HOOK API

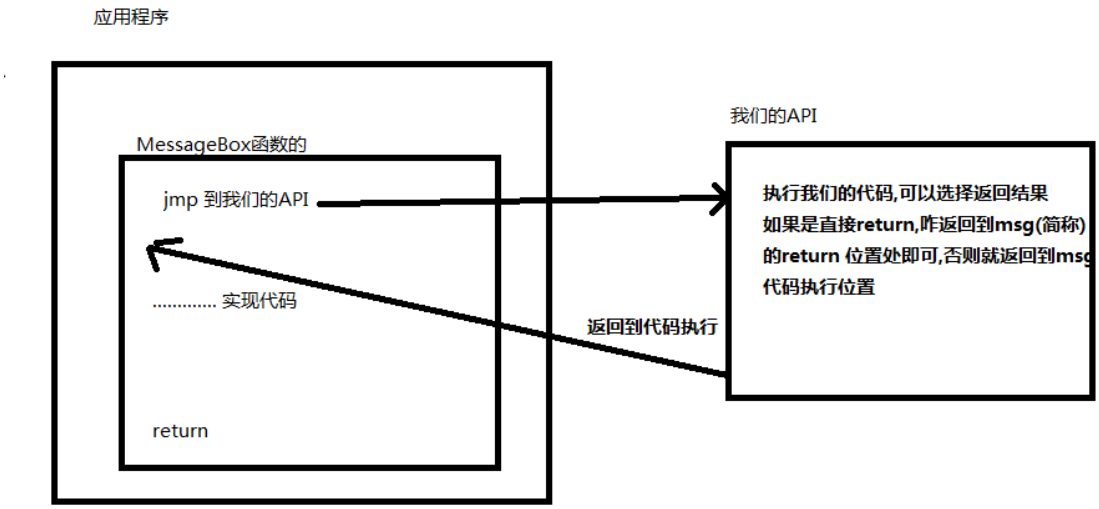
原理

在windows的大部分API中，开头的5个字节的指令是

```
8B FF    mov edi, edi
55       push ebp
8BEC     mov ebp, esp
```

而`mov edi, edi`没有什么功能用途上的意义，恰好微软给我们构造了一个绝佳的开头用来将其覆盖为同样5个字节的`jmp xxx`指令，就有了如下图的hook方法

假定我们需要hook的API是`MessageBox`



其实相当于就是我们在这个API之前，跳转到我们的函数执行了，然后跳转之后，我们执行完毕之后，可以在跳转回去

jmp指令的偏移计算

`jmp xxx`的偏移计算公式遵循`目的地址 - 源地址 = 偏移`的原则

- 目的地址
  - 要跳往的地址处
- 源地址
  - 当前指令的下一条指令的地址处

例如：

```
; 跳到我们的代码
mov eax, @buf_ptr   ; @buf_ptr是在目标进程中申请的空间
add eax, offset NEW_API - offset INJECT_BEGIN   ; 定位到注入代码的功能代码开头作为目的地址
sub eax, @target_api    ; @target_api为hook的API的地址 -- 源地址
sub eax, 5      ; 覆盖了5个字节，公式为 eax - (@target_api + 5)


; 跳转回去
mov eax, @buf_ptr   ; @buf_ptr是在目标进程中申请的空间
add eax, offset INJECT_END - offset INJECT_BEGIN    ; 定位到注入代码的尾部
mov ebx, @target_api    ; @target_api为hook的API的地址
add ebx, 5  ; 因为覆盖了5个字节，所以回去的目标地址为 @target_api + 5
sub ebx, eax    ; 公式为 (@target_api + 5) - eax
```

我们注入的代码

在我们注入的代码中，需要还原被我们覆盖为`jmp`指令的原先的指令组，即

```
mov edi, edi
push ebp
mov ebp, esp
```

而`mov edi, edi`并没有什么实际的功能用途，故只还原后两条指令即可，大致为如下：

```
INJECT_BEGIN:
    msgbox dd 0


NEW_API:
    ; ----------
    ; 注入的代码开始
    push ebx
    call NEXT
NEXT:
    pop ebx
    sub ebx, offset NEXT    ; ebx保存重定位的偏移量

    push MB_OK
    push NULL
    push [esp + 10h]    ; 从我们hook的API中获取参数
    push NULL

    call [ebx +offset msgbox]   ; MessageBox

    pop ebx

    ; 还原我们hook的API的指令组
    push ebp
    mov ebp, esp
API_END_JMP:
    db 0e9h, 00, 00, 00, 00 ; 跳转硬编码
```

```
INJECT_END:
    nop
```