关于绿盟科技

北京神州绿盟信息安全科技股份有限公司（简称绿盟科技）成立于 2000 年 4 月，总部位于北京。在国内外设有 30 多个分支机构，为政府、运营商、金融、能源、互联网以及教育、医疗等行业用户，提供具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。

基于多年的安全攻防研究，绿盟科技在网络及终端安全、互联网基础安全、合规及安全管理等领域，为客户提供入侵检测 / 防护、抗拒绝服务攻击、远程安全评估以及 Web 安全防护等产品以及专业安全服务。

北京神州绿盟信息安全科技股份有限公司于 2014 年 1 月 29 日起在深圳证券交易所创业板上市交易。

股票简称：绿盟科技　股票代码：300369

目录

目录

目录

目录

目录

文件类

kernel32!CreateFile

功能：这是一个多功能的函数，可打开或创建以下对象，并返回可访问的句柄：控制台，通信资源，目录（只读打开），磁盘驱动器，文件，邮槽，管道

函数原型：

```
HANDLE WINAPI CreateFile(
    _In_ LPCTSTR lpFileName,
    _In_ DWORD dwDesiredAccess,
    _In_ DWORD dwShareMode,
    _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    _In_ DWORD dwCreationDisposition,
    _In_ DWORD dwFlagsAndAttributes,
    _In_opt_ HANDLE hTemplateFile
);
```

参数介绍：

lpFileName 要打开的文件的名或设备名。这个字符串的最大长度在 ANSI 版本中为 MAX_PATH，在 unicode 版本中为 32767。

dwDesiredAccess 指定类型的访问对象。如果为 GENERIC_READ 表示允许对设备进行读访问；如果为 GENERIC_WRITE 表示允许对设备进行写访问（可组合使用）；如果为零，表示只允许获取与一个设备有关的信息 。

dwShareMode， 如果是零表示不共享； 如果是 FILE_SHARE_DELETE 表示随后打开操作对象会成功只要删除访问请求；如果是 FILE_SHARE_READ 随后打开操作对象会成功只有请求读访问；如果是 FILE_SHARE_WRITE 随后打开操作对象会成功只有请求写访问。

lpSecurityAttributes， 指向一个 SECURITY_ATTRIBUTES 结构的指针，定义了文件的安全特性（如果操作系统支持的话）

dwCreationDisposition，创建配置

dwFlagsAndAttributes，扩展属性

hTemplateFile，hTemplateFile 为一个文件或设备句柄，表示按这个参数给出的句柄为模板创建文件（就是将该句柄文件拷贝到 lpFileName 指定的路径，然后再打开）。它将指定该文件的属性扩展到新创建的文件上面，这个参数可用于将某个新文件的属性设置成与现有文件一样，并且这样会忽略 dwAttrsAndFlags。通常这个参数设置为 NULL，为空表示不使用模板，一般为空。

备注：

kernel32!CreateFileMapping

功能：创建一个新的文件映射内核对象。

函数原型：

```
HANDLE WINAPI CreateFileMapping(
    _In_HANDLE hFile,
    _In_opt_LPSECURITY_ATTRIBUTES lpAttributes,
    _In_DWORD flProtect,
    _In_DWORD dwMaximumSizeHigh,
    _In_DWORD dwMaximumSizeLow,
    _In_opt_LPCTSTR lpName
);
```

参数介绍：

hFile:Long，指定欲在其中创建映射的一个文件句柄。0xFFFFFFFF（-1，即 INVALID_HANDLE_VALUE）表示在页面文件中创建一个可共享的文件映射。

lpFileMappigAttributes:SECURITY_ATTRIBUTES，它指明返回的句柄是否可以被子进程所继承，指定一个安全对象，在创建文件映射时使用。如果为 NULL（用 ByVal As Long 传递零），表示使用默认安全对象。

结构体类型如下：保存对象的安全属性

```
typedef struct _SECURITY_ATTRIBUTES {
    DWORD  nLength;
    LPVOID lpSecurityDescriptor;
    BOOL   bInheritHandle;
} SECURITY_ATTRIBUTES, *PSECURITY_ATTRIBUTES, *LPSECURITY_ATTRIBUTES;
```

flProtect：保护属性设置

dwMaximumSizeHigh:Long，文件映射的最大长度的高 32 位。

dwMaximumSizeLow:Long，文件映射的最大长度的低 32 位。如这个参数和 dwMaximumSizeHigh 都是零，就用磁盘文件的实际长度。

lpName:String，指定文件映射对象的名字。如存在这个名字的一个映射，函数就会打开它。用 vbNullString 可以创建一个无名的文件映射。

返回值：

Long，新建文件映射对象的句柄；零意味着出错。会设置 GetLastError。即使函数成功，但倘若返回的句柄属于一个现成的文件映射对象，那么 GetLastError 也会设置成 ERROR_ALREADY_EXISTS。在这种情况下，文件映射的长度就是现有对象的长度，而不是这个函数指定的尺寸。

备注：

创建一个映射到文件的句柄，将文件装载到内存，并使得他可以通过内存地址进行访问。启动器，装载器和注入器会使用这个函数来读取和修改 PE 文件。

kernel32.OpenFile

功能：用于打开文件的

函数原型：

```
HFILE WINAPI OpenFile(
    _In_  LPCSTR     lpFileName,
    _Out_ LPOFSTRUCT lpReOpenBuff,
    _In_  UINT       uStyle
);
```

参数介绍：（参数与返回值）

lpFileName：文件名

lpReOpenBuff：变量指针，用于存储文件被首次打开时接收信息

uStyle：打开文件的常量类型

成功返回打开的文件句柄，失败返回 HFILE_ERROR

kernel32!FindFirstFile

功能：根据文件名查找文件。该函数到一个文件夹 (包括子文件夹) 去搜索指定文件

函数原型：

```
HANDLE FindFirstFile(
    LPCTSTR lpFileName,//filename
    LPWIN32_FIND_DATA lpFindFileData//databuffer
);
```

参数介绍：

LPCTSTR lpFileName 文件名（包括路径）

LPWIN32_FIND_DATA lpFindFileData 指向一个用于保存文件信息的结构体

结构体声明如下：

```
typedef struct _WIN32_FIND_DATA {
    DWORD dwFileAttributes;                 // 文件属性
    FILETIME ftCreationTime;                // 文件创建时间
    FILETIME ftLastAccessTime;              // 文件最后一次访问时间
    FILETIME ftLastWriteTime;               // 文件最后一次修改时间
    DWORD nFileSizeHigh;                    // 文件长度高 32 位
    DWORD nFileSizeLow;                     // 文件长度低 32 位
    DWORD dwReserved0;                      // 系统保留
    DWORD dwReserved1;                      // 系统保留
    TCHAR cFileName[ MAX_PATH ];            // 长文件名
    TCHAR cAlternateFileName[ 14 ];         // 8.3 格式文件名
```

```
} WIN32_FIND_DATA, *PWIN32_FIND_DATA;
```

返回值：

如果调用成功返回一个句柄，可用来做为 FindNextFile 或 FindClose 参数

调用失败 返回为 INVALID_HANDLE_VALUE(即 -1) , 可调用 GetLastError 来获取错误信息

备注：用来搜索文件目录和枚举文件系统的函数

kernel32!FindNextFile

功能：可以用来遍历目录或文件时，判断当前目录下是否有下一个目录或文件。

函数原型：

```
BOOLFindNextFile(
    HANDLE hFindFile, //searchhandle
    LPWIN32_FIND_DATA lpFindFileData //databuffer
);
```

参数介绍：

HANDLE hFindFile 搜索的文件句柄 函数执行的时候搜索的是此句柄的下一文件

LPWIN32_FIND_DATA lpFindFileData 指向一个用于保存文件信息的结构体

返回值：

非零表示成功，零表示失败。如不再有与指定条件相符的文件，会将 GetLastError 设置成 ERROR_NO_MORE_FILES

备注：用来搜索文件目录和枚举文件系统的函数

kernel32!GetModuleFileName

功能：获取当前进程已加载模块的文件的完整路径，该模块必须由当前进程加载。

函数原型：

```
DWORD WINAPI GetModuleFileName(
    _In_opt_  HMODULE hModule,
    _Out_     LPTSTR lpFilename,
    _In_      DWORD nSize
);
```

参数介绍：

hModule 一个模块的句柄。可以是一个 DLL 模块，或者是一个应用程序的实例句柄。如果该参数为 NULL，该函数返回该应用程序全路径。

lpFileName 指定一个字串缓冲区，要在其中容纳文件的用 NULL 字符中止的路径名，hModule 模块就是从这个文件装载进来的

nSize 装载到缓冲区 lpFileName 的最大字符数量

返回值：Long，如执行成功，返回复制到 lpFileName 的实际字符数量；零表示失败

备注：返回目前进程装载某个模块的文件名，恶意代码可以使用这个函数，在目前运行进程中修改或复制文件

kernel32!GetModuleHandle

功能：获取一个应用程序或动态链接库的模块句柄。

函数原型：HMODULEGetModuleHandle(LPCTSTRlpModuleName);

参数介绍：lpModuleName 模块名称

返回值：如执行成功成功，则返回模块句柄。零表示失败

备注：用来获取已装载模块句柄的函数，恶意代码可以使用此函数在一个装载模块中定位和修改代码，或者搜索一个合适位置来注入代码。

kernel32!GetProcAddress

功能：检索指定的动态链接库 (DLL) 中的输出库函数地址。

函数原型：

```
FARPROC GetProcAddress(
    HMODULE hModule,
     LPCSTR lpProcName
);
```

参数介绍：

hModule

[in] 包含此函数的 DLL 模块的句柄。LoadLibrary、AfxLoadLibrary 或者 GetModuleHandle 函数可以返回此句柄。

lpProcName

[in] 包含函数名的以 NULL 结尾的字符串，或者指定函数的序数值。如果此参数是一个序数值，它必须在一个字的底字节，高字节必须为 0。

返回值：

如果函数调用成功，返回值是 DLL 中的输出函数地址。

如果函数调用失败，返回值是 NULL。

备注：获取装载到内存中一个 DLL 程序的函数地址。用来从其他 DLL 程序中导入函数，以补充在 PE 文件头部中导入的函数。

kernel32!GetStartupInfo

功能：取得进程在启动时被指定的 STARTUPINFO 结构。

函数原型：

```
VOID GetStartupInfo(
    LPSTARTUPINFO lpStartupInfo
);
```

参数介绍：

lpStartupInfo 一个指向用来存放要获取的 STARTUPINFO 结构的指针。

结构体声明如下：

```
typedef struct _STARTUPINFO {
    DWORD cb; // 包含 STARTUPINFO 结构中的字节数
    LPTSTR lpReserved; // 保留。必须初始化为 N U L L
    LPTSTR lpDesktop; // 用于标识启动应用程序所在的桌面的名字。如果该桌面存在，新进程便与指定的桌面相关联。如果桌面不存在，便创建一个带有默认属性的桌面，并使用为新进程指定的名字。如果 lpDesktop 是 NULL（这是最常见的情况 ), 那么该进程将与当前桌面相关联
    LPTSTR lpTitle; // 用于设定控制台窗口的名称。如果 l p Ti t l e 是 N U L L，则可执行文件的名字将用作窗口名
    DWORD dwX; // 用于设定应用程序窗口在屏幕上应该放置的位置的 x 和 y 坐标（以像素为单位）。
    DWORD dwY;// 只有当子进程用 CW_USEDEFAULT 作为 CreateWindow 的 x 参数来创建它的第一个重叠窗口时 , 才使用这两个坐标。若是创建控制台窗口的应用程序，这些成员用于指明控制台窗口的左上角
    DWORD dwXSize; // 用于设定应用程序窗口的宽度和长度（以像素为单位）只有 wYsize
    DWORD dwYSize;// 当子进程将 CW_USEDEFAULT 用作 CreateWindow 的 nWidth 参数来创建它的第一个重叠窗口时，才使用这些值。
    DWORD dwXCountChars; // 用于设定子应用程序的控制台窗口的宽度和高度（以字符为单位）
    DWORD dwYCountChars;
    DWORD dwFillAttribute; // 用于设定子应用程序的控制台窗口使用的文本和背景颜色
    DWORD dwFlags;
    WORD wShowWindow;
    WORD cbReserved2; // 保留。必须被初始化为 0
    LPBYTE lpReserved2; // 保留。必须被初始化为 NULL
    HANDLE hStdInput; // 用于设定供控制台输入和输出用的缓存的句柄。
    HANDLE hStdOutput;// 用于标识控制台窗口的缓存
    HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

返回值：无

备注：获取一个包含当前进程如何自动运行配置信息的结构，比如标准句柄指向哪些位置。

kernel32!GetTempPath

功能：获取为临时文件指定的路径

函数原型：

```
DWORD WINAPI GetTempPath(
    _In_  DWORD  nBufferLength,
    _Out_ LPTSTR lpBuffer
);
```

参数介绍：

nBufferLength：表示 lpBuffer 的大小

lpBuffer：接收路径的一块内存

返回值：如果成功，返回 lpBuffer 的长度，失败返回 0

备注：返回临时文件路径，如果看到恶意代码使用了这个函数，需要检查他是否在临时文件路径中读取或写入了一些文件。

kernel32!GetWindowsDirectory

功能：获取 Windows 目录的完整路径名。

函数原型：UINTGetWindowsDirectory(LPTSTR lpBuffer,UINT uSize)

参数介绍:

lpBuffer，指定一个字串缓冲区，用于装载 Windows 目录名。除非是根目录，否则目录中不会有一个中止用的“\”字符

nSize，lpBuffer 字串的最大长度

返回值：复制到 lpBuffer 的一个字串的长度。如 lpBuffer 不够大，不能容下整个字串，就会返回 lpBuffer 要求的长度，零表示失败

备注：返回 Windows 目录的文件系统路径，恶意代码经常使用这个函数来确定将其他恶意程序安装到哪个目录。

kernel32!MapViewOfFile

功能：将一个文件映射对象映射到当前应用程序的地址空间

函数原型：

```
LPVOID WINAPI MapViewOfFile(
    __in HANDLE hFileMappingObject,
    __in DWORD dwDesiredAccess,
    __in DWORD dwFileOffsetHigh,
```

```
    __in DWORD dwFileOffsetLow,
    __in SIZE_T dwNumberOfBytesToMap
);
```

参数介绍：

hFileMappingObject：hFileMappingObject 为 CreateFileMapping() 返回的文件映像对象句柄。

dwDesiredAccess：dwDesiredAccess 映射对象的文件数据的访问方式，而且同样要与 CreateFileMapping() 函数所设置的保护属性相匹配。

dwFileOffsetHigh：dwFileOffsetHigh 表示文件映射起始偏移的高 32 位 .

dwFileOffsetLow：dwFileOffsetLow 表示文件映射起始偏移的低 32 位 .(64KB 对齐不是必须的)

dwNumberOfBytesToMap：dwNumberOfBytes 指定映射文件的字节数 .

返回值：

如果成功 , 则返回映射视图文件的开始地址值 . 如果失败，则返回 NULL

备注：映射一个文件到内存，将文件内容变得通过内存地址可访问。启动器，装载器和注入器使用这个函数来读取和修改 PE 文件。通过使用此函数，恶意代码可以避免使用 WriteFile 来修改文件内容。

NtosKrnl!NtQueryDirectoryFile

功能：返回多种指定的文件信息

函数原型：

```
NTSTATUS
 ZwQueryDirectoryFile(
    __in HANDLE  FileHandle,
    __in_opt HANDLE  Event,
    __in_opt PIO_APC_ROUTINE  ApcRoutine,
    __in_opt PVOID  ApcContext,
    __out PIO_STATUS_BLOCK  IoStatusBlock,
    __out PVOID  FileInformation,
    __in ULONG  Length,
    __in FILE_INFORMATION_CLASS  FileInformationClass,
    __in BOOLEAN  ReturnSingleEntry,
    __in_opt PUNICODE_STRING  FileName,
    __in BOOLEAN  RestartScan
);
```

参数介绍：

FileHandle：ZwCreateFile 和 ZwOpenFile 返回的句柄，代表着被查询信息的文件夹。

Event：调用者创建的一个可选的事件句柄。

ApcRoutine：调用者提供的一个 APC 例程，当操作完成时，调用此例程。

ApcContext：如果调用者提供了 APC 例程，则此参数位 APC 例程的上下文。

IoStatusBlock：指向 IO_STATUS_BLOCK 结构体，返回操作的完成状态和信息。

结构体声明如下：

```
typedef struct _IO_STATUS_BLOCK {
union { NTSTATUS Status; PVOID Pointer; };
ULONG_PTR Information;
} IO_STATUS_BLOCK, *PIO_STATUS_BLOCK;
```

Status 是请求被处理的状态，如果被成功的处理，则为 STATUS_SUCCESS ，其他情况为 STATUS_XXX

Information 如果处理成功，Information 表示处理的字节数，如果处理失败，此值为 0。

FileInformation：接收文件的特定的信息

结构体声明如下：

```
typedef enum _FILE_INFORMATION_CLASS
{
    FileDirectoryInformation = 1,
    FileFullDirectoryInformation = 2,
    FileBothDirectoryInformation = 3,
    FileBasicInformation = 4,
    FileStandardInformation = 5,
    FileInternalInformation = 6,
    FileEaInformation = 7,
    FileAccessInformation = 8,
    FileNameInformation = 9,
    FileRenameInformation = 10,
    FileLinkInformation = 11,
    FileNamesInformation = 12,
    FileDispositionInformation = 13,
    FilePositionInformation = 14,
    FileFullEaInformation = 15,
    FileModeInformation = 16,
    FileAlignmentInformation = 17,
    FileAllInformation = 18,
    FileAllocationInformation = 19,
    FileEndOfFileInformation = 20,
    FileAlternateNameInformation = 21,
    FileStreamInformation = 22,
    FilePipeInformation = 23,
    FilePipeLocalInformation = 24,
    FilePipeRemoteInformation = 25,
    FileMailslotQueryInformation = 26,
    FileMailslotSetInformation = 27,
```

```
    FileCompressionInformation = 28,
    FileObjectIdInformation = 29,
    FileCompletionInformation = 30,
    FileMoveClusterInformation = 31,
    FileQuotaInformation = 32,
    FileReparsePointInformation = 33,
    FileNetworkOpenInformation = 34,
    FileAttributeTagInformation = 35,
    FileTrackingInformation = 36,
    FileIdBothDirectoryInformation = 37,
    FileIdFullDirectoryInformation = 38,
    FileValidDataLengthInformation = 39,
    FileShortNameInformation = 40,
    FileIoCompletionNotificationInformation = 41,
    FileIoStatusBlockRangeInformation = 42,
    FileIoPriorityHintInformation = 43,
    FileSfioReserveInformation = 44,
    FileSfioVolumeInformation = 45,
    FileHardLinkInformation = 46,
    FileProcessIdsUsingFileInformation = 47,
    FileNormalizedNameInformation = 48,
    FileNetworkPhysicalNameInformation = 49,
    FileMaximumInformation = 50
} FILE_INFORMATION_CLASS;
```

Length：FileInformation 信息的长度。

FileInformationClass：需要返回的文件夹信息的类型。

ReturnSingleEntry：如果要返回单一入口，就将此值设为 TRUE，否则设为 FALSE。

FileName：调用者申请的文件名

RestartScan：如果第一次访问文件夹，将此参数设为 TRUE，否则设为 FALSE。

返回值：成功返回 STATUS_SUCCESS，失败返回错误码

备注：返回一个目录中文件的信息，内核套件普遍会挂钩这个函数来隐藏文件。

kernel32!SetFileTime

功能：设置文件的创建、访问及上次修改时间 函数原型：

```
BOOL WINAPI SetFileTime(
    _In_        HANDLE   hFile,
    _In_opt_ const FILETIME *lpCreationTime,
    _In_opt_ const FILETIME *lpLastAccessTime,
    _In_opt_ const FILETIME *lpLastWriteTime
```

```
);
```

参数介绍：

hFile Long，系统文件句柄

lpCreationTime FILETIME，文件的创建时间

lpLastAccessTime FILETIME，文件上一次访问的时间

lpLastWriteTime FILETIME，文件最近一次修改的时间

返回值：非零表示成功，零表示失败

备注：修改一个文件的创建，访问或者最后修改时间，恶意代码经常使用这个函数来隐藏恶意行为。

kernel32!Wow64DisableWow64FsRedirection

功能：禁用文件系统重定向机制

函数原型：

```
BOOL WINAPI Wow64DisableWow64FsRedirection(
    _Out_ PVOID *OldValue
);
```

参数介绍：OldValue：Wow64 文件系统重定向值

返回值：成功返回非 0，失败返回 0.

备注：禁用 32 为文件在 64 位操作系统中装载后发生的文件重定向机制，如果一个 32 位应用程序在调用这个函数后向 C:\Windows\System32 写数据，那么它将会直接写到真正的 C:\Windows\System32，而不是被重定向至 C:\Windows\SysWOW64

网络类

ws2_32!socket

功能：用于根据指定的地址族、数据类型和协议来分配一个套接口的描述字及其所用的资源。如果协议 protocol 未指定（等于 0），则使用缺省的连接方式

函数原型：int socket(int af, int type, int protocol);

参数介绍：（参数与返回值）

af：一个地址描述。目前仅支持 AF_INET 格式，也就是说 ARPA Internet 地址格式。

type：指定 socket 类型。新套接口的类型描述类型，如 TCP（SOCK_STREAM）和 UDP（SOCK_DGRAM）。常用的 socket 类型有，SOCK_STREAM、SOCK_DGRAM、SOCK_RAW、SOCK_PACKET、SOCK_SEQPACKET 等等。

protocol：顾名思义，就是指定协议。套接口所用的协议。如调用者不想指定，可用 0。常用的协议有，IPPROTO_TCP、IPPROTO_UDP、IPPROTO_STCP、IPPROTO_TIPC 等，它们分别对应 TCP 传输协议、UDP 传输协议、STCP 传输协议、TIPC 传输协议。

若无错误发生，socket() 返回引用新套接口的描述字。否则的话，返回 INVALID_SOCKET 错误，应用程序可通过 WSAGetLastError() 获取相应错误代码。

ws2_32!accept

功能：用来监听入站网络连接，此函数预示着程序会在一个套接字上监听入站网络连接。

函数原型：

int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

参数介绍：

sockfd：套接字描述符，该套接口在 listen() 后监听连接。

addr：（可选）指针，指向一缓冲区，其中接收为通讯层所知的连接实体的地址。Addr 参数的实际格式由套接口创建时所产生的地址族确定。

结构体声明如下：

```
struct sockaddr {
    unsigned short sa_family; /* 地址家族 , AF_xxx */
    char sa_data[14]; /* 14 字节的协议地址 */
};
```

sa_family：是 2 字节的地址家族，一般都是“AF_xxx”的形式，它的值包括三种：AF_INET，AF_INET6 和 AF_UNSPEC。

addrlen：（可选）指针，输入参数，配合 addr 一起使用，指向存有 addr 地址长度的整型数。

返回值：如果没有错误产生，则 accept() 返回一个描述所接受包的 SOCKET 类型的值。否则的话，返回

INVALID_SOCKET 错误，应用程序可通过调用 WSAGetLastError() 来获得特定的错误代码。

addrlen 所指的整形数初始时包含 addr 所指地址空间的大小，在返回时它包含实际返回地址的字节长度。

ws2_32!bind

功能：将一本地地址与一套接口捆绑

函数原型：

int bind(int sockfd,const struct sockaddr *addr,socklen_t *addrlen);

参数介绍：

sockfd: 已经建立的 socket 编号 (描述符)

addr: 是一个指向 sockaddr 结构体类型的指针；

addrlen: 表示 addr 结构的长度

返回值：成功返回 0，失败返回 -1

备注：

ws2_32!connect

功能：用来将参数 sockfd 的 socket 连至参数 serv_addr 指定的网络地址

函数原型：

int connect (int sockfd, struct sockaddr * serv_addr, int addrlen);

参数介绍：

Sockfd：套接字描述符

serv_addr：指向数据结构 sockaddr 的指针，其中包括目的端口和 IP 地址

addrlen：参数二 sockaddr 的长度，可以通过 sizeof（struct sockaddr）获得

返回值：成功则返回 0，失败返回非 0，错误码 GetLastError()。

备注：用来连接一个远程套接字。恶意代码经常使用底层功能函数来连接一个命令控制服务器。

wininet.InternetOpenA

功能：初始化一个应用程序，以使用 WinINet 函数

函数原型：

```
HINTERNET InternetOpen(
    _In_ LPCTSTR lpszAgent,
    _In_ DWORD dwAccessType,
```

```
    _In_ LPCTSTR lpszProxyName,
    _In_ LPCTSTR lpszProxyBypass,
    _In_ DWORD dwFlags
);
```

参数介绍：

lpszAgent：指向一个空结束的字符串，该字符串指定调用 WinInet 函数的应用程序或实体的名称。使用此名称作为用户代理的 HTTP 协议。

dwAccessType：指定访问类型

lpszProxyName：指针指向一个空结束的字符串，该字符串指定的代理服务器的名称，不要使用空字符串；如果 dwAccessType 未设置为 INTERNET_OPEN_TYPE_PROXY，则此参数应该设置为 NULL。

lpszProxyBypass：指向一个空结束的字符串，该字符串指定的可选列表的主机名或 IP 地址。如果 dwAccessType 未设置为 INTERNET_OPEN_TYPE_PROXY 的，参数省略则为 NULL。

dwFlags：参数可以是下列值的组合：

INTERNET_FLAG_ASYNC：使异步请求处理的后裔从这个函数返回的句柄

INTERNET_FLAG_FROM_CACHE：不进行网络请求，从缓存返回的所有实体，如果请求的项目不在缓存中，则返回一个合适的错误，如 ERROR_FILE_NOT_FOUND。

INTERNET_FLAG_OFFLINE：不进行网络请求，从缓存返回的所有实体，如果请求的项目不在缓存中，则返回一个合适的错误，如 ERROR_FILE_NOT_FOUND。

返回值：

成功：返回一个有效的句柄，该句柄将由应用程序传递给接下来的 WinINet 函数。

失败：返回 NULL。

备注：该函数是第一个由应用程序调用的 WinINet 函数。它告诉 Internet DLL 初始化内部数据结构并准备接收应用程序之后的其他调用。当应用程序结束使用 Internet 函数时，应调用 InternetCloseHandle 函数来释放与之相关的资源。

应用程序可以对该函数进行任意次数的调用，不过在一般情况下一次调用就已经足够了。如果要调用多次该函数，应用程序则有必要定义独立的函数实例的行为，诸如不同的代理服务器等。

ole32!CoCreateInstance

功能：用指定的类标识符创建一个 Com 对象 , 用指定的类标识符创建一个未初始化的对象

函数原型：

```
STDAPI CoCreateInstance(
    REFCLSID rclsid, // 创建的 Com 对象的类标识符 (CLSID)
    LPUNKNOWN pUnkOuter, // 指向接口 IUnknown 的指针
```

```
    DWORD dwClsContext, // 运行可执行代码的上下文
    REFIID riid, // 创建的 Com 对象的接口标识符
    LPVOID * ppv // 用来接收指向 Com 对象接口地址的指针变量
);
```

参数说明：

rclsid

[in] 用来唯一标识一个对象的 CLSID(128 位), 需要用它来创建指定对象。

pUnkOuter

[in] 如果为 NULL, 表明此对象不是聚合式对象一部分。如果不是 NULL, 则指针指向一个聚合式对象的 IUnknown 接口。

dwClsContext

[in] 组件类别 . 可使用 CLSCTX 枚举器中预定义的值 .

riid

[in] 引用接口标识符 , 用来与对象通信。

ppv

[out] 用来接收指向接口地址的指针变量。如果函数调用成功 ,*ppv 包括请求的接口指针。

返回值：

S_OK：指定的 Com 对象实例被成功创建。

REGDB_E_CLASSNOTREG：指定的类没有在注册表中注册 . 也可能是指定的 dwClsContext 没有注册或注册表中的服务器类型损坏

CLASS_E_NOAGGREGATION：这个类不能创建为聚合型。

E_NOINTERFACE：指定的类没有实现请求的接口 , 或者是 IUnknown 接口没有暴露请求的接口 .

备注：创建一个 COM 对象，COM 对象提供了非常多样化的功能。类标识（CLSID）会告诉你哪个文件包含着实现 COM 对象的代码。

wininet!FtpPutFile

功能：将本地文件上传到 FTP 服务器

函数原型：

```
BOOL WINAPI FtpPutFile( HINTERNET hConnect,
    LPCTSTR lpszLocalFile,
    LPCTSTR lpszNewRemoteFile,
    DWORD dwFlags,
```

```
    DWORD dwContext
);
```

参数介绍：

hConnect： FTP 会话句柄

lpszLocalFile 本地文件路径

lpszNewRemoteFile 上传到 ftp 服务器的文件保存路径

dwFlags 指示文件上传的条件

dwContext 指定应用数据该搜索相关联的应用程序定义的值。此参数仅当应用程序已调用 InternetSetStatusCallback 成立一个状态回调。所有的状态请求都得到同步处理。

返回值：TRUE 表示成功，FALSE 表示失败

备注：一个高层次上的函数，用来向一个远程 FTP 服务器上传文件。

Iphlpapi!GetAdaptersInfo

功能：获取网卡详细信息

函数原型：

```
DWORD GetAdaptersInfo(
    _Out_   PIP_ADAPTER_INFO pAdapterInfo,
    _Inout_ PULONG           pOutBufLen
);
```

参数介绍：

pAdapterInfo：一个缓冲区的指针，用来接收 IP_ADAPTER_INFO 结构的信息。

结构体声明如下：

```
typedef struct _IP_ADAPTER_INFO {
    struct _IP_ADAPTER_INFO* Next;                          // 指向链表中下一个适配器信息的指针
    DWORD ComboIndex;/                                      / 预留值
    char AdapterName[MAX_ADAPTER_NAME_LENGTH + 4];          // 适配器名称
    char Description[MAX_ADAPTER_DESCRIPTION_LENGTH + 4];   // 适配器描述
    UINT AddressLength;                                     // 适配器硬件地址以字节计算的长度
    BYTE Address[MAX_ADAPTER_ADDRESS_LENGTH];               // 硬件地址以 BYTE 数组所表示
    DWORD Index;                                            // 适配器索引
    UINT Type;// 适配器类型
    UINT DhcpEnabled;                                       // 指定这个适配器是否开启 DHCP
    PIP_ADDR_STRING CurrentIpAddress;                       // 预留值
    IP_ADDR_STRING IpAddressList;                           // 该适配器的 IPv4 地址链表
    IP_ADDR_STRING GatewayList;                             // 该适配器的网关 IPv4 地址链表
```

```
    IP_ADDR_STRING DhcpServer;                    // 该适配器的 DHCP 服务器的 IPv4 地址链表
    BOOL HaveWins;
    IP_ADDR_STRING PrimaryWinsServer;
    IP_ADDR_STRING SecondaryWinsServer;
    time_t LeaseObtained;
    time_t LeaseExpires;
} IP_ADAPTER_INFO,*PIP_ADAPTER_INFO;
```

pOutBufLen：表示 pAdapterInfo 的大小

返回值：成功返回 ERROR_SUCCESS，其他表示失败

备注：用来获取系统上网络适配器的相关信息。后门程序有时会调用此函数来取得关于受感染主机的摘要信息。在某些情况下，这个函数也会被使用来取得主机的 MAC 地址，在对抗虚拟机技术中用来检测 VMware 等虚拟机。

Ws2_32!gethostbyname

功能：返回对应于给定主机名的包含主机名字和地址信息的 hostent 结构指针

函数原型：

struct hostent *gethostbyname(const char *name);

参数介绍：

name：指向主机名的指针。

返回值：如果没有错误发生，gethostbyname() 返回如上所述的一个指向 hostent 结构的指针，否则，返回一个空指针

结构体声明如下：

```
struct hostent{
    char * h_name;// 地址的正式名称。
    char ** h_aliases;// 空字节 - 地址的预备名称的指针。
    short h_addrtype;// 地址类型 ; 通常是 AF_INET。
    short h_length;// 地址的比特长度。
    char ** h_addr_list;// 零字节 - 主机网络地址指针。网络字节顺序。
};
```

备注：在向一个远程主机发起 IP 连接之前，用来对一个特定域名执行一次 DNS 查询，作为命令控制服务器的域名通常可以用来创建很好的网络监测特征码

ws2_32!gethostname

功能：返回本地主机的标准主机名。

函数原型：int PASCAL FAR gethostname(char FAR *name, int namelen);

参数介绍：

name： 一个指向将要存放主机名的缓冲区指针。

namelen：缓冲区的长度。

返回值：如果没有错误发生，gethostname() 返回 0。否则它返回 SOCKET_ERROR

备注：获取计算机主机名。后门程序经常使用此函数来获取受害主机的摘要信息

ws2_32!inet_addr

功能：将一个点分十进制的 IP 转换成一个长整数型数

函数原型：in_addr_t inet_addr(const char* strptr);

参数介绍：strptr：字符串，一个点分十进制的 IP 地址

返回值：如果正确执行将返回一个无符号长整数型数。如果传入的字符串不是一个合法的 IP 地址，将返回 INADDR_NONE。

备注：将一个 IP 地址字符串，如 127.0.0.1，进行转化，使其能够在如 connect 等函数中使用。这些字符串有时也可以用作基于网络的特征码。

wininet!InternetOpen

功能：初始化一个应用程序，以使用 WinINet 函数。

函数原型：

```
HINTERNET InternetOpen(
    _In_ LPCTSTR lpszAgent,
    _In_ DWORD dwAccessType,
    _In_ LPCTSTR lpszProxyName,
    _In_ LPCTSTR lpszProxyBypass,
    _In_ DWORD dwFlags
);
```

参数介绍：

lpszAgent

指向一个空结束的字符串，该字符串指定的应用程序或实体调用 WinInet 函数的名称。使用此名称作为用户代理的 HTTP 协议。

dwAccessType

指定访问类型

lpszProxyName

指针指向一个空结束的字符串，该字符串指定的代理服务器的名称，不要使用空字符串；如果 dwAccessType 未设置为 INTERNET_OPEN_TYPE_PROXY，则此参数应该设置为 NULL。

lpszProxyBypass

指向一个空结束的字符串，该字符串指定的可选列表的主机名或 IP 地址。如果 dwAccessType 未设置为 INTERNET_OPEN_TYPE_PROXY 的 ，参数省略则为 NULL。

dwFlags：网络选项标志位

返回值：

成功：返回一个有效的句柄，该句柄将由应用程序传递给接下来的 WinINet 函数。

失败：返回 NULL。

备注：初始化 WinINet 中的一些高层次互联网访问函数，搜索此函数是找到互联网访问功能初始位置的一个好方法。该函数的一个参数是 User-Agent，有时也可以作为基于网络的特征码。

wininet!InternetOpenUrl

功能：通过一个完整的 FTP，Gopher 或 HTTP 网址打开一个资源

函数原型：

```
HINTERNET InternetOpenUrl(
    HINTERNET hInternetSession,
    LPCTSTR lpszUrl,
    LPCTSTR lpszHeaders,
    DWORD dwHeadersLength,
    DWORD dwFlags,
    DWORD dwContext
);
```

参数介绍：

hInternet

当前的 Internet 会话句柄。句柄必须由前期的 InternetOpen 调用返回。

lpszUrl

一个空字符结束的字符串变量的指针，指定读取的网址。只有以 ftp:, gopher:, http:, 或者 https: 开头的网址被支持。

lpszHeaders

一个空字符结束的字符串变量的指针，指定发送到 HTTP 服务器的头信息。欲了解更多信息，请参阅 HttpSendRequest 函数里 lpszHeaders 参数的说明。

dwHeadersLength：lpszHeaders 的长度。

dwFlags：行为标志的位掩码

dwContext

一个指向一个应用程序定义的值，将随着返回的句柄，一起传递给回调函数

返回值：如果已成功建立到 FTP，Gopher，或 HTTP URL 的连接，返回一个有效的句柄，如果连接失败返回 NULL。

备注：使用 FTP,HTTP 或 HTTPS 协议连接来打开一个特定的 URL，如果 URL 固定，则可以作为基于网络的特征码

wininet!InternetReadFile

功能：从一个由 InternetOpenUrl,FtpOpenFile, 或 HttpOpenRequest 函数打开的句柄中读取数据

函数原型：

```
BOOL InternetReadFile( __in HINTERNET hFile,
    __out LPVOID lpBuffer,
    __in DWORD dwNumberOfBytesToRead,
    __out LPDWORD lpdwNumberOfBytesRead
);
```

参数介绍：

hFile：由 InternetOpenUrl,FtpOpenFile, 或 HttpOpenRequest 函数返回的句柄 .

lpBuffer：缓冲区指针

dwNumberOfBytesToRead：欲读数据的字节量。

lpdwNumberOfBytesRead：接收读取字节量的变量。该函数在做任何工作或错误检查之前都设置该值为零

返回值：成功：返回 TRUE，失败，返回 FALSE

备注：从之前打开的 URL 中读取数据

wininet!InternetWriteFile

功能：向服务器上传文件

函数原型：

```
BOOL InternetWriteFile(
    _In_  HINTERNET hFile,
    _In_  LPCVOID   lpBuffer,
    _In_  DWORD     dwNumberOfBytesToWrite,
    _Out_ LPDWORD   lpdwNumberOfBytesWritten
);
```

参数介绍：hFile ： 由 InternetOpenUrl,FtpOpenFile, 或 HttpOpenRequest 函数返回的句柄 .

lpBuffer：缓冲区指针

dwNumberOfBytesToWrite ：欲写数据的字节量

lpdwNumberOfBytesWritten：实际写入的数据的字节量

返回值：成功：返回 TRUE，失败，返回 FALSE

备注：写数据到之前打开的一个 URL

Netapi32!NetShareEnum

功能：返回服务器上共享的资源的信息

函数原型：

```
NET_API_STATUS NetShareEnum(
    _In_   LPWSTR  servername,
    _In_   DWORD   level,
    _Out_   LPBYTE  *bufptr,
    _In_   DWORD   prefmaxlen,
    _Out_   LPDWORD entriesread,
    _Out_   LPDWORD totalentries,
    _Inout_   LPDWORD resume_handle
);
```

参数介绍：servername：函数要执行的远程服务的 DNS 或 NetBIOS 名字

Level：数据的信息级别

Bufptr：接收数据的缓冲区，数据的格式取决于参数 Level。释放此缓冲区需要使用 NetApiBufferFree

Prefmaxlen：指定返回的数据的最大长度

Entriesread：接收枚举到的实际的数据的个数。

Totalentries：接收枚举到的所有函数入口的个数。

resume_handle：用来接收一个句柄，此句柄被用来继续进行搜索。当第一次搜索时，此句柄需要为 0。

返回值：成功返回 NERR_Success，失败返回系统的错误码

备注：用来枚举网络共享的函数

ole32!OleInitialize

功能 是在当前单元（apartment）初始化组件对象模型（COM）库，将当前的并发模式标识为 STA（single-thread apartment——单线程单元），并启用一些特别用于 OLE 技术的额外功能

函数原型：WINOLEAPI OleInitialize(LPVOID pvReserved);

参数介绍：pvReserved 为保留参数，在使用函数时必须设定为 NULL

返回值：

这个函数除了支持标准的返回值 E_INVALIDARG，E_OUTOFMEMORY 和 E_UNEXPECTED 之外还可能产生以下返回值

S_OK：COM 库和 OLE 技术所特有的额外功能在当前单元被成功初始化。

S_FALSE：COM 库在当前单元已经被初始化过。

OLE_E_WRONGCOMPOBJ：本机上的文件 COMPOBJ.DLL 和 OLE2.DLL 的版本不相匹配。

RPC_E_CHANGED_MODE：之前有一个对函数 CoInitializeEx 的调用指明了这个单元的并发模式为 MTA（multithread apartment——多线程单元）。如果当前正在使用 Windows 2000，这个返回值也能表明发生了 NA（neutral threaded apartment——中立线程单元）到 STA 的转换。

备注：用来初始化 COM 库，使用 COM 对象的程序必须在调用任何其他 COM 功能之前，调用这个函数。

ws2_32!recv

功能：从已经连接的 socket 中接收数据

函数原型：

```
int recv( _In_ SOCKET s,
    _Out_ char *buf,
    _In_ int len,
    _In_ int flags
);
```

参数介绍：

s: 已经连接的 socket

buf：接收数据的缓冲区

len：buf 缓冲区的长度

flags：影响此函数行为的标志

返回值：成功返回接收的数据的长度，失败返回 0.

备注：从一个远程主机获取数据，恶意代码经常使用这个函数来从远程的命令控制服务器获取数据。

ws2_32!send

功能：向一个已经连接的 socket 发送数据

函数原型：int PASCAL FAR send(SOCKET s, const char FAR* buf, int len, int flags);

参数介绍：

s：一个用于标识已连接套接口的描述字。

buf：包含待发送数据的缓冲区。

len：缓冲区中数据的长度。

flags：调用执行方式。

返回值：如果无错误，返回值为所发送数据的总数，否则返回 SOCKET_ERROR。

备注：发送数据到远程主机，恶意代码经常使用这个函数来发送数据到远程的命令控制服务器。

urlmon!URLDownloadToFile

功能：从指定 URL 地址读取内容并将读取到的内容保存到特定的文件里

函数原型：

```
HRESULT URLDownloadToFile(
    LPUNKNOWN pCaller,
    LPCTSTR szURL,
    LPCTSTR szFileName,
    DWORD dwReserved,
    LPBINDSTATUSCALLBACK lpfnCB
);
```

参数介绍：

pCaller 控件的接口 , 如果不是控件则为 0

szURL 要下载的 url 地址 , 不能为空

szFileName 下载后保存的文件名 .

dwReserved 保留字段 , 必需为 0

lpfnCB 下载进度状态回调

返回值：成功返回 S_OK，失败返回相应的错误码。

备注：一个高层次的函数调用，来从一个 WEB 服务器下载文件并存储到硬盘上。这个函数在下载器中是非常普遍的，因为他以一个函数调用便实现了下载器的所有功能。

ws2_32!WSAStartup

功能：初始化底层级别的网络函数

函数原型：int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData);

参数介绍：

wVersionRequested：一个 WORD（双字节）型数值，在最高版本的 Windows Sockets 支持调用者使用，高阶字节指定小版本 (修订本) 号 , 低位字节指定主版本号。

lpWSAData 指向 WSADATA 数据结构的指针，用来接收 Windows Sockets 实现的细节。

返回值：成功返回 0，否则返回错误码

备注：用来初始化底层级别的网络功能，搜索此函数调用位置，经常是定诶网络相关功能最简单的方法。

注册表与服务类

advapi32!CreateService

功能：创建一个服务对象，并将其添加到指定的服务控制管理器数据库

函数原型：

```
SC_HANDLE CreateService(
SC_HANDLE hSCManager,
LPCTSTR lpServiceName,
LPCTSTR lpDisplayName,
DWORD dwDesiredAccess,
DWORD dwServiceType,
DWORD dwStartType,
DWORD dwErrorControl,
LPCTSTR lpBinaryPathName,
LPCTSTR lpLoadOrderGroup,
LPDWORD lpdwTagId,
LPCTSTR lpDependencies,
LPCTSTR lpServiceStartName,
LPCTSTR lpPassword
);
```

参数介绍：

hSCManager, // 服务控制管理程序维护的登记数据库的句柄，由系统函数 OpenSCManager 返回

lpServiceName, // 以 NULL 结尾的服务名，用于创建登记数据库中的关键字

lpDisplayName, // 以 NULL 结尾的服务名，用于用户界面标识服务

dwDesiredAccess, // 指定服务返回类型

dwServiceType, // 指定服务类型

dwStartType, // 指定何时启动服务

dwErrorControl, // 指定服务启动失败的严重程度

lpBinaryPathName, // 指定服务程序二进制文件的路径

lpLoadOrderGroup, // 指定顺序装入的服务组名

lpdwTagId, // 忽略，NULL

lpDependencies, // 指定启动该服务前必须先启动的服务或服务组

lpServiceStartName, // 以 NULL 结尾的字符串，指定服务帐号。如是 NULL, 则表示使用 LocalSystem 帐号

lpPassword // 以 NULL 结尾的字符串，指定对应的口令。为 NULL 表示无口令。但使用 LocalSystem 时填 NULL

返回值：

如果函数成功，返回值将是该服务的句柄。

如果函数失败，则返回值为 NULL

备注：创建一个可以再启动时刻运行的服务。恶意代码使用此函数来持久化，隐藏或者是启动内核驱动。

advapi32!ControlService

功能：向服务发送一个控制码

函数原型：

```
BOOL WINAPI ControlService(
    _In_  SC_HANDLE  hService,
    _In_  DWORD dwControl,
    _Out_ LPSERVICE_STATUS lpServiceStatus
);
```

参数介绍：

hService：服务的句柄，此句柄由 OpenServier 或者 CreateService 返回。

dwControl：控制码

lpServiceStatus：指向 SERVICE_STATUS 的指针，用来接收最近的服务状态信息。

结构体声明如下：

```
typedef struct _SERVICE_STATUS {
    DWORD dwServiceType;// 指明服务可执行文件的类型。
    DWORD dwCurrentState;// 用于通知 SCM 此服务的现行状态
    DWORD dwControlsAccepted;// 指明服务接受什么样的控制通知
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;// 是一个服务用来报告它当前的事件进展情况的。
    DWORD dwWaitHint;// 是一个服务用来报告它当前的事件进展情况的。
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

返回值：成功返回非 0，失败返回 0.

备注：用来启动，停止，修改或发送一个信号到运行服务。如果恶意代码使用了他自己的恶意服务，你就需要分析实现服务的代码，来确定出调用的用意。

advapi32!OpenSCManager

功能：建立了一个到服务控制管理器的连接，并打开指定的数据库。

函数原型：

```
SC_HANDLE WINAPI OpenSCManager(
```

```
    _In_opt_ LPCTSTR lpMachineName,
    _In_opt_ LPCTSTR lpDatabaseName,
    _In_    DWORD   dwDesiredAccess
);
```

参数介绍：

lpMachineName

指向零终止字符串，指定目标计算机的名称。如果该指针为 NULL ，或者它指向一个空字符串，那么该函数连接到本地计算机上的服务控制管理器。

lpDatabaseName

指向零终止字符串，指定将要打开的服务控制管理数据库的名称。此字符串应被置

为 SERVICES_ACTIVE_DATABASE。如果该指针为 NULL ，则打开默认的 SERVICES_ACTIVE_DATABASE

数据库。

dwDesiredAccess

指定服务访问控制管理器的权限

返回值：

如果函数成功，返回值是一个指定的服务控制管理器数据库的句柄。

如果函数失败，返回值为 NULL 。

备注：打开一个到服务控制管理器的句柄。任何想要安装，修改或是控制一个服务的程序，都必须要调用这个函数，才能使用其他服务操纵函数。

user32!RegisterHotKey

功能：定义一个系统范围的热键

函数原型：

```
BOOL RegisterHotKey（HWND hWnd，
    int id，
    UINT fsModifiers，
    UINT vk
）；
```

参数介绍：

hWnd：接收热键产生 WM_HOTKEY 消息的窗口句柄。若该参数 NULL，传递给调用线程的 WM_HOTKEY 消息必须在消息循环中进行处理。

id：定义热键的标识符。调用线程中的其他热键，不能使用同样的标识符

fsModifoers：定义为了产生 WM_HOTKEY 消息而必须与由 nVirtKey 参数定义的键一起按下的键。

vk：定义热键的虚拟键码。

返回值：

若函数调用成功，返回一个非 0 值。若函数调用失败，则返回值为 0

备注：用来注册一个热键，当用户任意时刻输入一个特定键值组合时，注册热键句柄将会被通知，无论当用户输入键值组合时哪个窗口是活跃的，这个函数通常被间谍软件使用，使其在键值组合中输入前对用户保持隐藏。

advapi32!RegOpenKey

功能：打开给定键

函数原型：

```
LONG RegOpenKey( HKEY hKey,
    LPCTSTR lpSubKey,
    PHKEY phkResult
);
```

参数介绍：

HKEY hKey：要打开键的句柄

LPCTSTR lpSubKey：要打开子键的名字的地址

PHKEY phkResult ：要打开键的句柄的地址

返回值：

如果调用成功，返回 ERROR_SUCCESS。

如果调用失败，返回一个非零错误码

备注：打开一个注册表键值的句柄，来进行读写。修改注册表键值通常是软件在主机上进行持久化保存的一种方法。这册表也包含了完整的操作系统和应用程序配置信息。

advapi32!StartServiceCtrlDispatcher

功能：将服务进程的主线程连接到服务控制管理器

函数原型：

```
BOOL WINAPI StartServiceCtrlDispatcher(
    _In_ const SERVICE_TABLE_ENTRY *lpServiceTable
);
```

参数介绍：lpServiceTable：指向 SERVICE_TABLE_ENTRY 结构体数组，包含进程可以运行的服务的入口。

结构体声明如下：

```
typedef struct _SERVICE_TABLE_ENTRY {
    LPTSTR  lpServiceName;// 服务名字
    LPSERVICE_MAIN_FUNCTION lpServiceProc;// 指向 ServiceMain 函数的指针
} SERVICE_TABLE_ENTRY, *LPSERVICE_TABLE_ENTRY;
```

返回值：成功返回非 0，失败返回 0.

备注：由服务使用来连接到服务管理控制进程的主线程。任何以服务方式运行的进程必须在启动后 30 秒内调用这个函数。在恶意代码中找到这个函数，可以知道他的功能应该以服务方式运行。

Advapi32!RegCloseKey

功能：释放指定注册键的句柄

函数原型：

```
LONG RegCloseKey(
    HKEY hKey // 释放键的句柄
);
```

参数介绍：hKey : [输入] 想要关闭的已经打开的键。

返回值：如果过程执行成功，返回值是 ERROR_SUCCESS。如果功能失败，返回一个非零值，错误码在 Winerror.h 定义。可以使用 FormatMessage 函数和 FORMAT_MESSAGE_FROM_SYSTEM 标记获得一个分类的错误描述。

备注：被关闭的句柄将不可以再使用，因为已经不再有效。

Advapi32!RegCreateKey

功能：用于创建或打开注册表项

函数原型：

```
LONG WINAPI RegCreateKey(
    _In_ HKEY hKey,
    _In_opt_ LPCTSTR lpSubKey,
    _Out_ PHKEY phkResult
);
```

参数介绍：

hKey: 指向当前打开表项的句柄，或者是下列预定义保留句柄值之一，实际上就是注册表中的几个分支。

HKEY_CLASSES_ROOT

HKEY_CURRENT_CONFIG

HKEY_CURRENT_USER

HKEY_USERS

HKEY_LOCAL_MACHINE

lpSubKey: 指向一个空终止的字符串指针，指示这个函数将打开或创建的表项的名称。这个表项必须是由 hKey 参数所标识的项的子项

phkResult: 这是一个返回值，指向一个变量的指针，用来接受创建或打开的表项的句柄。当不再需要此返回的注册表项句柄时，调用 RegCloseKey 函数关闭这个句柄。

返回值：0 表示成功，其他任何值都代表一个错误代码 .

Advapi32!RegCreateKeyEx

功能：用于创建指定的注册键

函数原型：

```
LONG WINAPI RegCreateKeyEx(
    __in HKEY hKey,
    __in LPCTSTR lpSubKey,
    DWORD Reserved,
    __in LPTSTR lpClass,
    __in DWORD dwOptions,
    __in REGSAM samDesired,
    __in LPSECURITY_ATTRIBUTES lpSecurityAttributes,
    __out PHKEY phkResult,
    __out LPDWORD lpdwDisposition
);
```

参数介绍：

hKey Long，一个打开项的句柄，或者一个标准项名

lpSubKey String，欲创建的新子项的名字

Reserved Long，设为零

lpClass String，项的类名

dwOptions Long，下述常数为零：REG_OPTION_VOLATILE——这个项不正式保存下来，系统重新启动后会消失

samDesired Long，带有前缀 KEY_?? 的一个或多个常数。它们组合起来描述了允许对这个项进行哪些操作

lpSecurityAttributes SECURITY_ATTRIBUTES，对这个项的安全特性进行描述的一个结构（用 ByVal As Long 传递空值）。不适用于 windows 95

phkResult Long，指定用于装载新子项句柄的一个变量

lpdwDisposition Long，用于装载下列某个常数的一个变量：

REG_CREATED_NEW_KEY——新建的一个子项

REG_OPENED_EXISTING_KEY——打开一个现有的项

返回值：成功则返回 ERROR_SUCCESS

备注：REG_OPTION_VOLATILE 不适用于 windows 95

Advapi32!RegDeleteKey

功能：用来删除一个注册表键值。

函数原型：

```
LONG WINAPI RegDeleteKey(
    __in HKEY hKey,
    __in LPCTSTR lpSubKey
);
```

参数介绍：

hKey：注册表打开的键值的句柄。删除的键值必须是拥有访问权限的

lpSubKey: 被删除的键值名称。它必须是 hkey 的一个子项，但它并不能有子项。此参数不能为空。

返回值：

如果函数成功，返回值是 ERROR_SUCCESS。

如果函数失败，返回值是非零错误代码定义在 Winerror.h。

备注：

删除的键值不能取消，直至最后处理它被关闭。

被删除的键值不能有子项。删除的一个键值和所有其子项，您需要枚举子项，并单独删除他们。删除键递归，使用 RegDeleteTree 或 SHDeleteKey 函数。

Advapi32!RegOpenKey

功能：打开给定键

函数原型：

```
LONG RegOpenKey(
    HKEY hKey,
    LPCTSTR lpSubKey,
    PHKEY phkResult
);
```

参数介绍：

hKey, // 要打开键的句柄

lpSubKey, // 要打开子键的名字的地址

phkResult // 要打开键的句柄的地址

返回值：

如果调用成功，返回 ERROR_SUCCESS。

如果调用失败，返回一个非零错误码（定义在 WINERROR.H）。

备注：RegOpenKey 函数使用默认的安全存取掩码打开一个键。如果打开的键需要一个不同的掩码，函数将发生错误，返回 ERROR_Access_DENIED。在这种情形下一个应用程序应该使用 RegOpenKeyEx 函数去指定存取掩码。

Advapi32!RegOpenKeyEx

功能：用于打开一个指定的注册表键

函数原型：

```
LONG RegOpenKeyEx(
    HKEY hKey, // 需要打开的主键的名称
    LPCTSTR lpSubKey, // 需要打开的子键的名称
    DWORD ulOptions, // 保留，设为 0
    REGSAM samDesired, // 安全访问标记，也就是权限
    PHKEY phkResult // 得到的将要打开键的句柄
)
```

参数介绍：

hKey：需要打开的主键的名称

lpSubKey：需要打开的子键的名称

ulOptions：保留，设为 0

samDesired：安全访问标记，也就是权限

phkResult： 得到的将要打开键的句柄

返回值：如果函数调用成功，则返回 0（ERROR_SUCCESS）。否则，返回值为文件 WINERROR.h 中定义的一个非零的错误代码。

Advapi32!RegDeleteValue

功能：用于删除一个键下的一个键值

函数原型：

LONGWINAPIRegDeleteValue(

```
    _In_HKEY hKey,
    _In_opt_LPCTSTR lpValueName
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或标准项名之一

lpValueName String，要删除的值名。可设为 vbNullString 或一个空串，表示删除那个项的默认值

返回值：成功返回 ERROR_SUCCESS ，失败返回一个非零值

Advapi32!RegQueryValue

功能：取得指定项或子项的默认（未命名）值

函数原型：

```
LONG RegQueryValue(
    HKEY hkey,
    LPCTSTR lpSubkey,
    LPTSTR lpValue,
    PLONG lpcbValue
);
```

参数介绍：

hKey: 一个已打开表项的句柄，或者指定一个标准项名（即注册表中的几个根注册表项）

lpSubKey: 指向一个空终止的字符串指针，指示这个函数将打开或创建的表项的名称。这个表项必须是由 hkey 参数所标识的子项。

lpValue: 一个返回值，指向一个缓存区，用来获得与指定子项默认值相关的一个字符串。

lpcbValue: 指定一个变量，用于装载 lpValue 缓冲区的长度。一旦返回，它会设为实际载入缓冲区的字节数量。该大小包含了数据长度还加上了终止符的空字符串。

返回值：0（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码。

Advapi32!RegSetValue

功能：设置指定注册表项的默认值或未命名值的数据

函数原型：

```
LONG WINAPI RegSetValue(
    _In_ HKEY hKey,
    _In_opt_ LPCTSTR lpSubKey,
    _In_ DWORD dwType,
    _In_ LPCTSTR lpData,
```

```
    _In_ DWORD cbData
);
```

参数介绍：

hKey: 指向当前打开表项的句柄，或者预定保留值之一（注册表中默认的几大注册表项）

lpSubKey: 指向一个空终止的字符串指针，指示这个函数将打开或创建表项的名称。可以为 NULL，或空字符串，那么此函数将为参数 hkey 所指定的注册表项默认设置值。这个表项必须是由 hkey 参数所标识的子项。

dwType: 指示将被存储的信息类型。该参数必须为 REG_SZ

lpData: 指向一个空终止的字符串，该字符串中包含了要为指定项的默认值设置的数据。

cdData: 指示 lpData 参数所指向的字符串的大小，单位是字节，但不包含字符串最后的空终止字符

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegSetValueEx

功能：设置指定值的数据和类型

函数原型：

```
LONG RegSetValueEx(
    HKEY hKey,
    LPCTSTR lpValueName,
    DWORD Reserved,
    DWORD dwType,
    CONST BYTE *lpData,
    DWORD cbData
);
```

参数介绍：

hKey: 一个已打开项的句柄，或指定一个标准项名

lpValueName: 指向一个字符串的指针，该字符串包含了欲设置值的名称。若拥有该值名称的值并不存在于指定的注册表项中，则此函数将其加入到该项。如果此值是 NULL，或指向空字符串，则此函数为该项的默认值或未命名值设置类型和数据。

Reserved: 保留值，必须强制为 0

dwType: 指定将被存储的数据类型，该参数可以为

REG_BINARY 任何形式的二进制数据

REG_DWORD 一个 32 位的数字

REG_DWORD_LITTLE_ENDIAN 一个“低字节在前”格式的 32 位数字

REG_DWORD_BIG_ENDIAN 一个“高字节在前”格式的 32 位数字

REG_EXPAND_SZ 一个以 0 结尾的字符串，该字符串包含对环境变量（如“%PAHT”）的未扩展引用

REG_LINK 一个 Unicode 格式的带符号链接

REG_MULTI_SZ 一个以 0 结尾的字符串数组，该数组以连接两个 0 为终止符

REG_NONE 未定义值类型

REG_RESOURCE_LIST 一个设备驱动器资源列表

REG_SZ 一个以 0 结尾的字符串

lpData: 指向一个缓冲区，该缓冲区包含了欲为指定值名称存储的数据。

cdData: 指定由 lpData 参数所指向的数据的大小，单位是字节。

返回值：0（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegQueryInfoKey

功能：获取与一个项有关的信息

函数原型：

```
LONG WINAPI RegQueryInfoKey(
    _In_       HKEY     hKey,
    _Out_opt_  LPTSTR   lpClass,
    _Inout_opt_ LPDWORD  lpcClass,
    _Reserved_  LPDWORD  lpReserved,
    _Out_opt_  LPDWORD  lpcSubKeys,
    _Out_opt_  LPDWORD  lpcMaxSubKeyLen,
    _Out_opt_  LPDWORD  lpcMaxClassLen,
    _Out_opt_  LPDWORD  lpcValues,
    _Out_opt_  LPDWORD  lpcMaxValueNameLen,
    _Out_opt_  LPDWORD  lpcMaxValueLen,
    _Out_opt_  LPDWORD  lpcbSecurityDescriptor,
    _Out_opt_  PFILETIME lpftLastWriteTime
);
```

参数介绍：

hKey HKEY，一个已打开项的句柄，或指定一个标准项名

lpClass LPWSTR，指定一个字串，用于装载这个注册表项的类名

lpcbClass LPDWORD，指定一个变量，用于装载 lpClass 缓冲区的长度。一旦返回，它会设为实际装载到缓冲区的字节数量

lpReserved LPDWORD，未用，设为 NULL

lpcSubKeys LPDWORD，用于装载（保存）这个项的子项数量的一个变量

lpcbMaxSubKeyLen LPDWORD，指定一个变量，用于装载这个项最长一个子项的长度。注意这个长度不包括空中止字符

lpcbMaxClassLen LPDWORD，指定一个变量，用于装载这个项之子项的最长一个类名的长度。注意这个长度不包括空中止字符

lpcValues LPDWORD，用于装载这个项的设置值数量的一个变量

lpcbMaxValueNameLen LPDWORD，指定一个变量，用于装载这个项之子项的最长一个值名的长度。注意这个长度不包括空中止字符

lpcbMaxValueLen LPDWORD，指定一个变量，用于装载容下这个项最长一个值数据所需的缓冲区长度

lpcbSecurityDescriptor LPDWORD，装载值安全描述符长度的一个变量

lpftLastWriteTime FILETIME，指定一个结构，用于容纳该项的上一次修改时间

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegEnumKey

功能：获取指定的子键值

函数原型：

```
LONG WINAPI RegEnumKey(
    _In_  HKEY   hKey,
    _In_  DWORD  dwIndex,
    _Out_ LPTSTR lpName,
    _In_  DWORD  cchName
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或者指定一个标准项名

dwIndex Long，欲获取的子项的索引。第一个子项的索引编号为零

lpName String，用于装载指定索引处项名的一个缓冲区

cbName Long，lpName 缓冲区的长度

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegEnumKeyEx

功能：枚举指定项下方的子项

函数原型：

```
LONG WINAPI RegEnumKeyEx(
    _In_      HKEY     hKey,
    _In_      DWORD    dwIndex,
    _Out_     LPTSTR   lpName,
    _Inout_   LPDWORD  lpcName,
    _Reserved_ LPDWORD  lpReserved,
    _Inout_   LPTSTR   lpClass,
    _Inout_opt_ LPDWORD  lpcClass,
    _Out_opt_  PFILETIME lpftLastWriteTime
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或者指定一个标准项名

dwIndex Long，欲获取的子项的索引。第一个子项的索引编号为零

lpName String，用于装载指定索引处项名的一个缓冲区

lpcbName Long，指定一个变量，用于装载 lpName 缓冲区的实际长度（包括空字符）。一旦返回，它会设为实际装载到 lpName 缓冲区的字符数量

lpReserved Long，未用，设为零

lpClass String，项使用的类名。可以为 vbNullString

lpcbClass Long，用于装载 lpClass 缓冲区长度的一个变量。一旦返回，它会设为实际装载到缓冲区的字符数量

lpftLastWriteTime FILETIME，枚举子项上一次修改的时间

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegEnumValue

功能：用来枚举指定项的值

函数原型：

```
LONG WINAPI RegEnumValue(
    _In_      HKEY    hKey,
    _In_      DWORD   dwIndex,
    _Out_     LPTSTR  lpValueName,
    _Inout_   LPDWORD lpcchValueName,
    _Reserved_ LPDWORD lpReserved,
    _Out_opt_  LPDWORD lpType,
    _Out_opt_  LPBYTE  lpData,
    _Inout_opt_ LPDWORD lpcbData
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或者指定一个标准项名

dwIndex Long，欲获取值的索引。注意第一个值的索引编号为零

lpValueName String，用于装载位于指定索引处值名的一个缓冲区

lpcbValueName Long，用于装载 lpValueName 缓冲区长度的一个变量。一旦返回，它会设为实际载入缓冲区的字符数量

lpReserved Long，未用；设为零

lpType Long，用于装载值的类型代码的变量

lpData Byte，用于装载值数据的一个缓冲区

lpcbData Long，用于装载 lpData 缓冲区长度的一个变量。一旦返回，它会设为实际载入缓冲区的字符数量

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegLoadKey

功能：从以前用 RegSaveKey 函数创建的一个文件里装载注册表信息

函数原型：

```
LONG WINAPI RegLoadKey(
    _In_     HKEY    hKey,
    _In_opt_ LPCTSTR lpSubKey,
    _In_     LPCTSTR lpFile
);
```

参数介绍：

hKey Long，HKEY_LOCAL_MACHINE、HKEY_USERS 或者用 RegConnectRegistry 创建的一个子项

lpSubKey String，要创建的新子项的名字

lpFile String，包含了注册信息的那个文件的名字

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegReplaceKey

功能：用一个磁盘文件保存的信息替换注册表信息；并创建一个备份，在其中包含当前注册表信息

函数原型：

```
LONG WINAPI RegReplaceKey(
    _In_     HKEY    hKey,
    _In_opt_ LPCTSTR lpSubKey,
```

```
    _In_    LPCTSTR lpNewFile,
    _In_    LPCTSTR lpOldFile
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或指定一个标准项名

lpSubKey String，要替换的子项名称。它必须直接位于 HKEY_LOCAL_MACHINE 或 HKEY_USERS 控制项的下方

lpNewFile String，包含了注册表信息的一个文件的名字。这个文件是用 RegSaveKey 函数创建的

lpOldFile String，对当前注册表信息进行备份的一个文件的名字

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegRestoreKey

功能：从一个磁盘文件恢复注册表信息

函数原型：

```
LONG WINAPI RegRestoreKey(
    _In_ HKEY hKey,
    _In_ LPCTSTR lpFile,
    _In_ DWORD dwFlags
);
```

参数介绍：

hKey [in]：可打开注册表项的句柄

hKey Long，一个已打开项的句柄，或者指定一个标准项名

lpFile String，要从中恢复注册表信息的一个文件的名字

dwFlags Long，0 表示进行常规恢复。REG_WHOLE_HIVE_VOLATILE 表示临时恢复信息（系统重新启动时不保存下来）。在这种情况下，hKey 必须引用 HKEY_LOCAL_MACHINE 或 HKEY_USERS

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegSaveKey

功能：将一个项以及它的所有子项都保存到一个磁盘文件

函数原型：

```
LONG WINAPI RegSaveKey(
    _In_    HKEY            hKey,
    _In_    LPCTSTR         lpFile,
```

```
    _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

参数介绍：

hKey Long，一个已打开项的句柄，或指定一个标准项名

lpFile String，要在其中保存注册表信息的一个磁盘文件的名字

lpSecurityAttributes SECURITY_ATTRIBUTES，为保存的信息提供的安全信息。可设为 NULL，表示采用默认的安全信息（变成 ByVal As Long，并传递零值）

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegConnectRegistry

功能：访问远程系统的部分注册表

函数原型：

```
LONG WINAPI RegConnectRegistry(
    _In_opt_ LPCTSTR lpMachineName,
    _In_    HKEY    hKey,
    _Out_   PHKEY   phkResult
);
```

参数介绍：

lpMachineName String，欲连接的系统。采用“\\ 计算机名”的形式

hKey Long，HKEY_LOCAL_MACHINE 或 HKEY_USERS

phkResult Long，用于装载指定项句柄的一个变量

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegNotifyChangeKeyValue

功能：使应用程序可以接收事件通知中指定的注册表项及其子项的更改

函数原型：

```
LONG WINAPI RegNotifyChangeKeyValue(
    _In_    HKEY   hKey,
    _In_    BOOL   bWatchSubtree,
    _In_    DWORD  dwNotifyFilter,
    _In_opt_ HANDLE hEvent,
    _In_    BOOL   fAsynchronous
);
```

参数介绍：

hKey Long，要监视的一个项的句柄，或者指定一个标准项名

bWatchSubtree Long，TRUE（非零）表示监视子项以及指定的项

dwNotifyFilter Long，下述常数的一个或多个

REG_NOTIFY_CHANGE_NAME 侦测注册表项名称的变化，以及侦测注册表的创建和删除事件

REG_NOTIFY_CHANGE_ATTRIBUTES 侦测属性的变化

REG_NOTIFY_CHANGE_LAST_SET 侦测上一次修改时间的变化

REG_NOTIFY_CHANGE_SECURITY 侦测对安全特性的改动

hEvent Long，一个事件的句柄。如 fAsynchronus 为 False，则这里的设置会被忽略

fAsynchronus Long，如果为零，那么除非侦测到一个变化，否则函数不会返回。否则这个函数会立即返回，而且在发生变化时触发由 hEvent 参数指定的一个事件

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

Advapi32!RegUnloadKey

功能：卸载指定的项以及它的所有子项

函数原型：

```
LONG WINAPI RegUnLoadKey(
    _In_    HKEY    hKey,
    _In_opt_ LPCTSTR lpSubKey
);
```

参数介绍：

hKey Long，HKEY_LOCAL_MACHINE、HKEY_USERS 或者用 RegConnectRegistry 打开的一个子项

lpSubKey String，要卸载的子项的名字。必须是早先用 RegLoadKey 函数载入的

返回值：零（ERROR_SUCCESS）表示成功。其他任何值都代表一个错误代码

进程线程类

user32!AttachThreadInput

功能：把一个线程的输入消息连接到另外的线程

函数原型：

```
BOOL WINAPI AttachThreadInput(
    __in  DWORD idAttach,
    __in  DWORD idAttachTo,
    __in  BOOL fAttach
);
```

参数介绍：

idAttach: 指定要连接到另外一个线程的线程。该线程不能是系统线程。

idAttachTo: 要连接其他线程的线程，该线程不能是系统线程。且线程不能自己连接到自己。

fAttach: 为 TRUE：连接； 为 FALSE: 释放连接

返回值：如果调用成功则返回非零值。

备注：

在一些情况下，自己的窗口没有输入焦点但是想要当前焦点窗口的键盘输入消息，可以使用 Win32 API 函数 AttachThreadInput() 来解决这个问题

kernel32!CheckRemoteDebuggerPresent

功能：检查一个特定进程是否被调试。这个函数通常在一个反调试技术中被使用。

函数原型：

```
BOOL WINAPI CheckRemoteDebuggerPresent(
    _In_   HANDLE hProcess,
    _Inout_ PBOOL  pbDebuggerPresent
);
```

参数介绍：

hProcess：进程句柄 .

pbDebuggerPresent：如果 hProcess 所代表的进程被调试，此变量返回 TRUE，否则返回 FALSE。

返回值：成功返回非 0，失败返回 0.

备注：

kernel32!ConnectNamedPipe

功能：指示一台服务器等待下去，直至客户机同一个命名管道连接。

函数原型：

```
BOOL WINAPI ConnectNamedPipe(
    _In_       HANDLE       hNamedPipe,
    _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

参数介绍：

hNamedPipe：管道的句柄。

lpOverlapped，如设为 NULL（传递 ByVal As int），表示将线程挂起，直到一个客户同管道连接为止。否则就立即返回；此时，如管道尚未连接，客户同管道连接时就会触发 lpOverlapped 结构中的事件对象。随后，可用一个等待函数来监视连接

返回值：

Long，如 lpOverlapped 为 NULL，那么：

如管道已连接，就返回 Ture（非零）；如发生错误，或者管道已经连接，就返回零（GetLastError 此时会返回 ERROR_PIPE_CONNECTED）

lpOverlapped 有效，就返回零；如管道已经连接，GetLastError 会返回 ERROR_PIPE_CONNECTED；如重叠操作成功完成，就返回 ERROR_IO_PENDING。在这两种情况下，倘若一个客户已关闭了管道，且服务器尚未用 DisconnectNamedPipe 函数同客户断开连接，那么 GetLastError 都会返回 ERROR_NO_DATA

备注：用来为进程间通信创建一个服务端管道，等待一个客户端管道连接进来。后门程序和反向 shell 经常使用此函数来简单的连接到一个命令控制服务器。

kernel32!CreateProcess

功能：创建一个新的进程和它的主线程，这个新进程运行指定的可执行文件。

函数原型：

```
BOOL CreateProcess(
    LPCTSTR lpApplicationName,
    LPTSTR lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL bInheritHandles,
    DWORD dwCreationFlags,
    LPVOID lpEnvironment,
    LPCTSTR lpCurrentDirectory,
    LPSTARTUPINFO lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

参数说明：

lpApplicationName

指向一个 NULL 结尾的、用来指定可执行模块的字符串。

这个字符串可以是可执行模块的绝对路径，也可以是相对路径，在后一种情况下，函数使用当前驱动器和目录建立可执行模块的路径。

这个参数可以被设为 NULL，在这种情况下，可执行模块的名字必须处于 lpCommandLine 参数最前面并由空格符与后面的字符分开。

lpCommandLine

指向一个以 NULL 结尾的字符串，该字符串指定要执行的命令行。

这个参数可以为空，那么函数将使用 lpApplicationName 参数指定的字符串当做要运行的程序的命令行。

如果 lpApplicationName 和 lpCommandLine 参数都不为空，那么 lpApplicationName 参数指定将要被运行的模块，lpCommandLine 参数指定将被运行的模块的命令行。新运行的进程可以使用 GetCommandLine 函数获得整个命令行。C 语言程序可以使用 argc 和 argv 参数。

lpProcessAttributes

指向一个 SECURITY_ATTRIBUTES 结构体，这个结构体决定是否返回的句柄可以被子进程继承。如果 lpProcessAttributes 参数为空（NULL），那么句柄不能被继承。

lpThreadAttributes

同 lpProcessAttribute, 不过这个参数决定的是线程是否被继承 . 通常置为 NULL.

bInheritHandles

指示新进程是否从调用进程处继承了句柄。

如果参数的值为真，调用进程中的每一个可继承的打开句柄都将被子进程继承。被继承的句柄与原进程拥有完全相同的值和访问权限。

dwCreationFlags

指定附加的、用来控制优先类和进程的创建的标志。还用来控制新进程的优先类，优先类用来决定此进程的线程调度的优先级。

lpEnvironment

指向一个新进程的环境块。如果此参数为空，新进程使用调用进程的环境。

lpCurrentDirectory

指向一个以 NULL 结尾的字符串，这个字符串用来指定子进程的工作路径。这个字符串必须是一个包含驱动器名的绝对路径。如果这个参数为空，新进程将使用与调用进程相同的驱动器和目录。这个选项是一个需要启动应用程序并指定它们的驱动器和工作目录的外壳程序的主要条件。

lpStartupInfo

指向一个用于决定新进程的主窗体如何显示的 STARTUPINFO 结构体。

lpProcessInformation

指向一个用来接收新进程的识别信息的 PROCESS_INFORMATION 结构体。

结构体声明如下：

typedef struct_PROCESS_INFORMATION{

HANDLE hProcess;// 返回新进程的句柄

HANDLE hThread;// 返回主线程的句柄。

DWORD dwProcessId;// 返回一个全局进程标识符。该标识符用于标识一个进程。从进程被创建到终止，该值始终有效。

DWORD dwThreadId;// 返回一个全局线程标识符。该标识符用于标识一个线程。从线程被创建到终止，该值始终有效。

}PROCESS_INFORMATION;

返回值：

如果函数执行成功，返回非零值。

如果函数执行失败，返回零

备注：创建并启动一个新进程。如果恶意代码创建了一个新进程，你需要同时分析这个新进程。

kernel32!CreateRemoteThread

功能：创建一个在其它进程地址空间中运行的线程 (也称 : 创建远程线程)

函数原型：

```
HANDLE WINAPI CreateRemoteThread(
    __in HANDLE hProcess,
    __in LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in SIZE_T dwStackSize,
    __in LPTHREAD_START_ROUTINE lpStartAddress,
    __in LPVOID lpParameter,
    __in DWORD dwCreationFlags,
    __out LPDWORD lpThreadId
);
```

参数介绍：

hProcess [in]

线程所属进程的进程句柄 .

该句柄必须具有 PROCESS_CREATE_THREAD, PROCESS_QUERY_INFORMATION, PROCESS_VM_OPERATION, PROCESS_VM_WRITE, 和 PROCESS_VM_READ 访问权限 .

lpThreadAttributes [in]

一个指向 SECURITY_ATTRIBUTES 结构的指针 , 该结指定了线程的安全属性 .

dwStackSize [in]

线程初始大小 , 以字节为单位 , 如果该值设为 0, 那么使用系统默认大小 .

lpStartAddress [in]

在远程进程的地址空间中 , 该线程的线程函数的起始地址 .

lpParameter [in]

传给线程函数的参数 .

dwCreationFlags [in]

线程的创建标志 .

lpThreadId [out]

指向所创建线程句柄的指针 , 如果创建失败 , 该参数为 NULL.

返回值：

如果调用成功 , 返回新线程句柄 .

如果失败 , 返回 NULL.

备注：用来在一个远程进程中启动一个线程。启动器和隐蔽性恶意代码通常使用这个函数，将代码注入到其他进程中执行。

kernel32!CreateToolhelp32Snapshot

功能：通过获取进程信息为指定的进程、进程使用的堆 [HEAP]、模块 [MODULE]、线程建立一个快照 .

函数原型：

```
HANDLE WINAPI CreateToolhelp32Snapshot(
    DWORD dwFlags,
    DWORD th32ProcessID
);
```

参数说明：

dwFlags, 用来指定“快照”中需要返回的对象，可以是 TH32CS_SNAPPROCESS 等

th32ProcessID 一个进程 ID 号，用来指定要获取哪一个进程的快照，当获取系统进程列表或获取 当前进程快照时可以设为 0

返回值：

调用成功，返回快照的句柄，调用失败，返回 INVALID_HANDLE_VALUE 。

备注：

用来创建一个进程，堆空间，线程和模块的快照。恶意代码经常使用这个函数，在多个进程或线程之间传播感染。

kernel32!EnumProcesses

功能：检索进程中的每一个进程标识符 .

函数原型：

```
BOOL WINAPI EnumProcesses（
    _Out_ DWORD * pProcessIds，
    _In_ DWORD CB，
    _Out_ DWORD * pBytesReturned
）；
```

参数介绍：

pProcessIds 接收进程标识符的数组

cb 数组的大小 .

pBytesReturned 数组返回的字节数 .

返回值：成功返回非零数 , 失败返回零 , 可以使用函数 GetLastError 获取错误信息 .

备注：用来在系统上枚举运行进程的函数，恶意代码经常枚举进程来找到一个可以注入的进程。

kernel32!EnumProcessModules

功能：枚举进程模块

函数原型：

```
BOOL WINAPI EnumProcessModules(
    _In_  HANDLE  hProcess,
    _Out_ HMODULE *lphModule,
    _In_  DWORD   cb,
    _Out_ LPDWORD lpcbNeeded
);
```

参数介绍：

hProcess：要枚举的进程的句柄

lphModule：该进程所包含的模块句柄的数组，我们可以定义一个数组来接受该进程所含有的 // 模块句柄

cb：lphModule 数组的大小

lpcbNeeded：该进程实际的模块的数量，以字节来计数

返回值：成功返回非 0，失败返回 0

备注：用来枚举给定进程的已装载模块（可执行文件和 DLL 程序），恶意代码在进行注入时经常枚举模块

kernel32!IsWow64Process

功能：确定指定进程是否运行在 64 位操作系统的 32 环境（Wow64）下

函数原型：BOOL WINAPI IsWow64Process(__in HANDLE hProcess, __out PBOOL Wow64Process);

参数介绍：

hProcess：进程句柄。该句柄必须具有 PROCESS_QUERY_INFORMATION 或者 PROCESS_QUERY_LIMITED_INFORMATION 访问权限

Wow64Process：指向一个 bool 值，如果该进程是 32 位进程，运行在 64 操作系统下，该值为 true，否则为 false。如果该进程是一个 64 位应用程序，运行在 64 位系统上，该值也被设置为 false。

返回值：

如果函数成功返回值为非零值。如果该函数失败，则返回值为零

备注：由一个 32 位进程使用，来确定它是否运行在 64 位操作系统上

ntdll!ZwQueryInformationProcess

功能：返回特定进程的信息

函数原型：

```
NTSTATUS WINAPI ZwQueryInformationProcess(
    _In_    HANDLE          ProcessHandle,
    _In_    PROCESSINFOCLASS ProcessInformationClass,
    _Out_   PVOID           ProcessInformation,
    _In_    ULONG           ProcessInformationLength,
    _Out_opt_ PULONG        ReturnLength
);
```

参数介绍：

ProcessHandle：需要查询的进程的句柄。

ProcessInformationClass：需要查询的进程信息的类型。

ProcessInformation：接收进程信息的缓冲区

ProcessInformationLength：ProcessInformation 内容的大小。

ReturnLength：返回信息的大小。

返回值：成功返回 STATUS_SUCCESS，失败返回错误码

备注：返回关于一个特定进程的不同信息。这个函数通常在反调试技术中被使用。

kernel32!OpenProcess

功能：用来打开一个已存在的进程对象，并返回进程的句柄

函数原型：

```
HANDLE OpenProcess(
    DWORD dwDesiredAccess,
    BOOL bInheritHandle,
    DWORD dwProcessId
);
```

参数介绍：

dwDesiredAccess：渴望得到的访问权限（标志）

bInheritHandle：是否继承句柄

dwProcessId：进程标示符

返回值：

如成功，返回值为指定进程的句柄。

如失败，返回值为空

备注：打开系统上运行其他进程的句柄。这个句柄可以被用来向其他进程内存中读写数据，或是注入代码到其他进程中。

kernel32!PeekNamedPipe

功能：预览一个管道中的数据，或取得与管道中的数据有关的信息。

函数原型：

```
BOOL WINAPI PeekNamedPipe(
    __in HANDLE hNamedPipe,                    // 管道句柄
    __out_opt LPVOID lpBuffer,                 // 读取输出缓冲区，可选
    __in DWORD nBufferSize,                    // 缓冲区大小
    __out_opt LPDWORD lpBytesRead,             // 接收从管道中读取数据的变量的指针，可选
    __out_opt LPDWORD lpTotalBytesAvail,       // 接收从管道读取的字节总数
    __out_opt LPDWORD lpBytesLeftThisMessage
);
```

参数介绍：

hNamedPipe：管道句柄。这个参数可以是一个命名管道实例句柄，返回，由 CreateNamedPipe 或

CreateFile 函数，或者它可以是一个匿名管道的读端句柄，返回由 CREATEPIPE 功能。句柄必须有 GENERIC_READ 权限的管道。

lpBuffer：接收从管道读取数据的缓冲区的指针。如果没有数据要读取，此参数可以为 NULL。

nBufferSize：lpBuffer 参数以字节为单位，由指定的缓冲区大小。如果 lpBuffer 是 NULL，则忽略此参数。

lpBytesRead：接收从管道中读取的字节数的变量的指针。此参数可以为 NULL，如果没有数据要读取。

lpTotalBytesAvail：一个指针变量，接收从管道读取的字节总数。此参数可以为 NULL，如果没有数据要读取。

lpBytesLeftThisMessage：指向剩余的字节数的变量的指针消息。此参数将是零字节类型的命名管道或匿名管道。此参数可以为 NULL，如果没有数据要读取。

返回值：非零表示成功，零表示失败

备注：用来从一个命名管道中复制数据，而无须从管道中移除数据。这个函数在反向 shell 中很常用。

kernel32!Process32First

功能：是一个进程获取函数，当我们利用函数 CreateToolhelp32Snapshot() 获得当前运行进程的快照后，我们可以利用 process32First 函数来获得第一个进程的句柄。

函数原型：

```
BOOL WINAPI Process32First(
    HANDLE hSnapshot,
    LPPROCESSENTRY32 lppe
)
```

参数介绍：

hSnapshot：调用 CreateToolhelp32Snapshot 返回的快照句柄。

lppe：指向 PROCESSENTRY32 结构体

结构体声明如下：

```
typedef struct tagPROCESSENTRY32
{
    DWORD dwSize;                    // 结构体的大小
    DWORD cntUsage;                  // 此进程的引用计数
    DWORD th32ProcessID;
    ULONG_PTR th32DefaultHeapID;     // 进程默认堆 ID
    DWORD th32ModuleID;              // 进程模块 ID
    DWORD cntThreads;                // 此进程开启的线程计数
    DWORD th32ParentProcessID;
    LONG pcPriClassBase;             // 线程优先权
    DWORD dwFlags;                   // 这个成员已经不再被使用，总是设置为零
    TCHAR szExeFile[MAX_PATH];       // 进程全名
```

```
} PROCESSENTRY32, *PPROCESSENTRY32;
```

返回值：成功返回 true，失败返回 FALSE

备注：在调用 CreateToolhelp32Snapshot 之后，使用此函数和 Process32Next 来枚举进程。恶意代码通常枚举进程，来找到一个可以注入的进程。

kernel32!Process32Next

功 能：当我们利用函数 CreateToolhelp32Snapshot() 获得当前运行进程的快照后，我们可以利用 Process32Next 函数来获得下一个进程的句柄。

函数原型：

```
BOOLWINAPIProcess32Next(
    __in HANDLEhSnapshot,
    __out LPPROCESSENTRY32lppe
);
```

参数介绍：

hSnapshot：从 CreateToolhelp32Snapshot 返回的句柄。

Lppe：指向 PROCESSENTRY32 结构的指针。

返回值：成功返回 TRUE，失败返回 FALSE。

备注：在调用 CreateToolhelp32Snapshot 之后，使用此函数和 Process32First 来枚举进程。恶意代码通常枚举进程，来找到一个可以注入的进程。

kernel32!ReadProcessMemory

功能：根据进程句柄读入该进程的某个内存空间

函数原型：

```
BOOL ReadProcessMemory(
    HANDLE hProcess,
    PVOID  pvAddressRemote,
    PVOID   pvBufferLocal,
    DWORD  dwSize,
    PDWORD  pdwNumBytesRead
);
```

参数介绍：

hProcess [in] 远程进程句柄。 被读取者

pvAddressRemote [in] 远程进程中内存地址。 从具体何处读取

pvBufferLocal [out] 本地进程中内存地址 . 函数将读取的内容写入此处

dwSize [in] 要传送的字节数。要写入多少

pdwNumBytesRead [out] 实际传送的字节数 . 函数返回时报告实际写入多少

返回值：成功返回 1，失败返回 0.

备注：用来从远程进程中读取内存。

kernel32!ResumeThread

功能：恢复挂起的线程

函数原型：DWORD WINAPI ResumeThread(__in HANDLE hThread);

参数介绍：hThread：需要恢复的线程的句柄。

返回值：成功返回线程先前的挂起次数，失败返回 -1.

备注：继续之前挂起的线程。此函数在几种注入技术中都会被使用。

kernel32!SetThreadContext

功能：修改给定线程的上下文

函数原型：

```
BOOL WINAPI SetThreadContext(
    _In_ HANDLE  hThread,
    _In_ const CONTEXT *lpContext
);
```

参数介绍：

hThread：目标线程的句柄

lpContext：目标线程的上下文

返回值：修改成功，返回非 0，失败返回 0

备注：一些注入技术会使用这个函数。

shell32!ShellExecute

功能：运行一个外部程序（或者是打开一个已注册的文件、打开一个目录、打印一个文件等等），并对外部程序有一定的控制

函数原型：

```
HINSTANCE ShellExecute(
    _In_opt_ HWND    hwnd,
```

```
    _In_opt_ LPCTSTR lpOperation,
    _In_     LPCTSTR lpFile,
    _In_opt_ LPCTSTR lpParameters,
    _In_opt_ LPCTSTR lpDirectory,
    _In_     INT     nShowCmd
);
```

参数介绍：

Hwnd：指定父进程句柄

lpOperation：指定动作，例如：open、runas、print、edit、explore、find

lpFile：指定要打开的文件或程序

lpParameters：给要打开的程序指定的参数，如果打开的是文件，这里应该是 null

lpDirectory。缺省目录

nShowCmd：打开选项

返回值：成功返回应用程序句柄。失败返回相应的错误码

备注：用来执行另一个程序，如果恶意代码创建了一个新的进程，需要分析这个新进程。

kernel32!SuspendThread

功能：暂停指定的线程

函数原型：

```
DWORD WINAPI SuspendThread(
    _In_HANDLE hThread
);
```

参数介绍：hThread：需要暂停的线程的句柄

返回值：如果成功，返回线程先前的挂起计数，失败返回 -1

备注：挂起一个线程，使得他停止运行。恶意代码有时会挂起一个线程，铜鼓代码注入技术来修改它。

kernel32!Thread32First

功能：返回进程中第一个线程的信息

函数原型：

```
BOOL WINAPI Thread32First(
    _In_    HANDLE          hSnapshot,
    _Inout_ LPTHREADENTRY32 lpte
);
```

参数介绍：

hSnapshot 句柄从先前的调用返回的快照 CreateToolhelp32Snapshot 功能。

lpte 一个指向一个 THREADENTRY32 结构。

返回值：如果第一个条目的线程列表复制到缓冲区返回 TRUE，否则返回 FALSE。

备注：用来轮询一个进程的所有线程。注入器会使用这些函数来找出可供注入的合适线程。

kernel32!Thread32Next

功能：返回进程中下一个线程的信息

函数原型：

```
BOOL WINAPI Thread32Next(
    _In_  HANDLE          hSnapshot,
    _Out_ LPTHREADENTRY32 lpte
);
```

参数介绍：

hSnapshot 句柄从先前的调用返回的快照 CreateToolhelp32Snapshot 功能。

lpte 一个指向一个 THREADENTRY32 结构。

返回值：如果下一个条目的线程列表复制到缓冲区返回 TRUE，否则返回 FALSE。

备注：用来轮询一个进程的所有线程。注入器会使用这些函数来找出可供注入的合适线程。

kernel32!WinExec

功能：运行指定的程序

函数原型：

```
UINT WINAPI WinExec(
    _In_ LPCSTR lpCmdLine,
    _In_ UINT   uCmdShow
);
```

参数介绍：

lpCmdLine：指向一个空结束的字符串，串中包含将要执行的应用程序的命令行（文件名加上可选参数）。

uCmdShow：定义 Windows 应用程序的窗口如何显示，并为 CreateProcess 函数提供 STARTUPINFO 参数的 wShowWindow 成员的值。

返回值：返回值大于 31 表示成功，否则返回相应的错误码

备注：

kernel32!WriteProcessMemory

功能：写入某一进程的内存区域。入口区必须可以访问，否则操作将失败

函数原型：

```
BOOL WriteProcessMemory(
    HANDLE hProcess,
    LPVOID lpBaseAddress,
    LPVOID lpBuffer,
    DWORD nSize,
    LPDWORD lpNumberOfBytesWritten
);
```

参数介绍：

hProcess 由 OpenProcess 返回的进程句柄。

如参数传数据为 INVALID_HANDLE_VALUE 【即 -1】目标进程为自身进程

lpBaseAddress 要写的内存首地址，再写入之前，此函数将先检查目标地址是否可用，并能容纳待写入的数据。

lpBuffer 指向要写的数据的指针。

nSize 要写入的字节数。

返回值：非零代表成功，零代表失败

备注：用来向远程进程写数据的函数，恶意代码在进程注入中会用到此函数。

Rpccrt4!RpcServrRegisterIf

功能：注册与 RPC 运行时库的接口。

函数原型：

```
RPC_STATUS RPC_ENTRY RpcServerRegisterIf(
    RPC_IF_HANDLE IfSpec,
    UUID        *MgrTypeUuid,
    RPC_MGR_EPV   *MgrEpv
);
```

参数介绍：

IfSpec: 表名要注册的接口

MgrTypeUuid: 指向与 MgrEpv 参数关联的类型 UUID 的指针

MgrEpv: 例程的入口点向量，使用 MIDL 生成默认的 EPV

返回值：成功返回 RPC_S_OK

Rpccrt4!RpcServerListen

功能：指示 RPC 运行时库来监听远程过程调用

函数原型：

```
RPC_STATUS RPC_ENTRY RpcServerListen(
    unsigned int MinimumCallThreads,
    unsigned int MaxCalls,
    unsigned int DontWait
);
```

参数介绍：

MinimumCallThreads：指定应在给定服务器中创建和维护的最小调用线程数

MaxCalls：建议服务器可以执行的并发远程过程调用的最大数量

DontWait：控制从 RpcServerListen 返回的标志。非零值表示完成功能处理后，RpcServerListen 应立即返回。值为零表示 RpcServerListen 不会返回，直到 RpcMgmtStopServerListening 函数已被调用并且所有远程调用都已完成。

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_ALREADY_LISTENING	服务端已经在监听
RPC_S_NO_PROTSEQS_REGISTERED	没有注册协议序列
RPC_S_MAX_CALLS_TOO_SMALL	MaxCalls 值太小

Rpccrt4!RpcServerUseProtseqEp

功能：告诉 RPC 运行时库使用指定的协议序列与指定的端点组合来接收远程过程调用。

函数原型：

```
RPC_STATUS RPC_ENTRY RpcServerUseProtseqEp(
    unsigned char *Protseq,
    unsigned int  MaxCalls,
    unsigned char *Endpoint,
    void          *SecurityDescriptor
);
```

参数介绍：

Protseq：指向要向 RPC 运行时库中注册的协议序列的字符串标识符

MaxCalls：ncacn_ip_tcp 协议序列的积压队列长度。 所有其他协议序列忽略此参数。 使用 RPC_C_PROTSEQ_MAX_REQS_DEFAULT 指定默认值

Endpoint：指向用于创建 Protseq 参数中指定的协议序列的绑定的端点地址信息。

SecurityDescriptor：指向为安全子系统提供的可选参数。 仅用于 ncacn_np 和 ncalrpc 协议序列

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_PROTSEQ_NOT_SUPPORTED	此主机不支持协议序列
RPC_S_INVALID_RPC_PROTSEQ	协议序列无效
RPC_S_INVALID_ENDPOINT_FORMAT	端点格式无效
RPC_S_OUT_OF_MEMORY	系统内存不足
RPC_S_DUPLICATE_ENDPOINT	端点是重复的
RPC_S_INVALID_SECURITY_DESC	安全描述符无效

Rpccrt4!RpcMgmtStopServerListening

功能：告诉服务端停止对 RPC 的监听

函数原型：

```
RPC_STATUS RPC_ENTRY RpcMgmtStopServerListening(
    RPC_BINDING_HANDLE Binding
);
```

参数介绍：

Binding：要指示远程应用程序停止监听远程过程调用，请为该应用程序指定服务器绑定句柄。 要指导您自己的（本地）应用程序停止监听远程过程调用，请指定一个 NULL 值。

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_INVALID_BINDING	绑定的句柄无效
RPC_S_WRONG_KIND_OF_BINDING	对操作的错误的约束

Rpccrt4!RpcServerUnregisterIf

功能：从 RPC 运行时库注册表中删除一个接口

函数原型：

```
RPC_STATUS RPC_ENTRY RpcServerUnregisterIf(
    RPC_IF_HANDLE IfSpec,
    UUID        *MgrTypeUuid,
    unsigned int  WaitForCallsToComplete
);
```

参数介绍：

IfSpec：从注册表中删除的界面

MgrTypeUuid：指向要从注册表中删除的 EPV 的 UUID 类型的指针

WaitForCallsToComplete：指示是否立即从注册表中删除接口或等待所有当前调用完成的标志。

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_UNKNOWN_MGR_TYPE	manager 类型未知
RPC_S_UNKNOWN_IF	interface 未知

Rpccrt4!RpcStringBindingCompose

功能：创建一个字符串绑定句柄。

函数原型：

```
RPC_STATUS RPC_ENTRY RpcStringBindingCompose(
    TCHAR *ObjUuid,
    TCHAR *ProtSeq,
    TCHAR *NetworkAddr,
    TCHAR *EndPoint,
    TCHAR *Options,
    TCHAR **StringBinding
);
```

参数介绍：

ObjUuid：指向对象 UUID 的以空值终止的字符串的指针

ProtSeq：指向协议序列的以 null 结尾的字符串。

NetworkAddr：指向网络地址的以 null 结尾的字符串

EndPoint：指向端点的以 null 结尾的字符串

Options：指向网络选项的以 null 结尾的字符串

StringBinding：返回一个指针，指向绑定句柄的以 null 结尾的字符串的指针。

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_INVALID_STRING_UUID	指明 UUID 无效

Rpccrt4!RpcBindingFromStringBinding

功能：从绑定句柄的字符串表示中返回绑定句柄。

函数原型：

```
RPC_STATUS RPC_ENTRY RpcBindingFromStringBinding(
    unsigned char     *StringBinding,
    RPC_BINDING_HANDLE *Binding
```

);

参数介绍：

StringBinding：指向绑定句柄的字符串指针。

Binding：返回指向服务器绑定句柄的指针

返回值：

Value	Meaning
RPC_S_OK	成功
RPC_S_INVALID_STRING_BINDING	字符串绑定无效
RPC_S_PROTSEQ_NOT_SUPPORTED	协议序列不支持
RPC_S_INVALID_RPC_PROTSEQ	协议序列无效
RPC_S_INVALID_ENDPOINT_FORMAT	断点格式无效
RPC_S_STRING_TOO_LONG	字符串太长
RPC_S_INVALID_NET_ADDR	网络地址无效
RPC_S_INVALID_ARG	参数无效
RPC_S_INVALID_NAF_ID	网络地址族标识符无效。

Rpccrt4!RpcStringFree

功能：释放由 RPC 运行库分配的字符串

函数原型：

```
RPC_STATUS RPC_ENTRY RpcStringFree(
    unsigned char **String
);
```

参数介绍：

String：指向要释放的字符串

返回值：成功返回 RPC_S_OK

Rpccrt4!RpcExceptionCode

功能：返回一个 code，标识异常出现的类型。

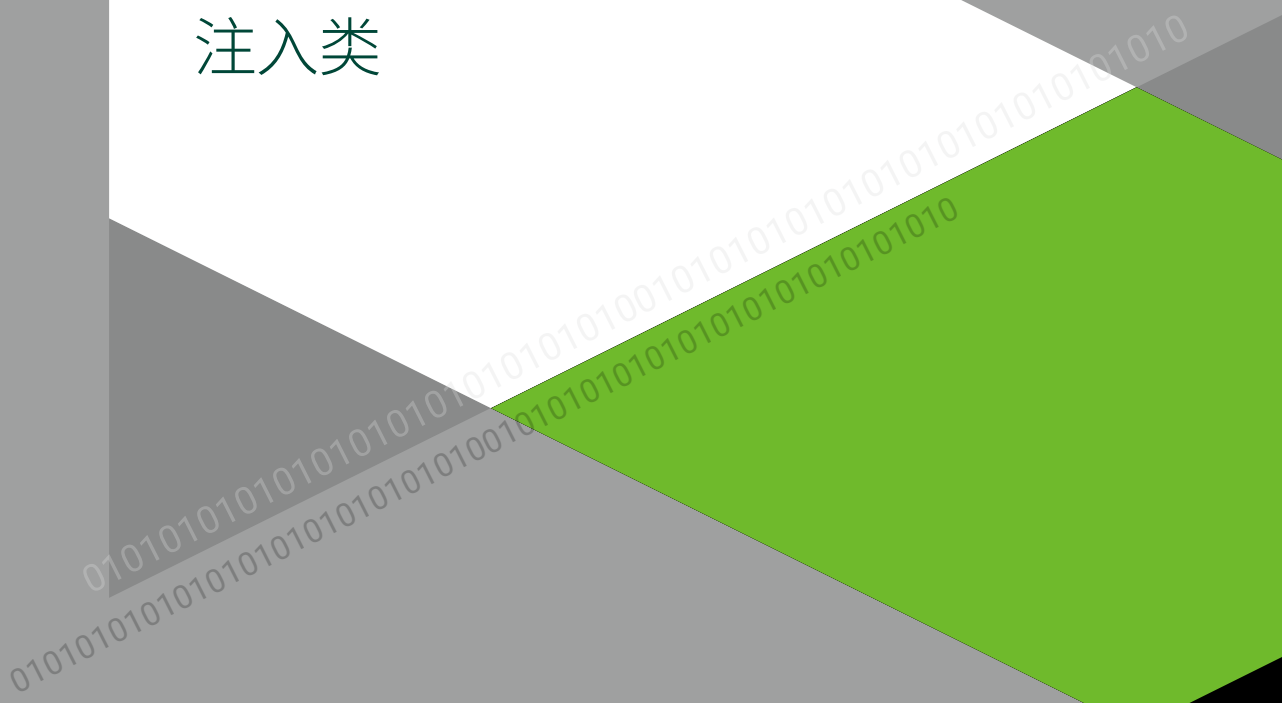
函数原型：

```
unsigned long RpcExceptionCode(void);
```

返回值：

可能的返回值包括由 RPC 函数返回的前缀为“RPC_S_”和“RPC_X”的错误代码集以及 Windows 操作系统返回的异常集。

注入类



kernel32!GetThreadContext

功能：获取目标线程的上下文。

函数原型：

BOOL GetThreadContext(HANDLE hThread, LPCONTEXT lpContext);

参数介绍：

hThread：获取信息目标进程的线程句柄。（用 OpenThread 获取）

lpContext：一个用于接收信息的 CONTEXT 结构指针

返回值：如果成功，返回值不为零 . 如果不成功，返回值为零

备注：返回一个给定线程的上下文结构。线程上下文结构中存储了所有线程信息，比如寄存器值和目前状态。

kernel32!QueueUserAPC

功能：把一个 APC 对象加入到指定线程的 APC 队列中。

函数原型：

```
DWORD QueueUserAPC(
    PAPCFUNC  pfnAPC,
     HANDLE  hThread,
     ULONG_PTR  dwData
);
```

参数介绍：

pfnAPC：指向一个用户提供的 APC 函数的指针

hThread：指定特定线程的句柄

dwData：指定一个被传到 pfnAPC 参数指向的 APC 函数的值。

返回值：成功返回非 0，失败返回 0

备注：用来在其他线程中执行代码，恶意代码有时会使用这个函数注入代码到其他进程。

kernel32!VirtualAllocEx

功能：在指定进程的虚拟空间保留或提交内存区域，除非指定 MEM_RESET 参数，否则将该内存区域置 0。

函数原型：

```
LPVOID VirtualAllocEx(
    HANDLE hProcess,
    LPVOID lpAddress,
    SIZE_T dwSize,
```

```
    DWORD flAllocationType,
    DWORD flProtect
);
```

参数介绍：

hProcess：

申请内存所在的进程句柄。

lpAddress：

保留页面的内存地址；一般用 NULL 自动分配 。

dwSize：

欲分配的内存大小，字节单位；注意实际分 配的内存大小是页内存大小的整数倍

flAllocationType：申请空间的类型。

flProtect：申请空间的保护属性。

返回值：执行成功就返回分配内存的首地址，不成功就是 NULL

备注：一个内存分配的例程，支持在远程进程中分配内存。恶意代码有时会在进程注入中使用此函数。

kernel32!VirtualProtectEx

功能：改变在特定进程中内存区域的保护属性。

函数原型：

```
BOOL VirtualProtectEx(
    HANDLE hProcess,                // 要修改内存的进程句柄
    LPVOID lpAddress,               // 要修改内存的起始地址
    DWORD dwSize,                   // 页区域大小
    DWORD flNewProtect,             // 新访问方式
    PDWORD lpflOldProtect           // 原访问方式 用于保存改变前的保护属性 易语言要传址
);
```

参数介绍：

hProcess：目的进程的句柄。

lpAddress：要修改内存的起始地址。

dwSize：要修改内存的大小

flNewProtect：保护选项

lpflOldProtect：必须指向一个有效的变量，如果是 NULL 或者无效的变量，函数将失败

返回值：如果成功，返回非零。失败返回零

备注：修改一个内存区域的保护机制，恶意代码可能会使用这个函数来将一块只读的内存节修改为可执行代码。

驱动类

kernel32!DeviceIoControl

功能：直接发送控制代码到指定的设备驱动程序，使相应的移动设备以执行相应的操作。

函数原型：

```
BOOL WINAPI DeviceIoControl(
    _In_ HANDLE hDevice,
    _In_ DWORD dwIoControlCode,
    _In_opt_ LPVOID lpInBuffer,
    _In_ DWORD nInBufferSize,
    _Out_opt_ LPVOID lpOutBuffer,
    _In_ DWORD nOutBufferSize,
    _Out_opt_ LPDWORD lpBytesReturned,
    _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

参数介绍：

hDevice，设备句柄

dwIoControlCode，应用程序调用驱动程序的控制命令，就是 IOCTL_XXX IOCTLs。

lpInBuffer，应用程序传递给驱动程序的数据缓冲区地址。

nInBufferSize，应用程序传递给驱动程序的数据缓冲区大小，字节数。

lpOutBuffer，驱动程序返回给应用程序的数据缓冲区地址。

nOutBufferSize，驱动程序返回给应用程序的数据缓冲区大小，字节数。

lpBytesReturned，驱动程序实际返回给应用程序的数据字节数地址。

lpOverlapped，这个结构用于重叠操作。针对同步操作，请用 ByVal As Long 传递零值

返回值：非零表示成功，零表示失败。

备注：从用户空间向设备驱动发送一个控制消息。此函数在驱动级的恶意代码中是非常普遍使用的，因为他是一种最简单和灵活的方式，在用户空间和内核空间之间传递信息。

secur32!LsaEnumerateLogonSessions

功能：枚举当前系统上的登录会话

函数原型：

```
NTSTATUS NTAPI LsaEnumerateLogonSessions(
    _Out_ PULONG LogonSessionCount,
    _Out_ PLUID  *LogonSessionList
);
```

参数介绍：LogonSessionCount：指向一个 long 型的指针，返回 LogonSessionList 链表中的参数个数。

LogonSessionList：指向 LUID 的指针，返回登录会话标识符数组中的第一个成员的地址。

返回值：成功返回 STATUS_SUCCESS，失败返回相应的错误码

备注：枚举当前系统上的登录会话，往往是一个登录凭证窃取器使用的部分功能。

ntoskrnl!MmGetSystemRoutineAddress

功能：返回特定函数的地址

函数原型：

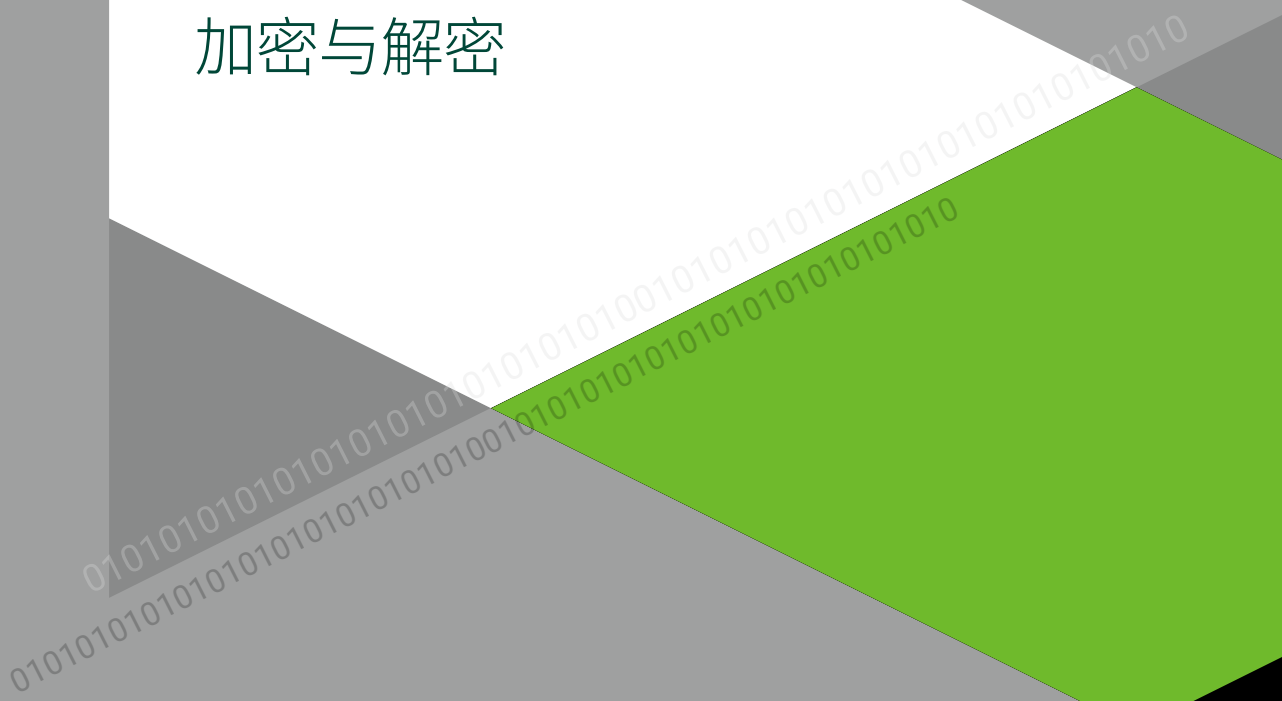
```
PVOID MmGetSystemRoutineAddress(
    _In_ PUNICODE_STRING SystemRoutineName
);
```

参数介绍：SystemRoutineName ： 函数名称

返回值：成功则返回指定函数的地址，否则返回 NULL。

备注：与 GetProcAddress 类似，但是这个函数是内核代码使用的。这个函数从另外一个模块中获取函数的地址，但仅仅可以用来获得 ntoskrnl.exe 和 hal.dll 的函数地址。

加密与解密



Advapi32!CryptAcquireContext

功能：获取有某个容器的 CSP 模块的指针

函数原型：

```
BOOL WINAPI CryptAcquireContext(
    _Out_ HCRYPTPROV  *phProv,
    _In_  LPCTSTR    pszContainer,
    _In_  LPCTSTR    pszProvider,
    _In_  DWORD      dwProvType,
    _In_  DWORD      dwFlags
);
```

参数说明：

phProv：向一个 CSP 模块句柄指针，里面用指定的容器

pszContainer：密钥容器名称， 指向密钥容器的字符串指针。如果 dwFlags 为 CRYPT_VERIFYCONTEXT，pszContainer 必须为 NULL。

pszProvider：密钥容器名称， 指向密钥容器的字符串指针。如果 dwFlags 为 CRYPT_VERIFYCONTEXT，pszContainer 必须为 NULL。

dwProvType：CSP 类型，下表为常见的 CSP 类型

CSP 类型	交换算法	签名算法	对称加密算法	Hash 算法
PROV_RSA_FULL	RSA	RSA	RC2 RC4	MD5 SHA
PROV_RSA_SIG	NONE	RSA	NONE	MD5 SHA
PROV_RSA_SCHANNEL	RSA	RSA	RC4 DES Triple DES	MD5 SHA
PROV_DSS	DSS	NONE	DSS	MD5 SHA
PROV_DSS_DH	DH	DSS	CYLINK_MEK	MD5 SHA
PROV_DH_SCHANNEL	DH	DSS	DES Triple DES	MD5 SHA
PROV_DH_SCHANNEL	KEA	DSS	Skipjack	SHA
PROV_DH_SCHANNEL	RSA	RSA	CAST	MD5
PROV_DH_SCHANNEL	RSA	RSA	Varies	Varies

dwFlags：常被设置为 0，也可以使用如下几个标志：

CRYPT_VERIFYCONTEXT	指出应用程序不需要使用公钥 / 私钥对，如程序只执行哈希和对称加密。只有程序需要创建签名和解密消息时才需要访问私钥。
CRYPT_NEWKEYSET	使用指定的密钥容器名称创建一个新的密钥容器。如果 pszContainer 为 NULL，密钥容器就使用却省的名称创建。
CRYPT_MACHINE_KEYSET	由此标志创建的密钥容器只能由创建者本人或有系统管理员身份的人使用。
CRYPT_DELETEKEYSET	删除由 pszContainer 指定的密钥容器。如果 pszContainer 为 NULL，缺省名称的容器就会被删除。此容器里的所有密钥对也会被删除。
CRYPT_SILENT	应用程序要求 CSP 不显示任何用户界面。

返回值：成功返回 true，失败返回 false

备注：

这个函数是用来取得指定 CSP 密钥容器句柄，以后的任何加密操作就是针对此 CSP 句柄而言。函数首先查找由 dwProvType 和 pszProvider 指定的 CSP，如果找到了 CSP，函数就查找由此 CSP 指定的密钥容器。由适当的 dwFlags 标志，这个函数就可以创建和销毁密钥容器，如果不要求访问私钥的话，也可以提供对 CSP 临时密钥容器的访问。

Advapi32!CryptReleaseContext

功能：释放 CSP 句柄

函数原型：

```
BOOL WINAPI CryptReleaseContext(
    HCRYPTPROV hProv,
    DWORD dwFlags
);
```

参数说明：

hProv[in] 由 CryptAcquireContext 获得的 CSP 句柄。

dwFlags[in] 保留。必须为 0

返回值：成功返回 true，失败返回 false

备注：此函数释放 CSP 的句柄。对于每一次调用，CSP 的引用计数都减 1。当引用计数为 0 时，CSP 上下文就会被系统释放变成无效句柄，以后针对此 CSP 句柄的函数不再可用。

此函数并不销毁密钥容器或密钥对。

Advapi32!CryptEnumProviders

功能：此函数得到第一个或下一个可用的 CSP。如果使用循环，就可以得到计算机上所有可用的 CSP。

函数原型：

```
BOOL WINAPI CryptEnumProviders(
    DWORD dwIndex,
    DWORD *pdwReserved,
    DWORD dwFlags,
    DWORD *pdwProvType,
    LPTSTR pszProvName,
    DWORD *pcbProvName
);
```

参数说明：

dwIndex[in] 枚举下一个 CSP 的索引。

pdwReserved[in] 保留。必须为 NULL。

dwFlags[in] 保留。必须为 NULL。

pdwProvType[out]CSP 的类型。

pszProvName[out] 指向接收 CSP 名称的缓冲区字符串指针。此指针可为 NULL，用来得到字符串的大小。

pcbProvName[in/out] 指出 pszProvName 字符串的大小。

返回值：成功返回 true，失败返回 false

Advapi32!CryptCreateHash

功能：此函数初始化哈希数据流。它创建并返回了一个 CSP 哈希对象的句柄。此句柄由 CryptHashData 和 CryptHashSessionKey 来调用。

函数原型：

```
BOOL WINAPI CryptCreateHash(
    HCRYPTPROV hProv,
    ALG_ID Algid,
    HCRYPTKEY hKey,
    DWORD dwFlags,
    HCRYPTHASH *phHash
);
```

参数说明：

hProv[in]CSP 句柄

Algid[in] 哈希算法的标示符。

hKey[in] 如果哈希算法是密钥哈希，如 HMAC 或 MAC 算法，就用此密钥句柄传递密钥。对于非密钥算法，此参数为 NULL。

dwFlags[in] 保留。必须为 0。

phHash[out] 哈希对象的句柄。

返回值：成功返回 true，失败返回 false

Advapi32! CryptGetHashParam

功能：得到指定哈希对象的数据。

函数原型：

```
BOOL WINAPI CryptGetHashParam(
    HCRYPTHASH hHash,
    DWORD dwParam,
```

```
    BYTE *pbData,
    DWORD *pdwDataLen,
    DWORD dwFlags
);
```

参数说明：

hHash[in] 哈希对象的句柄

dwParam[in] 查询类型。可以是下列：

参数名称	作用
HP_ALGID	哈希算法
HP_HASHSIZE	哈希值长度
HP_HASHVAL	哈希值，由 hHash 指定的哈希值或者消息哈希

返回值：成功返回 true，失败返回 false

Advapi32!CryptDestroyHash

功能：此函数销毁由 hHash 指定的哈希对象。当一个哈希对象被销毁后，它对程序来说不可用。

函数原型：

```
BOOL WINAPI CryptDestroyHash(
    HCRYPTHASH hHash
);
```

参数说明：hHash[in] 要销毁的哈希对象句柄

返回值：成功返回 true，失败返回 false

Advapi32!CryptHashData

功能：添加数据到 hash 对象

函数原型：

```
BOOL WINAPI CryptHashData(
    _In_ HCRYPTHASH hHash,
    _In_ BYTE      *pbData,
    _In_ DWORD     dwDataLen,
    _In_ DWORD     dwFlags
);
```

参数说明：

hHash：Hash 对象的句柄

pdData：指向一块内存，包含要添加到 hash 对象的数据

dwDataLen：数据的长度，如果 dwFlags 设置为 CRYPT_USERDATA，这个值必须为 0.

dwFlags：取值如下：

value	meaning
CRYPT_OWF_REPL_LM_HASH 0x00000001	已经不再使用这个值
CRYPT_USERDATA 1 (0x1)	如果设置此标志，则 CSP 将提示用户直接输入数据。 该数据被添加到 hash。 该应用程序不允许访问此数据。 该标志可用于允许用户在系统中输入 PIN 码。

返回值：成功返回 true，失败返回 false

Advapi32!CryptDeriveKey

功能：此函数从一基本数据值中派生会话密钥。函数保证当 CSP 和算法相同时，从相同基本数据值中产生的密钥是唯一的。

函数原型：

```
BOOL WINAPI CryptDeriveKey(
    HCRYPTPROV hProv,
    ALG_ID Algid,
    HCRYPTHASH hBaseData,
    DWORD dwFlags,
    HCRYPTKEY *phKey
);
```

参数说明：

hProv[in]CSP 句柄

Algid[in] 要产生密钥的对称加密算法

hBaseData[in] 哈希对象的句柄

dwFlags[in] 指定密钥的类型

参数	作用
CRYPT_CREATE_SALT	由哈希值产生一个会话密钥，有一些需要补位。如果用此标志，密钥将会赋予一个盐值
CRYPT_EXPORTABLE	如果置此标志，密钥就可以用 CryptExportKey 函数导出。
CRYPT_NO_SALT	如果置此标志，表示 40 位的密钥不需要分配盐值。
CRYPT_UPDATE_KEY	有些 CSP 从多个哈希值中派生会话密钥。如果这种情况，CryptDeriveKey 需要多次调用。

phKey[in/out] 密钥的句柄

返回值：成功返回 true，失败返回 false

Advapi32!CryptGetProvParam

功能：此函数获得 CSP 的各种参数。

函数原型：

```
BOOL WINAPI CryptGetProvParam(
```

```
    HCRYPTPROV hProv,
    DWORD dwParam,
    BYTE *pbData,
    DWORD *pdwDataLen,
    DWORD dwFlags
);
```

参数说明：

hProv[in]CSP 句柄。

dwParam[in] 指定查询的参数

参数名	作用
PP_CONTAINER	指向密钥名称的字符串
PP_ENUMALGS	不断的读出 CSP 支持的所有算法
PP_ENUMALGS_EX	比 PP_ENUMALGS 获得更多的算法信息
PP_ENUMCONTAINERS	不断的读出 CSP 支持的密钥容器
PP_IMPTYPE	指出 CSP 怎样实现的
PP_NAME	指向 CSP 名称的字符串
PP_VERSION	CSP 的版本号
PP_KEYSIZE_INC	AT_SIGNATURE 的位数
PP_KEYX_KEYSIZE_INC	AT_KEYEXCHANGE 的位数
PP_KEYSET_SEC_DESCR	密钥的安全描述符
PP_UNIQUE_CONTAINER	当前密钥容器的唯一名称
PP_PROVTYPE	CSP 类型
PP_USE_HARDWARE_RNG	指出硬件是否支持随机数发生器
PP_KEYSPEC	返回 CSP 密钥的信息

pbData[out] 指向接收数据的缓冲区指针。

pdwDataLen[in/out] 指出 pbData 数据长度。

dwFlags[in] 如果指定 PP_ENUMCONTAINERS，就指定 CRYPT_MACHINE_KEYSET。

返回值：成功返回 true，失败返回 false

Advapi32!CryptSetKeyParam

功能：自定义会话密钥各方面的操作。此函数设置的值不会保留到内存中，只能在单个会话中使用。

函数原型：

```
BOOL WINAPI CryptSetKeyParam(
    _In_   HCRYPTKEY    hKey,
    _In_   DWORD    dwParam,
    _In_      const BYTE   *pbData,
    _In_   DWORD    dwFlags
);
```

参数说明：

hKey: 要设置值的键的句柄。

dwParam: 可取值过多，详情请参考 msdn

pdData: 在调用 CryptSetKeyParam 之前，使用要设置的值初始化缓冲区的指针。 此数据的形式取决于 dwParam 的值。

dwFlags: dwFlags 参数用于传递启用密钥的标志值。

返回值：成功返回 true，失败返回 false

Advapi32!CryptEncrypt

功能：此函数用于加密数据。加密数据所需要的算法由 hKey 的密钥指定

函数原型：

```
BOOL WINAPI CryptEncrypt(
    HCRYPTKEY hKey,
    HCRYPTHASH hHash,
    BOOL Final,
    DWORD dwFlags,
    BYTE *pbData,
    DWORD *pdwDataLen,
    DWORD dwBufLen
);
```

参数说明：

hKey[in] 加密密钥的句柄

hHash[in] 哈希对象的句柄。如果数据需要同时被哈希并且加密，hHash 就指出了哈希对象。

Final[in] 指出是否是最后一次加密操作。如果 Final 为 TRUE，就为最后一次，否则为 FALSE。

dwFlags[in] 保留

pbData[in/out] 指向被加密的数据地址。

pdwDataLen[in/out] 指向一个 DWORD 值的地址。在调用此函数前，这个值就是需要加密的数据长度。在调用此函数后，这个值就是已经加密的数据长度。如果此值为 NULL，函数就返回需要数据的长度。

dwBufferLen[in] 指出 pbData 的数据长度。

返回值：成功返回 true，失败返回 false

Advapi32!CryptDecrypt

功能：此函数对由 CryptEncrypt 加密过的数据进行解密。

函数原型：

```
BOOL WINAPI CryptDecrypt(
```

```
    HCRYPTKEY hKey,
    HCRYPTHASH hHash,
    BOOL Final,
    DWORD dwFlags,
    BYTE *pbData,
    DWORD *pdwDataLen
);
```

参数说明：

hKey[in] 解密密钥的句柄

hHash[in] 哈希对象的句柄。如果需要解密数据并且同时作哈希，hHash 传递此参数。

Final[in] 指出是否是最后一次解密操作。

dwFlags[in] 保留

pbData[in/out] 需要解密数据的地址

pdwDataLen[in/out] 指向 DWORD 值的指针，此值指出解密数据的长度。在调用此函数前，此值为需要解密数据的长度，调用此函数后，此值为已经解密的数据长度。

返回值：成功返回 true，失败返回 false

Advapi32!CryptDestroyKey

功能：此函数释放密钥句柄。

函数原型：

```
BOOL WINAPI CryptDestroyKey(
    HCRYPTKEY hKey
);
```

参数说明：hKey[in] 需要销毁的密钥句柄

返回值：成功返回 true，失败返回 false

Advapi32!CryptGenKey

功能：生成一个密钥

函数原型：

```
BOOL WINAPI CryptGenKey(
    _In_  HCRYPTPROV hProv,
    _In_  ALG_ID     Algid,
    _In_  DWORD      dwFlags,
    _Out_ HCRYPTKEY  *phKey
);
```

参数说明：

hProv: 指定 CSP 模块的句柄

Algid: 标识要生成密钥的算法的 ALG_ID 值。 该参数的值取决于所使用的 CSP。

dwFlags: 指定了生成密钥的类型

phKey: 接收生成的 Key

返回值：成功返回 true，失败返回 false

Advapi32!CryptGetUserKey

功能：获取用户的公钥 / 私钥对其中一个的句柄，该功能仅由公钥 / 私钥对的所有者使用，并且仅当 CSP 及其相关联的密钥容器的句柄可用时

函数原型：

```
BOOL WINAPI CryptGetUserKey(
    _In_  HCRYPTPROV hProv,
    _In_  DWORD      dwKeySpec,
    _Out_ HCRYPTKEY  *phUserKey
);
```

参数说明：

hProv: 指向 CryptAcquireContext 产生的 CSP

dwKeySpec: 标识从密钥容器使用的私钥。

phUserKey: 接收 key

返回值：成功返回 true，失败返回 false

Advapi32!CryptContextAddRef

功能：增加 CSP 的引用计数器的数值

函数原型：

```
BOOL WINAPI CryptContextAddRef(
    _In_ HCRYPTPROV hProv,
    _In_ DWORD      *pdwReserved,
    _In_ DWORD      dwFlags
);
```

参数说明：

hProv: 指定要增加计数的 CSP

pdwReserved: 必须为 NULL。

dwFlags: 必须事 0

返回值：成功返回 true，失败返回 false；

Advapi32!CryptReleaseContext

功能：释放 CSP 句柄的，当这个函数调用一次的时候，CSP 里面的引用计数就减少一，当引用计数减少的 0 的时候。CSP 将不能再被这个程序中的任何函数调用了

函数原型：

```
BOOL WINAPI CryptReleaseContext(
    _In_ HCRYPTPROV hProv,
    _In_ DWORD      dwFlags
);
```

参数说明：

hProv：指向 CSP 的句柄

dwFlags：必须是 0

返回值：成功返回 true，失败返回 false

Advapi32!CryptExportKey

功能：

函数原型：

```
BOOL WINAPI CryptExportKey(
    _In_    HCRYPTKEY hKey,
    _In_    HCRYPTKEY hExpKey,
    _In_    DWORD     dwBlobType,
    _In_    DWORD     dwFlags,
    _Out_   BYTE      *pbData,
    _Inout_ DWORD     *pdwDataLen
);
```

参数说明：

hKey: 指向要导出的 key

kExpKey: 用户最终使用到的密钥的句柄

dwBlobType: 指定要在 pbData 中导出的键 BLOB 的类型

dwFlags: 指定函数的其他选项

pbData: 指向一块内存，用来接收 key Blob

pdwDataLen: 函数返回时，指明 pbData 的长度。

返回值：成功返回 true，失败返回 false。

消息传递

User32!SendMessage

功能：将指定的消息发送到一个或多个窗口

函数原型：

```
LRESULT WINAPI SendMessage(
    _In_ HWND   hWnd,
    _In_ UINT   Msg,
    _In_ WPARAM wParam,
    _In_ LPARAM lParam
);
```

参数介绍：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息特定信息。

lParam：指定附加的消息特定信息。

返回值：返回值指定消息处理的结果，依赖于所发送的消息

User32!SendMessageCallback

功能：将指定的消息发送到一个或多个窗口，此函数为指定的窗口调用窗口程序，并立即返回；当窗口程序处理完消息后，系统调用指定的回调函数，将消息处理的结果和一个应用程序定义的值传给回调函数

函数原型：

```
BOOL WINAPI SendMessageCallback(
    _In_ HWND          hWnd,
    _In_ UINT          Msg,
    _In_ WPARAM        wParam,
    _In_ LPARAM        lParam,
    _In_ SENDASYNCPROC lpCallBack,
    _In_ ULONG_PTR     dwData
);
```

参数介绍：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

IpResultCallBack：指向回收函数的指针，窗口程序处理完消息后调用该回调函数。参见 SendAsyncProc 可得到合适的回调函数的信息。如果 hwnd 为 HWND_BROADCAST，系统为每个顶层窗口调用一次 SendASyncProc 回调函数。

dwData：一个应用程序定义的值，被传给由参数 IPResultCallBack 指向的回调函数。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，返回值是零。若想获得更多的错误信息，请调用 GetLastError 函数

备注：

如果发送一个低于 WM_USER 范围的消息给异步消息函数（PostMessage，SendNotifyMesssge；SendMessageCallback），消息参数不能包含指针。否则，操作将会失败。函数将在接收线程处理消息之前返回，发送者将在内存被使用之前释放。

需要以 HWND_BROADCAST 方式通信的应用程序应当用函数 RegisterWindwosMessage 来获得应用程序间通信的独特的消息。

此回调函数仅当调用 SendMessagecallback 的线程调用 GetMessage，PeekMessage 或 WaitMessage 时调用。

User32!SendNotifyMessage

功能：将指定的消息发送到一个窗口。如果该窗口是由调用线程创建的；此函数为该窗口调用窗口程序，并等待窗口程序处理完消息后再返回。如果该窗口是由不同的线程创建的，此函数将消息传给该窗口程序，并立即返回，不等待窗口程序处理完消息。

函数原型：

```
BOOL SendNotifyMessage（
    HWND hWnd，
    UINT Msg，
    WPARAM wParam，
    LPARAM IParam
）；
```

参数介绍：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口、被覆盖的窗口和弹出式窗口，但消息不被发送到子窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

IParam：指定附加的消息指定信息。

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零。

User32!SendMessageTimeout

功能：将指定的消息发送到一个或多个窗口。此函数为指定的窗口调用窗口程序，并且，如果指定的窗口属于不同的线程，直到窗口程序处理完消息或指定的超时周期结束函数才返回。如果接收消息的窗口和当前线程属于同一个队列，窗口程序立即调用，超时值无用。

函数原型：

```
LRESULT SendMessageTimeout（
    HWND hwnd，
    UINT Msg，
    WPARAM wParam，
    LPARAM lParam，
    UINT fuFlags，
    UINT uTimeout，
    LPDWORD lpdwResultult
）；
```

参数介绍：

hWnd：其窗口程序将接收消息的窗口的句柄。如果此参数为 HWND_BROADCAST，则消息将被发送到系统中所有顶层窗口，包括无效或不可见的非自身拥有的窗口。

Msg：指定被发送的消息。

wParam：指定附加的消息指定信息。

lParam：指定附加的消息指定信息。

fuFlags；指定如何发送消息。此参数可为下列值的组合：

SMTO_ABORTIFHUNG：如果接收进程处于“hung”状态，不等待超时周期结束就返回。

SMTO_BLOCK：阻止调用线程处理其他任何请求，直到函数返回。

SMTO_NORMAL：调用线程等待函数返回时，不被阻止处理其他请求。

SMTO_NOTIMEOUTIFNOTHUNG：Windows 95 及更高版本：如果接收线程没被挂起，当超时周期结束时不返回。

uTimeout：为超时周期指定以毫秒为单位的持续时间。如果该消息是一个广播消息，每个窗口可使用全超时周期。例如，如果指定 5 秒的超时周期，有 3 个顶层窗回未能处理消息，可以有最多 15 秒的延迟。

lpdwResult：指定消息处理的结果，依赖于所发送的消息。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，或超时，返回值是零

User32!PostMessage

功能：将一条消息放入到消息队列中。消息队列里的消息通过调用 GetMessage 和 PeekMessage 取得。

函数原型：

```
BOOL WINAPI PostMessage(
    HWND hWnd,
    UINT Msg,
```

```
    WPARAM wParam,
    LPARAM lParam
);
```

参数介绍：

hWnd：其窗口程序接收消息的窗口的句柄。可取有特定含义的两个值：

HWND_BROADCAST：消息被寄送到系统的所有顶层窗口，包括无效或不可见的非自身拥有的窗口、 被覆盖的窗口和弹出式窗口。消息不被寄送到子窗口

NULL：此函数的操作和调用参数 dwThread 设置为当前线程的标识符 PostThreadMessage 函数一样

Msg：指定被寄送的消息。

wParam：指定附加的消息特定的信息。

LParam：指定附加的消息特定的信息。

返回值：如果函数调用成功，返回非零，否则函数调用返回值为零

User32!PostThreadMessage

功能：将一个消息放入（寄送）到指定线程的消息队列里，不等待线程处理消息就返回。

函数原型：

```
BOOLPostThreadMessage(
    DWORDidThread,
    UINTMsg,
    WPARAMwParam,
    LPARAMlParam
);
```

参数介绍：

idThread

其消息将被寄送的线程的线程标识符。如果线程没有消息队列，此函数将失败。当线程第一次调用一个 Win 32 USER 或 GDI 函数时，系统创建线程的消息队列。要得到更多的信息，参见备注。

Msg

指定将被寄送的消息的类型。

wParam

指定附加的消息特定信息。

lParam

指定附加的消息特定信息。

返回值：如果函数调用成功，返回非零值。如果函数调用失败，返回值是零

User32!PostQuitMessage

功能：向系统表明有个线程有终止请求

函数原型：

void PostQuitMessage（int nExitCode）

参数介绍：

nExitCode：指定应用程序退出代码。此值被用作消息 WM_QUIT 的

返回值：无

User32!BroadcastSystemMessage

功能：该函数发送消息给指定的接受者。接受者可以是一个应用程序、安装驱动器、网络驱动器、系统级设备驱动器或这些系统组件的组合。

函数原型：

long BroadcastSystemMessage（DWORD dwFlags，LPDWORD lpdwRecipients,UINT UiMessage，WPARAMwParam,LPARAM lParam）；

参数介绍：

dwFlags：选项标志。可取下列值的组合：

BSF_FLUSHDISK：接受者处理消息之后清洗磁盘。

BSF_FORCEIFHUNG：继续广播消息，即使超时周期结束或一个接受者已挂起。BSF_IGNORECURRENTTASK：不发送消息给属于当前任务的窗口。这样，应用程序就不会接收自己的消息。

BSF_NOHANG：强制挂起的应用程序超时。如果一个接受者超时，不再继续广播消息。BSF_NOTIMEOUTIFNOTHUNG：只要接受者没挂起，一直等待对消息的响应。不会出现超时。BSF_POSTMESSAGE：寄送消息。不能和 BSF_QUERY 组合使用。

BSF_QUERY：每次发送消息给一个接受者，只有当前接受者返回 TRUE 后，才能发送给下一个接受者。lpdwRecipients：指向变量的指针，该变量含有和接收消息接受者的信息。此变量可为下列值的组合： BSM_ALLCOMPONENTS：广播到所有的系统组件。

BSM_ALLDESKTOPS：Windows NT 下，广播到所有的桌面。要求 SE_TCB_NAME 特权。

BSM_APPLICATIONS：广播到应用程序。

BSM_INSTALLABLEDRIVERS：Windows 95 下，广播到安装驱动器。

BSM_INTDRIVER：Windows 95 下，广播到网络驱动器。

BSM_VXDS：Windows 95 下，广播到所有系统级设备驱动器。

当函数返回时，此变量接受上述值的组合，以确定真正接受消息的接受者。如果此参数为 NULL，则将消息广播到所有的组件。

uiMessage：系统消息标识符。

WParam：32 位消息特定值。

lParam：32 位消息特定值。

返回值：如果函数调用成功，返回值是正数。如果函数不能广播消息，返回值是 C1。如果参数 dwFlags 为 BSF_QUERY 且至少一个接受者返回 BROADCAST_QUERY_DENY 给相应的消息，返回值是零

User32!GetMessage

功能：从调用线程的消息队列里取得一个消息并将其放于指定的结构。此函数可取得与指定窗口联系的消息和由 PostThreadMessage 寄送的线程消息。

函数原型：

```
BOOL WINAPI GetMessage(
    _Out_   LPMSG lpMsg,
    _In_opt_ HWND  hWnd,
    _In_    UINT  wMsgFilterMin,
    _In_    UINT  wMsgFilterMax
);
```

参数介绍：

lpMsg：指向 MSG 结构的指针，该结构从线程的消息队列里接收消息信息。

hWnd：取得其消息的窗口的句柄。当其值取 NULL 时，GetMessage 为任何属于调用线程的窗口检索消息，线程消息通过 PostThreadMessage 寄送给调用线程。

wMsgFilterMin：指定被检索的最小消息值的整数。

wMsgFilterMax：指定被检索的最大消息值的整数。

返回值: 如果函数取得 WM_QUIT 之外的其他消息，返回非零值。如果函数取得 WM_QUIT 消息，返回值是零。如果出现了错误，返回值是 -1。

User32!PeekMessage

功能：为一个消息检查线程消息队列，并将该消息（如果存在）放于指定的结构

函数原型：

```
BOOL PeekMessage(
    LPMSG lpMsg,
    HWND hWnd,
    UINT wMSGfilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg
);
```

参数介绍：

lpMsg

接收消息信息的 MSG 结构指针。

hWnd

其消息被检查的窗口句柄。

wMsgFilterMin

指定被检查的消息范围里的第一个消息。

wMsgFilterMax

指定被检查的消息范围里的最后一个消息。

wRemoveMsg

确定消息如何被处理。

返回值：如果消息可得到，返回非零值；如果没有消息可得到，返回值是零

User32!WaitMessage

功能：当一个线程的消息队列中无其它消息时，该函数就将控制权交给另外的线程，同时将该线程挂起，直到一个新的消息被放入线程的消息队列之中才返回。

函数原型：

BOOL WaitMessage（VOID）

参数介绍：无

返回值：如果函数调用成功，返回非零值；如果函数调用失败，返回值是零

User32!DispatchMessage

功能：该函数分发一个消息给窗口程序。通常消息从 GetMessage 函数获得。消息被分发到回调函数（过程函数），作用是消息传递给操作系统，然后操作系统去调用我们的回调函数，也就是说我们在窗体的过程函数中处理消息。

函数原型：

LONG DispatchMessage（CONST MSG*lpmsg）；

参数介绍：

lpmsg：指向含有消息的 MSG 结构的指针。

返回值：返回值是窗口程序返回的值。尽管返回值的含义依赖于被调度的消息，但返回值通常被忽略。

备注：MSG 结构必须包含有效的消息值。如果参数 lpmsg 指向一个 WM_TIMER 消息，并且 WM_TIMER 消息的参数 lParam 不为 NULL，则调用 lParam 指向的函数，而不是调用窗口程序。

其他

advapi32!AdjustTokenPrivileges

功能：启用或禁止指定访问令牌的特权。

函数原型：

```
BOOL AdjustTokenPrivileges(
    HANDLE TokenHandle,
    BOOL DisableAllPrivileges,
    PTOKEN_PRIVILEGES NewState,
    DWORD BufferLength,
    PTOKEN_PRIVILEGES PreviousState,
    PDWORD ReturnLength
);
```

参数介绍：

TokenHandle, 包含特权的句柄

DisableAllPrivileges, 禁用所有权限标志

NewState, 新特权信息的指针 (结构体)

结构体说明如下：

```
typedef struct _TOKEN_PRIVILEGES
{
    ULONG PrivilegeCount; // 数组元素的个数
    LUID_AND_ATTRIBUTES Privileges[ANYSIZE_ARRAY]; // 数组 . 类型为 LUID_AND_ATTRIBUTES
} TOKEN_PRIVILEGES, *PTOKEN_PRIVILEGES;
typedef struct _LUID_AND_ATTRIBUTES
{ // luaa
    LUID Luid; // 标识了一个 LUID 值
    DWORD Attributes; // 标识了 LUID 属性 . 这个值包含了多达 32 位 (one-bit) 的标识 . 它的意思是取决于 LUID 的定义和使用
} LUID_AND_ATTRIBUTES;
```

BufferLength, 缓冲数据大小 , 以字节为单位的 PreviousState 的缓存区 (sizeof)

PreviousState, 接收被改变特权当前状态的 Buffer

ReturnLength 接收 PreviousState 缓存区要求的大小

如果这个函数成功 , 返回非 0. 失败返回 0. 为了确定这个函数是否修改了所有指定的特权 , 可以调用 GetLastError 函数 , 当这个函数返回下面的值之一时就代表函数成功 :

ERROR_SUCCESS: 这个函数修改了所有指定的特权。

ERROR_NOT_ALL_ASSIGNED: 这个令牌没有参数 NewState 里指定一个或多个的权限。（一个或多个没有修改成功）. 即使权限没有被修改。这个函数也可能成功（返回这个 error 值）表明 参数 PreviousState 被修改。

Gdi32!BitBlt

功能：对指定的源设备环境区域中的像素进行位块（bit_block）转换，以传送到目标设备环境。

函数原型：

```
BOOL BitBlt(
    int x,
    int y,
    int nWidth,
    int nHeight,
    CDC* pSrcDC,
    int xSrc,
    int ySrc,
    DWORD dwRop
);
```

参数介绍：

X：指定目标矩形的左上角的逻辑 x 坐标。

Y：指定目标矩形的左上角的逻辑 y 坐标。

nWidth：指定宽度 (以逻辑单位) 的目标矩形和源位图。

nHeight：指定高度 (以逻辑单位) 目标矩形和源位图。

pSrcDC：设置为标识设备上下文位图要复制的 CDC 对象的指针。 它必须是 NULL，如果 dwRop 指定不包括一个源的光栅操作。

xSrc：指定源位图的左上角的逻辑 x 坐标。

ySrc：指定源位图的左上角的逻辑 y 坐标。

dwRop：指定要执行的光栅操作。 光栅操作代码定义 GDI 如何组合在涉及一个当前画笔、一个可能的源位图和一个目标位图的输出操作的颜色

返回值：返回非 0 表示成功，0 表示失败

备注：用来将图形数据从一个设备复制到另一个设备。间谍软件有时候会使用这个函数来捕获屏幕。这个函数也经常被编译器作为共享代码而添加。

user32!CallNextHookEx

功能：可以将钩子信息传递到当前钩子链中的下一个子程，一个钩子程序可以调用这个函数之前或之后处理钩子信息。

函数原型：

```
LRESULT WINAPI CallNextHookEx(
    _In_opt_ HHOOK hhk,
    _In_ int nCode,
    _In_ WPARAM wParam,
    _In_ LPARAM lParam
);
```

参数介绍：

hhk[可选]

说明：当前钩子的句柄

nCode [in]

说明：钩子代码 ; 就是给下一个钩子要交待的

传递给当前 Hook 过程的代码。下一个钩子程序使用此代码，以确定如何处理钩的信息。

wParam[in]

说明：要传递的参数 ; 由钩子类型决定是什么参数

wParam 参数值传递给当前 Hook 过程。此参数的含义取决于当前的钩链与钩的类型。

lParam[in]

说明：要传递的参数 ; 由钩子类型决定是什么参数

lParam 的值传递给当前 Hook 过程。此参数的含义取决于当前的钩链与钩的类型。

返回值：返回这个值链中的下一个钩子程序。当前 Hook 过程也必须返回该值。返回值的含义取决于钩型。

备注：在由 SetWindowsHookEx 函数设置了挂钩时间的代码中使用。CallNextHookEx 函数会调用链上的下一个挂钩函数。分析调用 CallNextHookEx 的函数可以确定出 SetWindowsHookEx 设置挂钩的用意。

crypt32!CertOpenSystemStore

功能 : 用来访问本地系统中的证书。

函数原型：

```
HCERTSTORE WINAPI CertOpenSystemStore(
    HCRYPTPROV hProv,
    LPCTSTR szSubsystemProtocol
);
```

参数介绍：

hProv：CSP 句柄，NULL 为默认句柄，或者由 CryptAcquireContext 返回

szSubsystemProtocol：打开的系统存储区的名字。假如名字不为 CA，MY，ROOT，SPC 则新建一个证书存储区域，可以使用 CertEnumSystemStore 列出所有的已存在的系统存储区

返回值：NULL 表示失败，成功则返回证书句柄。

备注：用来访问在本地系统中的证书。

kernel32!CreateMutex

功能：用来创建一个有名或无名的互斥量对象

函数原型：

```
HANDLE CreateMutex(
    LPSECURITY_ATTRIBUTES lpMutexAttributes,        // 指向安全属性的指针
    BOOL bInitialOwner,                             // 初始化互斥对象的所有者
    LPCTSTR lpName                                  // 指向互斥对象名的指针
);
```

参数介绍：

lpMutexAttributes SECURITY_ATTRIBUTES，指定一个 SECURITY_ATTRIBUTES 结构，或传递零值（将参数声明为 ByVal As Long，并传递零值），表示使用不允许继承的默认描述符

bInitialOwner Long，如创建进程希望立即拥有互斥体，则设为 TRUE。一个互斥体同时只能由一个线程拥有

lpName String，指定互斥体对象的名字。用 vbNullString 创建一个未命名的互斥体对象。如已经存在拥有这个名字的一个事件，则打开现有的已命名互斥体。这个名字可能不与现有的事件、信号机、可等待计时器或文件映射相符

返回值：

Long，如执行成功，就返回互斥体对象的句柄；零表示出错。会设置 GetLastError。即使返回的是一个有效句柄，但倘若指定的名字已经存在，GetLastError 也会设为 ERROR_ALREADY_EXISTS

备注：创建一个互斥对象，可以被恶意代码用来确保在给定时刻只有一个实例在系统上运行。恶意代码经常使用固定名字为互斥对象命名，这样他们就可以成为一个很好的主机特征，来检测系统是否感染了这个恶意样本。

crypt32!CryptAcquireContext

功能：连接 CSP，获得指定 CSP 的密钥容器的句柄；

函数原型：

```
BOOL WINAPI CryptAcquireContext(
    __out   HCRYPTPROV *phProv,
    __in    LPCTSTR pszContainer,
    __in   LPCTSTR pszProvider,
    __in   DWORD dwProvType,
    __in    DWORD dwFlags
);
```

参数介绍：

*phProv, CSP 句柄指针

pszContainer, 密钥容器名称，指向密钥容器的字符串指针；如果 dwFlags 为 CRYPT_VERIFYCONTEXT，pszContainer 必须为 NULL

pszProvider, 指向 CSP 名称的字符串指针；为 NULL，表示使用默认的 CSP

dwProvType, CSP 类型

dwFlags 标志位

返回值：操作成功返回 TRUE，否则返回 FALSE。

备注：这经常是恶意代码用来初始化使用 Windows 加密库的第一个函数。还有跟多其他函数是和加密相关的，绝大多数的函数都以 Crypt 开头。

EnbaleExecuteProtectionSupport

备注：一个未经文档化的 API 函数，用来修改宿主上的数据执行保护（DEP）设置，是的系统更容易被攻击。

kernel32!FindResource

功能：确定指定模块中指定类型和名称的资源所在位置。

函数原型：

```
HRSRC WINAPI FindResource(
    _In_opt_    HMODULE hModule,
    _In_    LPCTSTR lpName,
    _In_    LPCTSTR lpType
);
```

参数介绍：

hModule：处理包含资源的可执行文件的模块。NULL 值则指定模块句柄指向操作系统通常情况下创建最近过程的相关位图文件。

lpName：指定资源名称。若想了解更多的信息，请参见注意部分。

lpType：指定资源类型。若想了解更多的信息，请参见注意部分。作为标准资源类型。这个参数的含义同 EnumResLangProc\lpType。

返回值：如果函数运行成功，那么返回值为指向被指定资源信息块的句柄。为了获得这些资源，将这个句柄传递给 LoadResource 函数。如果函数运行失败，则返回值为 NULL

备注：用来在可执行文件和装在 DLL 程序中寻找资源的函数。恶意代码有时会使用资源来存储字符串，配置信息或者其他恶意文件。如果看到使用了这个函数，需要进一步检查恶意代码 PE 头中的 .rsrc 节

user32!FindWindow

功能：检索处理顶级窗口的类名和窗口名称匹配指定的字符串

函数原型：

```
HWND FindWindow
(
    LPCSTR lpClassName,
    LPCSTR lpWindowName
);
```

参数介绍：

lpClassName

指向一个以 NULL 字符结尾的、用来指定类名的字符串或一个可以确定类名字符串的原子。如果这个参数是

一个原子，那么它必须是一个在调用此函数前已经通过 GlobalAddAtom 函数创建好的全局原子。这个原子（一个 16bit 的值），必须被放置在 lpClassName 的低位字节中，lpClassName 的高位字节置零。

如果该参数为 null 时，将会寻找任何与 lpWindowName 参数匹配的窗口。

lpWindowName

指向一个以 NULL 字符结尾的、用来指定窗口名（即窗口标题）的字符串。如果此参数为 NULL，则匹配所有窗口名。

返回值：

如果函数执行成功，则返回值是拥有指定窗口类名或窗口名的窗口的句柄。

如果函数执行失败，则返回值为 NULL

备注：在桌面上搜索打开窗口的函数，有时这个函数会在反调试技术中使用，来搜索 OllyDbg 工具的窗口。

user32!GetAsyncKeyState

功能：用来判断函数调用时指定虚拟键的状态

函数原型：

SHORT GetAsyncKeyState(int nVirtKey);

参数介绍：nVirtKey: 指定 256 个可能的虚拟键盘值中的一个 .

返回值：指定虚拟键瞬时的状态值，它有四种返回值：

0--- 键当前未处于按下状态，而且自上次调用 GetAsyncKeyState 后改键也未被按过；

1--- 键当前未处于按下状态，但在此之前（自上次调用 GetAsyncKeyState 后）键曾经被按过；

-32768（即 16 进制数 &H8000） --- 键当前处于按下状态，但在此之前（自上次调用 GetAsyncKeyState 后）键未被按过；

-32767（即 16 进制数 &H8001） --- 键当前处于按下状态，而且在此之前（自上次调用 GetAsyncKeyState 后）键也曾经被按过。

备注：用来确定一个特定键是否被输入。恶意代码有时会使用这个函数来实现一个击键记录器

user32!GetDC

功能：该函数检索一指定窗口的客户区域或整个屏幕的显示设备上下文环境的句柄，以后可以在 GDI 函数中使用该句柄来在设备上下文环境中绘图。

函数原型：HDC GetDC(HWND hWnd)；

参数介绍：hWnd：设备上下文环境被检索的窗口的句柄，如果该值为 NULL，GetDC 则检索整个屏幕的设备上下文环境。

返回值：如果成功，返回指定窗口客户区的设备上下文环境；如果失败，返回值为 Null。

备注：返回一个窗口或者是整个屏幕的设备上下文句柄。间谍软件经常使用这个函数来抓取桌面截屏

user32!GetForegroundWindow

功能：获取一个前台窗口的句柄

函数原型：HWND GetForegroundWindow(void);

参数介绍：无

返回值：返回值是一个前台窗口的句柄。在某些情况下，如一个窗口失去激活时，前台窗口可以是 NULL。

备注：返回桌面目前正在处于前端的窗口句柄，击键记录器普遍使用这个函数，来确定用户正在往哪个窗口输入

user32!GetKeyState

功能：该函数检取指定虚拟键的状态。该状态指定此键是 UP 状态，DOWN 状态，还是被触发的

函数原型：SHORT GetKeyState（int nVirtKey）；

参数介绍：nVrtKey：定义一虚拟键。若要求的虚拟键是字母或数字（A ～ Z，a ～ z 或 0 ～ 9），nVirtKey 必须被置为相应字符的 ASCII 码值，对于其他的键，nVirtKey 必须是一虚拟键码。若使用非英语键盘布局，则取值在 ASCIIa ～ z 和 0 ～ 9 的虚拟键被用于定义绝大多数的字符键

返回值：返回虚拟键的状态，若高序位为 1，则键处于 DOWN 状态，否则为 UP 状态。

若低序位为 1，则键被触发

备注：被击键记录器使用，来获取键盘上一个特定键的状态。

kernel32!GetSystemDefaultLangId

功能：返回系统默认的语言设置

函数原型：LANGID GetSystemDefaultLangID(void);

参数介绍：无

返回值：返回本地系统的语言标识符

备注：返回系统默认语言设置的函数。这可以用来定制显示与文件名，作为对感染主机摘要信息的获取，这个函数也会被一些由“爱国主义”所驱动的恶意代码使用，从而对一些特定区域的系统进行感染。

kernel32!GetTickCount

功能：返回从操作系统启动所经过的毫秒数

函数原型：DWORD GetTickCount(void);

参数介绍：无

返回值：返回从操作系统启动到当前所经过的毫秒数

备注：返回从启动后到当前时间的微秒数。这个函数有时在反调试技术中被使用来获取时间信息。此函数也经常有编译器添加并包含在许多可执行程序中，因此简单的在导入函数列表中看到这个函数只能提供少量的线索信息

kernel32!GetVersionEx

功能：

函数原型：

```
BOOL WINAPI GetVersionEx(
    _Inout_ LPOSVERSIONINFO lpVersionInfo
);
```

参数介绍：lpVersionInfo：用于装载版本信息的结构。在正式调用函数之前，必须先将这个结构的 dwOSVersionInfoSize 字段设为结构的大小（148）

结构体声明如下：

```
typedef struct _OSVERSIONINFO{
    DWORD dwOSVersionInfoSize;          // 指定该数据结构的字节大小
    DWORD dwMajorVersion;               // 操作系统的主版本号
    DWORD dwMinorVersion;               // 操作系统的副版本号
    DWORD dwBuildNumber;                // 操作系统的创建号
    DWORD dwPlatformId;                 // 操作系统 ID 号
    TCHAR szCSDVersion[ 128 ];          // 关于操作系统的一些附加信息
} OSVERSIONINFO;
```

返回值：成功返回非 0，失败返回 0

备注：返回目前正在运行的 Windows 操作系统版本信息。可以被用来获取受害主机的摘要信息，或是在不同 Windows 版本中选择未经文档化结构的一些偏移地址。

kernel32!IsDebuggerPresent

功能：确定调用进程是否由用户模式的调试器调试

函数原型：BOOL WINAPI IsDebuggerPresent(void);

参数介绍：无

返回值：

如果当前进程运行在调试器的上下文，返回值为非零值。

如果当前进程没有运行在调试器的上下文，返回值是零。

备注：检查当前进程是否被调试，经常在反调试技术中使用，这个函数经常有编译器添加并包含在许多可执行程序中，因此简单的在导入函数列表中看到这个函数，只能提供少量的线索信息。

kernel32!LoadLibrary

功能：装载一个 dll 程序到进程中

函数原型：HMODULE WINAPI LoadLibrary(_In_ LPCTSTR lpFileName);

参数介绍：lpLibFileName 指定要载入的动态链接库的名称。采用与 CreateProcess 函数的 lpCommandLine

参数指定的同样的搜索顺序

返回值：成功则返回库模块的句柄，零表示失败

备注：装载一个 dll 程序到进程中，而这个程序在程序启动时还没有被装载。几乎每一个 Win32 程序都会导入这个函数。

kernel32!LoadResource

功能：该函数装载指定资源到全局存储器。

函数原型：HGLOBAL LoadResource（HMODULE hModule，HRSRC hReslnfo）；

参数介绍：

hModule：处理包合资源的可执行文件的模块句柄。若 hModule 为 NULL，系统从当前过程中的模块中装载资源。

hReslnfo：将被装载资源的句柄。它必须由函数 FindResource 或 FindResourceEx 创建。

返回值：如果函数运行成功，返回值是相关资源的数据的句柄。如果函数运行失败，返回值为 NULL

备注：从 PE 文件中装载资源进入到内存中。恶意代码有时候会使用资源来存储字符串，配置信息或者其他恶意软件。

user32!MapVirtualKey

功能：该函数将一虚拟键码翻译（映射）成一扫描码或一字符值，或者将一扫描码翻译成一虚拟键码

函数原型：UINT MapVirtualKey（UINT uCode，UINT uMapType）

参数介绍：

uCode：定义一个键的扫描码或虚拟键码。该值如何解释依赖于 uMapType 参数的值。

uMapType：定义将要执行的翻译。该参数的值依赖于 uCode 参数的值。取值如下：

0：代表 uCode 是一虚拟键码且被翻译为一扫描码。若一虚拟键码不区分左右，则返回左键的扫描码。若未进行翻译，则函数返回 O。

1：代表 uCode 是一扫描码且被翻译为一虚拟键码，且此虚拟键码不区分左右。若未进行翻译，则函数返回 0。

2：代表 uCode 为一虚拟键码且被翻译为一未被移位的字符值存放于返回值的低序字中。死键（发音符号）则通过设置返回值的最高位来表示。若未进行翻译，则函数返回 0。

3：代表 uCode 为一扫描码且被翻译为区分左右键的一虚拟键码。若未进行翻译，则函数返回 0。

返回值: 返回值可以是一扫描码，或一虚拟键码，或一字符值，这完全依赖于不同的 uCode 和 uMapType 的值。若未进行翻译，则函数返回 0。

备注：将一个虚拟键值代码转化成字符值，经常被击键记录器恶意代码所使用

kernel32!Module32First

功能：检索与进程相关联的第一个模块的信息

函数原型：

```
BOOL WINAPI Module32First(
    HANDLE hSnapshot,
    LPMODULEENTRY32 lpme
);
```

参数介绍：

hSnapshot

调用 CreateToolhelp32Snapshot 函数返回的快照句柄

lpme

MODULEENTRY32 结构的指针。用来返回数据

结构体声明如下：

typedef struct tagMODULEENTRY32 {

DWORD dwSize;// 指定结构的长度，以字节为单位

DWORD th32ModuleID;// 此成员已经不再被使用，通常被设置为 1

DWORD th32ProcessID;// 正在检查的进程标识符

DWORD GlblcntUsage;// 全局模块的使用计数，即模块的总载入次数

DWORD ProccntUsage;// 全局模块的使用计数

BYTE *modBaseAddr;// 模块的基址，在其所属的进程范围内。

DWORD modBaseSize;// 模块的大小，单位字节

HMODULE hModule;// 所属进程的范围内，模块句柄

TCHAR szModule[MAX_PATH];// NULL 结尾的字符串，其中包含模块名。

TCHAR szExePath[MAX_PATH];// NULL 结尾的字符串，其中包含的位置，或模块的路径。

} MODULEENTRY32, *PMODULEENTRY32, *LPMODULEENTRY32;

返回值：成功返回 TRUE 失败返回 FALSE

备注：用来枚举装载到进程中的模块，注入器通常使用这个函数来确定注入代码的位置。

kernel32!Module32Next

功能：用来枚举装载到进程中的模块

函数原型：

```
BOOL WINAPI Module32Next(
    _In_  HANDLE        hSnapshot,
    _Out_ LPMODULEENTRY32 lpme
);
```

参数介绍：

hSnapshot

调用 CreateToolhelp32Snapshot 函数返回的快照句柄

lpme

MODULEENTRY32 结构的指针。用来返回数据

返回值：如果模块链中的下一个模块拷贝到了缓冲区中，就返回 TRUE，否则返回 FALSE。

备注：用来枚举装载到进程中的模块，注入器通常使用这个函数来确定注入代码的位置。

Netapi32!NetScheduleJobAdd

功能：

函数原型：

```
NET_API_STATUS NetScheduleJobAdd(
    _In_opt_ LPCWSTR Servername,
    _In_    LPBYTE  Buffer,
    _Out_   LPDWORD JobId
);
```

参数介绍：

Servername：函数要执行的远程服务的 DNS 或 NetBIOS 名字

Buffer：执行 AT_INFO 结构的指针，描述了要提交的任务

JobId：新提交的任务 ID

返回值：成功返回 NERR_Success，失败返回系统错误码

备注：提交一个计划安排，来让一个程序在某个特定日期时间运行。恶意代码可以使用这个函数来运行一个其他程序，作为一位恶意代码分析师，需要定位与分析会在未来某个时间运行的程序。

kernel32!OpenMutex

功能：为现有的一个已命名互斥体对象创建一个新句柄。

函数原型：

函数原型：

```
 HANDLE OpenMutex(
DWORD dwDesiredAccess,                  // access
    BOOL bInheritHandle,                // inheritance option
    LPCTSTR lpName                      // object name
);
```

参数介绍：

dwDesiredAccess：

MUTEX_ALL_ACCESS 请求对互斥体的完全访问

MUTEX_MODIFY_STATE 允许使用 ReleaseMutex 函数

SYNCHRONIZE 允许互斥体对象同步使用

bInheritHandle : 如希望子进程能够继承句柄，则为 TRUE

lpName ： 要打开对象的名字

返回值：如执行成功，返回对象的句柄；零表示失败

备注：打开一个双向互斥对象的句柄，可以被恶意代码用来确保在任意给定时间在系统上只能有一个运行实例。恶意代码通常使用固定名字来为互斥对象命名，因此可以用作很好的主机特征。

kernel32!OutputDebugString

功能：输出字符串到一个附加的调试器上。

函数原型：void WINAPI OutputDebugString(__in_opt LPCTSTR lpOutputString);

参数介绍：lpOutputString：需要输出的字符串

返回值：无

备注：可以被使用在反调试技术上。

user32!SetWindowsHookEx

功能：设置一个挂钩函数，使其在某个特定时间触发时被调用。

函数原型：

```
HHOOK WINAPI SetWindowsHookEx(
    __in int idHook,                \\ 钩子类型
    __in HOOKPROC lpfn,             \\ 回调函数地址
    __in HINSTANCE hMod,            \\ 实例句柄
    __in DWORD dwThreadId);         \\ 线程 ID
```

参数介绍：

idHook：钩子类型

lpfn：回调函数地址

hMod：实例句柄

dwThreadId：线程 ID

返回值：若此函数执行成功 , 则返回值就是该挂钩处理过程的句柄 ; 若此函数执行失败 , 则返回值为 NULL

备注：此函数普遍被击键记录器和间谍软件使用，这个函数也提供了一种轻易的方法，将一个 DLL 程序装载入系统的所有 GUI 进程中。这个函数有时也会被编译器添加。

《2017 绿盟科技恶意样本分析手册 -API 函数篇》
由如下部门撰写

- 绿盟科技安全能力中心（SAC）

如需了解更多，请联系：

官方网站

技术博客

微信公众号

特别声明

为避免客户数据泄露，所有数据在进行分析前都已经匿名化处理，不会在中间环节出现泄露，任何与客户有关的具体信息，均不会出现在本报告中。

版权声明

THE EXPERT
BEHIND GIANTS
巨人背后的专家

多年以来，绿盟科技致力于安全攻防的研究，
为政府、运营商、金融、能源、互联网以及教育、医疗等行业用户，提供
具有核心竞争力的安全产品及解决方案，帮助客户实现业务的安全顺畅运行。
在这些巨人的背后，他们是备受信赖的专家。

www.nsfocus.com