Ring 3 的 IAT HOOK 和 EAT HOOK 是一种改函数地址的 HOOK 法，类似于 SSDT HOOK。但实际应用较少，目前应用此技术而又使用得比较广泛的软件，似乎只有 CHROME 和 IE。废话不多说，直接进入正题，说说这两种 HOOK 的实现。

一、EAT HOOK

根据模块名称和函数名称，找到此函数在模块导出表中的位置。然后修改导出表中记录的数据即可。此数据的计算公式是：代理函数地址-模块基址。

```
VOID EAT_HOOK_TEST64(char *ModName, char *FunName, ULONG64 ProxyFunAddr)
{
    HANDLE hMod;
    PVOID BaseAddress = NULL;
    IMAGE_DOS_HEADER * dosheader;
    IMAGE_OPTIONAL_HEADER64 * opthdr;
    PIMAGE_EXPORT_DIRECTORY exports;
    USHORT index=0 ;
    ULONG addr, i;
    PUCHAR pFuncName = NULL;
    PULONG pAddressOfFunctions;
    PULONG pAddressOfNames;
    PUSHORT pAddressOfNameOrdinals;
    BaseAddress= GetModuleHandleA(ModName);
    MODULEINFO mi={0};
    //获取模块信息
    GetModuleInformation(GetCurrentProcess(),(HMODULE)BaseAddress,&mi,sizeof(MODULEINFO));
    DWORD ass;
    //修改页属性
    VirtualProtect(BaseAddress,mi.SizeOfImage,PAGE_EXECUTE_READWRITE,&ass);
    hMod = BaseAddress;
    dosheader = (IMAGE_DOS_HEADER *)hMod;
    opthdr =(IMAGE_OPTIONAL_HEADER64 *) ((BYTE*)hMod+dosheader->e_lfanew+24);
    //查找导出表
    exports = (PIMAGE_EXPORT_DIRECTORY)((BYTE*)dosheader+ opthdr->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    pAddressOfFunctions=(ULONG*)((BYTE*)hMod+exports->AddressOfFunctions);
    pAddressOfNames=(ULONG*)((BYTE*)hMod+exports->AddressOfNames);
    pAddressOfNameOrdinals=(USHORT*)((BYTE*)hMod+exports->AddressOfNameOrdinals);
    //对比函数名
    for (i = 0; i < exports->NumberOfNames; i++)
    {
        index=pAddressOfNameOrdinals[i];
```

```
                        addr=pAddressOfFunctions[index];
                        pFuncName = (PUCHAR)( (BYTE*)hMod + pAddressOfNames[i]);
                        addr = pAddressOfFunctions[index];
                        if(!strcmp((const char*)pFuncName,FunName))
                        {
                                //最关键一步：修改地址
                                pAddressOfFunctions[index]=(ULONG)((ULONG64)ProxyFunAddr-
(ULONG64)hMod);
                                printf("eat fix!!!\n");;
                        }
                }
}
```

二、IAT HOOK

根据模块名称和函数名称，找到此函数在模块导入表中的位置。然后修改导入表中记录的数据即可。此数据直接就是代理函数的地址，不需要做任何计算。

```
        BOOL IAT_HOOK_TEST64(char *DllName, HMODULE hMod, ULONG64 g_orgProc,
ULONG64 g_newProc)
        {
                IMAGE_DOS_HEADER* pDosHeader = (IMAGE_DOS_HEADER*)hMod;
                IMAGE_OPTIONAL_HEADER64* pOptHeader = (IMAGE_OPTIONAL_HEADER64
*)((BYTE*)hMod + pDosHeader->e_lfanew + 24);    //24=4+sizeof(IMAGE_FILE_HEADER)
                IMAGE_IMPORT_DESCRIPTOR* pImportDesc =
(IMAGE_IMPORT_DESCRIPTOR*)((BYTE*)hMod +
pOptHeader->DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress);
                // 在导入表中查找 user32.dll 模块。因为 MessageBoxA 函数从 user32.dll 模
块导出
                while(pImportDesc->FirstThunk)
                {
                        char* pszDllName = (char*)((BYTE*)hMod + pImportDesc->Name);
                        if(lstrcmpiA(pszDllName, DllName) == 0)
                        {
                                break;
                        }
                        pImportDesc++;
                }
                if(pImportDesc->FirstThunk)
                {
                        // 一个 IMAGE_THUNK_DATA 就是一个双字，它指定了一个导入函数
                        // 调入地址表其实是 IMAGE_THUNK_DATA 结构的数组，也就是 DWORD
数组
                        IMAGE_THUNK_DATA* pThunk = (IMAGE_THUNK_DATA*)((BYTE*)hMod +
pImportDesc->FirstThunk);
                        while(pThunk->u1.Function)
```

```
            {
                // lpAddr 指向的内存保存了函数的地址
                ULONG64* lpAddr = (ULONG64*)&(pThunk->u1.Function);
                if(*lpAddr == g_orgProc)
                {
                    DWORD dwOldProtect;
                    VirtualProtect(lpAddr,                    sizeof(ULONG64),
PAGE_EXECUTE_READWRITE, &dwOldProtect);
                    *lpAddr=(ULONG64)g_newProc;
                    printf("iat fix!!!\n");
                    return TRUE;
                }
                pThunk++;
            }
        }
        return FALSE;
}
```

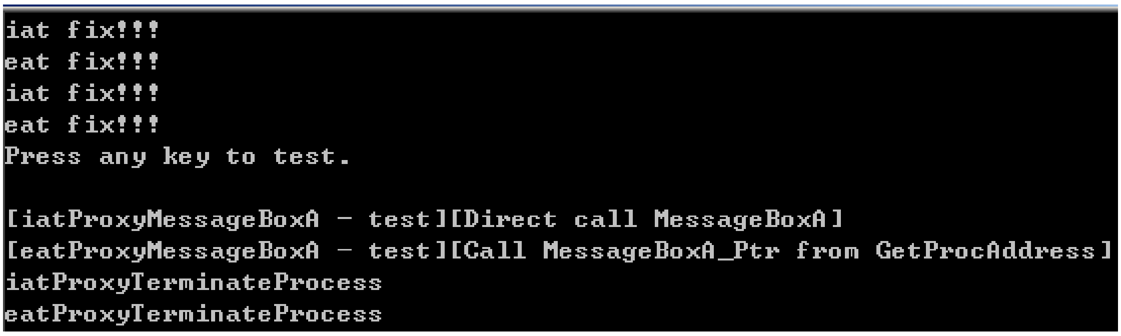
分别针对 MessageBoxA 和 TerminateProcess 函数，修改本进程的导入表和对应 DLL（USER32.DLL 和 KERNEL32.DLL）的导出表：

```
Void test()
{
        OriMsgBoxA=(ULONG64)MessageBoxA;
        IAT_HOOK_TEST64("user32.dll",GetModuleHandleA(0),(ULONG64)MessageBox
A,(ULONG64)iatProxyMessageBoxA);
        EAT_HOOK_TEST64("user32.dll","MessageBoxA",(ULONG64)eatProxyMessageB
oxA);
        OriTerminateProcess=(ULONG64)TerminateProcess;
        IAT_HOOK_TEST64("kernel32.dll",GetModuleHandleA(0),(ULONG64)TerminateP
rocess,(ULONG64)iatProxyTerminateProcess);
        EAT_HOOK_TEST64("kernel32.dll","TerminateProcess",(ULONG64)eatProxyTerm
inateProcess);
        printf("Press any key to test.\n");getchar();

        //test MessageBoxA
        MessageBoxA(0,"Direct call MessageBoxA","test",0);
        MSGBOXA
msgboxA=(MSGBOXA)GetProcAddress(LoadLibraryA("user32.dll"),"MessageBoxA");
        msgboxA(0,"Call MessageBoxA_Ptr from GetProcAddress","test",0);

        //test TerminateProcess
        TerminateProcess((HANDLE)1234,0);
        TERMINATEPROCESS
tp=(TERMINATEPROCESS)GetProcAddress(GetModuleHandleA("kernel32.dll"),"Terminate
Process");
	tp((HANDLE)1234,0);
}
```

测试的效果如下（无论是直接调用 API 还是通过函数指针调用 API，都被拦截）：



```
iat fix!!!
eat fix!!!
iat fix!!!
eat fix!!!
Press any key to test.

[iatProxyMessageBoxA - test][Direct call MessageBoxA]
[eatProxyMessageBoxA - test][Call MessageBoxA_Ptr from GetProcAddress]
iatProxyTerminateProcess
eatProxyTerminateProcess
```

本文代码稍加改造，即可实现内核级别的 EAT HOOK 和 IAT HOOK。在 WIN64 系统里，对 NTOSKNRL.EXE 进行 EAT HOOK 会触发 PatchGuard 导致 BSOD，但是对第三方驱动进行 IAT HOOK，PatchGuard 是不会管的。如果在 IMAGE NOTIFY 里，对加载的驱动进行 IAT HOOK，即可实现不触发 PatchGuard 的内核 HOOK。不过内核 IAT HOOK 的局限性也是很大的，因为如果通过函数指针来调用内核 API，内核 IAT HOOK 就无法拦截了。