

1 Windows系统基础原理

- 1.1 Windows的系统架构
- 1.2 Windows的管理机制
- 1.3 Windows的进程和线程
- 1.4 Windows的内存管理
- 1.5 Windows的输入输出系统和驱动
- 1.6 Windows的文件系统
- 1.7 Windows的网络机制
- 1.8 Windows的域和活动目录

2 Windows木马后门检测

- 2.1 Windows木马后门的技术
 - 2.1.1 Windows木马后门的植入技术
 - 2.1.2 Windows木马后门的开发技术
 - 2.1.2.1 管控功能实现技术
 - 2.1.2.2 自启动技术
 - 2.1.2.3 用户态的进程隐藏技术
 - 2.1.2.3.1 基于DLL插入的进程隐藏
 - 2.1.2.3.2 进程内存替换
 - 2.1.2.3.3 基于SvcHost服务的进程隐藏
 - 2.1.2.4 数据穿透和躲避技术
 - 2.1.2.5 内核态的隐藏技术(Rootkit)
 - 2.1.2.6 磁盘启动级的隐藏技术(Bootkit)
 - 2.1.3 Windows木马后门的免杀技术
- 2.2 Windows木马后门的检测分析
 - 2.2.1 Windows木马后门的检测技术
 - 2.2.1.1 基于自启动信息的检测
 - 2.2.1.2 基于进程信息的检测
 - 2.2.1.3 基于数据传输的检测
 - 2.2.1.4 Rootkit/Bootkit的检测
 - 2.2.2 Windows木马后门的分析技术
 - 2.2.2.1 木马后门的动态分析
 - 2.2.2.2 木马后门的静态分析

3 Windows系统安全概述

- 3.1 Windows系统安全性的历史与发展
- 3.2 Windows的基础安全防护机制

4 Windows核心安全机制

- 4.1 Windows系统的安全等级
- 4.2 Windows系统的安全组件
- 4.3 Windows的身份认证
 - 4.3.1 Windows账户的标识
 - 4.3.2 Windows的身份认证登录
 - 4.3.3 安全账户管理器(SAM)及数据库
 - 4.3.4 Windows身份验证协议
- 4.4 Windows的访问控制
 - 4.4.1 主体的安全访问令牌
 - 4.4.2 客体的安全描述
 - 4.4.3 用户权限控制
 - 4.4.4 应用程序控制策略

4.5 Windows的安全审核

5 Windows文件数据安全

5.1 NTFS文件系统简介

5.2 文件访问控制

5.2.1 NTFS文件访问权限

5.2.2 共享文件访问权限

5.3 文件加密

5.3.1 EFS文件加密系统

5.3.2 Bitlocker磁盘加密

5.4 文件数据安全

5.4.1 文件数据的备份和还原

5.4.2 文件的彻底删除和反删除

5.5 磁盘配额

6 Windows安全策略配置

6.1 本地安全策略

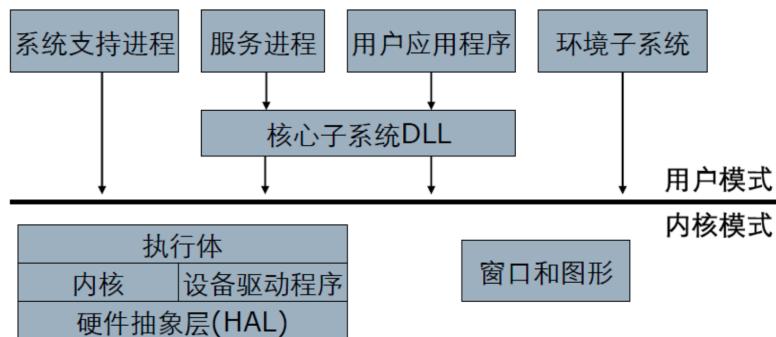
1 Windows系统基础原理

9x内核：无安全防护机制

NT内核：

- Windows 2000 (NT 5.0)：引入活动目录
- Windows Vista (NT 6.0)：引入UAC，即用户账户控制，不友好
- Windows 7 (NT 6.1)：引入白名单，降低安全，但体验好

1.1 Windows的系统架构



内核模式和用户模式：

- **内核模式：**
 - Ring 0
 - 可访问所有内存空间
 - 可直接操纵硬件
- **用户模式：**
 - Ring 3
 - 无法访问系统空间的内存页面

- 无法直接操纵硬件
 - 蓝屏：系统空间出问题
-

用户模式下的基本模块：

- 系统支持进程：Windows开机自动启动的系统内建服务进程，如winlogon
 - 服务进程：通过Windows的服务管理机制所启动的一系列系统及网络服务，如自动更新服务
 - 恶意代码很喜欢
 - 用户应用软件：在用户态执行的各类用户应用软件
 - **核心子系统DLL**：kernel32.dll、user32.dll、gdi32.dll、advapi32.dll等动态链接库文件，作为用户态代码和系统内核的交互接口，将用户态代码调用的API函数映射到相应的一个或多个Windows内部的系统服务调用
 - 易挂钩(防御、监视)，也易绕过
 - 只是传递者
 - 利于接口统一化，文档MSDN，任意Windows版本一样
 - 开发者的底层，安全人员的顶层
 - 环境子系统服务进程(不重要)：为操作系统运行环境提供支持的服务进程
-

内核模式下的基本模块：

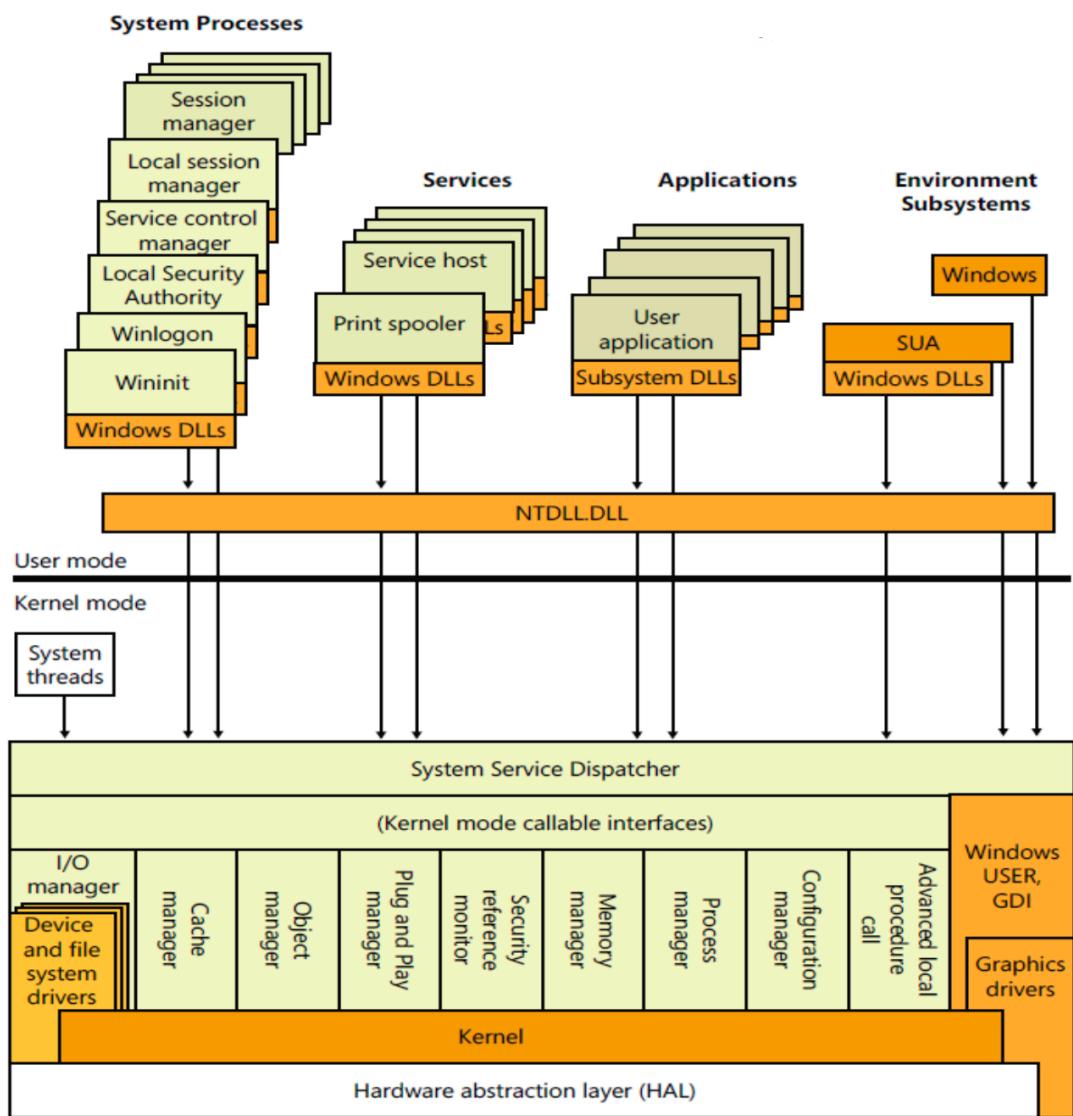
- **Windows执行体**(不重要)：包含了基本的操作系统服务，比如进程与线程管理、内存管理、I/O管理、网络和进程间通信以及安全服务等
 - **Windows内核体**(不重要)：实现了底层的操作系统功能，如线程调度、中断和异常调度、多处理器同步等，还提供了一组例程和基本对象，供其余的执行体来使用
 - **设备驱动程序**：既包括硬件设备驱动程序(将用户I/O函数调用转换为特定的硬件设备I/O请求)，也包括非硬件设备驱动程序(如文件系统和网络驱动程序)
 - Windows官方开发 + 第三方开发
 - 正常情况下恶意代码进入内核态的唯一途径(特殊情况：利用其它漏洞)
 - **硬件抽象层**(不重要)：将内核、驱动程序和执行体其余部分与特定于平台的硬件差异隔离开来
 - **窗口和图形系统**：实现了图形用户界面(GUI)函数(通常称为Windows用户和GDI功能)，包括窗口的处理、用户界面控制以及绘制等
 - 放在内核态，减少模式切换，流畅，带来了很多安全问题
-

Windows核心系统文件：

文件名	说明
Ntoskrnl.exe	实现了执行体、内核体，真正核心文件
Ntkrnlpa.exe	32位单处理器
Hal.dll	硬件抽象层
Win32k.sys	窗口和图形
Ntdll.dll	执行功能的内部支持函数和系统服务功能
Kernel32.dll、User32.dll、Gdi32.dll、Advapi32.dll	核心子系统DLL，64位也叫xxxxx32.dll

- 任务管理器中不可能有内核态进程，凡是任务管理器中带有kernel的都是假的
- 冰河：winkernel.exe，伪装
 - 有独立进程，现在的木马一般没有

详细图：



Hardware interfaces (buses, I/O devices, interrupts, interval timers, DMA, memory cache control, etc.)

- Ntdll.dll: 原始API, 无官方文档化, 不保证不同版本接口一致
 - 被若干操作系统重要的进程所使用 (如smss、csrss等)
 - 几个示例函数: NtCreateFile、NtReadFile、NtOpenProcess
 - System Service Dispatcher: 系统服务派遣, 由派遣表分发, 主动防御软件一般在此挂钩
 - 内核部分: 浅色执行体, 下面深色内核体, 左边驱动, 右边图形
 - 卡巴斯基: 所有都挂钩, 同步
 - 360: 异步, 回调机制
-

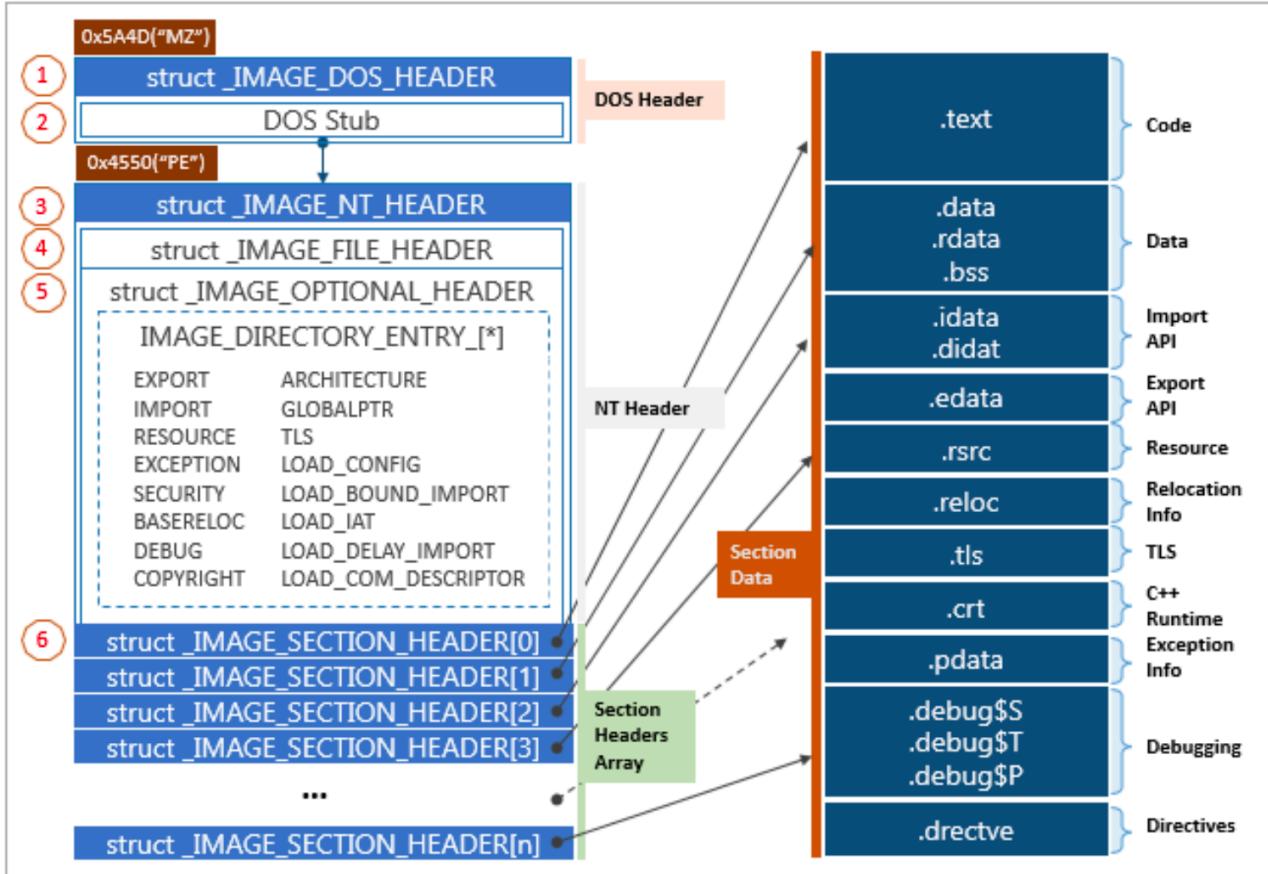
重要的系统进程:

- Smss.exe: 会话(Session)管理器, 系统启动时第一个运行的进程
 - 系统进程、不同用户进程都在不同会话中, 相互隔离
- Csrss.exe (不重要): Windows子系统进程, 即客户端/服务器运行进程
- Winlogon.exe: 处理交互式登录
- Services.exe: 服务控制管理器, 负责启动、停止、暂停和恢复服务
- Svchost.exe (重要): 共享进程服务的宿主进程
 - 有很多该进程
 - 同组服务共享一个进程
 - 很多服务程序寄生于此进程, 是一个容器, 藏污纳垢好地方
- Lsass.exe (非常重要): 本地安全授权子系统, 验证用户登录、授权和审计
 - UAC: 主动降权的概念, 如果你是管理员, 但UAC会对你降权, 某些要以管理员身份运行, 必须要提权

Windows的应用程序:

- EXE: 可执行程序, 包括DOS可执行程序和32位/64位的PE格式可执行程序
- DLL: 动态链接库程序; OCX文件, 即ActiveX控件, 本质上还是DLL
 - 相比于EXE, 无可执行的入口点函数
- SYS: 驱动程序, 加载到内核态
- OBJ: 编译器创建的对象文件, 作为链接器的一个输入
- Windows的可执行文件格式为PE(Portable Executable), 也就是可移植执行的意思
- PE和ELF文件格式同根同源, 都是从COFF发展而来
- PE文件被装载时直接映射到进程的虚拟空间中运行, 它是进程虚拟空间的映像, 因此也被称为映像文件
- PE相比COFF, 多了两个主要的变化:
 - PE的开始部分多了 IMAGE_DOS_HEADER 和 DOS_Stub(桩代码)
 - COFF中的 IMAGE_FILE_HEADER 被扩展成为 MAGE_NT_HEADER

PE Format

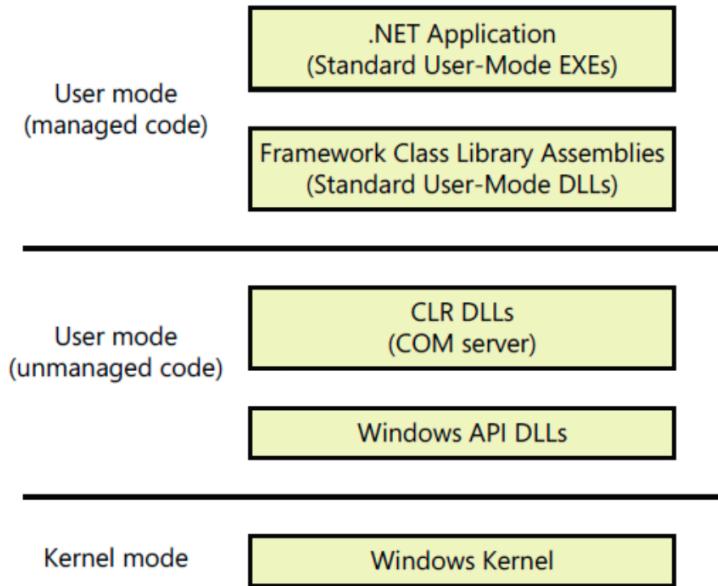


- “MZ”与“PE”
- 对数据的压缩加密：打包
- 对可执行代码的压缩加密：加壳
 - upx软件
 - 上面的字段会不一样
 - 难以静态分析，要设法脱壳

Windows的 API (应用编程接口) 函数：

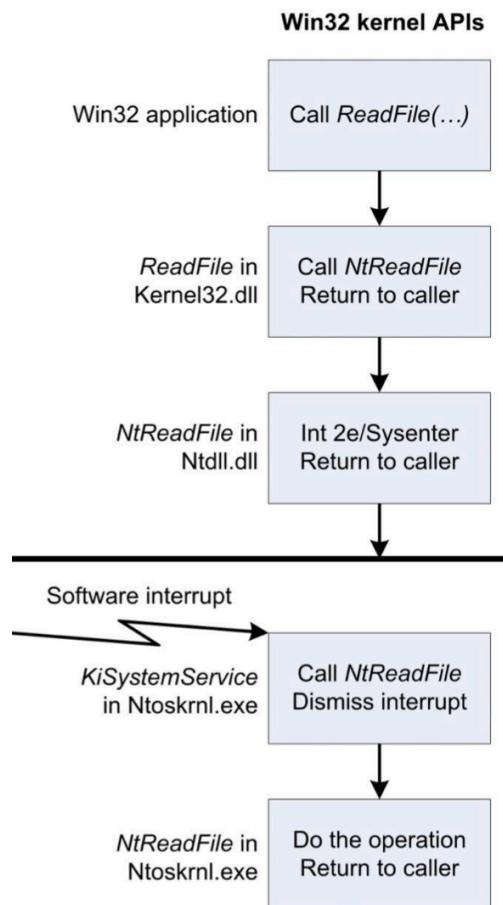
- 操作系统用户模式下的接口
 - 被若干个DLL文件导出：kernel32.dll、user32.dll、gdi32.dll
 - 即核心子系统DLL
 - 文档化的函数

.NET Framework



- 微软的.NET框架包含一个被称之为FCL的类库，一个“公共语言运行时”(CLR)，提供受管理的代码执行环境，包含实时编译、类型检查、垃圾回收、代码访问安全等特性
- CLR作为一个传统的COM服务器形式实现，代码依赖于标准的用户模式Windows DLL提供的不受管理的API函数，如kernel32.dll等
- .NET框架没有任何代码运行在内核模式下

系统服务派遣 (System Service Dispatching):



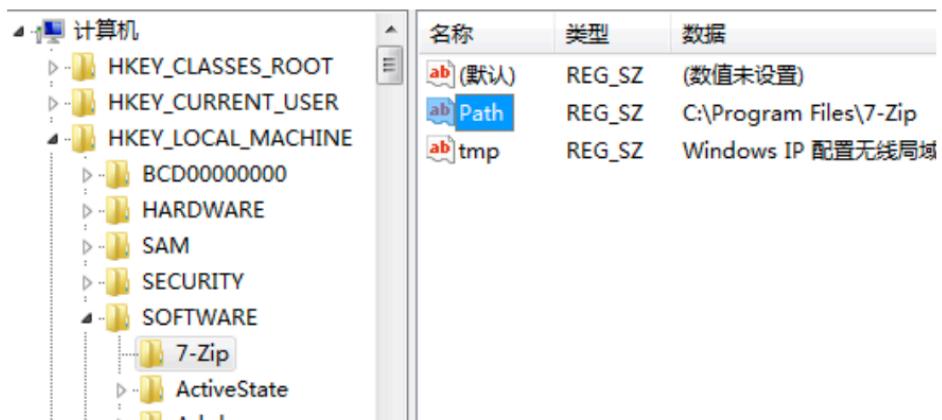
- 应用程序调用核心子系统导出的API: `Kernel32.dll.ReadFile(...)`

- 核心子系统调用Ntdll.dll导出的API： Ntdll.dll.NtReadFile(...)
- 进入内核态，通过系统服务派遣表 (SSDT)，找到函数Ntoskrnl.exe.NtReadFile(...)
- SSDT：
 - 从用户态到内核态后依靠此表寻找相应的函数
 - 攻防主战场

1.2 Windows的管理机制

注册表：

- 包含操作系统和其它软件的所有设置和配置相关数据的目录
- 注册表的逻辑结构类似于磁盘上的文件系统
 - 注册表包含了键 (key) 和值 (value)
 - 键类似文件系统中的目录，而值就像文件，键可以包含子键 (subkey) 和值
 - 值中存储着配置数据 (对应于文件内容)，数据有多种类型
 - 最顶层的键称为根键 (root key)
- 在内存中包含有“易变”的数据，比如当前的计算机和用户配置



根键 (Root Key)：

根键	缩写	描述	链接
HKEY_CURRENT_USER	HKCU	当前登录用户，登录生效，当前用户即可修改	HKU子键，对应于当前登录用户，链接，相当于文件的快捷方式
HKEY_USERS	HKU	系统所有用户，SID、安全身份标识符	不是链接
HKEY_CLASSES_ROOT	HKCR	注册信息、文件关联信息	不是直接的链接
HKEY_LOCAL_MACHINE	HKLM	计算机级别的全局配置，随计算机启动生效，管理员及以上具有修改权限	不是链接
HKEY_CURRENT_CONFIG	HKCC	当前硬件信息	链接

- 不可见的用户：SYSTEM
- 权限最高的用户：TrustedInstaller
- HKCU、HKLM：配置生效的时间点不同，可修改的权限不同

注册表的Hive文件：

- 注册表其实是以磁盘上的Hive (储巢) 文件进行存储的，大多数位于“%SystemRoot%\System32\Config”目录中
- “HKLM\SYSTEM\CurrentControlSet\Control\Hivelist” 中列出了注册表键和Hive文件的对应关系

服务：

- 服务程序是后台运行的进程，常用来执行特定的任务，不需要和用户进行交互
 - 自动更新服务、后台智能传输服务、事件日志服务等
- 服务程序受 Service Control Manager (SCM，即services.exe进程) 所控制
- 服务程序的配置数据位于“HKLM\System\CurrentControlSet\Services”
- 服务程序的种类 (Type)：
 - 内核驱动服务——内核态

Type	SERVICE_KERNEL_DRIVER (1)	设备驱动程序
	SERVICE_FILE_SYSTEM_DRIVE R (2)	内核模式的文件系统驱动程序
	SERVICE_ADAPTER(4)	已废弃
	SERVICE_RECOGNIZER_DRIVE R (8)	文件系统识别器驱动程序

- 独立进程服务——用户态
- 共享进程服务——用户态

Type	SERVICE_WIN32_OWN_PROCESS (16)	该服务运行在一个只能容纳一个服务的进程中
	SERVICE_WIN32_SHARE_PROCESS (32)	该服务运行在一个可容纳多个服务的进程中
	SERVICE_INTERACTIVE_PROCESS (256)	允许该服务在控制台上显示窗口，并且接收用户的输入，但是仅限于在控制台会话(0)上，防止与其他会话上的用户/控制台应用程序进行交互

- 服务的启动类型 (Start) :

- 自动 (Auto)
- 手动 (Demand)
- 禁用 (Disable)

Start	SERVICE_BOOT_START (0)	Winload预先加载该驱动程序，所以在引导过程中该驱动程序一直待在内存中。这些驱动程序恰好在SERVICE_SYSTEM_START驱动程序之前被初始化
	SERVICE_SYSTEM_START (1)	在内核初始化过程中，在SERVICE_BOOT_START驱动程序已初始化之后，该驱动程序被加载到内存中，并进行初始化
	SERVICE_AUTO_START (2)	在SCM进程 (Services.exe) 启动以后SCM启动该驱动程序或者服务
	SERVICE_DEMAND_START (3)	SCM根据需要启动该驱动程序或者服务
	SERVICE_DISABLED (4)	驱动程序或者服务不加载到内存中，也不初始化

- 前两个内核态 (先启动、后启动)，后三个用户态 (自动、手动、禁用)

- 服务的基本信息

- 显示名称 (DisplayName)

DisplayName	服务的名称	此服务应用在显示服务时使用该名称。如果没有指定名称的话，该服务的注册表键的名称变成该服务的名称
-------------	-------	---

- 服务描述 (Description)

Description	服务的说明	该服务的说明，最多可达32767个字节
-------------	-------	---------------------

- 可执行文件路径 (ImagePath)

■ 服务 (用户态) 和驱动 (内核态) 都是服务

ImagePath	服务或驱动程序可执行文件的路径	如果没有指定ImagePath的话，I/O管理器在%SystemRoot%\System32\ Drivers中寻找驱动程序。对于Windows服务，这是必需的
-----------	-----------------	--

- 登录身份账户 (ObjectName)

ObjectName 通常是 LocalSystem，但也可以是一个账户名称，比如.\Administrator

指定了该服务将在哪个账户下运行。如果没有指定 ObjectName 的话，则使用 LocalSystem 账户。该参数不适用于设备驱动程序

- beep.sys、null.sys：内核，自启动，没啥用，易被替换
- 停某一服务、替换(比较容易)、启动，即可进入内核态，不被主动防御软件发现
- 驱动程序大本营：C:\windows\System32\drivers

1.3 Windows的进程和线程

万物皆对象

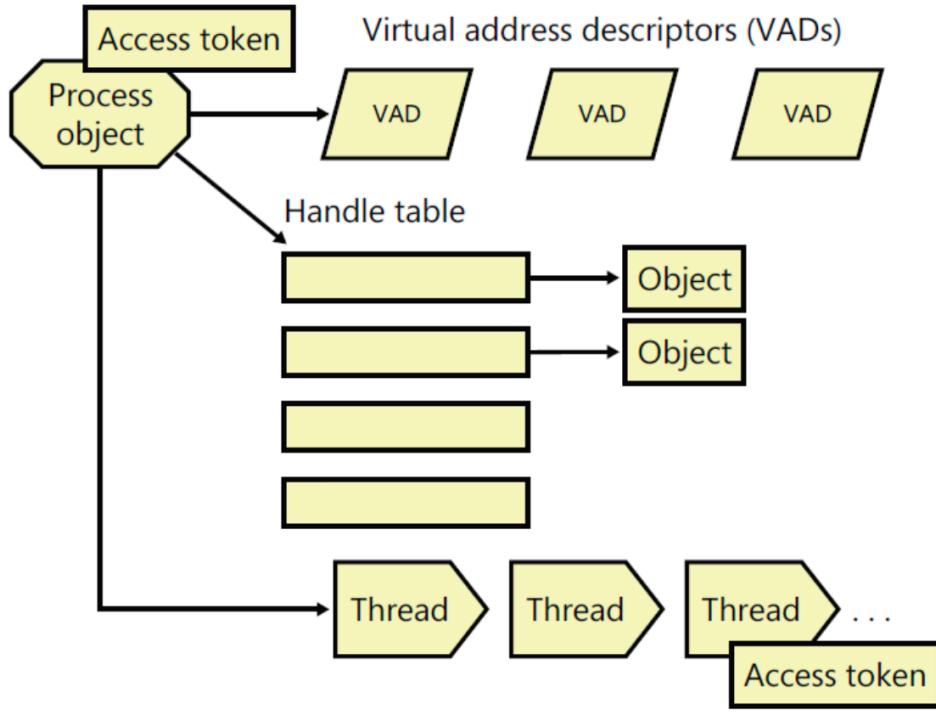
进程：

- 进程是一个应用程序运行的实例
- 进程包含以下一些基本组件：
 - 私有虚拟地址
 - 可执行体程序
 - 被操作系统分配的一份资源句柄(指向对象的指针)列表
 - 访问控制令牌(Token)，用以唯一的标识所有者及其所属组以及和该进程相关联的特权(Privilege)信息
 - 特权：做某一件事的能力，如关机、安装驱动、调试、打开文件……
 - 进程标识号
 - 一个或多个线程

线程：

- 线程是CPU调度执行的基本单元
- 线程包含以下一些基本组件：
 - CPU状态：初始、就绪、延迟就绪、备用、运行、等待、转换、终止等
 - 两个栈：分别用于内核模式和用户模式
 - 线程本地存储(TLS)：私有存储空间，用来保存子系统、运行时库以及DLL文件等
 - 线程标识号
 - 访问控制令牌(Token)，用以唯一的标识所有者及其所属组以及和该线程相关联的特权信息
 - 一般与进程共享

进程的资源及其和线程的关系：



- VAD: 虚拟地址描述符
- Handle table: 句柄列表
- 线程默认共享进程的访问控制令牌
 - 线程可拥有模拟访问控制令牌，以服务线程为主，利用客户身份创建新的令牌，替别人访问、以他的身份发出请求更安全

进程和线程的数据结构：

- 每个Windows进程都是由一个执行体进程块 (EPROCESS) 来表示，包含与进程有关的属性及其它的相关数据结构 (一个或者多个线程)
- 每个线程由执行体线程块 (ETHREAD) 来表示
- EPROCESS与其相关的数据结构大部分位于系统地址空间(内核态) 中，除了 PEB(进程环境块) 和 TEB(线程环境块) 位于进程地址空间(用户态)中
- EPROCESS 的 Active process link, 活动进程链表
 - 遍历活动进程，可以隐藏某一进程，比如相关的恶意代码

1.4 Windows的内存管理

内存管理器：

- 每个进程都有属于它的巨大的、连续的私有内存地址空间 (即虚拟地址)
- 内存管理器的功能：
 - 将虚拟的私有内存翻译/映射到真正存放数据的物理内存，可以隔离不同进程之间的数据读写
 - 当物理内存耗尽时，将内存换页到磁盘上去，然后当需要时再换页回来

虚拟内存：

- 每个进程都有它自己的虚拟地址空间
- 虚拟内存提供了和物理内存不相关的逻辑视图
- 换页是和磁盘传输内存内容的过程
 - 虚拟内存的大小可以超过可用的物理内存

Windows的内存地址布局：

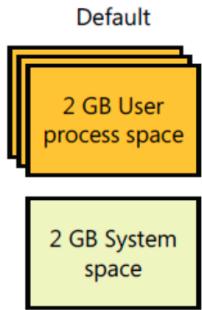


FIGURE 1-4 Typical address space layouts for 32-bit Windows

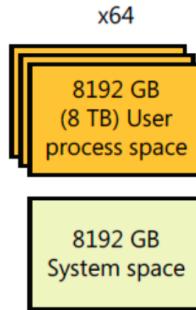
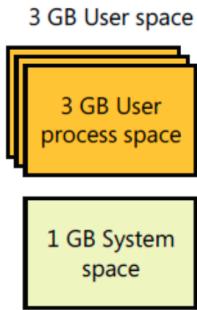
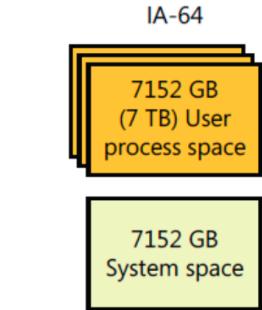


FIGURE 1-5 Address space layouts for 64-bit Windows

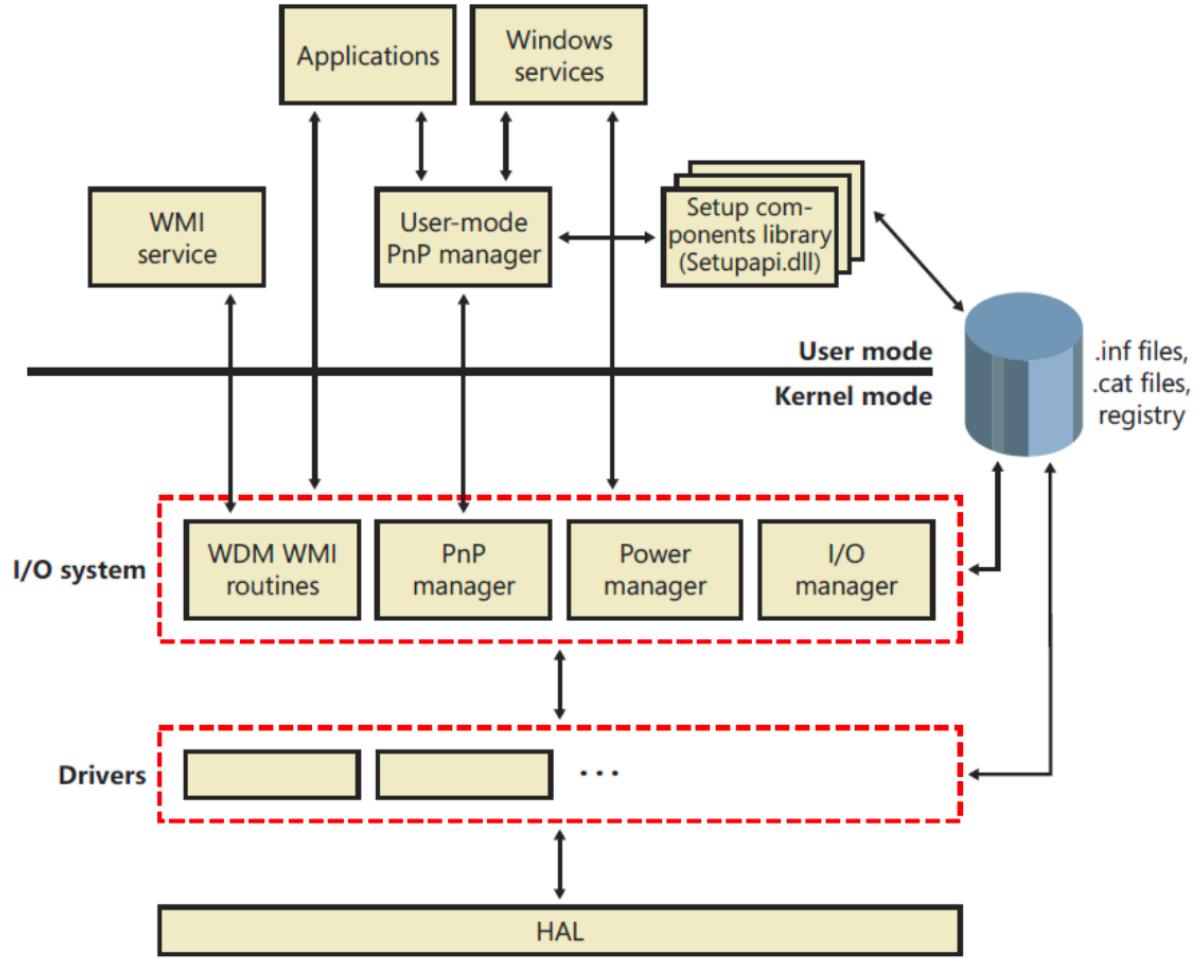


- 低地址用户，高地址内核

1.5 Windows的输入输出系统和驱动

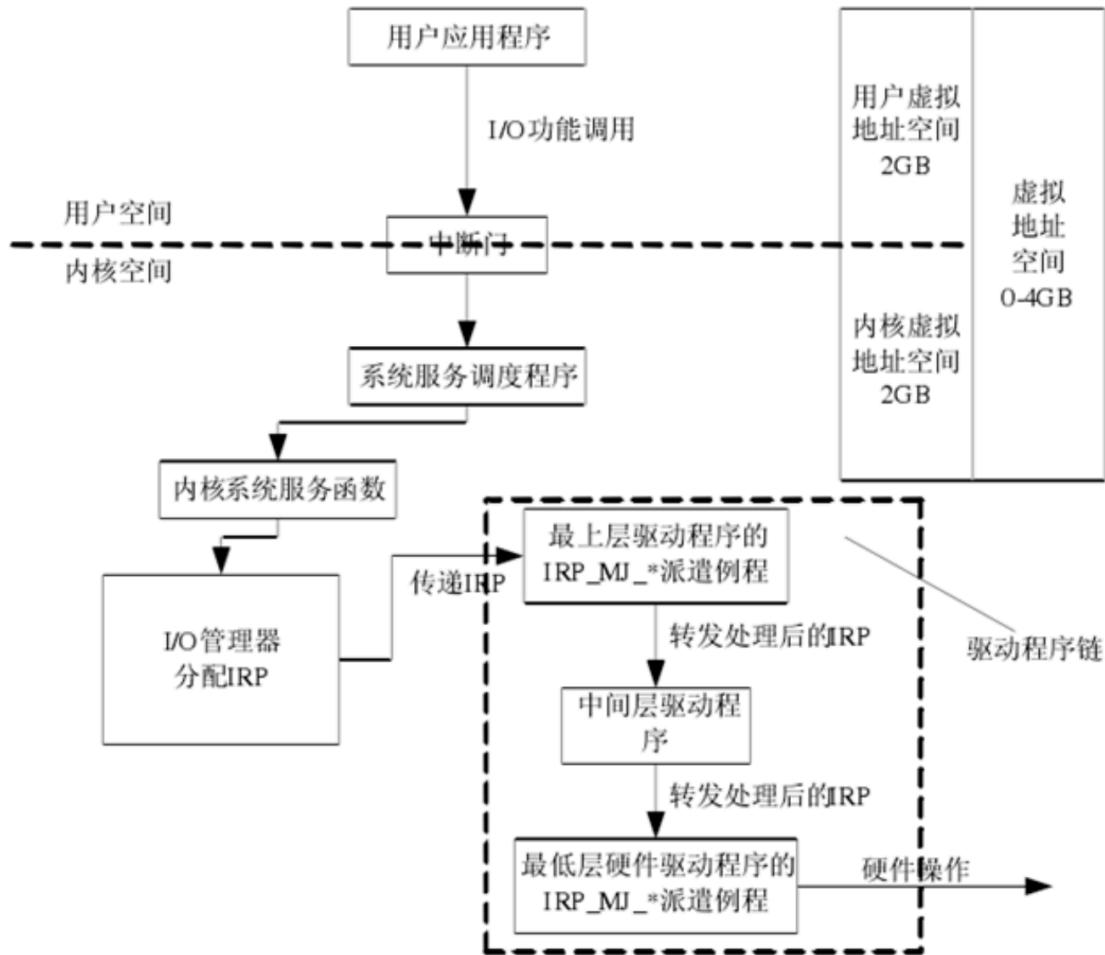
Windows的输入输出系统组件：

- 输入输出管理器 (I/O Manager)
 - 管理驱动程序
 - 可能利用一个空的设备，从而进入内核，访问系统地址空间
- 即插即用管理器 (Plug and Play/PnP Manager)
- 电源管理器 (Power Manager)



输入输出管理器接收应用程序的请求后，创建相应的 IRP (I/O request packets，输入输出请求数据包) 并传递至驱动程序进行处理，驱动程序可能有以下几种操作：

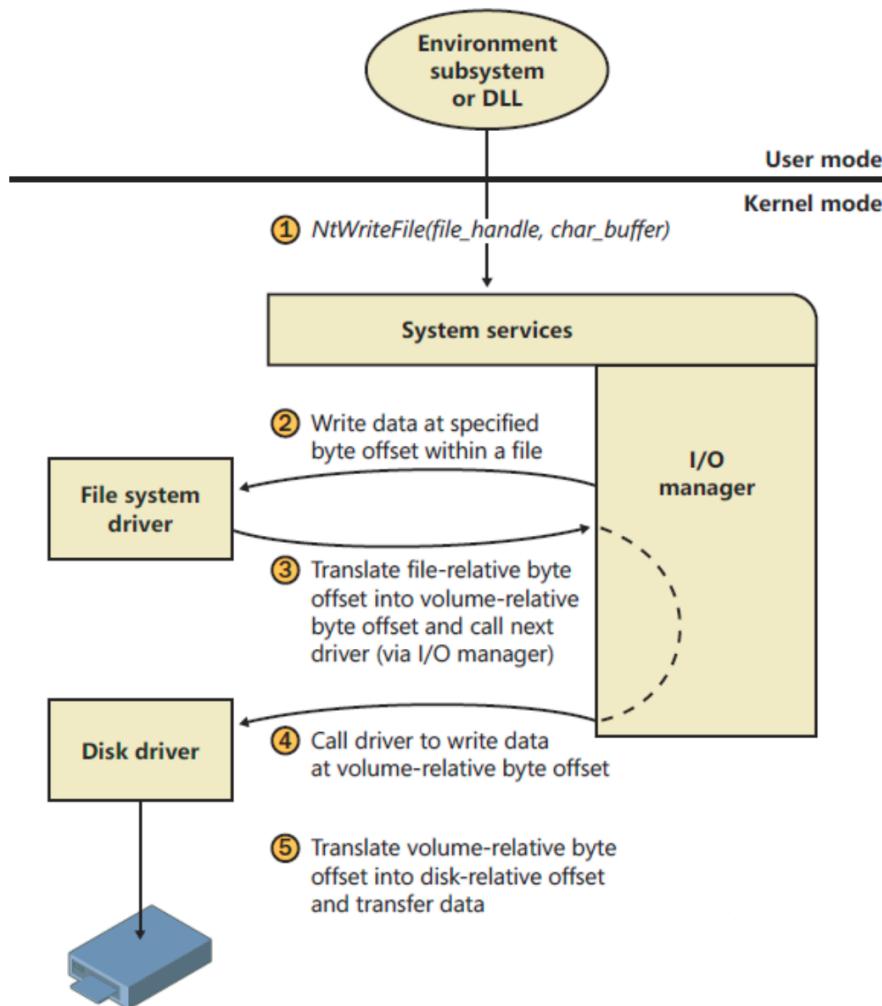
- 根据 IRP 的请求，直接操作具体硬件设备，然后完成此 IRP，并返回
- 将此 IRP 的请求，转发到更底层的驱动程序中去，并等待底层驱动的返回
- 接收到 IRP 请求后，不是急于完成，而是分配新的 IRP 发送到其他驱动程序中去，并等待返回



- 内核系统服务函数基于系统服务派遣表查找
- 跟驱动打交道的唯一途径：IRP

分层驱动模型：

- 驱动程序是分层的，可往里进行插入，比如文件加解密
- 分层的文件系统驱动和磁盘驱动程序
- 查看系统加载的驱动程序：Msinfo32.exe



1.6 Windows的文件系统

Windows支持以下几种文件系统格式：

- CDFS：适用于CD的只读文件系统
- UDF：适用于DVD的只读文件系统
- FAT12、FAT16、FAT32：
 - 文件分配表文件系统
 - 文件容量有限，最大4G，适合U盘
- EXFAT：
 - 大文件，空间消耗小，不大讲究安全性
- NTFS：
 - Windows NT以及之后Windows的标准文件系统
 - 有安全描述符
 - 格式化硬盘，空间损失大(预留空间给安全描述符等)

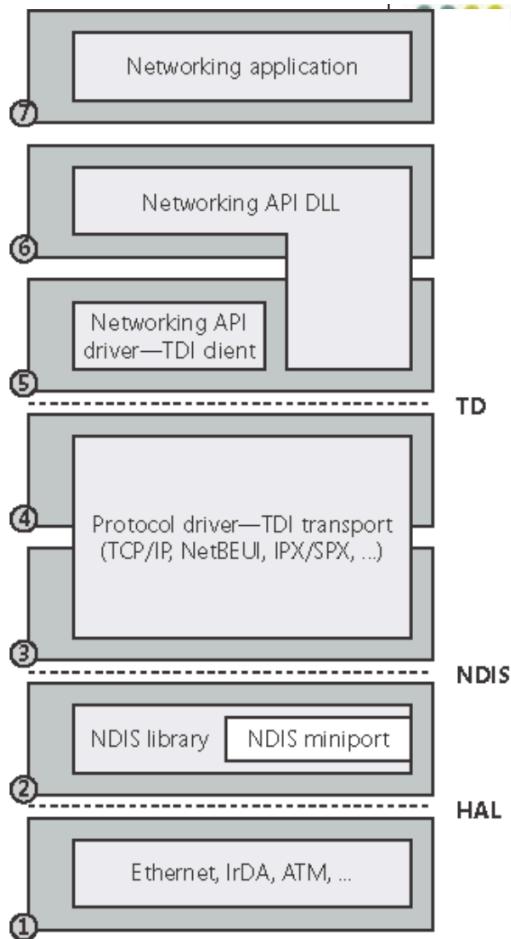
NTFS文件系统格式：

- NTFS是一种日志记录系统，其中的文件更改会被载入正式记录。其主要组件是主文件表(MFT)，其存储必要信息以便从分区找回文件。此文件系统还包括主文件表副本，当在初始表发生问题时可发挥作用
- 单文件大容量 > 4G

- 长文件名 > 255个字符
- 基于Unicode的命名方式
- 访问控制
- 磁盘配额
- 加密文件系统
- 支持多重数据流
- 支持硬链接和联结点

1.7 Windows的网络机制

Windows的网络结构和组件：



- 网络应用程序与服务进程：对应OSI应用层，通常使用各类网络API DLL来实现网络交互与通信功能
- 网络API DLL及TDI客户端：
 - 用户——对应OSI会话层与表示层，为应用程序提供了独立于具体协议的网络交互实现方式，包括Windows套接字(Winsock)、远程过程调用(RPC)、Web访问API、命名管道和邮件槽以及其它网络API接口
 - 内核(及以下)——TDI客户端作为内核模式驱动程序，是网络API接口内核部分的具体实现：将网络API的请求转换成IRP，通过TDI标准格式化后，发送给下层的协议驱动(TDI传输器)
- TDI(Transport Driver Interface)传输器：
 - 传输驱动程序接口，也被称之为TDI传输器、NDIS协议驱动程序、协议驱动程序等，对应OSI的网络层和传输层，实现了TCP/IP、NetBEUI、IPX等协议栈，接受上层TDI客户端的IRP请求，并调用NDIS库中提供的网卡驱动程序功能进行网络传输

- TDI传输器通过透明地进行如分片与重组、排序确认与重传等一系列消息操作，为上层网络应用提供了便捷的支持
- Windows Vista之后不再使用TDI，而是使用Windows Filter Platform(WFP)和Winsock Kernel(WSK):
 - WSK, Winsock内核：提供内核模式下的网络通信，使用类似于用户态Winsock的编程语法，也提供IRP和事件回调函数的异步I/O操作等特性。WSK还在下一代TCP/IP协议栈中默认支持IPv6
 - 面向编程，原来的太复杂了
 - WFP, Windows过滤平台：一套API函数和系统服务，提供创建网络过滤应用程序的能力，允许应用程序追踪、过滤甚至修改网络数据包
 - 安全特性，面向拦截、过滤(比较偏上层的过滤)
- NDIS(Network Driver Interface Specification)库及小端口(miniport)驱动程序：
 - 位于OSI的链路层，为各种不同的网卡驱动程序和TDI传输层之间构建一个封装接口
 - NDIS库(ndis.sys)对于上层为网络程序模块屏蔽了不同的网卡硬件类型
 - NDIS miniport驱动对于下层，为网卡制造商开发设备驱动程序提供了屏蔽Windows内核环境细节信息的编程接口
 - 中间层过滤驱动，wincap库基于NDIS实现
- 各种网卡硬件的设备驱动程序：处于OSI模型的物理层

1.8 Windows的域和活动目录

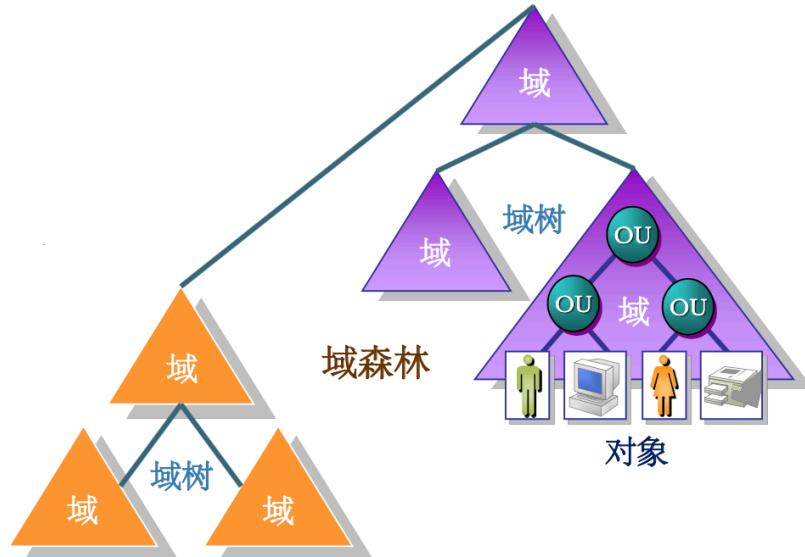
Windows系统的网络组织形式：

- 工作组：
 - 主机之间角色平等，没有上下层级关系
 - 主机数量较多时，管理和安全防护工作将会极其混乱
- 域：
 - 域是一个有安全边界的主机集合
 - 一个域中的主机用户无法访问另一域中的资源
 - 域有集中的身份验证、权限控制和安全策略管控机制
 - 便于管理
 - 单一登录验证：验证一次就够了，无需重复验证(类似于Jaccount)

域的逻辑结构：

- 单域：
 - 对象
 - 组织单元/单位(OU)：容器对象，是域中对象的逻辑组合
 - 有了域为什么还要OU：分发组策略最小单位，域太大了，不够细化
- 域树
 - 具有连续名称空间的多个域通过建立信任关系而形成的层级结构
 - 父域和子域
- 域森林

- 一个或多个没有连续名称空间的域树通过建立信任关系组成的集合



活动目录 (Active Directory, AD) :

- 软件概念
- 目录是一种存储网络对象(包括用户、组、计算机、共享资源、打印机、联系人等)信息的层次结构
- 目录服务是指帮用户快速、准确地从目录中查找到所需信息的服务
- 活动目录为域实现了目录服务，提供了网络环境的集中式管理机制
- 单一视图——在活动目录中，管理员不需要考虑被管理对象的地理位置，而只需要按照一定的方式将这些对象放置到不同的容器之中
- 活动目录是Microsoft提供的统一管理基础平台，众多服务(IsA防火墙、Exchange邮件收发、SMS系统管理等)都依赖它
- 活动目录：文件目录；目录服务：文件系统的查询

域控制器 (Domain Controller, DC) :

- 硬件概念，渗透最终目标
- DC是整个域的管理和通讯中枢，负责所有接入域的计算机和用户的身份验证、权限控制
- DC存储域范围内的所有目录数据信息，包括用户账号信息、身份验证信息(比如密码散列)，安全策略
- 多主复制(容灾)：在一个域中，可以有多个域控制器来分担用户登录、信息检索等操作。多个域控制器之间能够互相同步和复制数据
- Windows服务器安装活动目录(AD)后就变成了域控制器

域中主机的类型：

- 域控制器
- 成员服务器
 - 加入域，Server版Windows系统，没有安装活动目录
 - 主要任务是提供网络资源和应用服务，比如文件服务器、邮件服务器、数据库服务器、Web服务器，远程访问服务器、打印服务器等
- 客户机

- 加入域，其他操作系统，普通客户主机
- 使用域帐户和身份验证机制登录到域后，就可以使用活动目录提供的各种服务，访问域中的各种资源，但同时也被强制接受管理员设置的统一安全管理
- 独立服务器
 - 没有加入域，也没有安装活动目录
 - 可以创建工作组，与网络中其它计算机共享资源，但不能使用活动目录提供的任何服务

2 Windows木马后门检测

木马启动的3种可能：随开机、随用户登录、随程序启动

勒索病毒无需自启动，无需实时网络回连，简单！但木马不一样

木马后门：即远程控制代码，是一种隐蔽运行在受害者机器上的恶意代码/软件，可以让攻击者通过网络远程控制受害者的机器，窃取受害者的信息资料和隐私数据

恶意代码有很多种，木马只是其中一种

Windows木马后门的技术发展历程：

- 第1代：功能简单、技术单一，主要是密码窃取和发送
 - 2000以前
- 第2代：远程管控技术全面成熟，如BO2000、冰河等
 - 木马是server，不要点
- 第3代：防火墙和内网穿透技术，采用ICMP等协议隧道，反弹连接模式等，如网络神偷等
- 第4代：用户态的进程隐藏技术，主要采用远程线程插入(灰鸽子)、Windows挂钩、Svchost服务(Gh0st)等方式
- 第5代：主动防御系统的突破技术；内核态的隐藏技术(Rootkit)；还原系统的对抗技术
- 第6代：磁盘启动级的隐藏技术(Bootkit)，如鬼影、魔影、暗云、异鬼、双枪等
 - 在OS外的隐藏
- 第X代：硬件级别的后门技术
- 发展趋势：对等模式、匿名网络
- 现在投放木马一般不是exe，而是Powershell
- 现在木马的网络连接形态：反弹式，别人连我(攻击者)、我是server

实现木马要解决2个问题：自启动 + 连出去(网络回连)

2.1 Windows木马后门的技术

2.1.1 Windows木马后门的植入技术

直接攻击、电子邮件、文件下载、浏览网页、合并文件

主动 + 被动：

- 主动：服务器为主，利用系统漏洞获取目标机权限
- 被动：个人为主，邮件附件、浏览器漏洞、社工

具体植入方式：

- 向目标用户发送捆绑有木马的可执行文件，诱骗用户打开嵌入在其中的木马程序
 - 更改木马程序可执行文件的后缀名：exe → com、scr、bat、cmd、pif等；dll → cpl
 - com：应用程序，scr：屏保程序(有图标，效果好)
 - 制作正常的CHM(.chm)电子书，嵌入木马程序
 - 利用Unicode字符中的“由右往左的阅读顺序”来反转可执行文件名
 - 控制字符RLO
 - cod.exe → exe.doc，看到rcs、moc等要引起注意
 - (一般还要压缩包加个密，从而逃过检测软件)
- 向目标用户发送嵌入有木马程序的畸形文件，例如doc、ppt、xls、pdf等类型的文件，诱骗用户打开，利用这些文档处理软件的溢出漏洞来释放木马程序运行
- 向目标用户发送挂有木马链接的网页，诱骗用户访问，利用各种浏览器或者Flash、Java等插件的漏洞来下载木马程序运行
- 利用操作系统或应用服务的各类漏洞，对目标计算机实施攻击，获取控制权后运行木马程序
- 窃取用户口令，登录计算机后直接运行木马程序

2.1.2 Windows木马后门的开发技术

2.1.2.1 管控功能实现技术

2.1.2.2 自启动技术

三种启动方式：计算机、用户登录、软件启动

基于Windows启动目录的自启动：

- Windows NT5.x
 - C:\Documents and Settings\All Users\「开始」菜单\程序\启动
 - 安全性差
- Windows NT6.x
 - %USERPROFILE%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
 - C:\Users\YYY\AppData\.....
 - 当前用户能修改，无需提权，木马不用UAC提权也能写入
 - 可添加exe、脚本等
 - 随用户登录而启动
 - %ProgramData%\Microsoft\Windows\Start Menu\Programs\Startup
 - C:\ProgramData\Microsoft\.....
 - 至少拥有管理员及以上权限才能修改
 - 随计算机启动而启动

基于注册表的自启动：

- HKLM/HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\
- HKLM/HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run\

- HKLM/HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce\
 - HKLM/HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnceEx\
 - ...
 - HKLM、HKCU的不同：自启动的生效时间、写入的权限问题
 - RunOnce只运行一次，启动一次后清空，木马在重启、关机前需要再次写入，所以万一异常关机，木马无效
 - 使用SysinternalsSuit工具集的Autoruns软件查看自启动项，主要看没有签名的，签名未通过的，可能为木马
 - 在64位系统中还存在重定向的注册表路径：
 - HKLM/HKCU\Software\Wow6432Node\Microsoft\Windows\CurrentVersion\Run\
 - 若木马没有拿到管理员权限，则能自启动的地方很少，容易排查
 - 自启动相关目录、HKCU下相关键值
-

常规方法设置注册表Run自启动：

- RegCreateKey函数——创建/打开注册表子键
 - RegSetValueEx函数——设置注册表键值
 - 分析木马的时候重点找这些API函数的调用
-

非常规方法设置注册表Run自启动：

- RegSaveKey函数——备份，将注册表键值及其所有的子键存储到一个HIVE文件中去
 - RegRestoreKey函数——恢复，从HIVE文件恢复注册表键值及其所有的子键到注册表中去
 - 以下方法曾经能够躲避所有的主动防御软件(HIPS)
 - 所有HIPS均未主动监视这2个函数
 - 使用RegSaveKey将需要更改的目标注册表子键TargetKey及其包含的所有内容存储到一个临时文件A中
 - 在注册表中一个不被HIPS监控的地方创建一个临时子键TempKey
 - 使用RegRestoreKey将临时文件A中存储的子键及其所有内容恢复到TempKey下
 - 在TempKey下通过RegSetValueEx函数添加我们的自启动程序(TempKey非自启动，不受HIPS的监控，因此能够躲避主动防御)
 - 使用RegSaveKey将TempKey及其所有内容存储到临时文件B中
 - 使用RegRestoreKey将临时文件B中存储的子键及其所有内容恢复到TargetKey下，从而达到间接更改它的目的
-

以下需要管理员权限才能设置自启动

基于服务程序的自启动：

- 服务程序是后台运行的进程，常用来执行特定的任务，不需要和用户进行交互。比如自动更新服务、后台智能传输服务、事件日志服务等

- 服务程序的配置数据位于注册表如下键值 “HKLM\System\CurrentControlSet\Services”
 - 系统通过服务管理器 (SCM, ServiceControlManager, 对应的进程为services.exe) 来管理服务
 - 通过管理工具中的“服务”管理组件可查看管理用户态的系统服务
 - 查看不到内核态的驱动服务
-

- 基于ActiveX控件的自启动
 - HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components\{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXX}
- 基于计划任务 (Scheduled Tasks) 的自启动
- 其它.....

2.1.2.3 用户态的进程隐藏技术

procexp.exe：进程查看器

基于DLL插入的进程隐藏：

- 远程线程创建技术——CreateRemote Thread
- Windows挂钩技术——Windows Hook
- APC注入技术——APC Inject
- 以上无需管理员权限
- 木马让一个好的进程注入一个坏的dll

进程内存替换——Process Hollowing：

- 保留一个躯壳，内存被替换为恶意代码

基于SvcHost服务的进程隐藏：

- 共享服务，需要管理员权限
- 停服务、起服务无需管理员权限；创建服务、更改服务需要管理员权限

2.1.2.3.1 基于DLL插入的进程隐藏

动态链接库 (DLL) 基础：

- DLL是Windows系统中的另外一种“可执行文件”，但它不能独立运行，一般需由EXE程序创建的进程加载并调用
- DLL只是一组源代码模块，每个模块包含了应用程序将要调用的一组函数
- 在应用程序能够调用DLL中的函数之前，DLL文件映像必须被映射到调用进程的地址空间中，然后DLL的函数就可以为进程中运行的所有线程使用
- DLL有自身的入口点函数——DllMain

```

BOOL WINAPI DllMain(HINSTANCE hinstDll, DWORD fdwReason, LPVOID lpvReserved)
{
    char szProcessId[128] = {0}; //存放进程号的字符串
    switch(fdwReason)
    {
        case DLL_PROCESS_ATTACH: //DLL被初次映射到进程的地址空间中时。
            //显示加载DLL的进程号。
            _itoa(GetCurrentProcessId(), szProcessId, 10);
            MessageBox(NULL, szProcessId, "TestDLL Loaded", MB_OK);
            break;

        case DLL_PROCESS_DETACH: //DLL从进程的地址空间中被卸载时。
            break;

        case DLL_THREAD_ATTACH: //DLL被映射到进程后的新线程被创建时。
            break;

        case DLL_THREAD_DETACH: //DLL被映射到进程后的线程被撤销时。
            break;

        default:
            break;
    }
    return true;
}

```

Test.dll的DllMain入口点函数示例代码

基于DLL插入的进程隐藏：

- 木马EXE程序利用一些技术手段进入并操作另一个进程的私有内存，将木马DLL插入到该进程的地址空间中，然后自身退出达到无进程的目的
 - 比如利用浏览器进程，实现往外连接，防火墙是放行的
- 常见的技术手段：
 - 远程线程创建——CreateRemoteThread
 - Windows挂钩——SetWindowsHookEx
 - APC注入——QueueUserAPC

远程线程创建技术：

- 对应2个不同的进程即为Remote，进程A在B中创建线程
- 在对方进程中创建一个线程，加载黑的dll

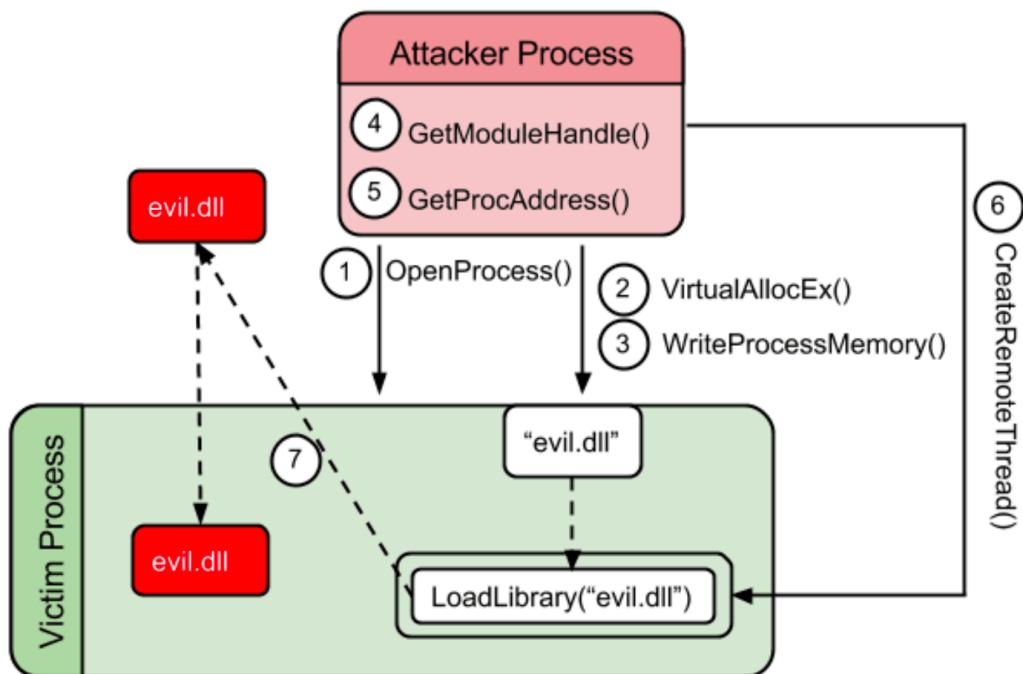
```

HANDLE CreateRemoteThread(
    HANDLE hProcess,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    SIZE_T dwStackSize,
    LPTHREAD_START_ROUTINE lpStartAddress,
    LPVOID lpParameter,
    DWORD dwCreationFlags,
    LPDWORD lpThreadId
)

```

- 参数说明：

- **hProcess**指拥有新创建线程的进程
- **lpThreadAttributes**指向安全属性结构的指针，置NULL为默认的安全属性
- **dwStackSize**指定线程的堆栈大小，置0表示与进程相同
- **lpStartAddress**为线程函数的起始地址
- **lpParameter**为传递给线程的上下文(参数)
- **dwCreationFlags**为线程创建标志，0表示立即启动，CREATE_SUSPENDED表明暂时挂起线程等待后续显式运行线程
- **lpThreadId**为内核给新生成的线程分配的线程ID(用于返回)



1. 通过OpenProcess来打开试图插入evil.dll的目标进程，如Explorer.exe、Web浏览器(iexplore.exe、Firefox.exe等)

```

hRemoteProcess = OpenProcess(
    PROCESS_CREATE_THREAD |      // 允许远程创建线程
    PROCESS_VM_OPERATION |     // 允许远程内存操作
    PROCESS_VM_WRITE,          // 允许远程内存写入
    FALSE, dwRemoteProcessId )

```

2. 向目标进程的内存写入evil.dll的文件路径信息

3.

```
//计算DLL文件路径名所需要的内存空间  
int cb = (1 + lstrlenW(pszLibFileName)) * sizeof(WCHAR);  
  
//VirtualAllocEx函数在远程进程的内存地址空间分配DLL文件名缓冲区  
pszLibFileRemote = (PWSTR) VirtualAllocEx (hRemoteProcess, NULL, cb,  
    MEM_COMMIT, PAGE_READWRITE);  
  
//WriteProcessMemory函数将DLL的路径名复制到远程进程的内存空间中去  
iReturnCode = WriteProcessMemory (hRemoteProcess, pszLibFileRemote, (PVOID)  
    pszLibFileName, cb, NULL);
```

4. 获取LoadLibrary (加载动态链接库) 函数的地址

5.

```
THREAD_START_ROUTINE pfnStartAddr =  
    (PTHREAD_START_ROUTINE)GetProcAddress (GetModuleHandle  
        (TEXT("Kernel32")), "LoadLibraryW");
```

6. 创建远程线程

7. 线程函数为“LoadLibrary”，参数为evil.dll的文件路径。即让目标进程去创建一个线程，该线程将会执行LoadLibrary函数去加载evil.dll文件

```
hRemoteThread = CreateRemoteThread (hRemoteProcess, NULL, 0,  
    pfnStartAddr, pszLibFileRemote, 0, NULL);
```

Windows挂钩技术：

- Windows的钩子(Hook)函数：
 - 这是Windows系统中消息处理机制的一个平台，应用程序可以在上面进行设置来监视特定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的
 - 钩子机制允许应用程序截获处理Window消息或特定事件。也就是说当消息到达后，应用程序在目标窗口处理函数之前处理它，也可以不作处理而继续传递该消息，甚至还可以强制结束消息的传递
- Windows钩子函数的典型应用：
 - 金山词霸等翻译软件的实时翻译功能
 - 恶意代码的击键记录功能
- 截获后监视、过滤、更改，目的是使得目标进程加载dll
- SetWindowsHookEx函数——安装Windows钩子
 - 非同步挂载，为异步感知；事情发生了通知我一下，我调用回调函数
 - 如果钩子回调函数由一个DLL提供，而被Hook的进程并没有加载这个DLL，那么系统会自动给这个进程加载这个钩子DLL。因此，我们只要使用SetWindowsHook为目标进程安装一个属于某DLL的钩子函数，就可以强迫该目标进程加载这个DLL

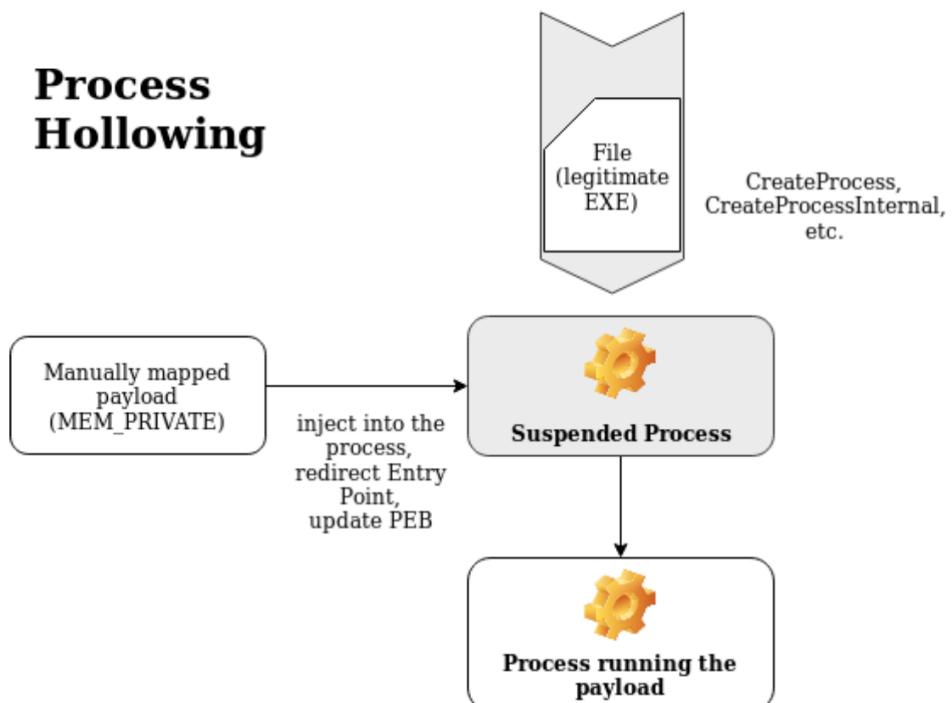
APC注入技术

- APC的基本概念：

- Asynchronous procedure call, 异步程序调用
- 一个进程当执行到SleepEx()或者WaitForSingleObjectEx()时，系统就会产生一个软中断，当线程再次被唤醒时，此线程会首先执行APC队列中的被注册的函数
- APC注入技术：
 - 使用QueueUserAPC()函数在目标进程的APC队列中注册函数，从而使得目标进程去执行指定的代码，比如“LoadLibrary”函数，并且向目标进程的内存写入DLL路径的参数，从而完成DLL注入的目的（这有点类似于CreateRemoteThread技术）
 - APC注入因为受目标进程使用API的条件而受限，并且处于等待的线程被注入后会立即返回，也有可能造成线程的运行错误，所以应用起来并不是很通用

2.1.2.3.2 进程内存替换

进程内存替换：将一个可执行文件（恶意代码文件）重写到一个运行进程（受信任的进程）的内存空间



1. 以挂起状态创建一个正常的合法进程，该进程的主线程被挂起
2. 调用 **NtUnmapViewOfSection** 来释放 **ImageBaseAddress**（被加载可执行文件的基地址）指向的内存
 - 实际调用 Ntdll.dll 导出的函数
3. 调用 **VirtualAllocEx** 为恶意代码分配新的内存
4. 调用 **WriteProcessMemory** 将恶意代码的 PE 头部和每个段（Section）写入到进程的内存空间
5. 调用 **SetThreadContext** 恢复进程环境，让入口点指向恶意代码
6. 调用 **ResumeThread** 恢复主线程的执行

非常规范，很容易被识破

2.1.2.3.3 基于SvcHost服务的进程隐藏

Svchost服务基础：

- Windows系统服务分为独立进程和共享进程两种，随着系统内置服务的增加，Windows就把很多系统服务变成共享进程的方式，由svchost.exe统一启动
- Svchost本身只是作为服务宿主，并不实现任何的服务功能。需要Svchost启动的服务以动态链接库(DLL)形式实现。在安装这些服务时，把服务的可执行程序指向svchost，启动这些服务时由svchost加载相应服务的DLL文件

以EventSystem服务为例，注册表子键

HKLM\SYSTEM\CurrentControlSet\Services\EventSystem 下存放了该服务相关的配置信息：

可执行程序路径 "ImagePath" = "%SystemRoot%\system32\svchost -k netsvcs"

动态链接库路径 "Parameters\ServiceDll" = "%SystemRoot%\system32\es.dll"

- netsvcs为组，此组可连出去，防火墙放行

Svchost服务进程：

- Windows把共享的服务分为几组，同组的服务共享一个svchost进程，不同组的服务则使用多个svchost进程
- 组的区别是由服务的可执行程序后边的参数决定的，如svchost -k netsvcs, svchost -k RpcSs
- svchost的所有组和组内的所有服务都在注册表的如下位置：
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
- 安装通过svchost启动的服务：
 - 需要在HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost下有该服务名，这可以通过如下方式来实现：
 - 添加一个新的服务组，在组里添加服务名
 - 在现有组里添加服务名
 - 直接使用现有服务组里的一个服务名，但本机没有安装的服务
 - 修改现有服务组里的现有服务，把它的Service Dll指向Trojan DLL
 - (前面三个改文件，最后一个改注册表)
 - Dll的程序实现：
 - Dll程序本身只要实现ServiceMain()函数和服务控制程序，在ServiceMain()函数里用RegisterServiceCtrlHandler()注册服务控制程序，并设置服务的运行状态即可
- 对抗svchost木马：
 - 检查注册表有无改动
 - 检查dll本身是否改动

2.1.2.4 数据穿透和躲避技术

控制者 C2 (command and control) 服务器，木马主动向此服务器发起连接，如何找到该服务器？

利用反弹端口技术 (3种经典技术)：

- 控制端作为服务器在运行后开启监听端口，木马主动向控制端发起通信请求，建立起通信信道
- 木马通过固定的域名解析，或者第三方服务器上某个约定的文本文件，或者某个微博的最新信息，或者某个Blog的最新日志内容，获取到攻击者的控制主机(服务器端)的网络地址，然后主动向其发起连接
- 反弹连接成功之后，可使用安全的协议隧道(如HTTP/HTTPS加密隧道等)进行通讯，实现远程控制的功能

域名/IP → 易被防守方画像，因此需要经常更换来逃过审查

协议隧道技术：

- 将远程控制传输的指令和数据封装在正常的应用协议中，作为数据部分(往往还进行加密)进行传输，用以躲避基于应用过滤的防火墙和IDS
- 常见的协议隧道：
 - HTTP/HTTPS
 - MSN, Google Talk(XMPP协议)
 - Google Docs
 - 各类云平台
- 匿名通信技术：
 - 来源匿名、躲避追踪和溯源
 - Tor(洋葱路由网络)
 - I2P(大蒜网络)

2.1.2.5 内核态的隐藏技术(Rootkit)

内核级别的隐藏，对权限要求高

工具agony：演示Rootkit

Rootkit检测工具：XueTr.exe → 以红色显示被隐藏的进程

Rootkit的基本概念：

- Rootkit现在指的是一种特殊的恶意软件或代码，用来在系统中隐藏自身及指定的文件、进程、模块、网络连接、注册表、服务等
- Rootkit的宗旨：一切为了隐藏

Rootkit技术的类型及发展：

- Ring3 → Ring0
 - Windows Rootkit往往需要加载恶意的驱动程序，进入到内核态，从而篡改系统内核来达到隐藏的目的
- 两大类技术：
 1. MEP (Modify Execution Path, 执行路径修改)
 2. DKOM (Direct Kernel Object Manipulation, 直接内核对象操纵)

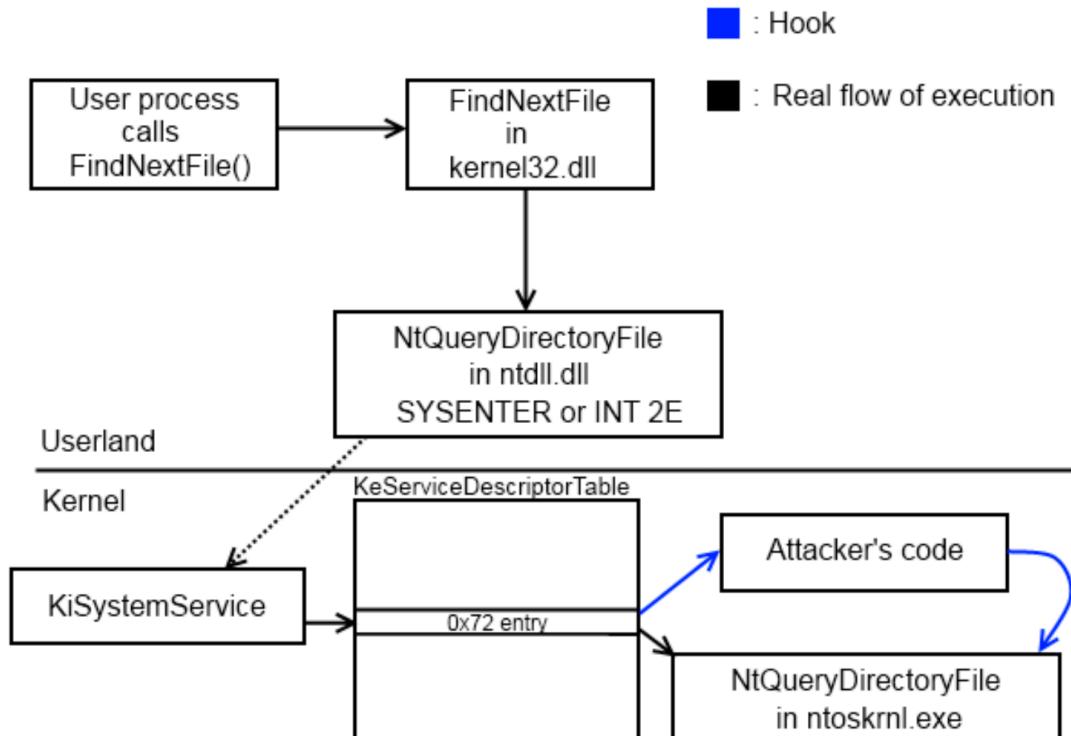
- 改对象存储，更底层

MEP技术：

- MEP，执行路径修改，主旨就是Hook(挂钩、挂接)
 - 拦截系统函数或相关处理例程，先转向我们自己的函数处理，这样就可以实现过滤参数或者修改目标函数处理结果的目的，实现进程、文件、注册表、端口之类的隐藏
 - 一般挂查询的函数，不影响系统使用，但查看时被欺骗
- 与主动防御软件的技术原理相同，都是挂钩
 - Rootkit：篡改正常函数的返回结果
 - 主动防御软件：对函数输入参数进行检查过滤
- Hook技术分类：
 - SSDT Hook，系统服务描述符表挂钩
 - Inline Hook，内联挂钩
 - IAT (Import Address Table) Hook，导入表挂钩
 - IDT (Interrupt Descriptor Table) Hook，中断描述符表挂钩
 - FilterDriver，过滤驱动
 - IRP Function Hook，IRP函数挂钩

SSDT Hook技术：

- SSDT (系统服务描述符表)：把Ring3的应用API函数和Ring0的内核API函数联系起来



SSDT的导出：

- KeServiceDescriptorTable是Windows内核导出的SSDT表，该表拥有一个指针，指向SSDT中包含由ntoskrnl.exe实现核心系统服务的相应部分

禁用Windows内存保护机制 (3种方法) :

1. 更改注册表，但需要重启

```
HKLM\SYSTEM\CurrentControlset\Control\Session Manager\Memory  
Management\EnforceWriteProtection = 0  
HKLM\SYSTEM\CurrentControlset\Control\Session Manager\Memory  
Management\DisablePagingExecutive = 1
```

2. 修改控制寄存器CR0，将wp位设置为0

- 只适合于单CPU

```
asm 去掉内存保护机制  
{  
    push eax  
    mov eax,CR0  
    and eax,0FFFFFFFh  
    mov CR0,eax  
    pop eax  
}
```

```
asm 恢复内存保护机制  
{  
    push eax  
    mov eax,CR0  
    or eax,NOT 0FFFFFFFh  
    mov CR0,eax  
    pop eax  
}
```

3. 利用内存描述符表 MDL 来描述一块可写内存 (推荐)

接下来开始挂钩：

- 例如，我们要Hook NtQuerySystemInformation 函数 (Taskmgr.exe等工具就是通过该函数来获取系统上的进程列表)

```
//假设，NtQuerySystemInformation在当前Windows系统的SSDT里面的索  
引号是0x97。那么，我们就需要把SSDT中偏移0x97 * 4处原来的一个  
DWORD类型的值读出来保存到一个全局变量中，然后再把它重新赋  
值成一个新的Hook函数的地址即可。
```

```
OldNtQuerySystemInformation =  
    KeServiceDescriptorTable.ServiceTableBase[0x97];  
  
KeServiceDescriptorTable.ServiceTableBase[0x97] =  
    NewNtQuerySystemInformation;  
  
//但问题是，这个索引号在不同的Windows操作系统版本上不一定相同。
```

获得内核函数索引号的通用办法

在ntoskrnl导出的ZwQuerySystemInformation中包含有索引号的硬编码。

```
kd> u ZwQuerySystemInformation
804011aa b897000000      mov    eax,0x97
804011af 8d542404      lea    edx,[esp+0x4]
804011b3 cd2e          int    2e
804011b5 c21000        ret    0x10
```

从中可以看出，只需要把ZwQuerySystemInformation入口处的第二个字节取出来就能得到相应的索引号了。

```
//获得函数在SSDT中的索引宏
#define SYSCALL_INDEX(_Function)
    *(PULONG)((PUCHAR)_Function+1)

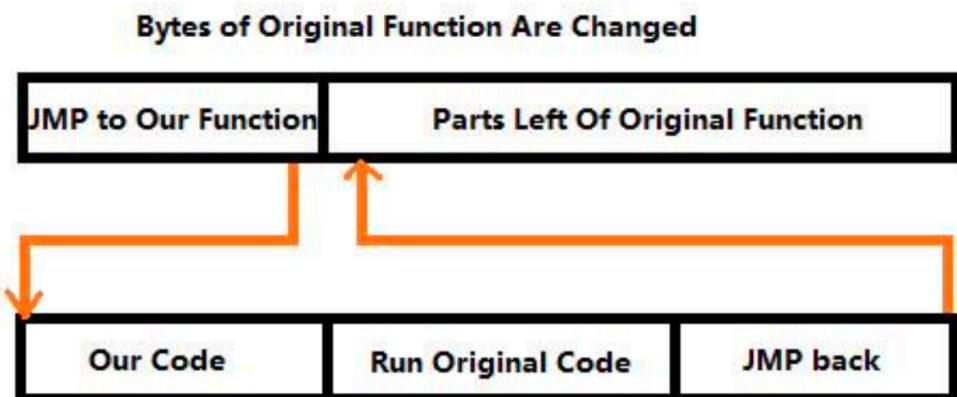
//获得SSDT基址宏
#define SYSTEMSERVICE(_function)
    KeServiceDescriptorTable.ServiceTableBase[ *(PULONG)((PUCHAR)_function+1)]
```

SSDT Hook技术方法总结——以隐藏进程为例：

1. 突破SSDT的内存保护(如MDL方法)
2. 实现自己的NewZwQuerySystemInformation函数，过滤掉包含特定字符串的进程名
3. 交换ZwQuerySystemInformation与我们自己的New*函数

Inline Hook(内联挂钩)技术：

- 又被称为Detour技术，运行时补丁技术
- Detour首先查找目标函数在内存空间中的地址，然后修改前几(5或7)个字节，把原有的指令代码改为跳转到替换函数的jmp、call等语句
- Detour技术只须修改单个函数就能够影响系统程序对该函数的调用，而无论程序是通过何种表索引到目标函数的



Windows API函数的前5个字节通常为：

```

8bff      mov edi, edi
55        push ebp
8bec      mov ebp, esp

```

- 第一行无效，是后门，为内联挂钩留下操作空间
- 后两行保留原来函数信息

因此，**可以更改函数的前5个字节为“JMP addr”来进行指令跳转**，从而改变函数执行流程，进而可以过滤函数的输入参数或者输出结果

7731EE9B	8BFF	MOV EDI, EDI
7731EE9D	55	PUSH EBP
7731EE9E	8BEC	MOV EBP, ESP
7731EEA0	83EC 40	SUB ESP, 40
7731EEA3	53	PUSH EBX

7731EE9B	E9 30236EF4	JMP 6BA011D0
7731EEA0	83EC 40	SUB ESP, 40
7731EEA3	53	PUSH EBX

DKOM技术：

- DKOM，直接内核对象操纵，偏底层
- 不同版本差异大，往往更改进程链表、服务链表
- Windows系统采用私有的数据结构在内存中存储对象信息，包括进程、服务、端口等内核对象
- DKOM技术通过直接修改内核对象的数据结构来影响上面各层函数的返回结果，从而实现隐藏特定对象的功能
- DKOM技术需要对Windows的系统机制非常熟悉，还需使用WinDbg等调试工具来查看Windows系统所没有公开的内部数据结构

使用DKOM技术隐藏系统进程：

- 进程的EPROCESS结构包含一个LIST_ENTRY的双向链表结构，该结构具有两个指针成员：FLINK和BLINK，分别指向该进程的前方和后方进程
- 通过当前进程来遍历此双向链表，就可以获得Windows系统的所有活动进程列表
- 从遍历得到的系统进程列表中匹配找到所需隐藏的进程节点，就可以将其从链表中断开，从而实现将该进程隐藏的目的

2.1.2.6 磁盘启动级的隐藏技术(Bootkit)

前面基于OS，此基于磁盘，OS无感知

Bootkit的概念：

- Bootkit最早于2005年由eEye Digital公司在其“BootRoot”项目中提及，该项目通过感染主引导记录(MBR)的方式，实现绕过内核检查和启动隐藏
 - 512个字节，后446个可改写，从而影响系统启动顺序

- 一般认为，在开机时比Windows内核更早加载，实现内核劫持和篡改的技术，都可以称之为Bootkit
- 比Rootkit要求更高，一般见于国家级对抗

Bootkit的技术种类：

- 基于MBR篡改的：eEye BootRoot、StonedBootkit、鬼影(ShadowGhost)、魔影(TDL)、暗云等
- 基于VBR篡改的：Rovnix、异鬼、双枪等
- 基于Bios感染的：IcLord等

基于修改MBR的Bootkit技术：

- MBR (主引导记录) 存在于磁盘最开始的一个扇区，共512字节，包括446字节的引导代码，64字节的4个分区表和2字节的结束标志位
- 可通过直接磁盘操作相应的API函数直接修改MBR的引导代码部分，还可直接给磁盘发送SCSI指令来对磁盘进行读写
- 修改MBR引导代码，在实模式下挂钩磁盘读取中断(INT 13h)或者内存读取中断(INT 15h)函数，然后在中断处理函数中对读到的内容加以过滤，匹配想要加载的指定文件或代码，修改返回的内容，使其跳转到病毒体的保护模式代码，这样便使病毒从实模式的执行权限转向到了保护模式的执行权限
- 病毒获取到执行权限之后就可以修改磁盘上的重要系统文件，比如beep.sys等驱动文件，从而实现隐蔽加载的功能
- 病毒此时还可以修改系统内核文件(ntoskrnl.exe)，去除PatchGuard、CodeSigning (防护内核) 等重要的保护功能

基于修改VBR的Bootkit技术：

- VBR (卷引导记录) 存在于激活分区 (引导分区表项的四个主分区里只有一个激活的) 的第一个扇区之中
- 病毒感染的对象不是MBR引导代码，而是VBR将要读入的 BootMgr Loader 的代码

2.1.3 Windows木马后门的免杀技术

对抗传统静态代码检测：加壳，添加花指令，输入表免杀

对抗启发式代码检测：动态函数调用

对抗云查杀：动态增大自身体积 (往云上传不动)，更改云查杀服务器域名解析地址 (连不上去)，断网，利用散列碰撞绕过云端“白名单”

攻击主防杀毒软件：更改系统时间，窗口消息攻击，主动发送IRP操纵主防驱动

利用证书信任：盗取合法证书签名，利用散列碰撞伪造证书，利用合法程序的DLL劫持缺陷(“白加黑”)

- DLL劫持是随软件启动而自启动的方法
- 先到当前目录加载，若没有则到系统目录，因此在当前目录放下黑的DLL即可劫持
 - UAC提权绕过很多利用了这个手段
 - sysprep.exe——Win7自动提权软件，存在DLL劫持缺陷
 - 绕过UAC不是不可能，方法很多
 - 真正对抗的是主动防御软件，而不是系统，系统漏洞太多了

白名单技术：Powershell、InstallUtil、IEExec、Rundll32、Regsvr32、Msbuild、...

2.2 Windows木马后门的检测分析

2.2.1 Windows木马后门的检测技术

2.2.1.1 基于自启动信息的检测

Autoruns：查看系统中所有的自启动项信息，包括注册表启动、服务启动、浏览器组件启动、组策略脚本启动、计划任务启动等许多信息，支持程序代码数字签名的验证

Autoruns支持查看的自启动种类：

- “Logon”: 基于常规注册表的启动
- “Internet Explorer”: IE浏览器的BHO对象
- “Explorer”: 包括ActiveX在内的资源管理器加载对象
- “Services”: 系统服务
- “Drivers”: 系统驱动程序
- “Scheduled Tasks”: 计划任务
- “Boot Execute”: 启动执行
- “Image Hijacks”: 映像劫持
- “Winlogon”: 用户登录通知程序

2.2.1.2 基于进程信息的检测

ProcessExplorer:

- 显示进程的基本信息
- 显示进程加载的DLL模块：名称、描述、公司、版本
- 显示进程打开的句柄 (Handle) 信息：类型、名称
- 显示进程 (主要是svchost) 包含的服务信息
- 显示进程的网络连接信息
- 验证可执行代码的数字签名

Process Name, 进程名称
PID, 进程ID
User Name, 用户名称
Description, 描述
Company Name公司名称
Verified Signer, 数字签名签发者
Version, 版本
Image Path, 执行文件路径
Window Title, 窗口标题
Windows Status, 窗口状态
Session, 所在的会话ID
Command Line, 命令行
DEP Status, DEP安全机制状态
Integrity Level, 完整性级别
Virtualized, 虚拟化机制是否启用
ASLR Enabled, ASLR安全机制是否启用

TcpView:

- 实时动态显示进程的网络连接情况
- 看一个进程的网络连接：
 1. ProcessExplorer查看某一进程的网络连接属性
 2. TcpView查看
 3. netstat -anob -p tcp
 - -a: 显示所有选项
 - -n: 拒绝显示别名
 - -o: 显示拥有的与每个连接关联的进程ID
 - -b: 提权, 连接对应的可执行程序
 - -p tcp: 协议, 只看tcp

2.2.1.3 基于数据传输的检测

WireShark:

- WireShark是现今最好的开源的网络数据包协议分析软件。它的前身为Ethereal。2006年6月, 因为商标的问题, Ethereal更名为Wireshark
- Wireshark适用于Unix和Windows平台, 支持多种网络接口
- Wireshark可以详细显示包的详细协议信息, 打开/保存捕捉的包, 导入导出其他捕捉程序支持的包数据格式, 通过多种方式过滤包, 以多种方式查找包, 创建多种统计分析等

2.2.1.4 Rootkit/Bootkit的检测

此项一般先做, 保证用户态看到的信息是准确的, 不是被骗的

检测系统中是否有隐藏的文件、进程、注册表项、通信端口、系统服务、内核模块等, 也可用于去除系统中的底层钩子, 保证普通应用层检测软件的结果正确性

- XueTr(XT)
- PowerTool(PT)
- RootkitUnhooker(RKU)
- Kaspersky TDSSKiller
- 其它(不再更新)：Icesword(冰刃)、Wsyscheck、DarkSpy、RootkitRevealer等

2.2.2 Windows木马后门的分析技术

2.2.2.1 木马后门的动态分析

动态分析基础技术：

- 基于虚拟机执行/沙箱(Sandbox)的动态分析(把代码交给沙盒, 自动分析)：
 - SSM(System Safety Monitor)
 - Anubis沙箱
 - Cuckoo
 - 恶意代码可能分析自己是否在虚拟机中, 此工具可对抗
 - Norman沙箱

- GFI沙箱(CWSandbox)
 - ...
 - 检测行为：
 1. 进程：创建新进程？
 2. 文件：写文件、读文件？
 3. 注册表：自启动、配置了什么信息？
 4. 网络流量：域名解析、连什么地址？
 - 进程动态监测(手动分析)：
 - ProMon - ProcessMonitor
 - 设定监控过滤器
 - 实时显示监控信息，包括注册表、文件、网络连接、进程行为等
 - 网络数据监测：
 - 模拟网络：伪造DNS应答，伪造控制端监听
 - 截获数据包分析
-

动态分析高级技术：

- 用户模式调试
 - OllyDbg
 - 创建断点，怎么断比较困难

2.2.2.2 木马后门的静态分析

静态分析基础技术：

- 反病毒引擎扫描：VirusScan、VirusTotal、...
 - 文件格式识别：PEid、FileAnalyzer、...
 - 文件加壳识别及脱壳：VMUnpacker、UPX、UnPEPack、ASPack unpacker、...
 - 在从代码中提取敏感字符串前，必须要先加壳识别及脱壳
 - 明文字符串查找：BinText、Strings
 - 链接库及导入/导出函数分析：Depends、IDA Pro、...
-

静态分析高级技术：

- IDA Pro
- 分析恶意代码，主要分析函数调用，找关键函数

3 Windows系统安全概述

3.1 Windows系统安全性的历史与发展

2012年2月，网络上出现了能直接获取Windows当前用户登录明文密码的“神器”——mimikatz

- 用于对本地安全授权子系统lsass进程的调试和分析



Microsoft安全开发生命周期 SDL (Security Development Lifecycle)

3.2 Windows的基础安全防护机制

Windows文件保护(WFP)——Windows 5.x

Windows资源保护(WRP)——Windows 6.x

- 防止Windows重要的系统文件被恶意篡改、替换或删除
- 隐藏存储目录，这些文件被压缩保存

WFP: %systemroot%/system32/dllcache

WRP: %systemroot%/winsxs/backup

内核模式写保护：防止Windows内核被非法修改

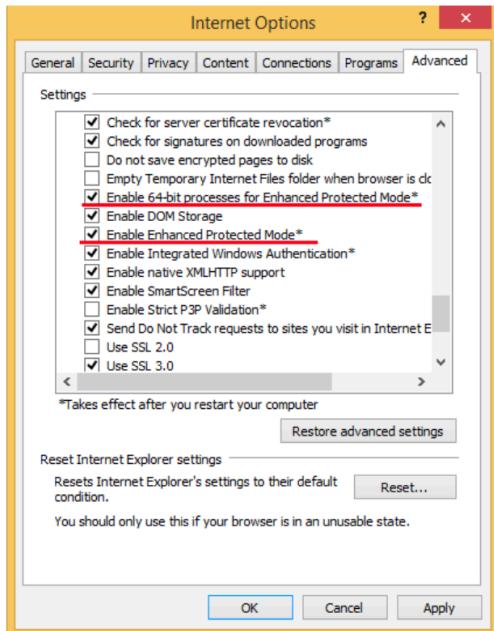
驱动程序签名：防止非法(无信任签名)的驱动程序加载到系统中

Windows的内存防护安全机制：

- DEP (数据执行保护):
 - 体现了数据代码分离原则
 - Windows XP SP2开始引入，缺省仅为基本的Windows程序和服务启用DEP
 - 基本原理是将数据所在的内存页标记为不可执行，当程序产生溢出，恶意代码试图在数据段执行指令时，CPU会产生异常而不去执行指令

- 这个机制需要CPU的支持。AMD公司推出了EVP(Enhanced Virus Protection), Intel推出了EDB(Execute Disable Bit), 这些技术在原理上均是在内存的页表中加入一个特殊的标识位(NX/XD)来标识是否允许在该页上执行指令
 - (各种内存溢出攻击的利用点: 数据、代码没分离)
 - DEP的四种可选参数:
 - OptIn: 对于多数用户版本的操作系统来说, 如Windows XP/7/10, 默认仅为一些基本的Windows程序和服务启用DEP保护。该模式可被应用程序动态关闭
 - OptOut: 多用于服务器版本的操作系统, 如Windows Server 2003/2008/2012等, 为所有在所选列表外的程序和服务启用DEP, 这种模式下, DEP仍可被应用程序关闭
 - AlwaysOn: 对所有的进程启用DEP保护, 不存在排除列表。该模式不可以被关闭, 仅在64位操作系统上才能实现, 最大限度的保证了所有程序都能够抵御溢出攻击
 - AlwaysOff: 对所有的进程都禁用了DEP保护, DEP也不能被应用程序动态开启, 这该模式一般只有在特殊场合才会使用
 - 绕过DEP的方法: 返回导向编程 ROP (Return Oriented Programming)
- ASLR (内存地址空间布局随机化):
 - 在未引入之前, 地址都是固定的
 - 能够对抗ROP技术
 - 在加载程序到内存空间时随机化各个模块的起始加载地址, 防止攻击者定位攻击指令代码的位置
 - ASLR需要操作系统及应用程序的双重支持才能发挥作用
 - 支持ASLR的程序在PE头中会设置 IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE 标识表明其支持ASLR
 - ASLR主要影响的部分:
 - 模块随机化: 系统将PE文件映射到内存时, 对其加载基址进行随机化处理, 基址在系统启动时确定, 系统重启后会变化
 - 堆栈随机化: 每次程序加载后, 其内存空间中堆、栈的基址都会发生变化。于是内存中的变量所在的地址也会发生变化
 - PEB/TEB随机化
 - Windows Vista/7引入实现, 但ASLR机制尚不完善, 攻击者还能在一定范围内进行漏洞利用
 - 存在没有随机化的系统或软件的EXE/DLL, 即编译生成代码时没有使用/DYNAMICBASE选项进行链接
 - MS Office 2010/2007加载的HXDS.dll, 支持Windows 7中的IE8/9
 - JRE 1.6.x加载的MSVCR71.DLL, 支持Windows 7中的IE8/9
 - .NET Framework 2.0 SP2加载的vsavb7rt.dll
 - 能被地址空间喷射技术绕过, 32位OS, 经常发生于针对浏览器的攻击
 - Heap Spray, 堆喷射
 - JIT (Just In Time Compilation, 即时编译)Spray, 比如Python解释器
 - Java Applet Spray
 - 可以通过内存区预判来绕过ASLR机制
 - 强制ASLR: Windows 8+, 安装了更新的Windows 7
 - 信息泄露漏洞对抗ASLR + ROP对抗DEP → 任意代码执行 → 沙盒逃逸漏洞 → 提权漏洞
 - EPM (增强保护模式):

- 也叫做“沙盒模式”(Sandbox Mode)，本质上是隔离进程和降低权限
- Windows 8的IE 10开始引入，从Windows 8.1开始默认启用



Enhanced Protected Mode (EPM)

启用增强保护模式

为浏览器的标签页开启AppContainer限制模式

Enable 64-bit processes for Enhanced Protection Mode

针对增强保护模式启用64位进程

为浏览器的标签页开启原始64位进程。在64位的Windows 8以后操作系统上，IE浏览器进程默认以32位进程运行浏览器标签页进程，容易遭受到堆喷射攻击来绕过ASLR机制。

- EMET (增强缓解体验工具包):

- 为应对网络中层出不穷的漏洞，在2010年10月推出的一款安全工具，使得用户即使在未安装补丁的情况下也可以免受攻击
- 包括数据执行保护(DEP)、结构化异常处理覆盖保护(SEHOP)、随机地址空间分配(ASLR)、空页(Null Page)保护等技术
- 最新版的还包括ASR(Attack Surface Reduction)、EAF+ (Export Address Table Filtering Plus)、Deep Hook等

Windows Vista/7引入的重要安全性机制：

- UAC (用户帐户控制)
- UIPI (用户界面权限隔离)
- 文件和注册表虚拟化
- 服务会话隔离
 - 服务会话0，本地控制台登录用户会话1，远程登录用户会话2，...
- IE低权限保护模式
 - 浏览器子进程是低权限的
- PatchGuard
 - Bootkit在系统启动前关闭此项
- CodeSigning
 - Bootkit在系统启动前关闭此项

PatchGuard (内核保护系统)：

- 在64位版本的Windows操作系统中提供的新功能，用于保护操作系统的内核结构，防止它们被其它程序修改

- 系统服务描述符表SSDT
- 全局描述符表GDT和中断描述符表IDT
- 系统映像，如ntoskrnl.exe、ndis.sys、hal.dll等
- 其它
- PatchGuard处在系统任务的一个较高层面上，通过每隔一定时间进行一些固定的检查来确定这些系统关键内容是否更改
 - 这些检查主要通过将核心内容与缓存中保存的已知正确的备份进行对比，检测时间间隔大约为5-10分钟的某一随机选择时间
 - 缺乏本地硬件水平的支持，只能通过轮询的形式，而不是采用事件驱动或硬件驱动的形式

Code Signing (代码签名检查机制)：

- 加载到系统内核中运行的驱动程序必须有数字签名以保证其代码的完整性，否则系统内核就不加载该驱动程序
 - 代码完整性检测被加载到内核中的驱动程序或系统文件是否已经被签名，或正在运行系统管理员账户权限的系统文件是否已被恶意软件篡改
 - 64位的Windows系统默认启用代码签名检查机制
-

Windows 8引入的重要安全性机制：

- ASLR随机熵提高
 - 在Windows 8中，特别是在64位下可以获得更高的随机熵
- 空值引用保护
 - 空值引用 (NULL dereference) 是由于声明变量时将变量初始化为Null，之后引用该对象却未进行空值判断造成的。如果攻击者能够故意触发空指针间接引用，攻击者就有可能利用引发的异常绕过安全逻辑，或致使应用程序泄漏调试信息，最终进行本地漏洞利用
 - Windows 8之前，内核允许用户进程映射NULL页，而Windows 8则禁止对前64k的空间进行映射
- 内核缓冲区完整性检查
- 禁止零页内存分配
 - Windows操作系统中，零页内存供16位虚拟机NTVDM使用，以确保16位代码正常运行。之前系统中存在一些内核漏洞，通过ZwAllocateVirtualMemory等系统调用可以在进程中分配出零页内存，触发未初始化对象指针/数据指针引用漏洞或辅助漏洞进行攻击
 - Windows 8中，禁止进程申请低地址内存(0x0~0x10000)；同时，默认禁用16位虚拟机，需要管理员权限才能开启。Windows 8在所有可能的内存分配位置检查零页分配
- 禁止Win32k系统调用
 - Win32k.sys是Windows内核漏洞高发对象，对它的调用不受进程权限的限制
- 不可执行的非分页池
- UEFI启动技术
 - Windows 8拥有新的保护机制来对抗Rootkit和Bootkit。所有版本的Windows 8都将包括统一可扩展固件接口(UEFI)安全开机功能，该功能取代了标准BIOS来作为电脑的固件接口
 - UEFI安全启动是实现跨平台和固件安全的基础，与体系结构无关。在执行固件映像之前，安全启动基于公钥基础结构的流程来验证固件映像，帮助降低遭受启动加载程序攻击的风险
- Intel Secure Key 技术
 - Intel在2012年4月正式发布的第三代Core处理器中加入了Intel Secure Key技术(代号“公牛山”)。该技术提供了对硬件实现的底层数字化随机数生成器(DRNG)的支持；提供了基于硬件的高性能、高质量

的熵和随机数生成器；引入了新的指令 RDRAND

- Windows 8系统内核开始使用该指令产生随机数，如重要的Security Cookie、ASLR的生成过程
-

Windows 10引入的重要安全性机制：

- 基于硬件虚拟化的安全隔离
 - Windows 10引入了CredentialGuard和DeviceGuard安全功能，运用硬件虚拟化技术，实现安全隔离。这两项功能主要存在于Windows 10企业版中
 - CredentialGuard使用硬件虚拟化功能(VSM)将证书/令牌的存储和管理和真实操作系统隔离，使得恶意程序即使拥有系统内核权限，也无法获取用户的证书，避免攻击者使用例如Mimikatz之类的工具来实现企业网络的进一步渗透
 - DeviceGuard则可以允许企业和锁定设备，禁止设备上安装未受信的软件
- 多因子安全认证的支持
 - 支持移动设备、生物识别、PIN码等多种方式的多因子认证保护，取代传统的Windows密码
- Edge浏览器的安全改进
 - 屏蔽了传统的ActiveX/BHO/Toolbars的扩展以及一些过时的组件如VBScript，减少了尤其是第三方控件引入的攻击面
 - 启用了全64位进程和增强保护模式沙箱保护 (IE11默认仅使用32位进程和保护模式沙箱)
 - 渲染引擎核心也增强了针对过去较多影响IE浏览器的一些漏洞的防护或缓解能力
- 内核模式字体引擎的安全改进
 - 将内核模式字体引擎部分分离并放入隔离的用户模式环境中运行 (即UMFD机制，User Mode Font Driver，用户模式字体驱动)，有效防止了通过字体漏洞直接入侵Windows内核的攻击方式，同时也增加了可以通过组策略禁止或审计非系统字体加载的功能
- CFG(控制流保护/执行流保护)内存保护机制
 - ControlFlowGuard(CFG)技术是Win 10中开始默认启用的一种抵御内存泄露攻击的新机制 (后来被Win 8.1 Update 3补丁加入)，用以弥补此前不完美的保护机制，例如地址空间布局随机化(ASLR)导致了堆喷射的发展，而数据执行保护(DEP)造成了返回导向编程(ROP)技术的发展。
 - CFG是一种编译器和操作系统相结合的防护手段，目的在于防止不可信的间接调用。漏洞攻击过程中，常见的利用手法是通过溢出覆盖或者直接篡改某个寄存器的值，篡改间接调用的地址，进而控制了程序的执行流程。CFG通过在编译和链接期间，记录下所有的间接调用信息，并把他们记录在最终的可执行文件中，并且在所有的间接调用之前插入额外的校验，当间接调用的地址被篡改时，会触发一个异常从而让系统介入处理
- 沙箱
 - 专用于执行潜在的危险软件和应用，使其无法触及主机操作系统或内核
 - Windows 10 Insider预览版19H1提供了该功能
 - 要求主机基于AMD64架构，且BIOS启用虚拟化功能，系统版本也必须是Windows 10 Pro或Windows 10企业版
 - 用户无需下载虚拟机通常要求的虚拟硬盘，只需拥有一个已安装有Windows 10的副本即可

4 Windows核心安全机制

4.1 Windows系统的安全等级

《可信计算机系统评估标准》(TCSEC, 橘皮书)

- 美国国家计算机安全中心，1983年定义的
- 将操作系统、网络组件和可信应用程序所提供的安全保护等级分为4大类，7小类，从高到低为：A1、B3、B2、B1、C2、C1、D
- 目前尚没有任何操作系统满足A1等级；仅有很少一部分操作系统满足B等级；C2等级被认为针对通用操作系统来说已经足够安全的了

TCSEC Rating Levels

Rating	Description
A1	Verified Design
B3	Security Domains
B2	Structured Protection
B1	Labeled Security Protection
C2	Controlled Access Protection
C1	Discretionary Access Protection (obsolete)
D	Minimal Protection

- D：9X, DOS, 无3A
- C1：废弃
- C2：含C1，任意访问控制、可控访问保护
- B1：军用

Windows从NT 3.5开始满足C2级别的安全性：

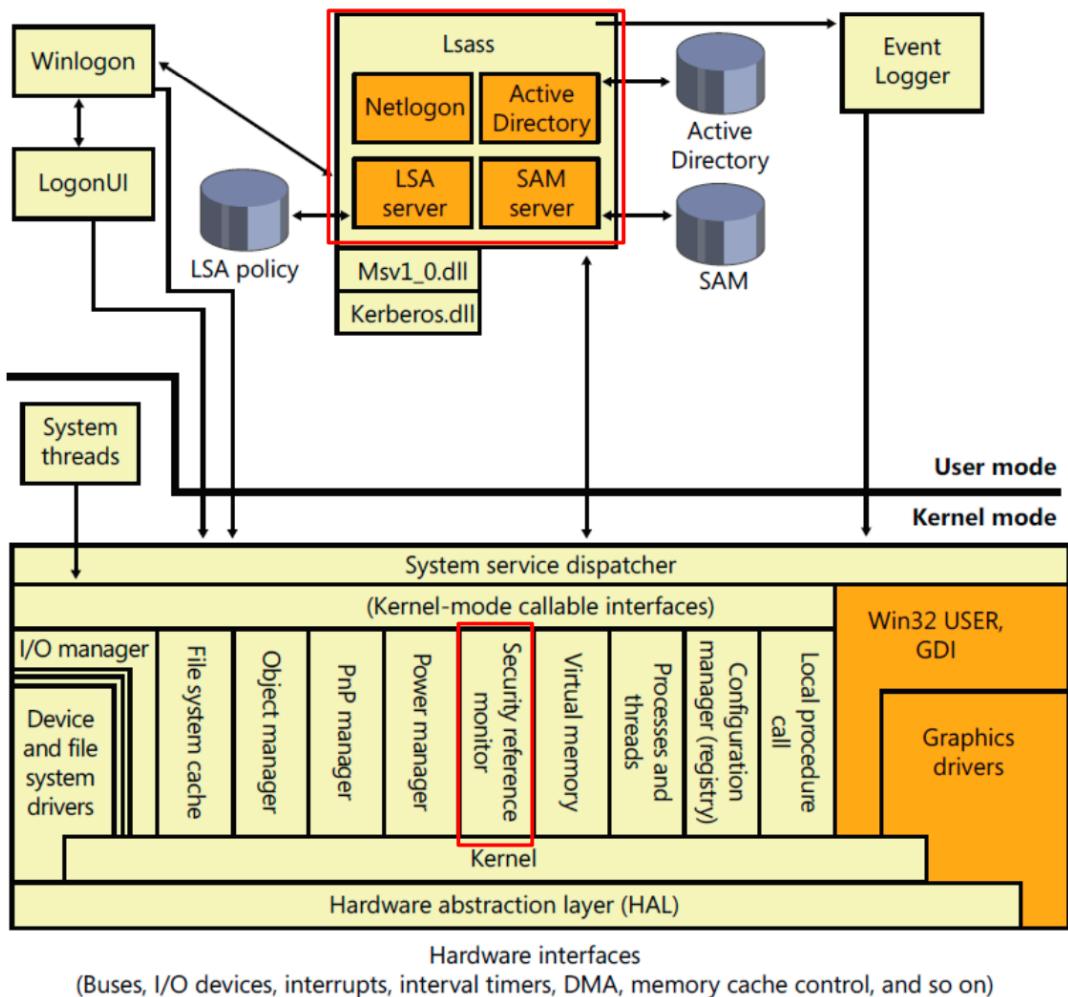
- 安全的登录设施 (身份认证)
 - 所有用户都必须以唯一的登录标识和密码来鉴别自身，从而跟踪用户的活动，而且只有授权的用户才能访问相应的系统资源
- 自主 (任意) 的访问控制 (DAC)
 - 资源的所有者能够决定谁可以访问该资源，以及他们可以对它做什么
 - 任意的访问控制 ~ C：创建文件的主体决定
 - 强制的访问控制 ~ B：统一由管理员安全策略决定
- 安全审计
 - 能够检测和记录与安全性相关的事件，或者任何想要创建、访问或删除资源的意图
- 对象重用保护
 - 防止用户看到其他用户已经删除的数据，或者访问到用户原先已经使用过后来又释放了的内存

Windows也满足B级别安全性的以下两个要求：

- 可信路径功能
 - 防止木马程序在用户登录的时候能够截取到用户的名称和口令。在Windows中，可信路径功能是以Ctrl+Alt+Del安全维护序列的形式来做到的，非特权应用程序无法截获到此序列

- 服务器电脑、加入域的个人电脑才能看到
- 可信设施管理
 - 要求针对管理功能有单独的账户角色作为支撑。例如，针对管理工作(管理员)、负责计算机备份的用户和标准用户分别提供单独的账户

4.2 Windows系统的安全组件



本地安全授权子系统 LSASS:

- 用户态
- **Lsass.exe:** 映像文件的用户态进程，负责本地系统安全策略、用户认证，以及发送安全审计信息到事件日志
- 用户身份验证和权限管理：
 - 负责交互式身份验证
 - 生成安全访问令牌
 - 身份验证成功后生成令牌
 - 用户之后创建的所有进程都有此令牌，代表用户身份
 - 令牌中无用户名，只有SID
 - 令牌内容：当前用户、所属组、拥有的特权 ...
 - 分配用户特权
 - 在令牌中

- 确定用户权限
 - 不在令牌中，实时由LSASS决定
 - 安全策略管理
 - 管理本地安全策略
 - 偏向于密码策略
 - 管理审核策略
 - 对象管理
 - 建立可信任域列表
 - 内存的配额管理
-

安全参考监视器 SRM

- 内核态，后2个A在内核态的实现在此完成，身份认证全在用户态完成
 - 是Windows执行体ntoskrnl.exe(内核态)的一个组件，负责执行对象的访问控制、管理特权(用户权限)以及生成所有的安全审计信息
 - 访问控制和特权管理
 - 根据LSASS配置的安全访问控制策略，联合对象管理器(Object Manager)，负责对所有安全主体访问Windows资源对象的授权访问控制
 - 安全审计
 - 根据LSASS配置的安全审计策略，对访问过程中关注的事件进行记录，并由事件日志服务生成系统审计日志
 - LSASS进行策略定制，由SRM生成，并在用户态的事件日志服务进行格式化、存储后查看
-

LSASS策略数据库

- 包含本地系统安全策略设置的数据库，数据库存储在注册表 HKLM\Security 子键下面，包含：
 - 哪些域是可信任的，从而可以认证用户的登录请求
 - 谁允许访问系统，以及如何访问(交互式登录、网络登录，或者服务登录)
 - 分配给谁哪些特权
 - 执行哪一种安全审计
 - 域登录在本地缓存的信息
 - Windows服务的用户—账户登录信息
-

Windows的核心安全机制 (3A)

- 身份认证——基石
- 访问控制和授权
- 审计

4.3 Windows的身份认证

Windows NT内核的操作系统都是多用户操作系统，需要进行身份认证/验证来保证安全性

4.3.1 Windows账户的标识

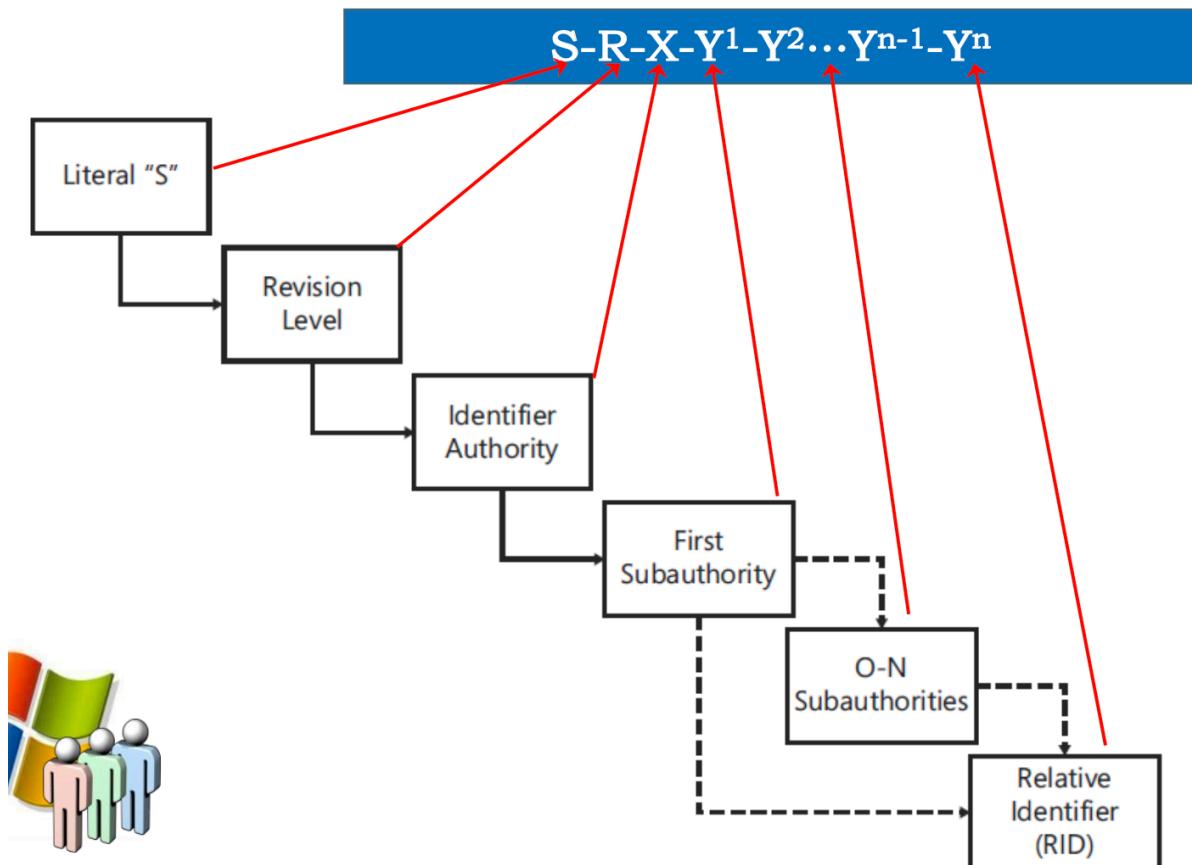
安全标识符 (SID)

- Windows使用SID来唯一表示安全主体，包括用户和组
- Windows NT 6.x中，系统服务也有SID标识
- 查看账户的SID
 - whoami /user
 - PsGetSid——能看任何用户的SID

一个用户，加密了一些文件，后来此用户被删除，这些文件成了无主文件，然而这些文件只有此用户能访问，怎么办？

若文件被加密，则没有正规解决方案，只有不正规的

- 添加用户：net user test /add
- 删除用户：net user test /del
- 添加用户：net user test /add
- SID不同



- S：表示该字符串为一个SID
- R：表示SID结构的版本号，Windows系统中都为1
- X：表示标示符颁发机构
 - Windows特定的账户/组，如Administrators，该值为5
 - Everyone等通用的账户/组，该值为1

- Y1-Yn-1：表示子级颁发机构，标识各级不同的域
 - Administrators组的域标识符是32(Builtin)
 - Everyone组没有域标识符
- Yn：表示域内特定的账户和组，也叫相对标识符
 - Administrators组的相对标识符是544
 - Administrator的相对标识符是500， Guest是501
 - Everyone组的相对标识符为空

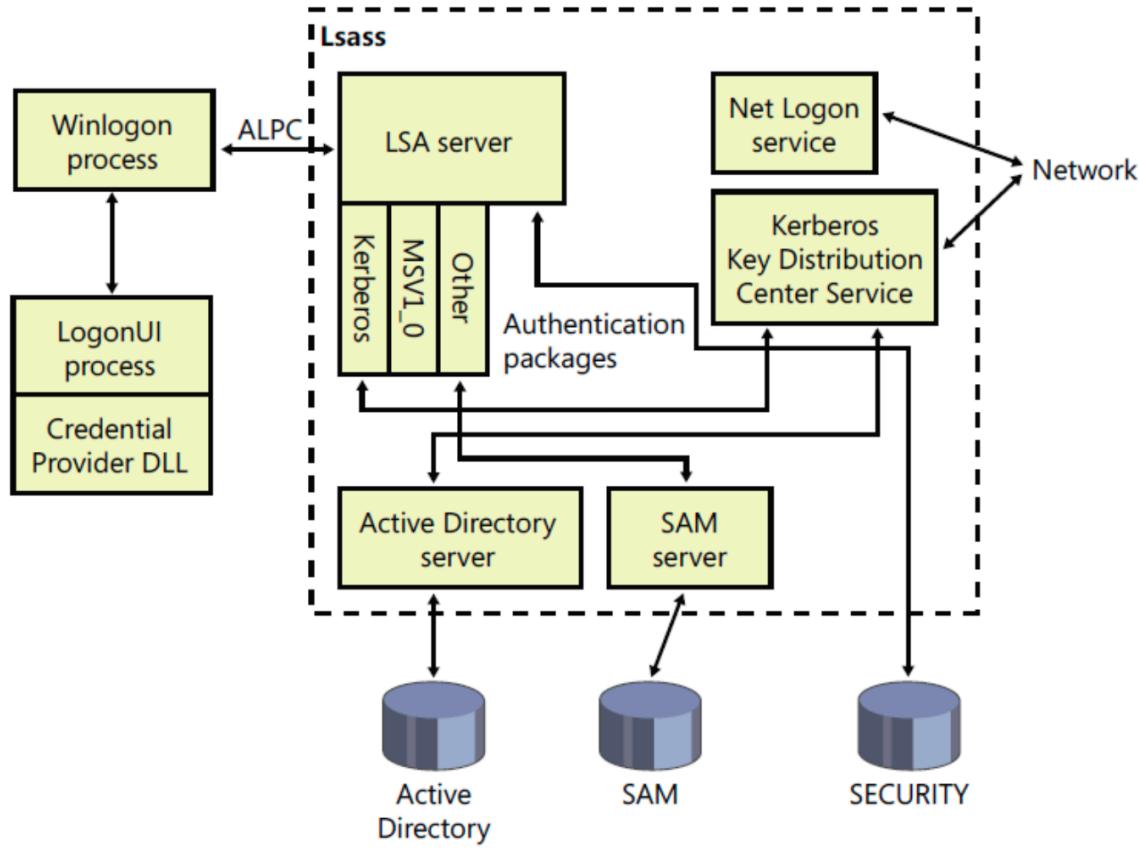
RID	Description
500	Administrator
501	Guest
502	Krbtgt
512	Domain Admins
513	Domain Users
514	Domain Guests
515	Domain Computers
516	Domain Controllers
544	Built-In Administrators
545	Built-In Users
546	Built-In Guests

- psgetsid guest、 psgetsid SID (反向解析用户名)

服务程序的SID：

- 从Windows Vista/Server 2008 (NT6.0)开始， 每个服务程序都有自己的SID， 而账户的名称则为“NT SERVICE \<服务名称>”
- 查看服务SID及其账户名称
 - sc.exe showsid <service name>
 - psgetsid <sid> (反向解析)

4.3.2 Windows的身份认证登录



左边为前端：

- Windows登录进程、界面、凭证提供dll
- 5.X与6.X的区别仅在于此，5.X没有后面2个

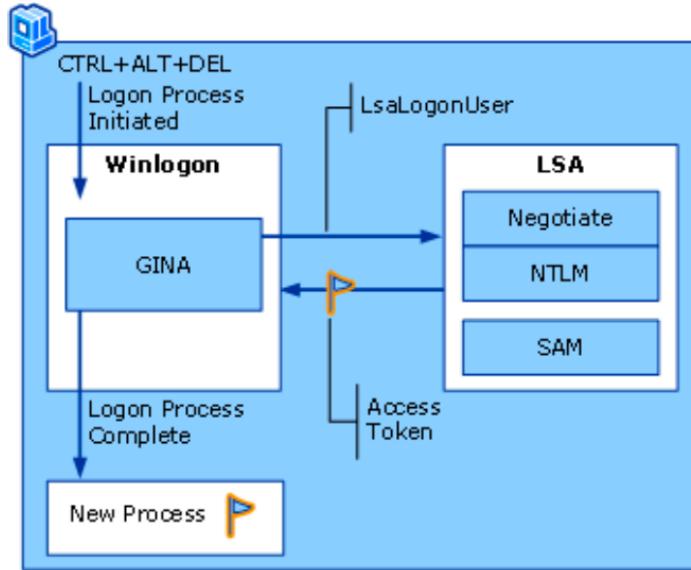
中间为后端：

- 身份认证程序包 (dll文件)：Kerberos (域登录)、MSV1_0 (本地登录)、Other
- 活动目录服务：用于域登录
- SAM服务：用于本地登录
- 右上角两个与网络有关，域登录涉及到网络，将身份验证信息通过网络发送到域控制器上

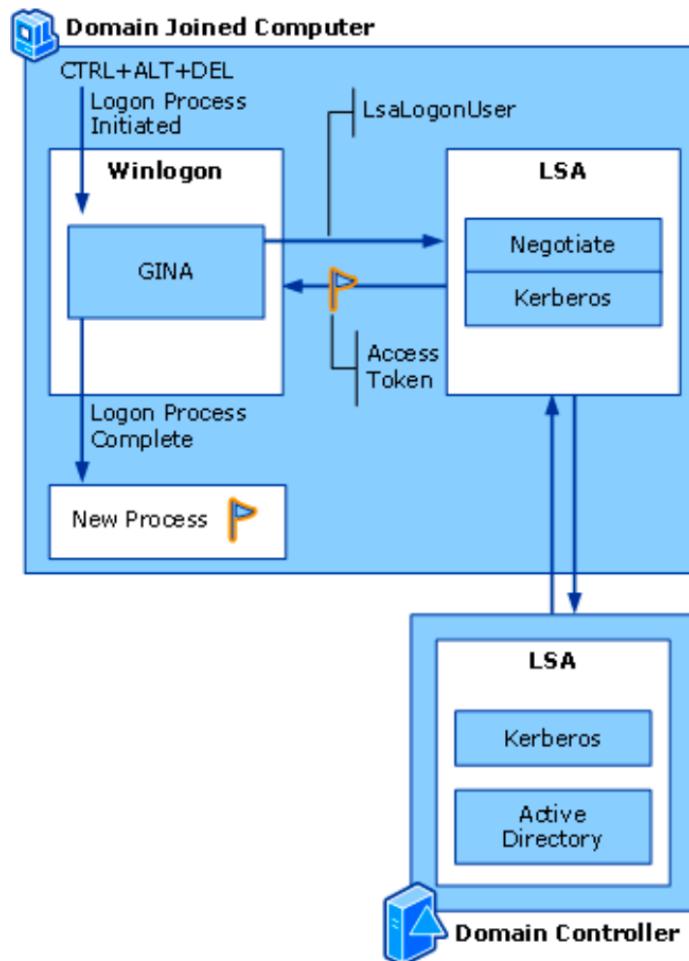
下面为数据库

Windows NT 5.x 身份交互式登录验证组件：

- 交互式本地登录



- 初始化后进入Winlogon界面，GINA为5.X独有
- LSA中协商，利用NTLM协议进行验证，返回访问控制令牌
- 数据在SAM中
- 由Winlogon生成新的进程，即桌面explorer.exe进程
- 交互式本地登录



- 验证的地方不一样，到域控制器上验证
- 利用Kerberos协议，数据在活动目录中，有所有域用户的凭证信息

Winlogon:

- 维护与用户登录相关状态切换的前端界面
- 可信任的进程，负责管理与安全有关的用户交互
 - 协调登录过程，在登录时启动用户第一个进程，处理注销过程，以及管理其他各种与安全有关的操作，包括在登录时输入口令、更改口令、锁住/解锁工作站等
 - 创建可用的桌面
 - 向操作系统注册一个安全维护序列(SAS)，默认为Ctrl+Alt+Delete
 - 维护工作站状态
 - 实现超时处理
- 向GINA发送事件通知消息，提供可供GINA调用的各种接口函数
- 保证其操作对其他进程不可见，从而防止登录密码等信息被截获
- 登录框被特殊处理，输入框理论上无法被挂钩

GINA动态链接库：

- 提供了Winlogon用户标识和验证用户的输出函数
- 默认的GINA是%systemroot%\system32\Msgina.dll
- 允许被用户替换来自行定制系统的用户识别和身份验证
 - [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon]
 - "GinaDLL"="ginadll.dll"
 - 若想获取用户输入口令，挂钩不可能，但可通过替换ginadll.dll来实现

LSA 本地安全授权：

- 主体文件：%SystemRoot%\System32\lsass.exe

身份验证程序包：

- 身份验证程序包的任务
 - 验证用户
 - 返回绑定到用户安全令牌中的SID
 - 返回SID给LSA，LSA再通过安全策略数据库确定特权，基于上述信息生成访问控制令牌
- 身份验证程序包位于DLL动态链接库中
 - 在系统启动期间被LSA所链接，接受输入的登录证书，通过验证程序决定是否允许用户登录
- Windows安装的身份验证程序包
 - 独立(工作组)环境：MSV1_0，%SystemRoot%\System32\Msv1_0.dll
 - 域环境：Kerberos，%SystemRoot%\System32\Kerberos.dll
- 当用户以交互方式登录到本地计算机上时，或者当没有域控制器可以访问时，Windows使用MSV1_0作为认证包
- 当用户以交互方式登录到一个域中时，Windows使用Kerberos作为认证包

SAM 安全账户管理器：

- 本地账户信息存储在SAM文件/注册表中
- %SystemRoot%\System32\config\SAM
- 通用，SAM子键即SAM文件的解析，安全认证心脏文件

Active Directory，活动目录：

- 域账户信息存储在域控制器(DC)上的活动目录中
- 文件名: Ntds.dit文件, 心脏文件

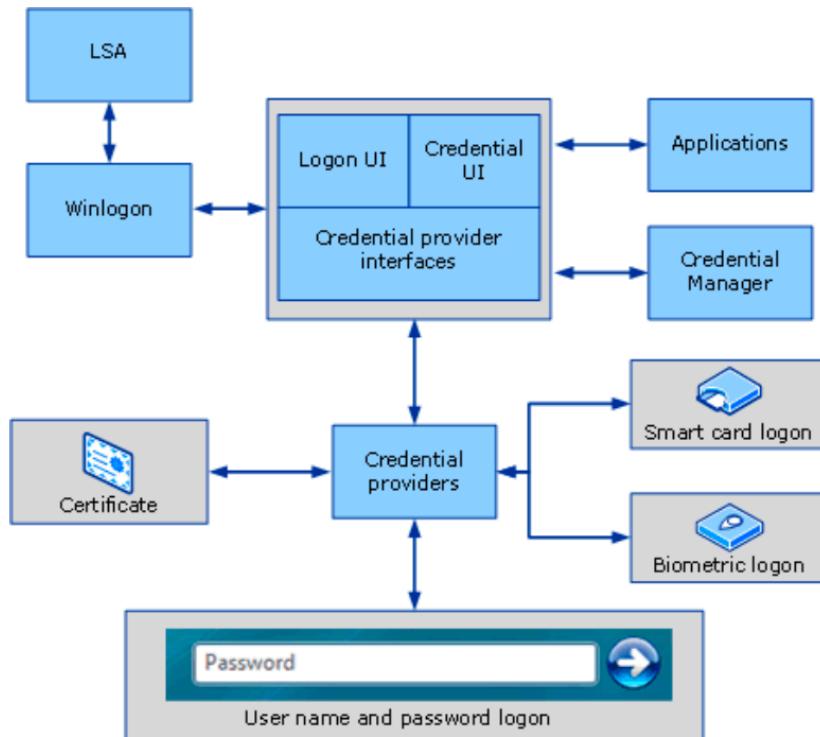
New Process, 新创建的过程:

- 用户初始化进程, %SystemRoot%\System32\userinit.exe
- 用户外壳进程, %SystemRoot%\explorer.exe
- userinit.exe消失, 并重新设置explorer.exe的父进程

Netlogon, 网络登录服务:

- 一种Windows服务, 用以创建和域控制器之间的安全通道, 发送安全请求, 既支持LM/NTLM的身份验证, 也支持活动目录登录
- %SystemRoot%\System32\Netlogon.dll

Windows NT 6.x 身份交互式登录验证组件



- 5.X是GINA

LogonUI, 登录用户接口:

- 为保护Winlogon进程不崩溃, LogonUI由Winlogon按需启动, 真正加载Credential provider, 为用户提供验证身份的图形界面
- LogonUI是用户模式下的进程, 即%SystemRoot%\System32\LogonUI.exe
- GINA天生由Winlogon加载, 而LogonUI按需启动

CP, 凭证提供者:

- Credential provider是运行在LogonUI进程内的COM对象, 位于一些DLL文件中(触发SAS热键后由Winlogon按需启动), 用来获取用户的用户名、密码、智能卡PIN码或者生物数据(比如指纹、视网膜信息)
- 标准的CP是%SystemRoot%\System32\authui.dll

和%SystemRoot%\System32\SmartcardCredentialProvider.dll

5.X与6.X身份验证登录的区别：不用GINA了，LogonUI按需启动，区别在前端，后端无区别

4.3.3 安全账户管理器(SAM)及数据库

SAM 服务：

- 运行在LSASS进程中的一组子例程，负责管理SAM数据库，即Windows用户账户数据库的控制与维护

SAM数据库

- 位于受到保护的注册表及对应的磁盘文件中。系统运行期间被SYSTEM帐号锁定，即使是Administrator帐号也无法打开
 - 注册表只是文件的解析，归根结底是在文件
 - 左下角有小锁的：被system账户锁定，打不开、删不掉、复制不了
- SAM数据库包含了所有的用户账户、组账户信息以及每个用户密码的两个加密散列值。
 - LM 密码散列
 - Windows Vista之后默认都不再保存LM散列
 - NTLM密码散列
- System32/config

secpol.msc：打开本地安全策略，域管理员定制的处理优先级比本地定制的高

Windows账户数据库(SAM)的加密保护：

- SAM锁定工具：“开始”→“运行”→ 输入“syskey”
- 密码启动：自己指定解密SAM所需的密码，这样每次启动系统时需要输入密码
 - 安全，忘了就坏了
- 在软盘上保存启动密钥：每次启动时需要插入这张软盘
- 在本机上保存启动密钥：默认设置
 - 不安全，syskey可以拿到
 - 在System32/config/SYSTEM，即HKLM/SYSTEM中，保存了syskey，可对SAM解密

SAM数据库的注册表分析：

- SAM数据库位于注册表 HKLM\SAM\SAM 下
 - 管理员组打不开这个子键，提权也打不开，用户SYSTEM可以
 - 想看到注册表内容的常规方法：psexec -s -i -d regedit.exe
 - s：以SYSTEM账户运行
 - i：服务与本地控制台可有交互
 - d：不等待进程结束而直接结束
 - 非常规方法，有写权限的权限(写入DAC)，所以可自己改权限
- SAM数据库在磁盘上保存在 %systemroot%\system32\config\ 目录下的“SAM”文件中
- 注册表HKLM\SAM\SAM\Domains\下就是域(或本机)中的SAM内容，包含“Account”和“Builtin”子键

- \Domains\Account是用户帐号内容
- \Domains\Account\Users下就是各个帐号的信息。其下的子键就是各个帐号的SID相对标志符，即RID。比如000001F4是Administrator的RID。每个帐号下面有两个子键，F和V：
 - “V”中保存的是账户的基本资料，用户名、用户全名、所属组、描述、密码散列(经过额外的加密保护)、注释、是否可以更改密码、账户是否启用、密码设置时间等
 - “F”中保存的是一些登录记录，比如上次登录时间、错误登录次数等，还有一个重要的地方就是这个帐号的RID
- Windows克隆账户：
 - V值不动，覆盖F，比如使得Guest登录后有管理员权限
 - SID从F值取出，RID导出，令牌里有管理员的SID
 - 检测：F值的RID与本身子键标题写着的是否相同

伪造一个用户，SID与原用户相同，注册表中可篡改SID

账户密码存储在SAM中的加密算法：

- 散列长度：128bits
- LM散列算法
 - 如果明文密码不足14位，就用0x00把密码补足14位，如果超出14个字节，则取前14个字节。将密码的所有字母转成大写字母，然后分为两个部分，每部分7个字节
 - 将每部分的7个字节(56bit)的每7bit后面添加1bit的0，从而扩充为64bit作为DES的密钥。然后使用该密钥来加密魔法字符串“KGS!@#\$%”，分别得到两个8字节的密文，最后将这两部分密文合并得到最终的16字节密码散列
- NTLM散列算法
 - 将密码转换成UNICODE字符形式，然后使用MD4算法进行散列运算
 - 20个字符以上的NTLM，可以认为安全

Windows账户密码的破解：

- SAM文件的获取方法
 - 物理接触主机、启动其它操作系统(PE系统)来获取Windows分区上的%SystemRoot%\system32\config\sam文件
 - 所有权限保护失效
 - 可通过文件加密防范
 - 使用pwdump (运用远程线程插入)、wce、gsecdump、copypwd等工具从Lsass内存或者注册表中导出SAM散列值，转储成类Unix系统的passwd格式文件
- Windows本地密码散列导出工具
 - Pwdump, wce, gsecdump, copypwd, QuarksPwDump
- Windows本地密码破解工具
 - L0phtCrack, SAMInside, Ophcrack
- Windows本地已登录账户的明文密码提取工具
 - mimikatz, wce

- mimikatz分析LSASS进程的内存，可能提取明文密码、散列、证书等
 - 彩虹表破解：
 - 一种“以空间交换时间”的技术
 - 由于LM/NTLM散列算法没有Salt机制，因此可以通过事先计算出所有可能密钥的加密值并存储成一张表，解密时只需在该表中进行搜索即可
 - 时间可以缩短至几秒至几分钟
 - 结合GPU硬件破解，更可以大大缩短破解时间
-

3.3.4 Windows身份验证协议

NTLM与Kerberos的区别：

1. NTLM是Windows固有的，Kerberos是微软把它应用进来
 2. NTLM在非域环境中采用，Kerberos是域环境默认采用
 3. NTLM底层逻辑基于质询/响应，Kerberos基于票据
 4. NTLM单向验证，Kerberos双方验证
-

LM和NTLM身份验证协议：

- 是Windows固有的身份验证协议
- 在Windows非域环境中采用，包括：
 - 工作组环境
 - 域环境，但是客户端或者服务器不是域成员
- 基于“Challenge/Response”(质询/响应)模式的验证协议，仅提供单向验证
 - 客户端无法验证服务端
 - Challenge由S发给C，每次都会改变，对抗重放攻击
 - 简化模型：Client → Hash(Hash(Pass) + Challenge) → Server
 - 中间人攻击：指定Challenge

Windows Server 2003之前：默认“发送LM和NTLM响应”

Windows Server 2003：默认“仅发送NTLM响应”

- LM/NTLM响应：在16字节的LM/NTLM Hash值后面加上5个空字节，总共21个字节，然后平均分割成三个7字节，再进行拼接而成三个8字节，分别对8字节的Challenge进行DES加密计算，最后得到24字节的Response

Windows Vista/7/Server 2008：默认“仅发送NTLM v2响应”

- NTLMv2：NTLM的改进版，使用NTLM Hash值(NTOWF)，在计算时还包含一个客户端的Challenge
- 太复杂，以下为实际使用的

协商支持NTLMv2会话安全(NTLMSSP, NTLM++)：

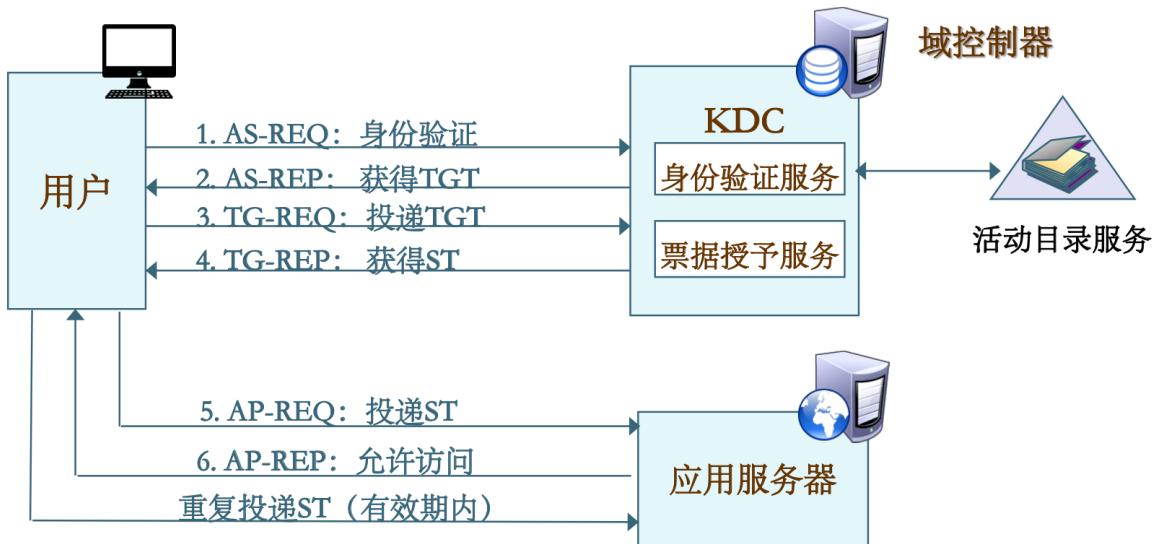
- NTLM Session Security，安全性介于NTLM和NTLMv2之间
 - 只要服务器支持，则协商一致后采用，现在实际上基本都采用这种级别的身份验证协议
-

Kerberos身份验证协议：

- 基于公私钥加密体制，为分布式环境提供双向验证
- Kerberos是Windows域环境中的默认身份验证协议
 - 满足用户的单一登录特性
 - 验证一次，用所有服务
- Kerberos协议是一个基于票据(Ticket)的系统
 - 客户通过网络向Kerberos密钥分发中心 (KDC) 进行身份验证并请求票据用以访问网络应用资源

Kerberos密钥分发中心 (KDC) 提供的服务

- 身份验证服务 (AS)：对用户进行验证，发行TGT (票据授予票据)，该票据供用户后续请求会话票据之用。
- 票据授予服务 (TGS)：在发行给客户的TGT的基础上，为应用服务发行ST (会话票据)



- TGT由krbtgt账户的密码散列加密，ST由应用服务器的机器账户密码散列加密
- Windows服务器和工作站都包括一个Kerberos安全支持提供程序 (SSP)

4.4 Windows的访问控制

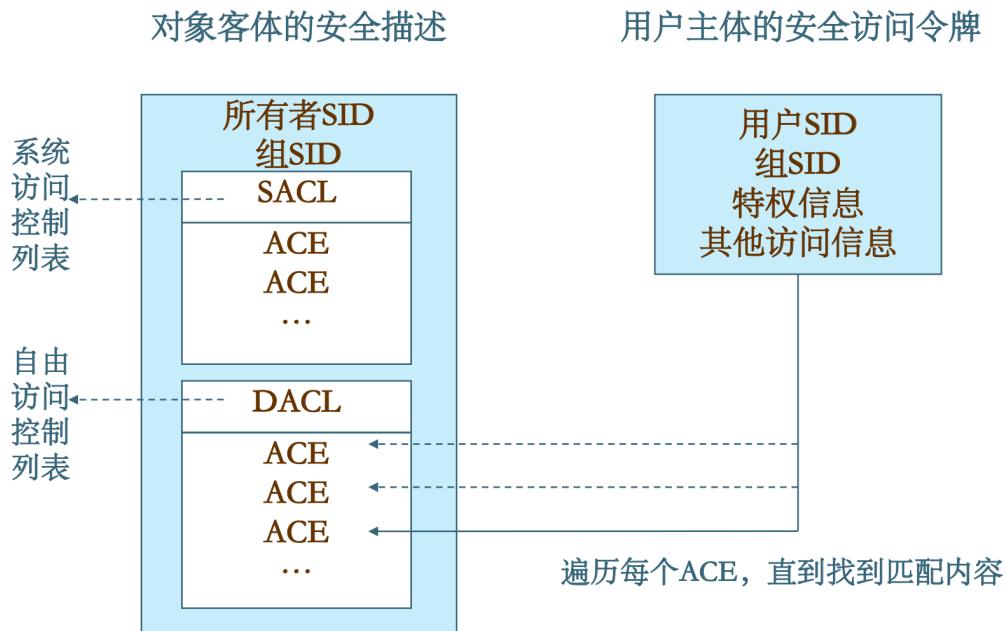
访问控制的目的：

- 限制安全主体(用户、进程、服务等)对客体(文件目录、注册表键等)的访问权限，从而使系统在安全合法范围内使用
- 客体包括：
 - 文件、目录
 - 注册表键
 - 活动目录对象
 - 内核对象(事件、信号量、互斥体)
 - 服务
 - 进程、线程
 - 防火墙端口
 - 窗口工作站和桌面
 - ...

Windows访问控制的本质：

安全主体的访问令牌 ← 比较 → 客体的安全描述

- 用户成功登录时，系统的本地安全授权(LSA)为其创建访问令牌
- 用户启动程序创建进程时，线程获取该令牌的拷贝
- 用户程序请求访问客体时，向系统提交该令牌
- 系统使用该令牌与客体的安全描述进行比较来执行访问检查和控制



- 客体所有者：默认谁创建谁拥有
- SACL：系统访问控制链表，只与审计有关
 - 谁访问我或怎么访问我，是否需要记录下来
- DACL：自由访问控制链表
 - 谁能读、写、执行 ...
- 用户名不在令牌中，可由SID解析得到
- NTFS支持访问控制、审计，每一个对象都有安全描述，占磁盘空间大

4.4.1 主体的安全访问令牌

Security Tokens:

- 用户帐号SID (User account SID)
- 用户所属组的SID (Group 1-n SID)
 - 可不止一个
- 受限SID (Restricted SID 1-n)
 - 限制一些权限
- 会话ID (Session ID)
 - 所属会话的ID
- 特权列表 (Privilege1-n)

- 与该令牌关联的特权列表
- 源 (Token Source)
 - 创建此令牌的实体，比如会话管理器、网络文件服务器，或者RPC服务器等
- 完整性级别 (Integrity Level)
 - Windows Vista开始引入，用以限制读取/写入对象的范围
 - 在进程中打一个标记，给进程分级别，不同场景级别不同
 - 5.X用户创建的进程一律平等，木马进程随意操作
 - 系统、高、中(默认)、低、容器
- 默认自主访问控制列表 (Default DACL)
 - 用以创建对象时为其设置的初始DACL
- 令牌类型 (Type)
 - 主令牌(primary)——默认
 - 模拟(implementation)令牌——一些服务提供者模拟用户访问
- 模拟级别 (Implementation Level)
 - 4个级别限制能够模拟的范围

查看访问令牌中的信息：

- process explorer：进程属性对话框的“安全属性”页面
-

令牌类型：

- 主令牌：
 - 每个进程都有一个主令牌来描述与该进程相关的用户帐号的安全上下文
 - 主线程使用主令牌
 - 模拟令牌：
 - 服务进程在自己的帐号下运行，使用自己的主令牌，但当服务接受一个客户的访问请求时，它创建一个线程来完成这项工作并将客户的访问令牌与工作线程相关联。客户的访问令牌是一个模拟令牌，用来标识客户、客户的组和特权
 - 当线程代表客户请求访问资源(比如文件、打印机、数据库等)时，在访问检查过程中使用该信息。在模拟结束后，线程重新使用主令牌并返回到服务自己的安全上下文里操作
 - 服务器只能在发起模拟请求的线程内模拟客户
-

模拟级别：

- 为了防止滥用模拟机制，Windows不允许服务器在没有得到客户同意的情况下执行模拟。客户进程在连接到服务器的时候可以指定一个安全服务质量，以此来限制服务器进程可以执行的模拟级别
 - SecurityAnonymous级别：最为受限制，服务器不能模仿或者识别出客户
 - SecurityIdentification级别：允许服务器得到客户的SID和特权，但不能模拟该客户
 - SecurityImpersonation级别：默认级别，允许服务器在本地系统上识别和模拟该客户
 - SecurityDelegation：最为随意，允许服务器在本地系统或远程系统上模拟该客户
 - 服务器完全变成了客户，一个级别不高的进程拥有级别更高的模拟令牌，实现提权，很多提权漏洞出自此

受限制的令牌：

- 出于安全原因，应用程序可创建一个受限的令牌并将它分配给子进程或模拟线程来为它们创建受限的安全上下文
 - 浏览器创建子进程
- 受限令牌在主令牌或模仿令牌的基础上创建的，来源于令牌的拷贝，但可能进行如下修改：
 - 从该令牌的特权列表中删除一些特权
 - 该令牌中的SID被标记为仅仅拒绝(Deny-only)
 - 该令牌中的SID被标记为受限制的(Restricted)
- 在进行对象访问检查的时候，系统要执行两次访问检查：一次使用令牌的“Enable SID”和“Deny SID”，另一个使用Restricted SID的列表。只有当这两个访问检查都允许请求的访问权限时才能允许访问

受限令牌的应用——被过滤的管理员令牌(UAC权限提升之前与之后的区别)：

- 完整性级别被设置为中等，提权后为高
- 除了ChangeNotify、Shutdown、Undock、IncreaseWorkingSet、Time Zone以外的其他特权都被去掉，提权后所有都有
- 给管理员以及类管理员的SID标记“Deny-only”，可以消除以下安全缺陷
 - 比如某个文件拒绝管理员组用户的访问，但允许其他包含这个管理员用户的组的访问，这将会导致这个用户最终能够访问该文件
 - 要对管理员限权，要明确拒绝其可以干某事
 - 拒绝优先于允许

账户权限和特权：

- 进程在运行过程中执行的许多操作是无法通过对象访问保护来授权控制的，因为这些操作并没有与一个特定的对象打交道。Windows使用特权(Privilege)和账户权限(Account Right)，以使得管理员可以控制哪些账户能够执行与安全相关的操作
- 特权：是指一个账户执行某个与系统相关的操作的权限，比如，关闭计算机或者改变系统的时间
 - 在令牌中
 - 与用户权限不同，不同的特权是由不同的组件来定义的，并且也是由这些组件来强制使用的。比如调试特权是由进程管理器来检查的，它使得一个进程在利用Windows API函数OpenProcess来打开另一个进程的句柄时可以绕过安全检查
 - 与账户权限不同，特权是可以被允许和禁止的。想让一个特权检查能够成功地通过，该特权必须出现在当前特定的令牌中，而且它必须是允许的
 - 超级特权：
 - 调试程序(SeDebugPrivilege)
 - 具有此特权的用户可以打开系统中的任何一个进程，而不必考虑该进程上的安全描述符
 - 接管所有权(SeTakeOwnershipPrivilege)
 - 特权持有者能够接管任何一个被保护对象的所有权
 - 管理员组有此权限

正规方法：管理员经UAC提权，重新指定无主文件所有者

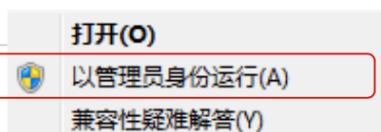
- 恢复文件和目录(SeRestorePrivilege)
- 加载和卸载设备驱动程序(SeLoadDriverPrivilege)
- 创建一个令牌对象(SeCreateTokenPrivilege)
 - 变成LSA了
 - 作为操作系统的一部分来执行(SeTcbPrivilege)
- 权限：把“执行某一特定登录类型(比如本地登录或交互式登录到一台计算机上)的能力”授予它所针对的账户，或拒绝分配给该账户
 - 与登录交互有关
 - 账户权限并不是由安全引用监视器(SRM)强制实施的，也不存储在令牌中
 - 当一个用户企图登录到系统中时，作为对登录请求的响应，LSA从LSA数据库中获取到已赋予该用户的账户权限，对登录类型(交互式登录、网络登录、批作业方式登录、服务方式登录、终端服务登录等)进行检查，如果该用户的账户没有“允许此种登录类型”的权限，或者具有“拒绝此种登录类型”的权限，则LSA拒绝该用户的登录请求

完整性级别：

- 强制完整性控制 (MIC)
 - Windows Vista之后开始引入，将进程分为不同的完整性级别，从而保证低级别的进程无法影响高级别的进程
 - 进程的完整性级别由令牌中的SID声明
 - 系统首先进行完整性级别的检查，然后再进行DACL的检查

Name (Level)	Use
Untrusted (0)	Used by processes started by the Anonymous group. It blocks most write access.
Low (1)	Used by Protected Mode Internet Explorer. It blocks write access to most objects (such as files and registry keys) on the system.
Medium (2)	Used by normal applications being launched while UAC is enabled.
High (3)	Used by administrative applications launched through elevation when UAC is enabled, or normal applications if UAC is disabled and the user is an administrator.
System (4)	Used by services and other system-level applications (such as Wininit, Winlogon, Smss, and so forth).

完整性级别的SID



- 1：保护模式下的IE、Adobe Reader
- 3：UAC启用时经提权的管理员、UAC禁用的管理员
- 4：服务程序、一些系统级别的程序

查看进程的完整性级别：

- Process Explorer

- AccessChk: accesschk -v -p <processname>

查看对象的完整性级别:

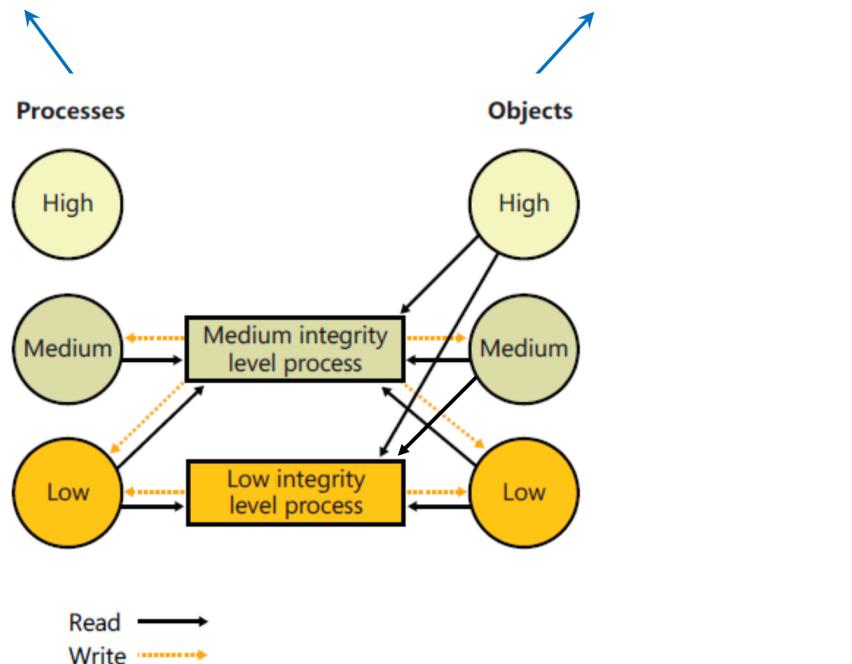
- AccessChk: accesschk -v <objectname>

进程的完整性策略中包括:

TOKEN_MANDATORY_NO_WRITE_UP
TOKEN_MANDATORY_NO_READ_UP

对象默认的完整性策略:

TOKEN_MANDATORY_NO_WRITE_UP
TOKEN_MANDATORY_NEW_PROCESS_MIN



- 中间为主体对象，左右为被访问客体
- 左边为进程，右边为文件目录等传统对象

以低完整性级别加载应用程序:

- psexec-l notepad.exe
- accesschk -v -p notepad.exe

Windows Vista之后的“低权限保护/沙盒”模式:

- 操作系统: Windows Vista之后版本
- 应用程序: >=Internet Explorer7、Adobe Reader X

观察比较:

1. 完整性级别
2. 特权列表
3. 受限制的令牌

4.4.2 客体的安全描述

安全描述附着于对象:

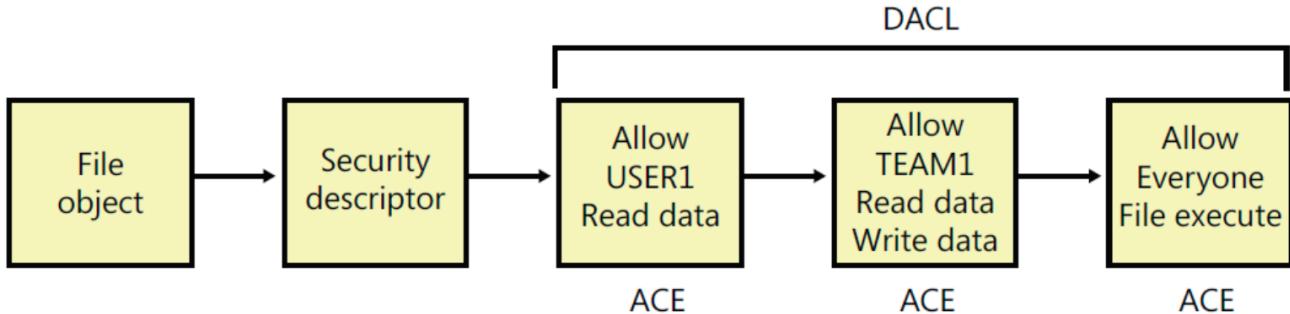
- 安全描述 (SD)
 - 版本号
 - 标志

- 所有者的SID
 - 所有者主要组的SID，仅被POSIX使用
 - DACL：自主访问控制列表，规定谁可以用什么方式访问对象
 - SACL：系统访问控制列表，规定哪些用户的哪些操作应该被记录到安全审计日志中
-

访问控制列表(ACL)：

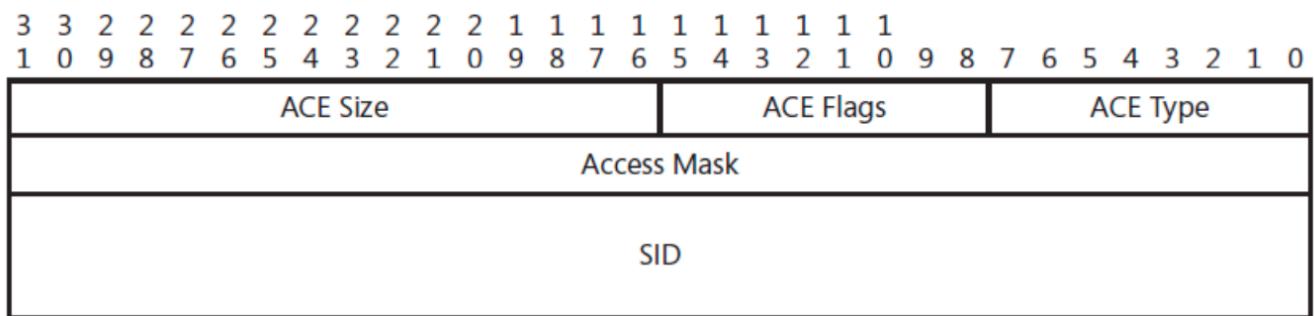
- 访问控制列表(ACL)是访问控制项(ACE)的有序列表
 - 若申请到的权限匹配到了，后面不会再匹配了
 - 有优先级，前面的先匹配到
-

DACL:



- 从左到右按顺序，每一项3个要素：
 1. 访问控制项类型：允许/拒绝
 2. 用户主体
 3. 干什么事，具体对应到访问屏蔽码
-

访问控制项(ACE)：



- ACE大小(Size)
- ACE标志(Flags)
- ACE类型(Type)
 - DACL包含ACE的类型：访问允许、访问拒绝、允许的-对象、拒绝的-对象等
 - SACL包含ACE的类型：系统审计、系统审计-对象
- 用户SID

访问屏蔽码(Mask):

- 32位，每一位对应着该对象的访问权限，可设置为1(置位，表示授予相应操作允许或拒绝的访问权)或0(未置位，未授予相应操作允许或拒绝的访问权)
- 0不是拒绝的意思，只是没有匹配上，继续往下扫描

3	3	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
G	G	G	G	Reserved	A	Standard Rights										Object-Specific Rights															
R	W	E	A	S																											

访问控制项在列表中的顺序：

- 直接设置(显式)的ACE在从父对象上继承下来的ACE前面
- 从第一层父对象继承下来的ACE在第二层父对象(祖父)上继承下来的ACE前面，以此类推
- 同一层中，拒绝类型的ACE在允许类型的ACE前面



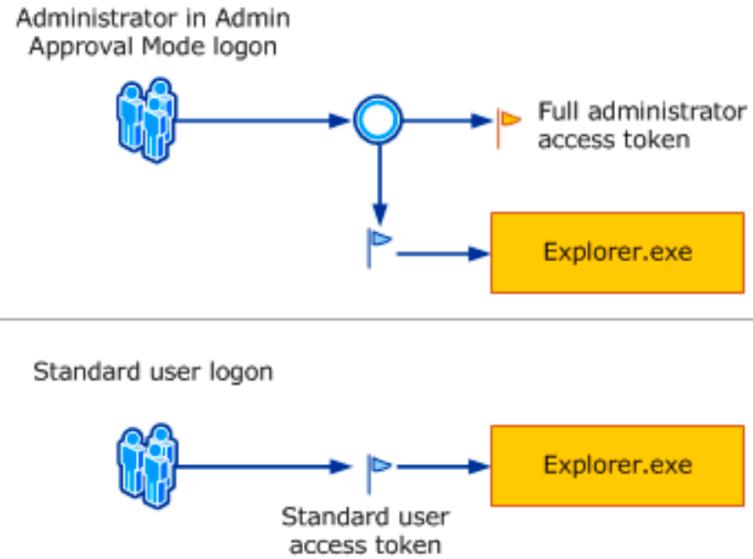
4.4.3 用户权限控制

RunAs服务(辅助登录服务)：

- 主要用于控制管理员，5.X管理员靠其限制，6.X之前只有此
- 管理员平时使用标准的用户账户登录，然后在必要的时候调用具有更高权限的管理员控制台来执行管理任务
- 不是默认，取决于管理员安全意识

用户账户控制(UAC)：

- 在Windows Vista之后的系统中，包括Windows7、8、10等，需要管理员权限的操作标有一个盾牌图标，表示首先需要经过系统UAC的提升确认才能进行进一步的操作
- 有盾牌的要强制提升，无盾牌的也可提升
- 默认行为，本质是对管理员进行降权，即缩减特权，用受限令牌创建外壳进程

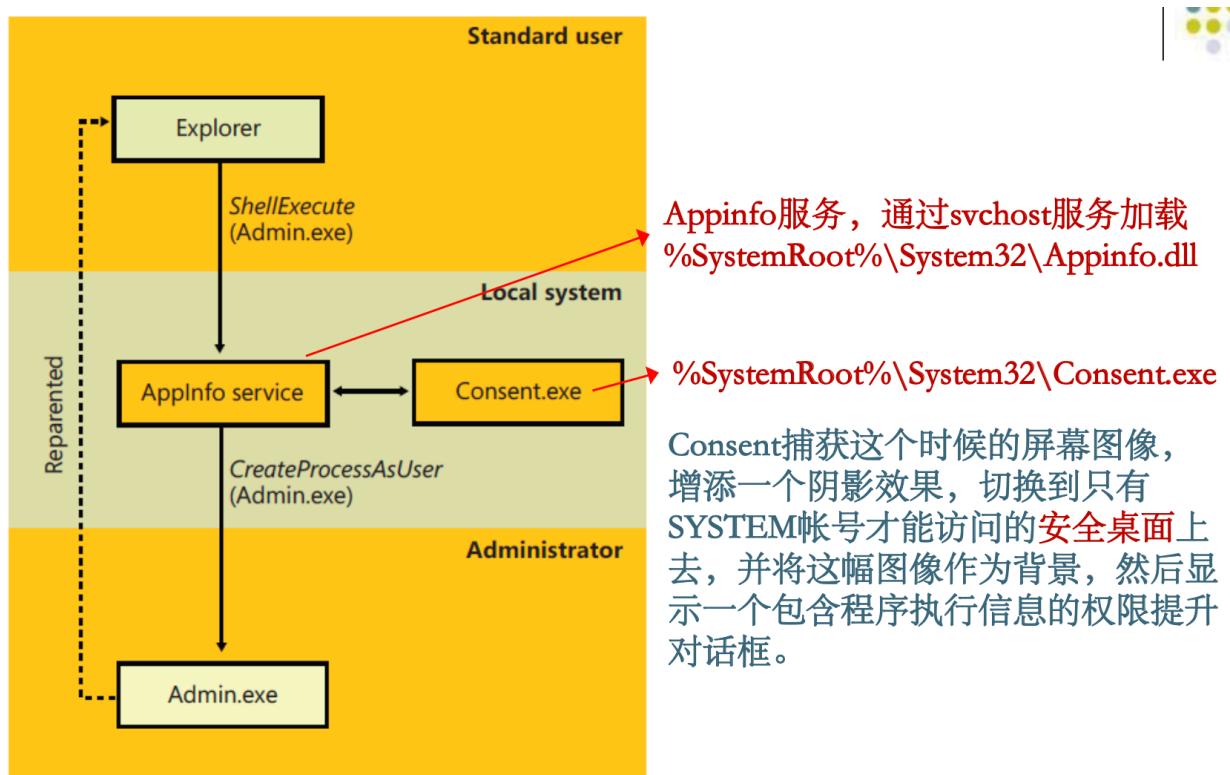


UAC权限提升对话框的颜色：

- 红色背景，带有红色盾牌图标：该程序的发布者被禁止，或者被组策略禁止
- 橘黄色背景，带有红色盾牌图标：该程序不被本地计算机信任(不包含可信任的数字签名，或数字签名损坏)
 - 比如自己编译的程序
- 蓝绿色背景：Windows自带的程序(包含微软的数字签名)
- 灰色背景：该程序带有签名且被本地计算机信任(带有信任的数字签名)

UAC权限提升的工作流程：

- 下面的安全桌面位于系统账户空间，恶意代码访问不到，不在一个上下文空间
- 最后重新设置其父进程为Explorer



应用程序请求管理员权限：

- 最常见的是在应用程序的“manifest”清单文件里包含一个“requestedExecutionLevel”标志
 - AsInvoker：默认值，此级别允许应用程序运行时使用与其父进程相同的令牌
 - HighestAvailable：此级别要求必须使用提供给当前用户的最高权限来运行应用程序
 - RequireAdministrator：此级别要求必须使用完整的管理员权限来运行应用程序

自动权限提升：

- Win7开始
- 大多数Windows可执行程序和控制面板组件并没有出现权限提升确认对话框，即便是需要管理员权限来运行
- 自动权限提升的条件：
 - 首先必须是Windows发行商（不只是Microsoft）签名的Windows可执行程序，而且必须位于某几个被认为安全的目录中，比如 %SystemRoot%\System32 及其大多数的子目录，%SystemRoot%\Ehome, %ProgramFiles% 里的一小部分目录（比如包含Windows Defender和Windows Journal的目录）
 - 然后是“manifest”清单里包含“autoElevate”元素的exe文件
- 但Windows还有一个权限自动提升的白名单，不需要它们包含“autoElevate”元素，但必须满足前一个条件（签名和路径位置），比如Pkgmgr.exe（包管理器）等
 - 有后门的感觉了

Windows 配置UAC：

- 内置管理员Administrator登录，UAC不起作用

滑块位置	当管理员用户没有以管理员权限运行…		备注
	尝试更改Windows设置，比如使用某些控制面板组件…	尝试安装软件，或者运行需要提升权限的程序，或者“以管理员身份运行”…	
最高的位置 (始终通知)	在安全桌面中出现UAC提升确认框	在安全桌面中出现UAC提升确认框	这是Windows Vista的行为
第二位置	自动产生UAC提升，没有提示或者通知	在安全桌面中出现UAC提升确认框	Windows 7的默认设置
第三位置	自动产生UAC提升，没有提示或者通知	在用户正常的桌面中出现UAC提升确认框	不推荐
最低的位置 (永不通知)	UAC为管理员用户关闭	UAC为管理员用户关闭	不推荐

用户界面权限隔离(UIPI)

- UIPI是UAC机制的一部分，目的在于防止窗口消息攻击
- 通过结合完整性级别控制，UIPI具体所做的保护包括：
 - 低级别进程无法对高级别的窗口句柄做验证
 - 低级别进程无法向高级别进程的窗口发送消息（SendMessage或PostMessage等）

- 低级别进程无法把线程注入到高级别进程
- 低级别进程无法对高级别进程进行消息或日志挂钩
- 低级别进程无法把DLL注入到高级别进程
- **低级别的进程不可以读取高级别的内存地址空间**

最小服务、最小权限——最大安全

文件和注册表虚拟化(Virtualization)

- 这里说的虚拟化本质上是“重定向”操作
- 某些老版本遗留下来的应用程序还需要往Windows系统目录中写入数据文件或者配置信息，那么Windows会自动将写入操作重定向到另外一个专用的文件夹，同时该程序读取相应文件夹时也将会被重定向
- 注册表的虚拟化和文件的虚拟化类似
- 程序进程不会被虚拟化的情况
 - 应用程序是64位的不会被虚拟化，虚拟化只为32位的应用程序启用，遵循开发标准的64位程序不启用虚拟化
 - 应用程序已经运行在管理员权限下
 - 本身已经提权了，用不着虚拟化
 - 要执行的操作来自于一个内核模式的调用者
 - 要执行的操作来自于一个模拟身份的调用者，比如网络共享访问
 - 进程的可执行映像文件带有一个与UAC兼容的清单文件(manifest，其中指定了requestedExecutionLevel设置)
 - 服务程序永远不会被虚拟化
- 可虚拟化：
 - 32位
 - 普通用户身份
 - 无与UAC兼容的清单文件
 - 非服务程序
- 文件虚拟化：
 - 虚拟化的路径包括：%ProgramFiles%、%ProgramData%以及%SystemRoot%，不包括某些特定的子目录。此外，不包括.exe、.bat、.scr、.vbs等可执行文件
 - 重定向的路径为“%LocalAppData%\VirtualStore”

4.4.4 应用程序控制策略

AppLocker：

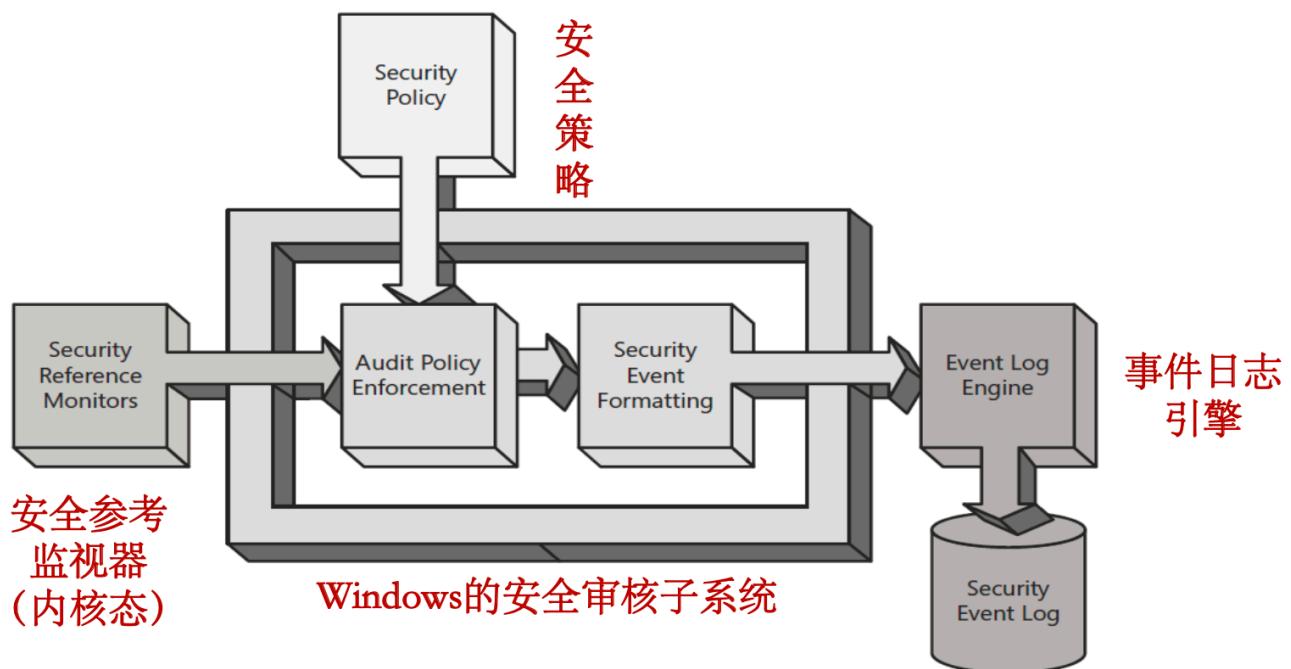
- 是Windows 7系统和Windows Server 2008 R2中新增的一项安全功能，用以代替以前的“软件限制策略”(Software Restriction Policies)
- 管理员可以非常方便地进行配置，以实现用户可在计算机上可运行哪些程序、安装哪些文件、运行哪些脚本，可以控制以下这些类型的应用程序文件：
 - 可执行程序(.exe、.com)

- 动态链接库(.dll、.ocx)
- 微软软件安装程序(.msi、.msp)
- Windows Powershell脚本程序(.ps1)
- 批处理(.bat、.cmd)
- Visual Basic脚本(.vbs)
- JavaScript(.js)
- 一般在域上通过组策略下发

4.5 Windows的安全审核

安全审计：

- 对象管理器(ObjectManager)可能会生成一些审计事件，并以此作为一次访问检查的结果；此外，有些可供用户应用程序使用的Windows函数也可以直接生成审计事件
- 内核模式的代码总是允许生成审计事件
- 有两个特权与审计有关：
 - 一个进程必须有SeSecurityPrivilege特权才能管理安全事件日志(Event Log)，以及查找或设置一个对象的SACL
 - 一个进程要调用审计系统服务，必须有SeAuditPrivilege特权才能成功地生成一条审计纪录



内核态中的安全参考监视器(SRM)产生安全事件，通知审核子系统(用户态)并告知具体细节；审核子系统格式化成事件记录，然后根据安全审核策略决定是否记录；如果需要记录的话，则将其传递给事件日志服务存储到安全事件日志中去

SRM与身份认证无关，只与访问控制、审计有关

SACL：系统访问控制列表，默認為空

审核策略的类型：

- 审核策略更改

- 确定是否对更改策略的每个事件进行审核
- **审核登录事件**
 - 确定是否审核每一个登录或注销计算机的用户实例
- **审核账户登录事件**
 - 确定是否审核在这台计算机用于验证账户时，用户登录到其它计算机或者从其它计算机注销的每个实例
 - “账户登录事件”是在账户验证信息所在的位置生成的，而“登录事件”是在登录发生的位置生成的
 - 比如：用户从本地成员主机成功登录到域，则在“域控制器”的安全日志中生成“账户登录事件”，而在“本地主机”的安全日志中生成“登录事件”
 - 因此本地登录无区别，登录到域就不一样了
- **审核账户管理**
 - 确定是否对计算机上的每个账户管理事件进行审核。包括创建、修改或删除用户账户或组；重命名、禁用或启用用户账户；设置或修改密码等
- **审核对象访问**
 - 确定是否对用户访问指定了自身 SACL (系统访问控制列表) 的对象 (如文件、文件夹、注册表项和打印机等) 的事件进行审核
- **审核目录服务访问**
 - 确定是否对用户访问指定了自身 SACL 的 Active Directory (活动目录) 对象的事件进行审核
- **审核过程追踪**
 - 确定是否审核事件的详细跟踪信息，如程序激活、进程退出、句柄复制和间接对象访问等
- **审核特权使用**
 - 确定是否对用户行使用户权限的每个实例进行审核
- **审核系统事件**
 - 确定在用户重新启动或关闭其计算机时，或者在影响系统安全或安全日志的事件发生时，是否进行审核

审核日志的存储文件

- Windows 5.x
 - 文件扩展名为evt，微软私有格式
 - %systemroot%\system32\config
 - SysEvent.evt, SecEvent.evt, AppEvent.evt
- Windows 6.x
 - 文件扩展名为evtx，XML格式
 - %systemroot%\system32\WinEvt\logs
 - System.evtx, Security.evtx, Application.evtx

5 Windows文件数据安全

5.1 NTFS文件系统简介

NTFS文件系统的优点：

- NTFS支持更大的文件以及更大的分区
 - NTFS支持多数据流属性
 - NTFS支持Unicode字符的文件名称
 - NTFS支持访问权限设置
 - NTFS支持EFS(加密文件系统)
 - NTFS支持文件压缩功能
 - NTFS具备磁盘配额功能
 - NTFS支持动态盘
 - NTFS具备恢复日志功能
-

NTFS的命名数据流特性：

- 多数据流属性
- NTFS将文件内容也当作属性
- 本身内容：未命名数据流
- 许多病毒代码利用这个特性隐藏自己

– 创建（命名）数据流

- echo This is a test file > test.txt
- echo This is an ADS > test.txt:hidden
- echo This is an ADS txt > test.txt:hidden.txt
- type c:\windows\system32\calc.exe > test.txt:calc2.exe

– 查看/运行（命名）数据流

- more < test.txt:hidden
 - notepad test.txt:hidden.txt
 - start c:\test\test.txt:calc2.exe
- wmic process call create c:\test\test.txt:calc2.exe
✓ Windows 7 “偷偷的”去除了执行ADS中代码的特性

– 检测（命名）数据流

- dir /r
- streams.exe
- Lads.exe

5.2 文件访问控制

5.2.1 NTFS文件访问权限

文件夹的标准权限：

权限	允许完成的操作
读取	查看文件夹内的子文件夹和文件 查看文件夹的所有权、权限和文件属性
写入	在文件夹内创建新的文件和子文件夹 更改文件夹属性，查看文件夹的所有权和权限
列出文件夹目录	查看该文件夹中的文件和子文件夹的名称
读取和执行	包含“读取”和“列出文件夹目录”权限所允许的操作 漫游各个文件夹，以便访问其他文件和文件夹，即使没有那些文件夹的权限
修改	包含“写入”和“读取和执行”权限所允许的操作 删除
完全控制	包含其他所有权限所允许的操作 更改权限，获取所有权，删除子文件夹和文件

文件的标准权限：

权限	允许完成的操作
读取	读取该文件和查看文件属性、所有者及权限
写入	覆盖该文件，更改文件属性和查看文件的所有者和权限
读取和执行	包含“读取”权限所允许的操作 运行应用程序
修改	包含“写入”和“读取和执行”权限所允许的操作 修改和删除文件
完全控制	包含其他所有权限所允许的操作 更改权限，获取所有权

多重权限：

- 权限是累加的：用户对一个资源的最终权限，是为该用户指定的全部权限和为该用户所属组指定的全部权限之和
- 同一级别中拒绝权限优先于其它权限
 - 底层逻辑：在DACL中，同一级别的拒绝访问控制项永远在允许访问控制项的前面
- 设置给文件的权限优先于设置给文件夹的权限
 - 文件权限优先于文件夹权限，用户只要有访问一个文件的权限，即使没有访问该文件所在文件夹的权限，也可以访问该文件
 - 无法打开FolderA文件夹，但可输入完整路径直接查看FileA文件

获取文件/文件夹所有权需遵循的规则：

- 当前的拥有者或者具有“完全控制”权限的任何用户，可以将“完全控制”这一标准权限或者“获得所有权”这一特殊访问权限授予另一个用户账户或者组
- Administrators组的成员可以取得某个文件或者文件夹的所有权，而不管该文件或者文件夹的权限如何
- 为了成为某个文件或者文件夹的拥有者，具有“获得所有权”权限的某个用户或者组的成员，必须明确地取得该文件或者该文件夹的所有权

假设曾对某文件夹设置了权限：只允许UserA用户访问，包括管理员在内的其他用户都禁止访问。后来我们无意中删除了UserA帐户，则该文件夹无法再被访问

这种情况下的正确的做法（前提是管理员组用户经UAC提权）：

1. 重新指派所有者（替换子容器和对象的所有者）
2. 使用所有者帐户登录，重新设置文件夹的访问权限

5.2.2 共享文件访问权限

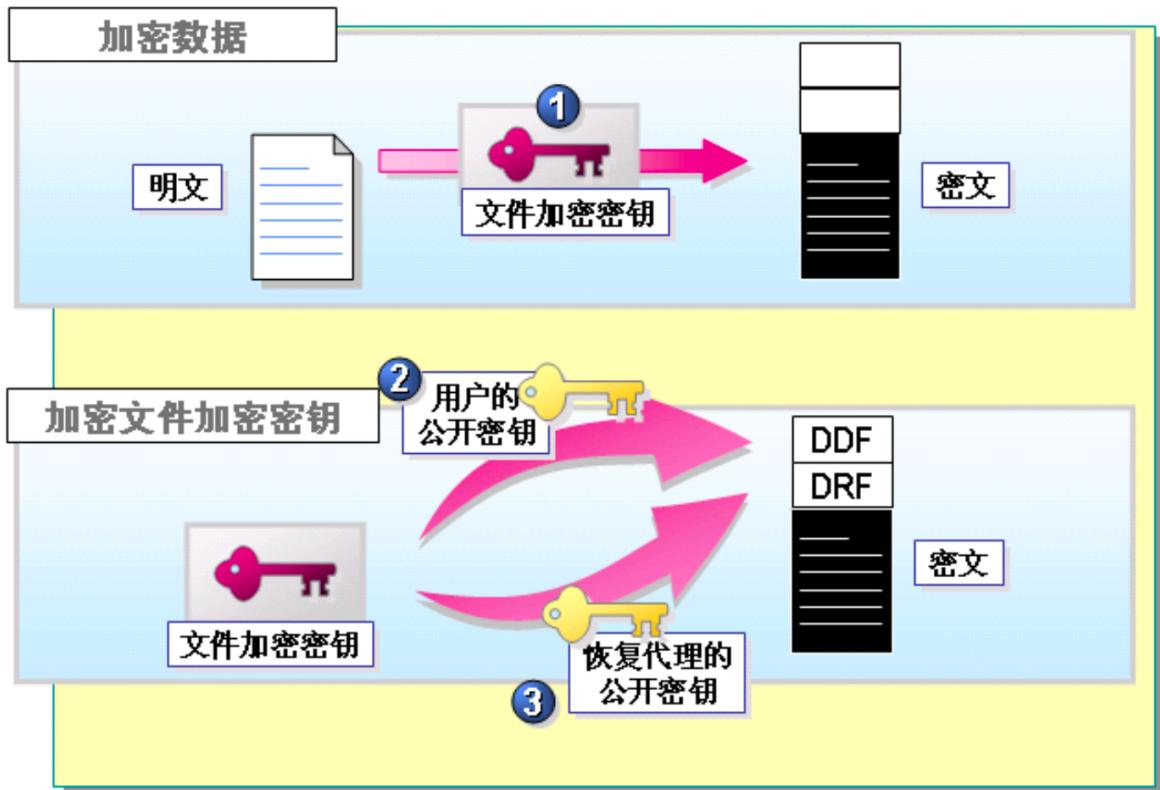
5.3 文件加密

5.3.1 EFS文件加密系统

EFS文件加密系统：

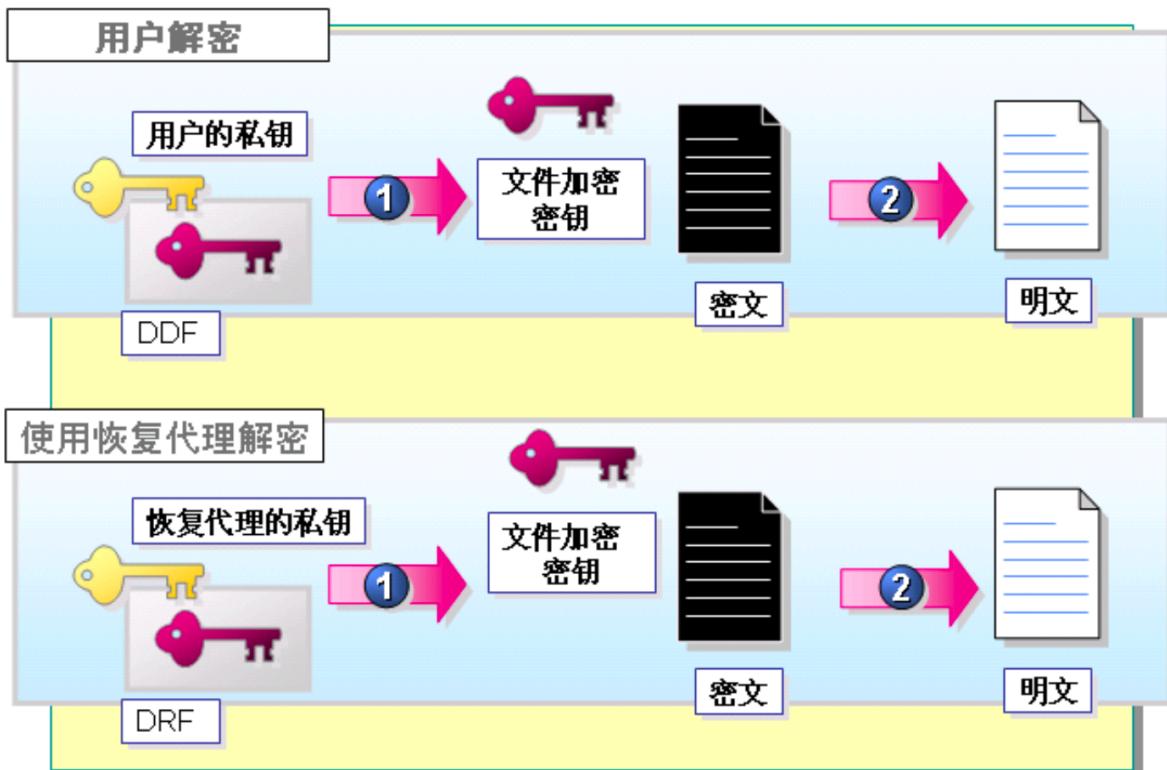
- EFS通过加密保护数据的安全
 - 默认情况下，将文件或文件夹加密后，只有进行加密的人才可以访问，其他人就算能够在物理上接触到电脑，只要没有使用加密者的Windows帐户登录，哪怕使用了其他管理员帐户，一样无法访问被加密的文件
 - 每个用户都有自己的公私钥，文件本身对称加密
- EFS最主要的优势：与操作系统结合紧密，对用户透明
- EFS采用了非对称加密算法的公私钥体系和对称加密算法的混合加密体制
 - 结合了对称加密算法的加密速度快，以及非对称加密算法的密钥管理方便这两方面的优势
- 每个用户的公私钥对在第一次加密时产生，不是创建用户时产生

EFS的数据加密过程：



- 文件加密密钥随机产生
- 恢复代理的公开密钥：防止用户私钥丢失，默认恢复代理的用户为管理员

EFS的数据解密过程：



- 透明，用户读取、写入无感

EFS的使用注意事项：

- 将EFS加密和NTFS权限配合使用
- 注意加密证书的创建时间：使用镜像备份软件还原系统时，会导致加密证书的丢失，从而无法打开加密文件
- 如果需要，可以添加右键快捷菜单项
- 加密过的文件，拷贝到别的计算机，会事先解密，在别的计算机上是未加密的

5.3.2 Bitlocker磁盘加密

Bitlocker：

- 是Windows Vista开始引入的一项数据保护新功能，可以解决计算机设备的物理丢失导致的数据失窃或恶意攻击泄漏
- Bitlocker通过将Windows的安装分区进行整卷加密，防止被攻击者通过启动其他操作系统来获取文件的“脱机攻击”

5.4 文件数据安全

5.4.1 文件数据的备份和还原

5.4.2 文件的彻底删除和反删除

5.5 磁盘配额

磁盘配额的作用：

- 管理员可以方便合理地分配存储资源，避免由于磁盘空间使用的失控可能造成的系统崩溃，从而提高了系统的安全性
- 适用场合：文件服务器、FTP服务器、邮件服务器等

磁盘配额的限制设置：

- 当用户超过指定的磁盘配额警告级别时，记录事件
- 当用户超过所指定的磁盘配额限度时，阻止进一步使用磁盘空间，并记录事件

磁盘配额是以文件所有权为基础的，并且不受卷中用户文件的文件夹位置的限制

6 Windows安全策略配置

6.1 本地安全策略

本地安全策略编辑器：secpol.msc

账户策略：

- 账户策略仅涉及和用户账户的凭据相关的设置
 - 通过设置账户策略、可以让所有本地账户更加安全，同时要破解账户密码所需的时间更长，所需技

术更高

- 账户策略的种类
 - 密码策略
 - 账户锁定策略
-

密码策略：

- **密码必须符合复杂性要求**
 - 建议设置为“启用”
 - 密码至少6个字符。
 - 密码至少拥有3种字符集中的字符（共有大写字符、小写字符、数字以及非字母和数字的字符等四类字符集）。
 - 密码不能有3个以上的字符来源于用户名分隔项。这些用户名分隔项是将用户的账户名称使用空格、下划线、Tab等符号分隔成的多个项，并丢弃小于3个字符的项。
 - 密码长度最小值
 - 建议设置为“14”
 - 密码最短使用期限
 - 建议设置为“5天”
 - 密码最长使用期限
 - 建议设置为“30天”
 - 强制密码历史
 - 建议设置为“10个”
 - 用可还原的加密来存储密码
 - 建议设置为“禁用”
-

账户锁定策略：

- 账户锁定阈值
 - 建议设置为“3次”
- 账户锁定时间
 - 建议设置为“30分钟”，可根据实际需要更改
- 复位账户锁定计数器
 - 建议设置为“30分钟”，可根据实际需要更改

在“复位账户锁定计数器”内错误了“账户锁定阈值”次，即锁定“账户锁定时间”这么多时间