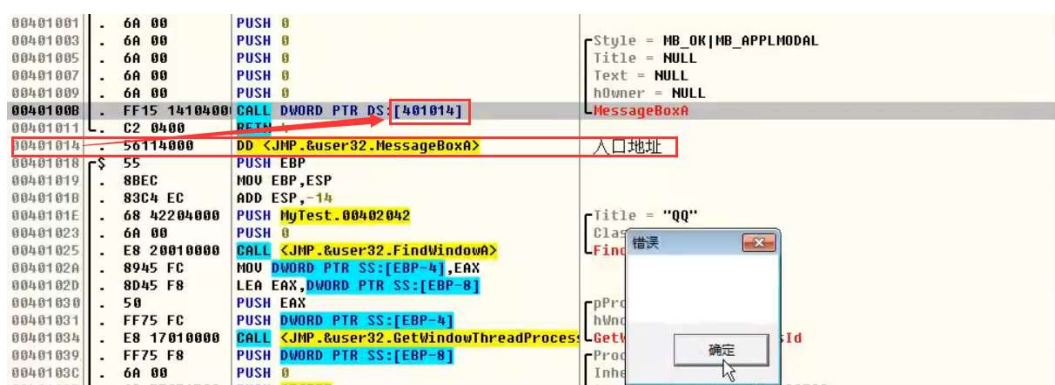


类似于对全局变量取内容，同一台机器上两个进程所使用相同的API，API的地址是否相同，需要根据所在库的模块基址是否一致进行判断。

代码注入，方法1：

```
.data?
    MY_MSG dd ?

.code
INJECT_CODE_START:
    push 0
    push 0
    psuh 0
    push 0
    call dword ptr [MSGBOX_ADD]
    retn 4
MSGBOX_ADD dd offset MessageBoxA
INJECT_CODE_END:
START
    psuh 0
    call INJECT_CODE_START
    xxx
end START
```



地址相对固定，目标进程中可能没有这个地址

方法2：代码重定位

API地址先使用四个字节数据进行填充，当目标进程运行起来的时候，获取目标进程的API地址，之后将其地址填到自己的进程调用API的地址上。

- 使用 LoadLibrary 获取模块基址 (user32.dll)
- 使用 GetProcAddress 获取 API 的地址

- 在API地址前添加一个标号，将获取到的API地址和编译时API的临时地址进程更换（需要使用 **VirtualProtect** 修改内存保护属性（可读可写可执行）），例如：

mov dword ptr [label1 + 1], eax ; +1 是因为需要跳过一个字节的操作码

0040100B	B8	DB B8	
0040100C	1EF08875	DD user32.MessageBoxA	
00401010	FFD0	CALL EAX	
00401012	C2 0400	RETN 4	
00401015	55	PUSH EBP	
00401016	8BEC	MOV EBP,ESP	
00401018	83C4 E4	ADD ESP,-1C	
0040101B	8D45 E4	LEA EAX,DWORD PTR SS:[EBP-1C]	
0040101E	50	PUSH EAX	
0040101F	6A 40	PUSH 40	
00401021	68 00100000	PUSH 1000	
00401024	68 00100000	PUSH 1000	

地址	HEX 数据	ASCII	0018FA28	00000000
00401000	CC 6A 00 6A 00 6A 00 6A 00 6A 00 6A 00 B8 1E FD 88 75	...	0018FA2C	75820000
00401010	FF D0 C2 04 00 55 8B EC 83 C4 E4 8D 45 E4 50 6A	...	0018FA30	0018FA28
00401020	40 68 00 10 00 00 00 00 10 40 00 2B 3E 01 00 00	...	0018FA34	00000001
00401030	68 40 20 40 00 B8 22 01 00 00 89 45 E8 68 58 20	...	0018FA38	00000000
00401040	40 00 FF 75 E8 28 0C 01 00 00 A3 0C 10 40 00 68	...	0018FA3C	00971E20
00401050	4A 20 40 00 6A 00 E8 2B 01 00 00 89 45 FC 8D 45 1E	...	0018FA40	00000000

代码示例：

```
.const
MY_USER32 "USER32.lib", 0
MY_MSGBOX "MessageBoxA", 0

.data?
    MY_MSG dd ?

.code
INJECT_CODE_START:
    ;push ebp
    ;push ebp, esp
    push 0
    push 0
    psuh 0
    push 0

label1:
    mov eax, 12345678h
    call eax
    ;call [ebp+8]leave
    ;leave
    ; leave 等价 mov esp, ebp pop ebp --> 过程函数中都被编译器翻译成
leave（指令周期更快）
    retn 4
INJECT_CODE_END:
START
    local @old:dword
    ; 修该内存保护属性
    invoke VirtualProtect, offset INJECT_CODE_START, 1000h,
PAGE_EXECUTE_READWRITE, add @old
    invoke LoadLibrary, offset MY_USER32
    invoke GetProcAddress, @hUser32, offset MY_MSGBOX
    mov dword ptr [label1+1], eax
```

```
XXX  
end START
```

不进行代码重定位也可以，获取到API的地址后，将其通过远程线程的参数进行传递也是可行的。

当目标程序使用的API和注入程序使用的API所加载模块的基址不一样时，就可以通过遍历目标进程模块找到使用API的模块地址 和 自己程序中API的地址减去模块地址得出的相对偏移 进行相加就可以得到目标进程中使用API在对应模块中的地址，再将地址写入到注入程序中即可。

遍历模块有两种方法：

- 通过快照，API: **CreateToolhelp32Snapshot** (获取进程以及进程所使用的堆、模块和线程的快照)、
**Module32First (不能遍历64位程序) **等
- 枚举进程模块，可以遍历32位程序，也可以遍历64位程序，API: EnumProcessModules, EnumProcessModulesEx, 对应的头文件为 "psapi.h"

在汇编编程中使用API时，ecx. edx 寄存器尽量不要当作局部变量使用，API内存中不会保存寄存器环境。

代码自重定位，位置无关代码

call 会将下一行执行代码的地址压入栈中

远程线程注入操作系统会默认传递四个字节的参数，代码注入寻址防止尽量使用寄存器（少使用立即数），原因：生成的汇编代码字节数更少

例如：

```
push MB_OK  
push NULL
```

可改为：

```
xor ebx, ebx  
push ebx
```

不是同一台计算机如何进行代码注入？

遍历模块基址方法不可靠（对方的操作系统版本也不确定）
远程目标计算机程序所使用的API地址不确定

代码运行起来对内存地址有要求就有重定位问题，反之没有重定位问题。

为目标进程加载模块