**（本提纲0至7章中的部分内容，参考陈婕同学提纲内容后添加，特此说明）**

**第0章**

1. **The Goals of Malware Analysis 要清楚**

* Exactly what happened
* Ensure you’ve located all infected machines and files
* Dissect the suspect files
* Find signatures for detection
* Build detection models based on machine learning
* How to measure and contain the damage
* 确定到底**发生了什么**
* 确保能够**定位**出所有受感染的主机和文件
* **解刨**可疑文件
* 找出可用于检测的**特征码**
* 基于**机器学习构建检测系统**
* 如何**检测并控制损害**

1. **Dissecting 解刨**

**Dissecting malware to understand**

* How it works
* How to identify it
* How to defeat or eliminate it
* 如何**检测**它
* 如何**识别**它
* 如何**防御或消除**它

1. **Signatures 特征码**

* **Host-based signatures 基于主机的特征码**
* Identify files or registry keys on a victim computer that indicate an infection
* Focus on what the malware did to the system, not the malware itself
* Different from antivirus signatures
* 又称为**感染迹象**，用于**在受感染主机上检测出**恶意代码。
* 经常是恶意代码所**创建或修改的文件**，或是**对注册表的特定修改**。
* 与反病毒软件所使用的**病毒特征码**不同

恶意代码的感染迹象关注的是**恶意代码对系统做了什么**

恶意代码的病毒特征码关注的是**恶意代码本身的特性**。

有时感染迹象更加有效，比如检测那些**经常变化自身形态**的多态性恶意代码，或者恶意代码已经**将自身文件从硬盘中删除**。

* **Network signatures 基于网络的特征码**
* Detect malware by analyzing network traffic
* More effective when made using malware analysis
* 通过**监测网络流量**来检测恶意代码
* 在**恶意代码分析帮助下**提取的基于网络的特征码往往是**更加有效**的，有**更高的检测率和更少的误报**

1. **Malware Analysis Techniques**

* **Static Analysis 静态分析基础技术**
* Examines malware without running it
* Tools: VirusTotal, strings, a disassembler like IDA Pro
* 不执行可执行文件，不查看具体指令的检测
* 工具：VirusTotal, strings
* **Dynamic Analysis 动态分析基础技术**
* Run the malware and monitor its effect
* Use a virtual machine and take snapshots
* Tools: RegShot, Process Monitor, Process Hacker, CaptureBAT
* RAM Analysis: Mandant Redline and Volatility
* **运行**恶意代码并观察其**在系统上的行为**
* 使用虚拟机并拍摄快照
* 工具：RegShot, Process Monitor, Process Hacker, CaptureBAT
* RAM 分析 : Mandant Redline and Volatility
* **Advanced static analysis 静态分析高级技术**
* Reverse-engineering with a disassembler
* Complex, requires understanding of assembly code, constructs, OS concepts
* 将可执行文件装载到**反汇编器**中，査看程序指令，来发现恶意代码到底做了什么的**逆向工程**
* 掌握汇编语言、代码结构、Windows操作系统概念等专业知识
* **Advanced Dynamic Analysis 动态分析高级技术**
* Run code in a debugger
* Examines internal state of a running malicious executable
* 使用**调试器**
* 来检查一个恶意可执行程序**运行时刻的内部状态**

1. **Types of Malware**

* **Backdoor后门**
* Allows attacker to control the system
* 恶意代码将**自身安装**到一台计算机来**允许攻击者访问**
* **Botnet 僵尸网络**
* All infected computers receive instructions from the same Command-and-Control (C&C) server
* 所有被同一个僵尸网络感染的计算机将会**从一台控制命令服务器**接收到**相同的命令**，允许攻击者访问系统
* **Downloader 下载器**
* Malicious code that exists only to download other malicious code
* Used when attacker first gains access
* 只是用来**下载其他恶意代码的恶意代码**
* 通常是在攻击者获得系统的访问时**首先进行安装**的
* **Information-stealing malware 间谍软件**
* Sniffers, keyloggers, password hash grabbers
* 从受害计算机上**收集信息**并**发送给攻击者**
* 比如：嗅探器、密码哈希采集器、键盘记录器等。
* **Launcher 启动器**
* Malicious program used to launch other malicious programs
* Often uses nontraditional techniques to ensure stealth or greater access to a system
* 用来**启动其他恶意程序**的恶意代码
* 通常使用一些**非传统的技术来启动**其他恶意程序，以确保其**隐蔽性**，或者以**更高权限**访问系统。
* **Rootkit内核套件**
* Malware that conceals the existence of other code
* Usually paired with a backdoor
* 用来**隐藏其他恶意代码**的恶意代码
* 通常是与其他恶意代码（如**后门**）**组合成工具套装**，来允许为攻击者提供远程访问
* **Scareware 恐吓软件（勒索软件？）**
* Frightens user into buying something
* **勒索用户购买某些东西**的恶意代码
* 通常有一个用户界面，使得它看起来像是一个杀毒软件或其他安全程序。它会通知用户系统中存在恶意代码，而唯一除掉它们的方法只有购买他们的“软件”，而事实上，他们所卖软件的全部功能**只不过是将勒索软件进行移除**而己。
* **Spam-sending malware 发送垃圾邮件的恶意代码**
* Attacker rents machine to spammers
* 通过为攻击者**出售垃圾邮件发送服务**而获得收益
* **Worms or viruses 蠕虫或病毒**
* Malicious code that can copy itself and infect additional computers
* 可以复制自身并感染其他计算机的恶意代码
* **Ransomware 勒索软件**
* encrypt victim's data as hostage
* ask for ransom to recover the data
* 将受害者的数据加密为人质
* 勒索财物后才恢复数据

1. **Mass v. Targeted Malware**

* **Mass malware 大众性恶意代码**
* Intended to infect as many machines as possible
* Most common type
* 设计为影响到尽可能多的机器
* 最为普遍
* **Targeted malware 针对性恶意代码**
* APT(Advanced Persistent Threat)：高级持续性威胁，本质是针对性攻击
* Tailored to a specific target
* Very difficult to detect, prevent, and remove
* Requires advanced analysis
* Ex: Stuxnet
* 针对特定组织而研制的
* 很难检测，防止和移除
* 需要高级分析
* 震网病毒

1. **General Rules for Malware Analysis**

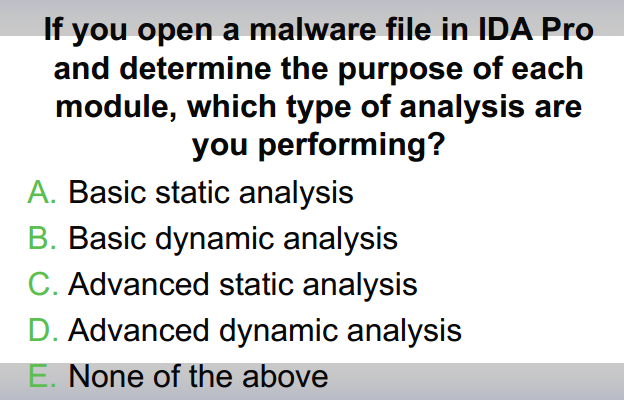
* **Don’t Get Caught in Details 不要过于陷入细节**
* You don’t need to understand 100% of the code
* Focus on key features
* 不需要了解每一个细节
* 需要关注最关键的主要功能
* **Try Several Tools 对于不同的工作任务，使用不同的工具和方法**
* If one tool fails, try another
* Don’t get stuck on a hard issue, move along
* 用一个工具失败的时候，尝试另外一种
* 如果在一个点上被卡住了，不要花太长时间，尝试转移到其他问题
* **Malware authors are constantly raising the bar**

**恶意代码编写者也在发明着可以挫败分析的新技术**

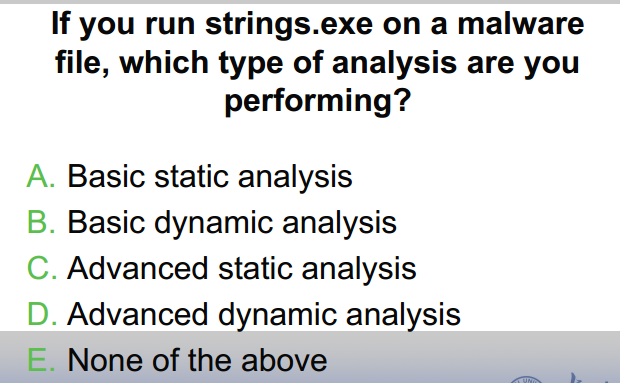
* cat-and-mouse game
* 猫捉老鼠的游戏

**第1章 静态分析基础技术**

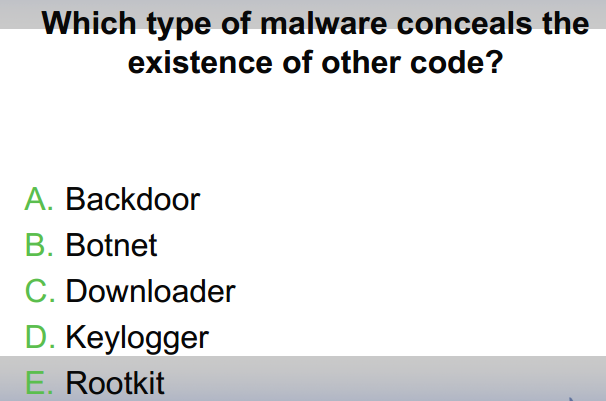
都是基本概念



C

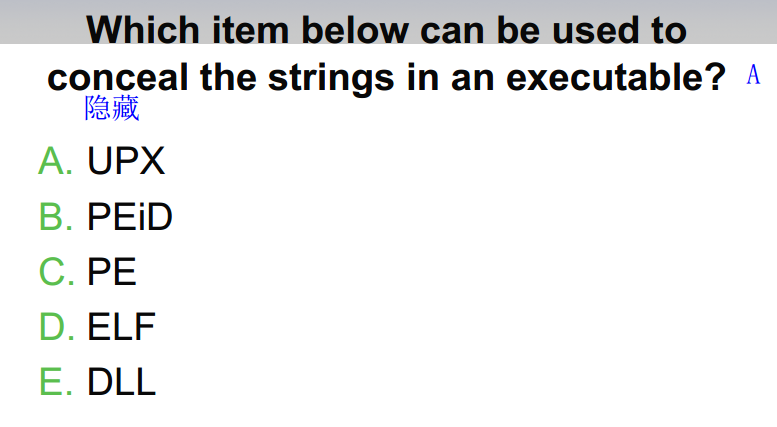


A

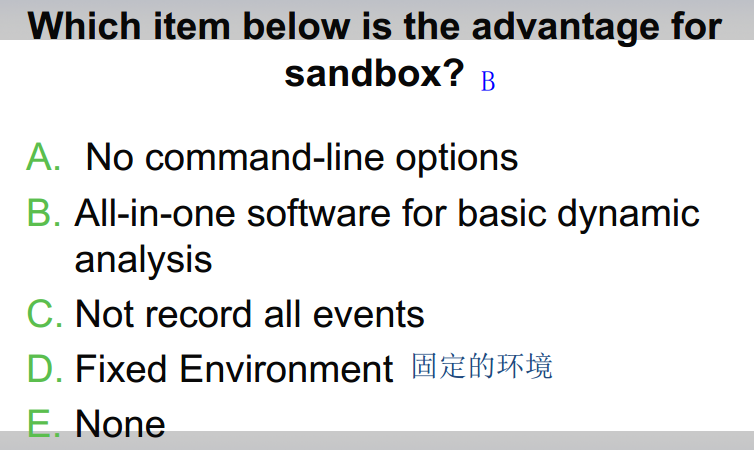


E（但是为什么呢）

**第3章 动态分析基础技术**



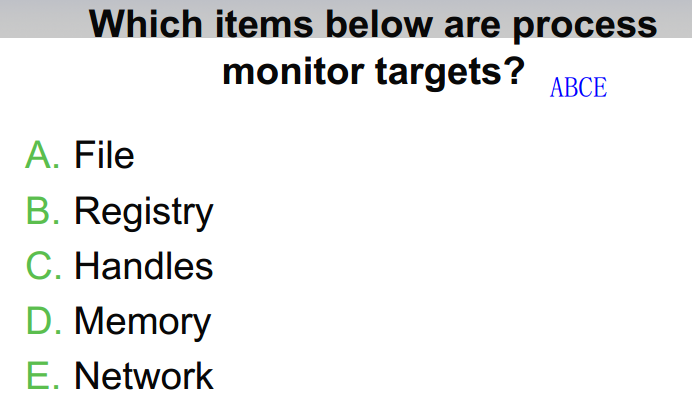
A



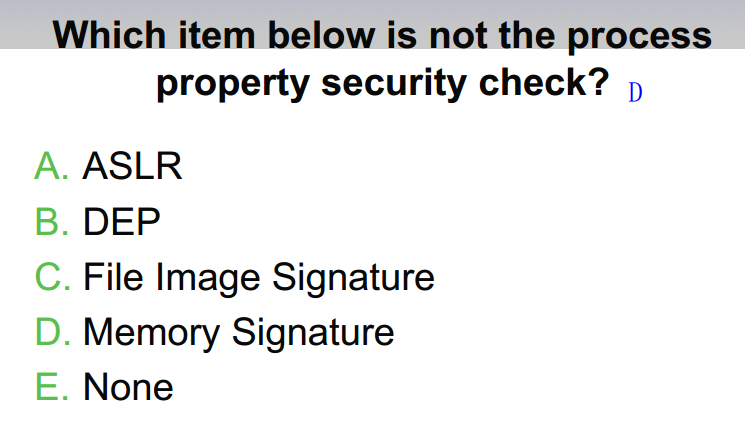
B



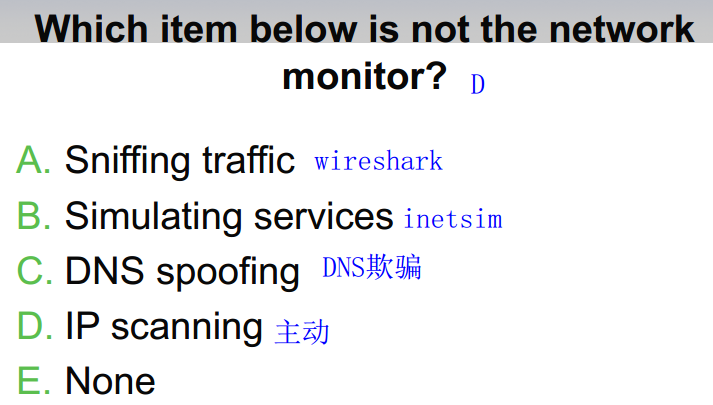




**这题选啥**

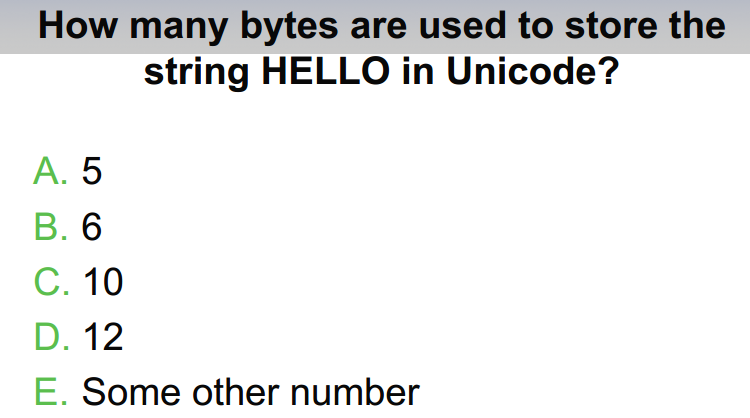


**Memory Signature是内存签名检查，内存变化太快，对其签名检查无意义**

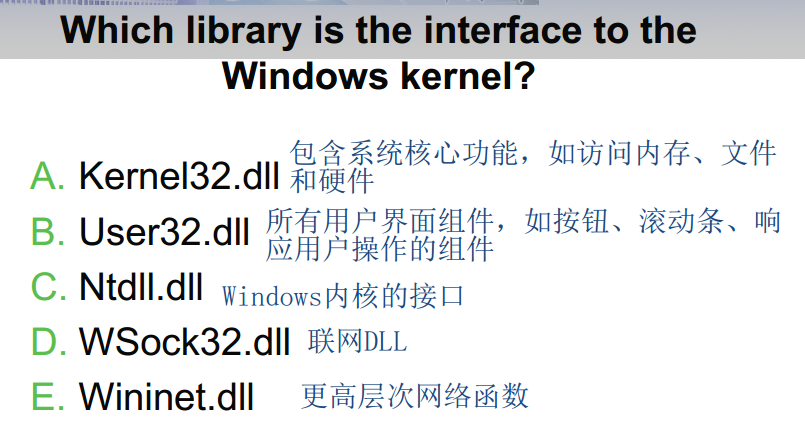




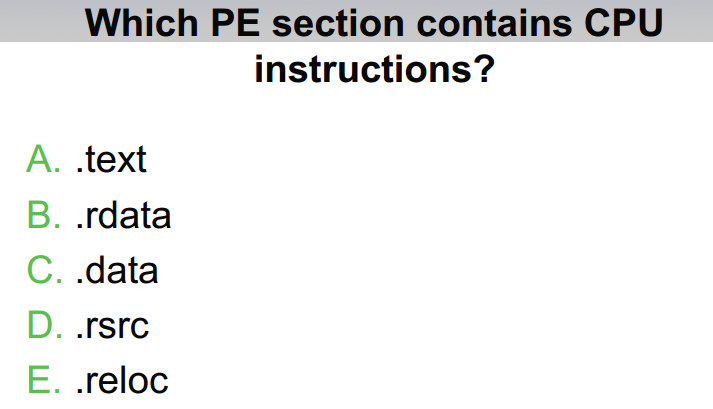
C



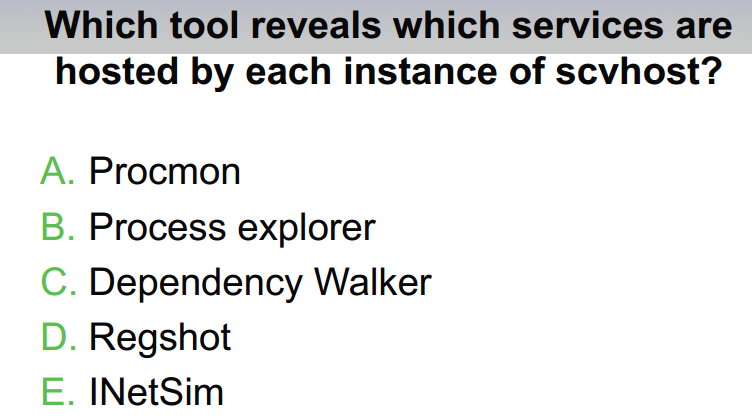
D



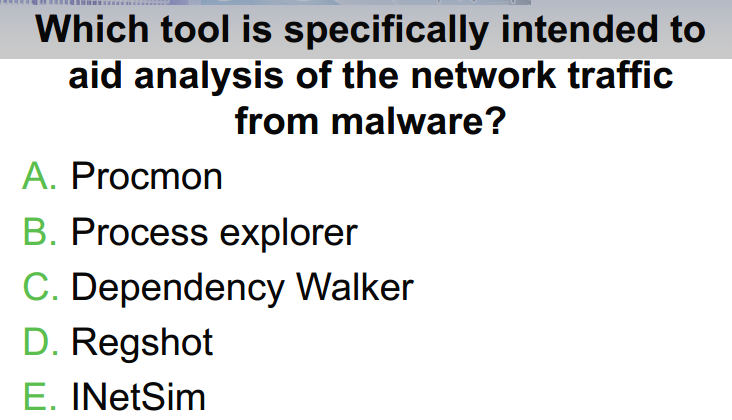
C



A

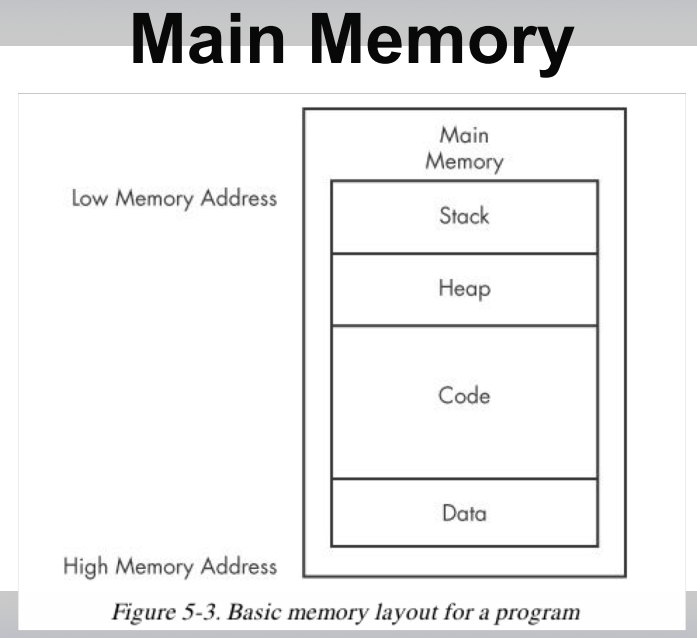


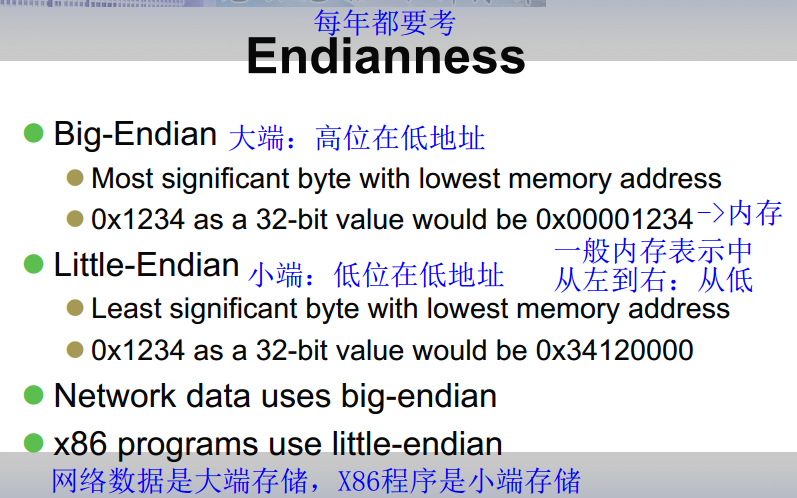
B

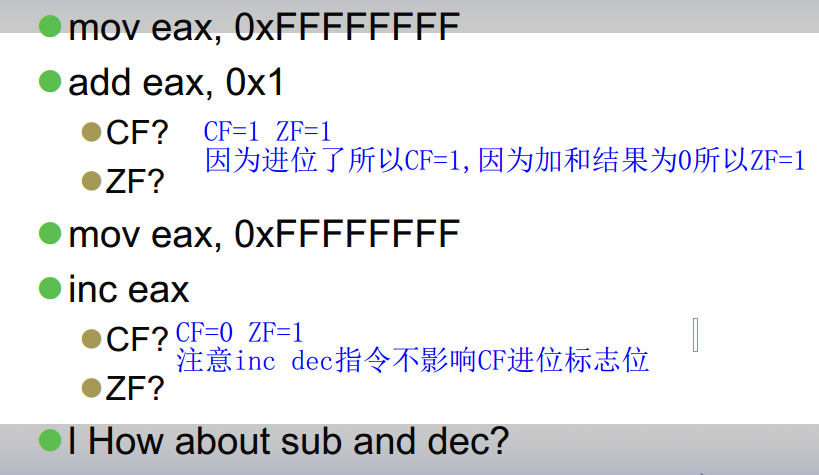


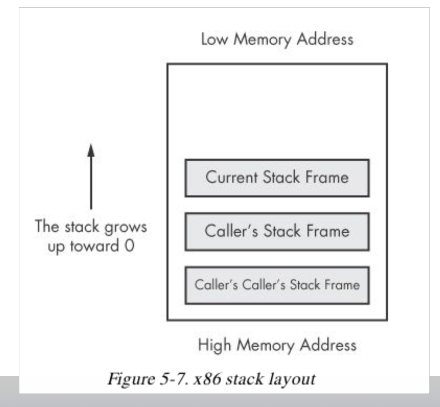
E

**第4章 X86反汇编速成**









**第6章 识别汇编中的C代码结构**

1:21:40

**第8章 动态调试**

1. **Source-Level vs. Assembly-Level Debuggers**

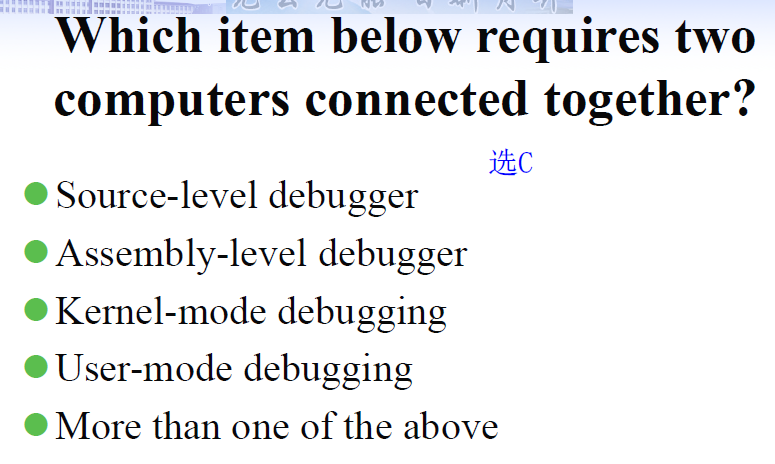
源码级与汇编级的调试器

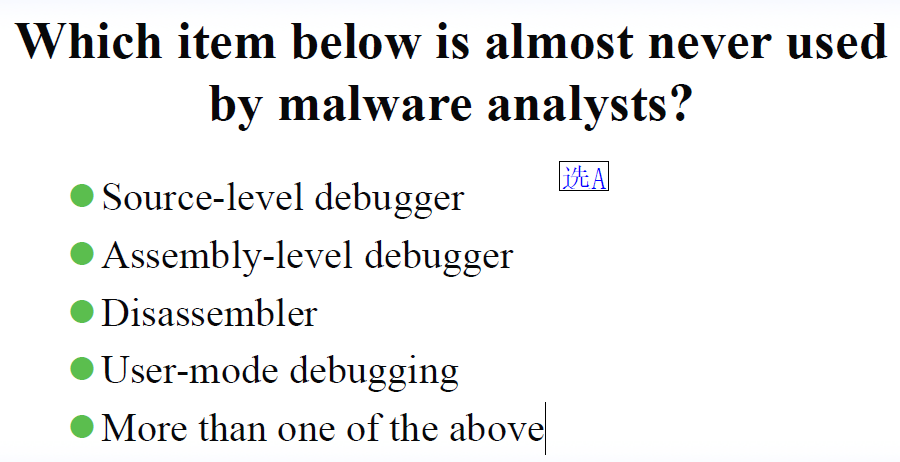
* **Source-Level**
* operate on source code
* step through program execution one line at a time
* **操作对象是源代码**
* **按照一次一行源代码的方式运行程序**
* 允许设置断点，让程序中断在源代码的某行
* **Assembly-Level**
* Low-level debugger
* Operate on assembly code
* step through program execution one instruction at a time
* 也称为底层调试器
* **操作对象是汇编代码**
* **按照一次一条指令的方式运行程序**
* 允许在某行汇编代码上设置断点

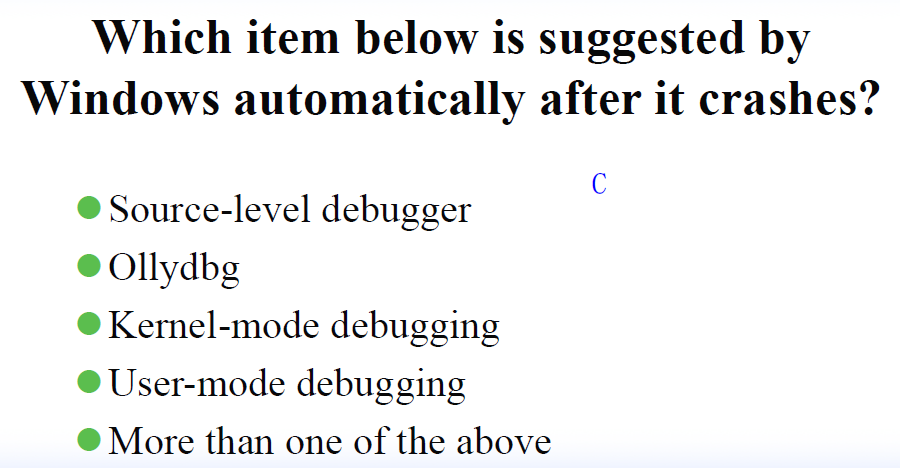
1. **Kernel vs. User-Mode Debugging**

内核模式与用户模式调试

* **Kernel mode debugger**
* **debug kernel mode codes**
* 需要在两个系统上进行
* **一个系统运行被调试的代码**，**另一个运行调试器**
* 必须配置操作系统使其开启内核调试功能，并将两个系统连通
* Windbg
* **User mode debugger**
* **debug user mode codes**
* 调试器与被调试的代码运行在同一个系统中
* 调试的是单个可执行程序，操作系统会将它**与其他可执行程序隔离**
* Ollydbg







1. **Using a Debugger：Two Ways**

* **利用调试器启动程序**

当启动程序后，它被加载到内存，程序在其**入口点运行之前立即停止运行**。此刻，你已经完全控制了程序

* **附加调试器到一个已经运行的程序上**

当**程序的所有线程暂停时**，你就可以调试它了。当你想调试一个已经运行的程序或者一个已经被恶意代码感染的进程时，这是一个很好的方法。

1. **Single-Stepping 单步调试**

* **用调试器能做的最简单事情，就是单步执行一个程序**
* **每运行一条指令，控制权就返回到调试器**

1. **单步跳过 (Stepping-Over) 和单步跳入 (Stepping-Into)**

* **Stepping-over 单步跳过**
* 单步跳过一条函数调用指令，会完成该函数的执行，在调试器中看到的下一条指令是**函数调用返回后要执行的第一条指令**。
* **Stepping-Into 单步跳入**
* 单步跳入一条函数调用指令，那么在调试器中看到的下一条指令是**被调用函数的第一条指令**。
* **Stepping-Out 跳出**
* 一些具有跳出（Step-out)功能的调试器会一直运行直到函数返回。

也就是当我们不幸进入了很深层的被调用的函数时，可以使用Step-Out功能返回caller函数

1. **Breakpoints**

* **Software execution breakpoint 软件执行断点**
* **Hardware execution breakpoint 硬件执行断点**
* **Conditional breakpoint 条件断点**

Eflags的TrapFlag设置为1，则CPU会单步执行

* **Software execution breakpoint 软件执行断点**
* 在特定指令执行后它会停止程序的运行
* 实现方式：
* 通过使用0XCC，即指令INT 3的机器码，**重写指令的首个字节**来实现软件断点；**设置和取消断点时**需要**保存和恢复**该字节
* 当0XCC指令执行后，操作系统会**产生一个异常**，然后将控制权转到调试器
* 如果程序执行过程中该字节发生了改变，断点将不会发生：如果这段代码可以自我修改或被另一个代码段修改，此时你设置的断点将被删掉
* 内核模式中会限制软件断点数量；用户模式下可以无限制地设置软件断点
* 代码变化很小时，调试器只需要少量的内存空间保存记录。
* **Hardware execution breakpoint 硬件执行断点**
* 实现方式：
* **DRO〜DR3**用来存储断点地址

**调试控制寄存器**（**DR7**)存储**DRO〜DR3寄存器中的值是否有效**，以及它们**是否表示读、写或者执行断点等信息**。

* 处理器每次执行一个指令时，硬件都会**检测指令指针是否与断点地址相等**
* 优点：可以设置访问中断的断点而不只是执行中断的断点。

例如你可以设置这样一个硬断点，**无论断点处的内存被读或被写都将引起中断**。如果你试着判断某个内存地址值的含义，你可以在那个内存地址上设置一个硬断点。然后，**无论那个内存地址处的指令是否被执行**，只要该地址有写操作，调试器都会中断。（你**也可以设置读、写或者读写访问时触发的断点**。）

* 缺陷：容易被运行的程序修改。恶意代码可以修改这 些寄存器来干扰调试器。

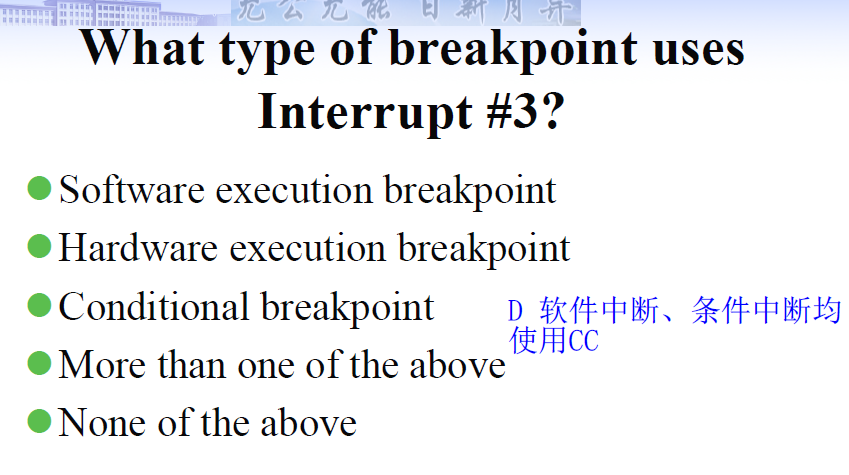
令人欣慰的是，x86芯片组具有防御该干扰的功能。通过设置DR7寄存器中的通用探测（**General Detect**）标志位，任何执行mov指令访问调试寄存器的操作都会触发中断，这让你能够探测调试寄存器的改变。

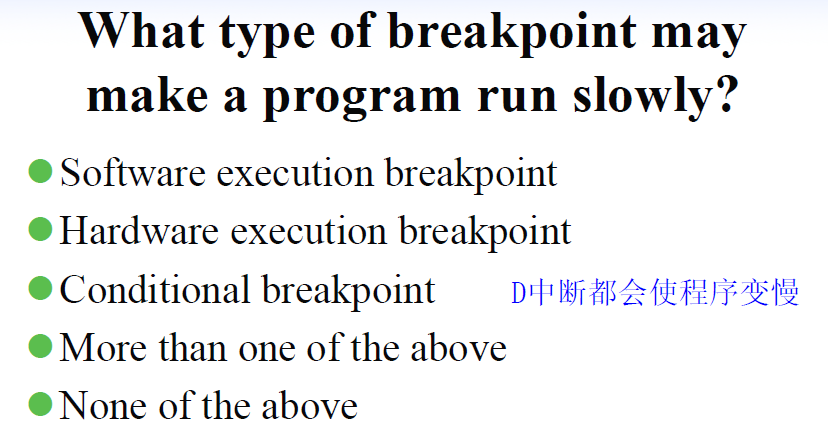
这种方法并不完美，它只能探测出访问调试寄存器的mov指 令，但它仍具有重要的意义。

* **Conditional breakpoint 条件断点**
* 在某些条件满足时才会被触发
* 例如，在函数GetProcAddress上设置了一个断点。假设你只想在传递给函数GetProcAddress的参数值为RegSetValue时才触发中断，那么条件断点可以帮助你来完成这个工作。

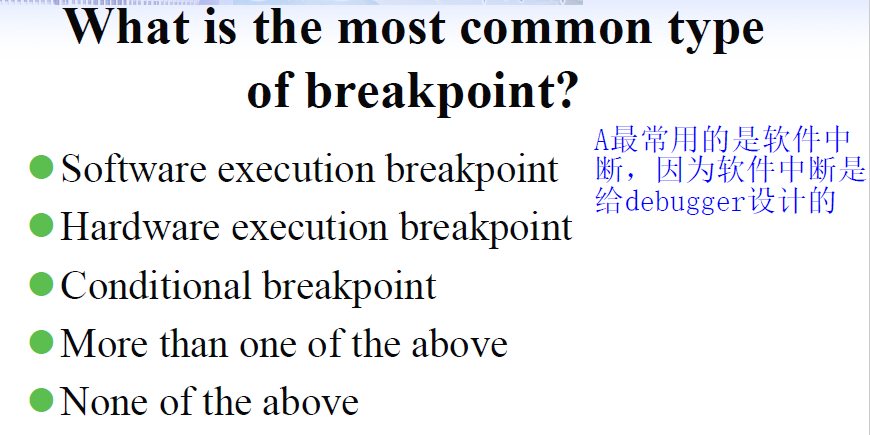
在本例中，条件就是栈中与第一个参数对应的值。

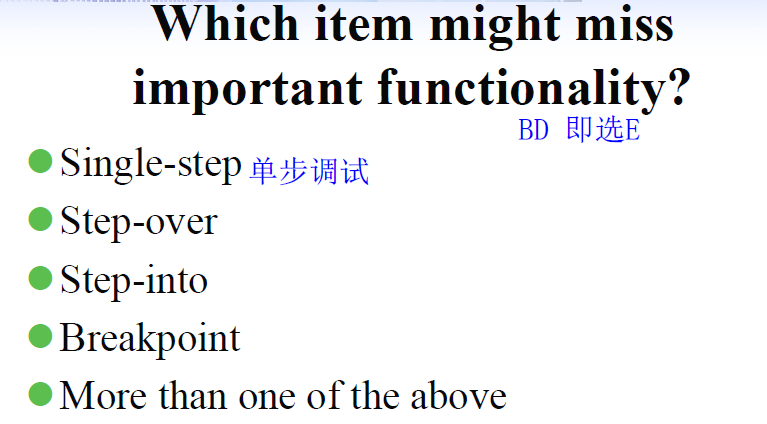
* 实现方式：
* 借助软件断点实现，调试器总是接收软断点
* 调试器计算条件，如果条件不满足，它自动 继续执行而不通知用户
* 断点比运行普通指令花费更多的时间。
* 如果你在一个经常被访问的指令上设置条件断点，程序的运行速度将会明显减慢。

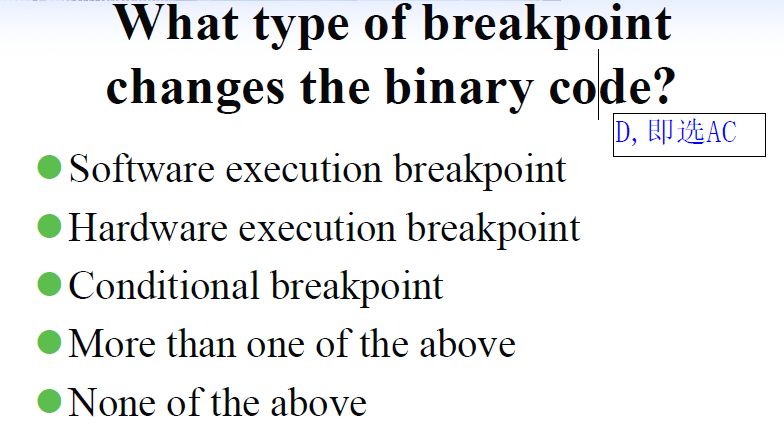


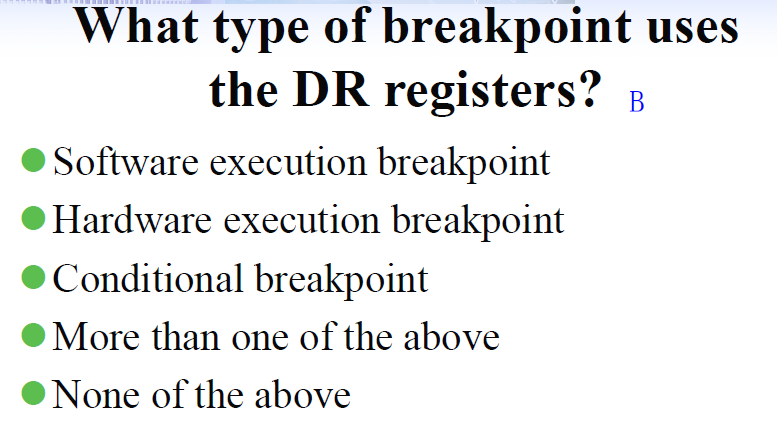


改选C









1. **Exceptions**

* **Exceptions 异常**
* 调试器取得运行程序控制权的基本方式
* 断点会产生异常
* 与调试无关的事件也可能产生异常
* 非法内存访问
* 除0操作
* **First- and Second-Chance Exceptions 首次和二次异常处理**
* 调试器通常有两次机会来处理同一个异常：**首次处理异常和二次处理异常**。
* 调试器第一次附加到进程时，会发生一个异常，

然后被调试的程序停止执行，此时调试器开始获得控制权。

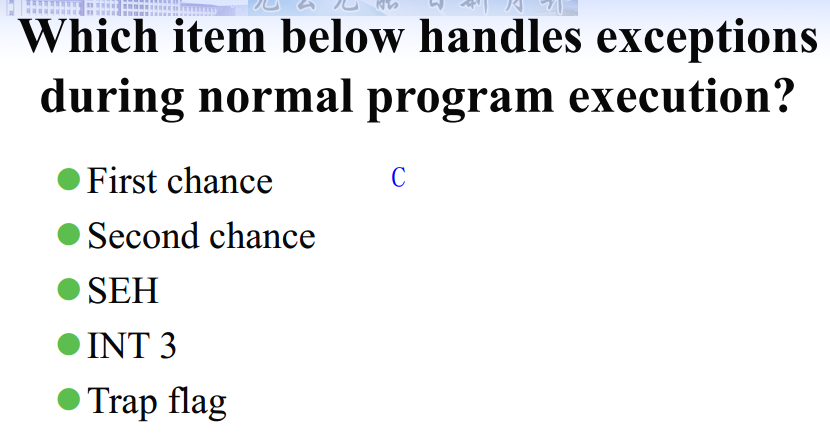
调试器**可以自己处理异常，也可以将异常转给被调试的应用程序处理**

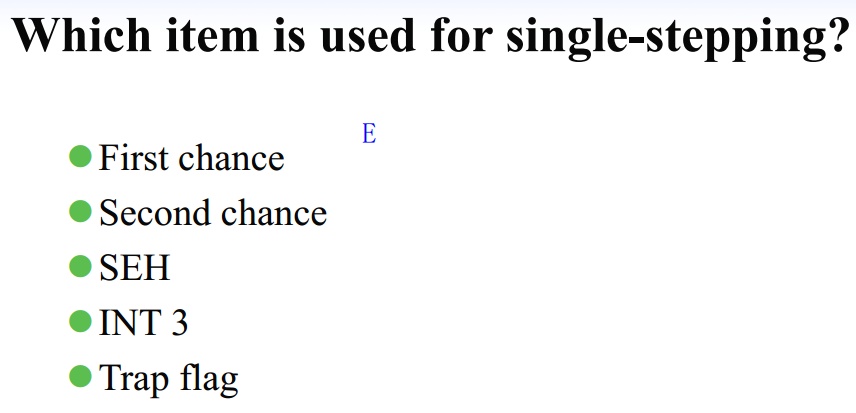
* 如果程序注册了一个异常处理函数，它会在调试器处理异常后，获得处理异常的权限。例如，一个程序注册一个除0异常的处理函数，如果程序执行了除0操作，异常函数会通知用户这个错误，并继续执行。这是没有连接到调试器时，应用程序的运行情况。
* 如果**应用程序没有处理异常**，**调试器将获得另一个处理它的机会**——**二次异常处理**。调试器接收到一个二次处理的异常后，意味着如果程序没有附加到调试器就会崩溃。为了让应用程序正常运行，调试器必须处理异常。
* 一般来说，分析恶意代码**并不是查找Bug，因此首次异常处理经常会被忽略**。
* **二次异常处理不能被忽略**，因为忽略以后程序将不能正常运行。如果在调试恶意代码时，你遇到了二次异常处理，则说明恶意代码中可能存在使它自身崩溃的Bug,但更为可能的是恶意代码并不想在当前环境中运行。
* **Common Exceptions常见异常**
* **INT 3: Software Breakpoint Trap 软件断点异常**
* 调试器有专门的代码处理INT 3 异常，而操作系统则把它和其他异常一样对待。
* 应用程序可能包含处理INT3异常的指令，但附加调试器到程序后，**调试器将获得首先处理异常权限**。**如果调试器将异常传给程序**，**程序的异常处理函数会处理它**。
* **INT 1: Single-stepping Trap 单步调试异常**
* 单步调试也作为操作系统内的一个异常来实现。
* 标志寄存器中的陷阱标志（trap flag）用于单步调试。

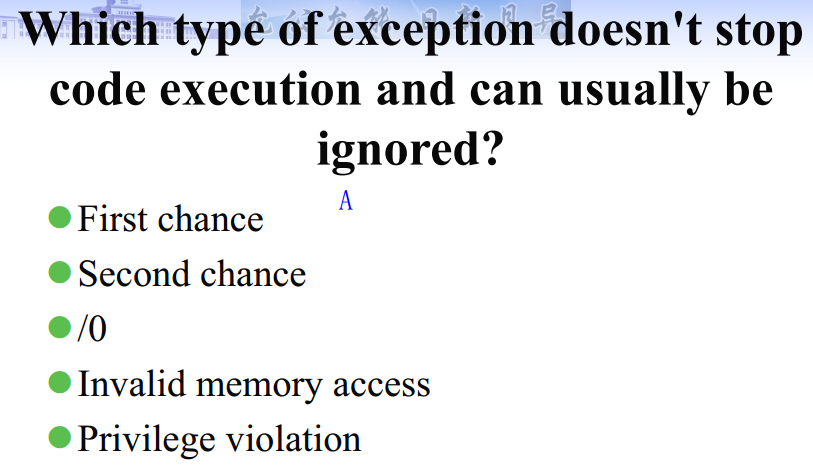
**陷阱标志置位后**，处理器**每执行一条指令就会产生异常**。

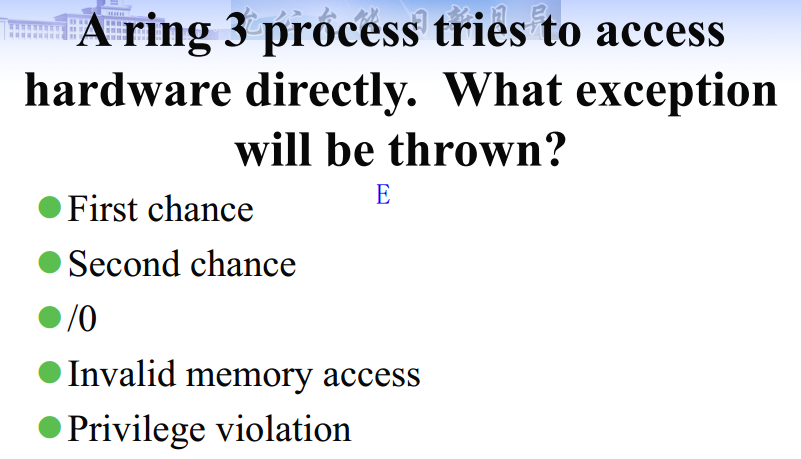
* **Memory-access violation exception 内存访问冲突异常**

可能因为内存地址无效，也可能因为访问了受保护而无权访问的内存位置









**第9章 OllyDbg 基本不考**

**第10章 Kernel Debugging with WinDbg**

* **Driver Drive Device及其关系要知道，谁在User Mode下谁在Kernel Mode下**
* **Driver 的main函数时DriverEntry**
* **命令不用死记硬背，可能给命令要求分析**
* **RootKit是重点，Kernel Debugging的主要目的是分析RootKit，OD分析不了它**

**RootKit的定义，常见的RootKit、SSDT RootKit、IDT** **RootKit**

1. **Driver Drive Device及其关系，谁在User Mode下谁在Kernel Mode下**

* **Driver 驱动**
* Driver让**第三方开发商**在Windows**内核模式下运行代码**。
* 分析驱动程序十分困难
* 驱动程序常驻内存，并且负责响应用户态应用程序的请求
* **应用程序不直接与驱动程序通信**
* **应用程序直接访问设备对象Device**，向具体的物理设备发送请求，使得驱动程序更加难以分析
* **Device 设备**
* 不是真实的物理硬件设备，是物理硬件设备的软件表示
* 设备对象由驱动程序Driver创建和销毁
* **可以被用户态的程序直接访问**
* **USB驱动器举例**

举例来说，USB闪存驱动器在**系统中**拥有一个**处理USB闪存驱动器请求的驱动**，但**用户态的应用程序并不能直接发送请求到这个Driver**，而**只能发送请求到一个具体的设备对象Device**。

当用户插入一个USB设备到计算机后，Windows**系统**为USB设备**创建一个如"F: drive"的设备对象Device object**。当应用程序**发送请求到"F: drive"这个Device object后**，**最终将由"F: drive"的Driver进行处理**。**第二个USB 设备**的处理**可由同一Driver完成**，但应用程序**必须通过不同Device object访问，如G:** **drive**。

* **Loading Drivers**
* 为了系统能够正常工作，**驱动程序必须加载到内核空间**，这与DLL需加载到进程空间是同样的道理。
* 当驱动首次被加载时，**DriverEntry函数将被调用**，这与DLLMain相似。

Driver中最先执行的函数是DriverEntry

* **DriverEntry**
* **DLL**通过**函数导出表export table**提供其功能接口
* **Driver**通过**注册回调函数register the address for callback functions**来提供功能
* 当用户态应用程序请求一个服务时，这些**回调函数将会被调用**
  + 1. DriverEntry程序中注册着回调函数
    2. **Windows**会**为每个驱动设备**创建**一个驱动对象driver object**
    3. **Windows**将**driver object**以参数形式传给**DriverEntry函数**
    4. **DriverEntry** 函数用回调函数**填充这个驱动对象driver object**
    5. **DriverEntry**创建一个可以被用户态应用程序访问的**设备对象device**，**应用程序与驱动driver**的交互请求都将通过这个**设备对象device**进行
* **举例：用户态读数据请求**
* 用户程序获得该硬件设备的一个文件句柄
* 用户在该句柄上调用函数ReadFile
* 内核将会处理ReadFile函数的请求，
* 最终由Driver的回调函数来响应对I/O设备的读请求
* **举例：请求内核态恶意组件**
* 请求内核态恶意组件的最常见函数是DeviceloControl
  + DeviceloControl是从用户态模块到内核设备的一种通用请求方法
  + 使用该函数时，用户态应用程序传递一个任意长度的缓冲区数据作为输入
  + 并且接收一个任意长度的缓冲区数据作为输出
* **Ntoskrnl.exe & Hal.dll**
* 恶意驱动通常不控制硬件设备
* 而是与Windows操作系统主要的内核组件Ntoskrnl.exe & Hal.dll进行交互。
  + **Ntoskrnl.exe**组件包含**操作系统核心功能**的代码

**NT Operating System Kernel**

* + **hal.dll**包含**与主要硬件设备交互**的代码

**Hardware Abstraction Layer**

* 恶意代码常通过从一个或者多个这样的内核组件中导入函数，来操纵内核。

1. **Rootkits 与SSDT**

Kernel Debugging的主要目的是分析RootKit，OD分析不了它

RootKit的定义，常见的RootKit（SSDT、IDT的概念）

* **Rootkit Basics**
* Rootkit通过**修改操作系统内部函数**，来隐藏自己
  + 通过这种修改，Rootkit可以**隐藏正在运行程序的文件、进程、网络连接**以及**其他资源**
  + 使其恶意活动难以被反病毒产品、管理员以及安全分析员发现
* 大部分Rootkit都是通过采用某种方式**修改操作系统内核来工作**的
* 在实际应用中，**系统服务描述表（SSDT: System Service Descriptor Table)挂钩技术**使用频繁
* **System Service Descriptor Table (SSDT)**
* 系统服务描述表（System Service Descriptor Table)
* 微软使用它来**查找进入内核的系统调用**
* 通常**不被第三方应用程序或者驱动程序访问**
* 用户态程序访问内核态代码只有三种方式：
* SYSCALL
* SYSENTER
* INT0X2E
* **SYSENTER**
* 当前Windows版本（Windows XP之后）使用SYSENTER 指令
* 它从**存储在寄存器EAX的函数序号Function code**中获取指令
* **Rootkit 原理**

Rootkit挂钩到这些函数后，它将会改变SSDT中某项的值，从而当内核态系统调用发生时，**将会调用Rootkit的代码**，而不是SSDT中预期的系统函数。在前面的例子中，0x25偏移处的值被改变成恶意驱动中的一个函数指针。这种改变可以修改函数，使其不能打开或是检查到恶意文件。

一般通过**在Rootkit代码中调用原始的NtCreateFile函数**，并**根据Rootkit的配置来过滤返回值**。为了防止其他应用程序获取这些文件的句柄，在**调用返回时，Rootkit需要删除需要隐藏的文件**。

* **Rootkit 实例**

1. 检查SSDT是否被挂钩：

* 获取NT模块的地址边界
* 找到SSDT表中落在地址范围之外的函数，这个可能就是被挂钩的函数

1. 定位驱动程序：

* 查找包含该地址的驱动程序

1. 分析驱动：

* 找到安装挂钩的代码段和执行挂钩的函数
* 安装挂钩的代码段：
* 创建要挂钩的函数名字符串NtCreateFile
* 调用MmGetSystemRoutineAddress函数获取NtCreateFile函数的地址
* 调用MmGetSystemRoutineAddress 函数是获取SSDT的地址
* 循环遍历SSDT，直到找到匹配NtCreateFile地址的值
* 使用钩子函数的地址覆盖该值
* 安装挂钩——将挂钩函数的地址复制到一个内存位置（方便访问？）
* 钩子函数：

**对于一些请求，挂钩函数直接调用原始的NtCreateFile函数，而对于其他一些请求，则返回 0xC0000034 (它对应的状态是STATUS\_OBJECT\_NAME\_NOT\_FOUND)。**

* 计算用户态程序尝试打开文件所使用的Object Attributes属性值（它包含对象的一些信息，如文件名等）。
* 如果NtCreateFile函数允许调用，挂钩函数会返回非零值
* 如果Rootkit阻止文件打开，则函数会返回零值。

如果挂钩函数返回零值，用户态应用程序 将收到一个代表指定文件不存在的错误。

* **挂钩函数的目的是阻止应用程序打开某些特殊文件，同时 不影响其他调用NtCreateFile的操作。**

1. **Interrupts**

* **Interrupts**
* 有时，Rootkit会**使用中断来干扰系统事件**。现代的处理器实现了用硬件方式触发软件事件的中断。**系统发送一条命令到硬件，硬件处理完请求事件后会中断处理器**。
* 有时，驱动或者Rootkit会**利用中断来执行代码**。驱动程序**调用IoConnectlnterrupt函数为特定中断注册一个处理程序handler**，然后**为这个中断指定一个中断服务例程**（**Interrupt Service Routine** , ISR）,每当**触发该中断**时，系统都会**调用注册的中断服务例程**。
* **中断描述表（IDT)**
* **存储着ISR的信息**
* 在WinDbg中可以通过**!idt命令**査看。
* **如果中断位于一个没有名字、没有签名或可疑的驱动中，则表明存在Rootkit或者恶意代码。**

**第11章 Malware Behavior**

* **Downloaders**
* **Launchers**
* **Backdoors**
* **Reverse Shell**
* **RATs**
* **Botnets**
* **RATs 和Botnets的区别**
* **Credential Stealers，什么是 Credential，什么是Credential Stealers，怎么去做Credential Stealers**
* **GINA，什么是GINA**
* **MITM Attack，什么是MITM，最主要**
* **Keystroker**
* **Persistence Mechanisms 怎么做到Persistence（三种方式，svchost是重点）**
* **Trojanized System Binaries**
* **Privilege Escalation**
* **IAT (Import Address Table) Hooking**
* **Inline Hooking**
* **Downloaders 下载器**
* 从互联网上下载其他的恶意代码，然后在本地系统中运行。
* **Launchers 启动器**
* 也称为加载器，是一类可执行文件
* 用来**安装立即运行或者将来秘密执行的恶意代码**。
* 可能在PE文件的rsrc节存储恶意代码
* **Backdoors 后门**
* 能让攻击者远程访问一个受害的机器
* 是一种最常见的恶意代码
* 常利用80端口使用HTTP协议。HTTP是出站流量最常使用的协议，所以它为恶意代码提供了一个与其他流量混淆的好机会。
* 后门拥有一套通用的功能，例如**操作注册表、列举窗口、创建目录、搜索文件，**等等。
* **Reverse Shell 反向shell**
* 从被感染机器上发起一个连接，向攻击者提供利用shell访问被感染机器的权限。
* **Netcat 反向 shell**

通过在两台机器上运行Netcat,来创建一个反向shell

* **Windows 反向 shell**

使用cmd.exe作为Windows系统中的反向shell

* **基础方法**

创建一个套接字，并与远程服务器建立连接。

绑定这个套接字与cmd.exe标准流（标准输入、标准 输出以及标准错误）。

调用CreateProcess函数用隐藏窗口的方式创建cmd.exe程序，对受害者隐藏 cmd.exe进程

* **多线程技术**

一个套接字、两个管道及两个线程的创建

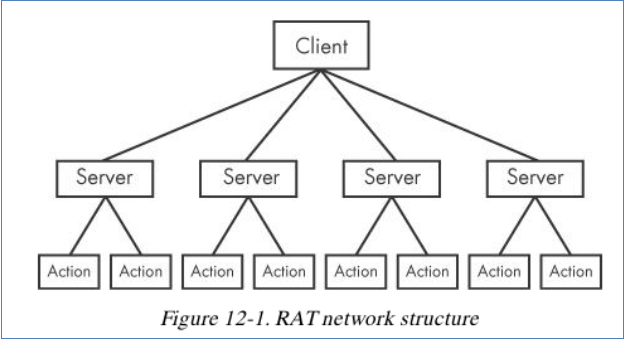
查询API调用函数CreateThread、CreatePipe。

CreatePipe用来绑定一个管道的读写端，如标准输入（stdin)和标准输出(stdout)。

CreateProcess方法用来绑定一个管道与标准流，而不是直接与套接字绑定。

调用 CreateProcess方法，恶意代码会产生两个线程：**一个用来从标准输入管道读数据**，并且**向套接字写数据**，另外**一个用来从一个套接字读数据**，并且**向一个标准输出管道写数据**。

* **RAT 远程控制工具**



* **服务器端**运行在一个被植入恶意代码的**受害主机**上。
* **客户端**作为攻击者远程操纵**运行命令和控制的单元**。
* 服务器端命令客户端开始一个连接，同时它们 也被客户端控制。
* Poison Ivy 是一个免费提供且流行的远程控制工具。
* **Botnets 僵尸网络**
* 僵尸网络是被感染主机的一个集合
* 由单一实体控制，通常由一个称为**僵尸控制器的机器作为服务器**。
* 僵尸网络的目标是尽可能多地感染机器，来构建一个更大的僵尸主机网络，从而使僵尸网络传播其他的恶意代码或蠕虫，或者执行分布式拒绝服务（DDoS)攻击。
* **Botnets v. RATs 僵尸网络与远程控制工具的比较**
* 僵尸网络感染和控制**数以百万计的主机**。

远程控制工具通常只控制**很少数量的主机**。

* 僵尸网络中的所有主机**在同一时刻被控制**

远程控制工具是**以每个受害者为单位进行远程控制**，因为远程控制工具要求攻击者与受害主机之间更紧密地交互。

* 僵尸网络用来进行**大规模攻击**

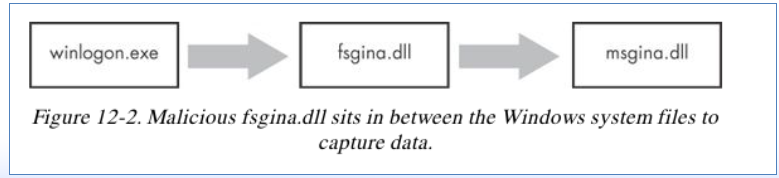
远程控制工具被用来执行**针对性的攻击**

* **Credential Stealers 登录凭证窃密器**

攻击者经常会不遗余力地去窃取登录凭证，他们主要使用以下三种类型的恶意代码：

* **等待用户登录**以窃取登录凭证的程序
* **转储Windows系统中存放信息**的程序，例如密码哈希值，程序直接使用它，或者对它进行离线破解
* **击键记**录程序
* **GINA Interception GINA拦截**
* GINA：微软图形识别和验证界面
* GINA的设计目的是让**合法第三方**通过添加一些代码，来自**定义登录过程**，如用**硬件无线射频标识（RFID)令牌或者智能卡来进行身份认证**等。
* 在Windows XP系统中，恶意代码使用GINA拦截技术来窃取用户的登录凭证。
* 恶意代码编写者利用GINA对第三方的支持来加载窃密器。
* **GINA在msgina.dll中实现**，这个DLL在用户登录系统过程中**由Winlogon可执行文件加载**。
* Winlogon也为第三方定制实现DLL程序工作，允许在Winlogon与GINA DLL之间加载第三方DLL。
* 为了方便，Windows用

HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\GinaDLL注册表项，来存储**需要Winlogon加载的第三方DLL**。



在一个实例中，我们发现恶意代码文件fsgina.dll作为GINA拦截器，安装到了上述的注册表位置。恶意文件在Winlogon 与之间工作，此时恶意代码能够拦截提交给系统认证的所有登录凭证。它可以选择将登录 凭证信息记录到硬盘，或者通过网络发送出去。

* **MITM Attack man-in-the-middle attack 中间人攻击**

恶意代码DLL必须包含真实的msgina.dll的所有导出函数

* 必须导出超过15个函数
* 大部分前缀是wlx。
* 如果你分析的一个DLL有许多包含前缀Wlx的导出函数，那么你就有很充分的理由判定这是一个GINA拦截器。



多数的导出函数只需简单地调用msgina.dll中的真正函数。

2处的调用功能是记录登录凭证信息，它将登录凭证信息作为参数，格式化字符串用来输出到.sys文件中

* **Hash Dumping 口令哈希存储（这个不是重点）**
* 转储Windows 口令哈希是恶意代码获取系统登录凭证的一种流行方法。
* 攻击者试图**抓取这些口令哈希**：

以便**离线暴力破解**，

或者**利用它们执行Pass-the-Hash攻击**。

Pass-the-Hash攻击在**无须破解或解密口令哈希**以获取明文密码的情况下，使用**LM或者NTLM哈希**来通过远程主机的身份验证，从而**获得访问权**。

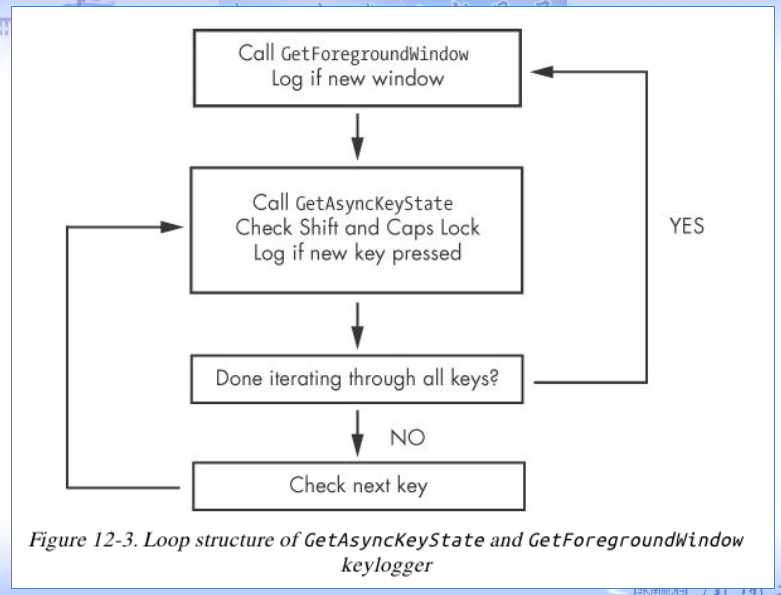
* **Pwdump和Pass-the-Hash (PSH)工具包**
* 提供哈希转储的免费黑客工具
* 开源
* 在恶意软件中经常被使用
* 多数反病毒软件**都拥有它们标准版本的特征码**, 所以通常情况下，攻击者要**逃避探测**就会**编译他们的专有版本**
* **Pwdump**

通过注入DLL到本地安全认证子系统服务（LSASS)进程进行攻击

* 从安全账户管理器（SAM)获得本地用户账户的LM和NTLM口令哈希
* DLL注入，在另一个进程中运行DLL
* 获得该进程的所有权限
* **Pass-the-Hash Toolkit PSH工具包**
* 将DLL注入lsass.exe以获取哈希值

程序命名为whosthere-alt

* 使用与Pwdump不同的API函数
* **Keystroke Logging 击键记录**
* **Kernel-Based Keyloggers 基于内核的击键记录器**
* 使用用户模式应用程序很难检测到
* 经常是rootkit的一部分
* 充当键盘驱动程序
* 绕过用户空间程序和保护
* **User-Space Keyloggers 用户空间的击键记录器**
* 常使用Windows API
* 常通过挂钩（hooking）或者轮询（polling）来实现
* **Hooking 挂钩**
* 使用SetWindowsHookEx函数在每次按下键时通知恶意软件
* **Polling 轮询**
* 使用GetAsyncKeyState＆GetForegroundWindow不断轮询键盘的状态
* GetForegroundWindow用于记录活动的窗口
* GetAsyncKeyState用来判断是否有一个按键被按下
* 如果一个按键被按下，应用程序会检查Shift键和Caps Lock键确定如何正确地记录按键。
* 一旦内部循环迭代遍历了整个按键列表，再次调用GetForegroundWindow确保用户仍在同一窗口。



* **Persistence Mechanisms 存活机制**
* **修改Run key 开机启动注册表项**
* **修改其他重要的注册表项**
* AppInit\_DLLs
* Winlogon Notify
* SvcHost DLLs
* **通过特洛伊二进制文件(trojanizing binaries)进程修改文件**
* **不修改注册表和文件——DLL加载顺序劫持**
* **修改Run key 开机启动注册表项**
* HKEY\_LOCAL\_MACHINE\ SOFTWARE\Microsoft\ Windows\ CurrentVersion\ Run
* **ProcMon shows registry modifications**
* **Applnit\_DLL**
* AppInit\_DLLs值可在以下注册表项中找到：

HKEY\_LOCAL\_MACHINE\ SOFTWARE\Microsoft\Windows NT\ CurrentVersion\Svchost

* AppInit\_DLLs包含空格分隔的DLL列表，这些DLL程序会在进程加载User32.dll时被加载。
* 插入DLL路径到注册表AppInit\_DLL，会让DLL程序获得加载机会。
* 注意：很多进程都会加载User32.dll，这些进程也会加载AppInit\_DLL。恶意代码编写者**通常只针对一个单独进程**，但是**AppInit\_DLL会被加载到多个进程中**。因此，恶意代码编写者**执行攻击负载(playload)前必须检查它运行在哪个进程中**。这种检查通常在恶意DLL程序的DllMain函数中完成。
* **Winlogon Notify**
* 恶意代码编写者可以**挂钩一个特殊的Winlogon事件**，如登录、注销、关机以及锁屏，等等。这甚至可以允许恶意代码在安全模式下加载。
* 下面注册表键中的注册表项包含Notify的值。 HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\
* 当winlogon.exe产生一个事件时，Windows系统会**检查Notify注册表键**来**查找处理这个事件的 DLL程序**。
* **SvcHost DLL**
* **Svchost.exe是从动态链接库中运行服务的通用主机进程**
* Windows操作系统通常同一时刻运行多个Svchost.exe实例
* 每个Svchost.exe实例包含**一组服务**，这些服务能够使开发、测试以及服务组管理变得更简单。
* 这些组被定义在下列的注册表位置（每个值代表一个不同的组）：

HKEY\_LOCAL\_MACHINE\ SOFTWARE\ Microsoft\Windows NT\ CurrentVersion\ Svchost

* 服务被定义在下列的注册表位置：

HKEY\_LOCAL\_MACHINE\ System\CurrentControlSet\ Services\ ServiceName

* Windows服务包含多个注册表值，其中多数提供关于服务的信息。
* **ServiceDLL：**

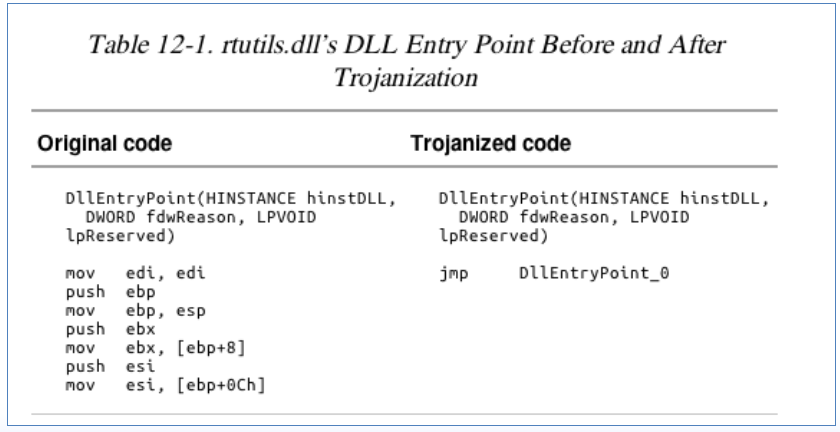
**所有的svchost.exe DLL**包含一个**拥有ServiceDLL值的Parameters键**

恶意代码编写者设置ServiceDLL的值为恶意DLL的位置

* Windows拥有很多预先定义的服务组，所以**恶意代码通常不会创建一个新的服务组**，因为新创建的服务组很容易被探测。相反，多数恶意代码会**将其加入一个已经存在的组或者覆盖一个无关紧要的服务**——经常使用netsvcs服务组中很少使用的服务。

要识别这种技术，使用**动态分析监视Windows的注册表**，或者在**反汇编中查找CreateServiceA函数**。如果恶意代码修改这些注册表键，你就会知道它们使用了这种存活技术。

* **Trojanized System Binaries 特洛伊木马化系统二进制文件**
* 修改系统的二进制文件，当被感染的二进制文件下次运行或者加载时，将会强制运行恶意代码
* DLL是恶意代码 编写者欢迎的目标。
* 通常修改一个系统文件的函数入口点，使它跳转到恶意代码。这些修改补丁，**覆盖函数的开始代码或者覆盖其他一些并不影响特洛伊木马化DLL正常操作的代码**。
* 为了不影响二进制文件的正常操作，**恶意的代码被加入到可执行文件的一个空节中**。插入的代码通常用来加载恶意代码，插入的函数并不关心它被插入到被感染的DLL的何处。**（这段话没看懂）**
* **加载恶意代码后**，为了让DLL的操作与插入前的代码一致，**代码会跳转回原始的DLL程序**。



* **DLL Load-Order Hijacking DLL加载顺序劫持**
* **下面是在Windows XP上加载DLL的默认搜索顺序：**

1. 加载应用程序的目录。

2. 当前目录。

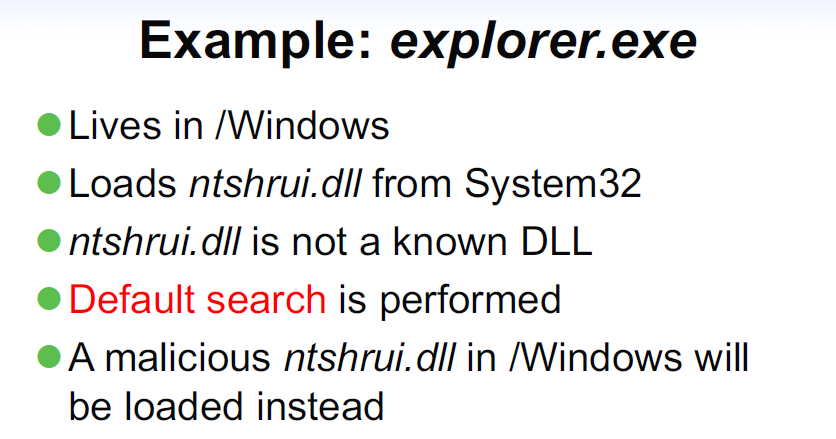
3. 系统目录（使用GetSystemDirectory函数获取的路径，例如.../Windows/System32/)。

4. 16位子系统的系统目录（例如.../Windows/System/)。

5. Windows 目录（使用 GetWindowsDirectory 获取的路径，例如.../Windows)。

6. PATH环境变量里列出的目录。

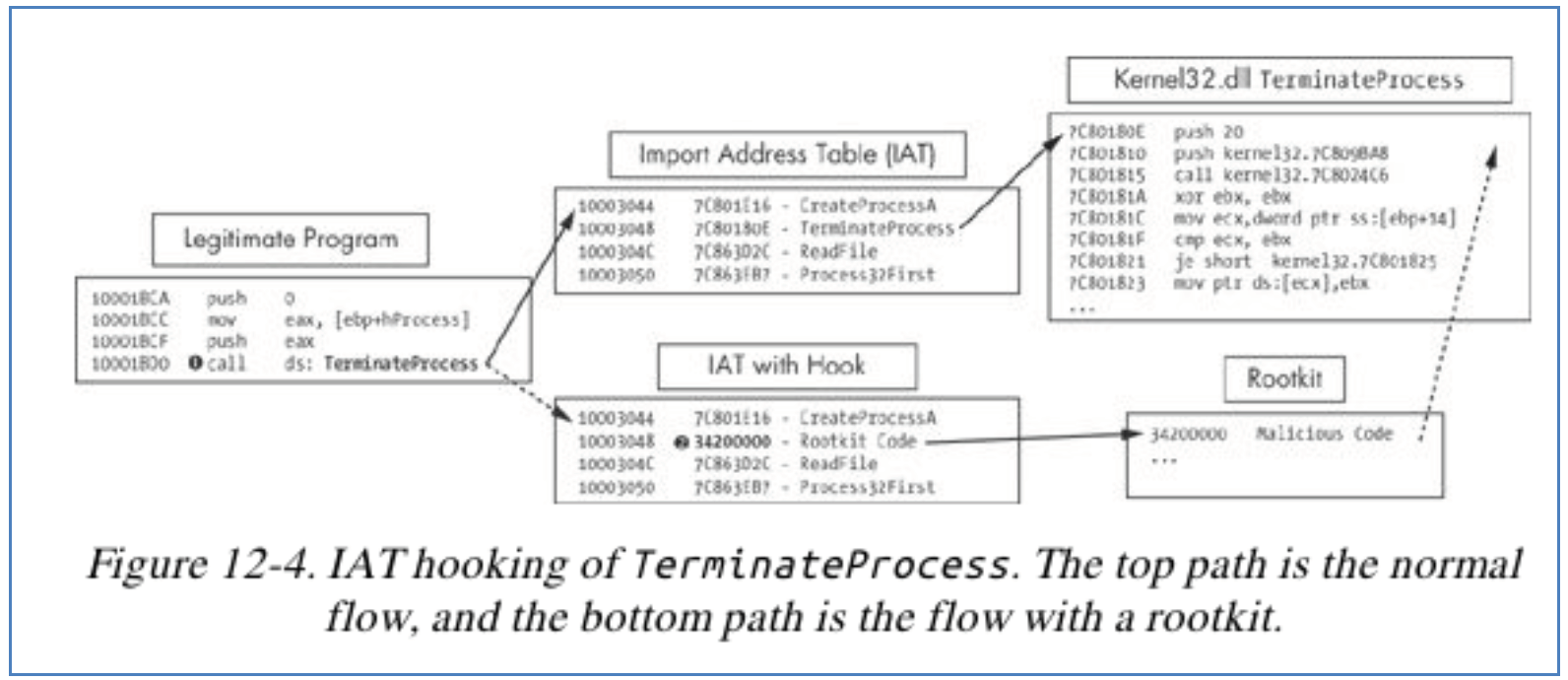
* **在Windows XP系统中，利用KnownDLL注册表键可以跳过DLL的加载过程**
* **DLL加载顺序劫持可以加载/System32目录中未被 KnownDLL保护的DLL程序。**



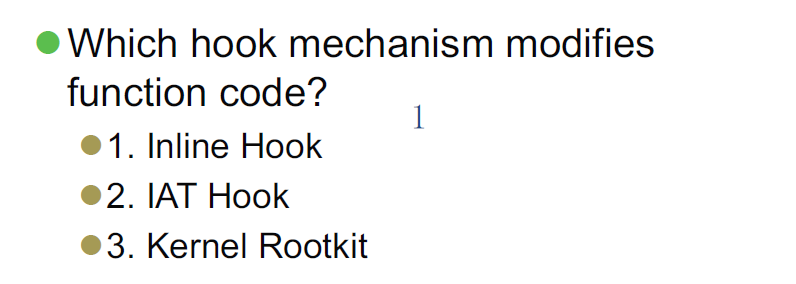
* **Privilege Escalation 提权**
* **No User Account Control**
* 多数提权攻击是利用本地系统已知漏洞或者0day漏洞进行攻击，其中多数可以在Metasploit Framework中找到
* DLL加载顺序劫持也可以被用来执行特权操作
* **使用SeDebugPrivilege**
* 用户运行的进程无法完成所有操作
* TerminateProcess或CreateRemoteThread等函数需要系统权限（管理员以上）
* SeDebugPrivilege特权用于调试
* 允许本地管理员帐户升级到系统权限
* 1获得访问令牌
* 2 调用AdjustTokenPrivileges为System提升特权
* **User-Mode Rootkits 用户态Rootkit**
* 修改操作系统的内部功能
* 隐藏文件，网络连接，进程等
* 内核模式rootkit更强大
* 本节介绍用户模式rootkit
* IAT (Import Address Table) Hooking
* Inline Hooking
* **IAT (Import Address Table) Hooking**
* **可能修改**

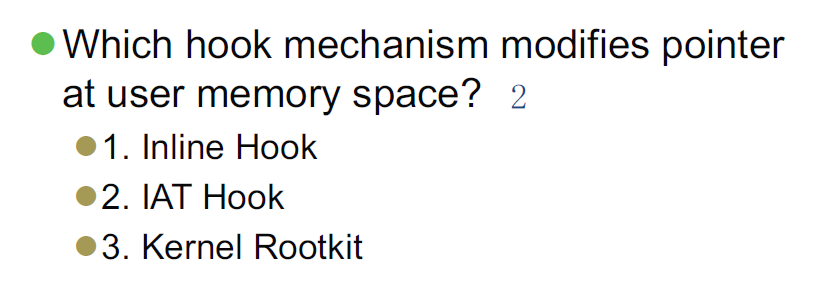
**IAT（导入地址表）或 EAT（出口地址表）**

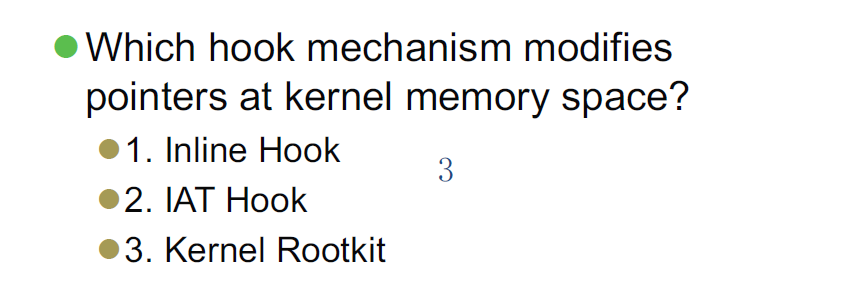
* **PE文件的一部分**



* **Inline Hooking**
* 通过覆盖导入DLL中API函数的代码来实现
* 必须等到DLL被加载后才能执行
* IAT Hook只简单地修改函数指针，但Inline Hook将修改实际的函数代码

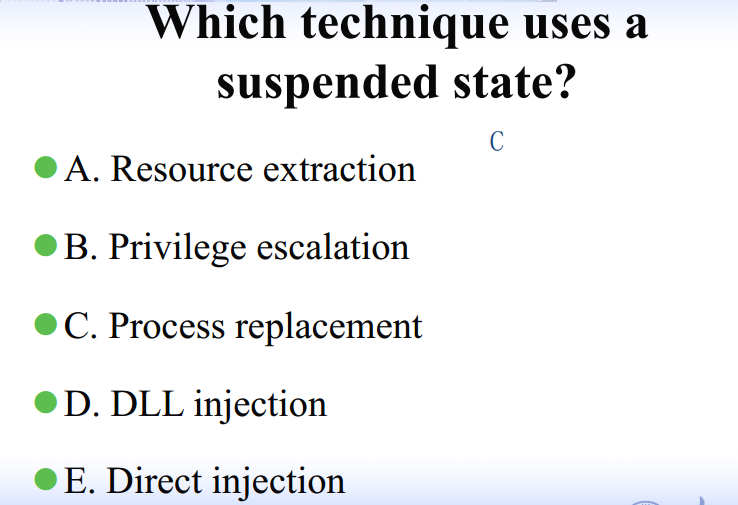






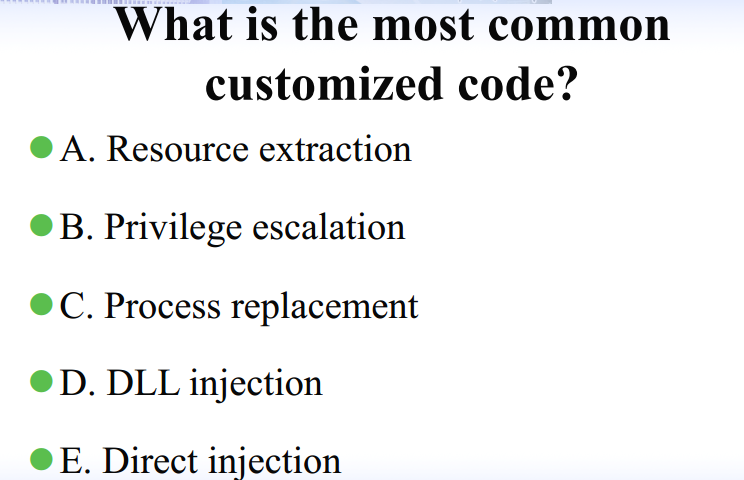
**第12章 Covert Malware Launching 隐蔽的恶意代码启动**

* **Launcher**
* **Process Injection**
* **DLL Injection**
* **Direct Injection**
* **Process Replacement**

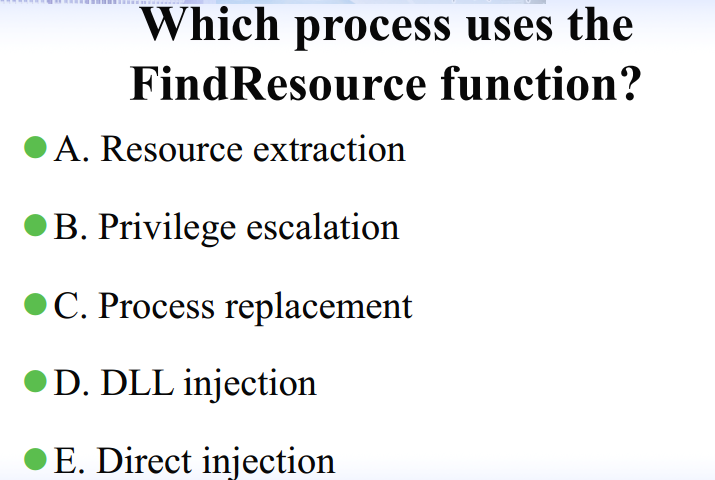




E：直接注入需要大量的定制代码

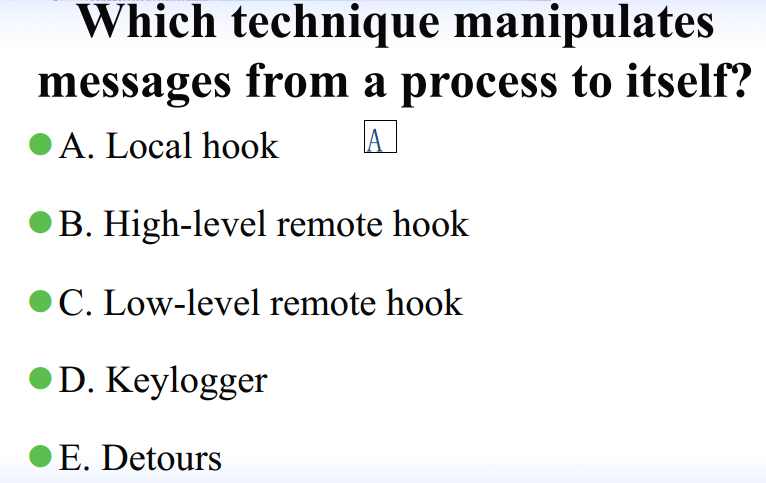


D：DLL注入是最常使用的秘密加载技术



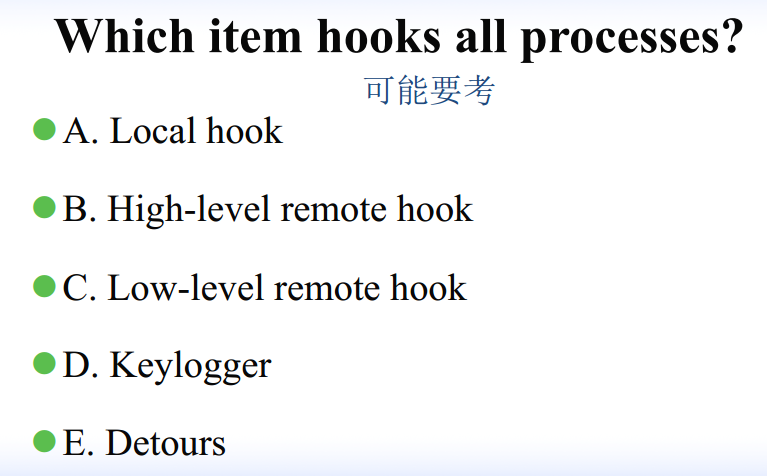
A

* **Hook Injection**

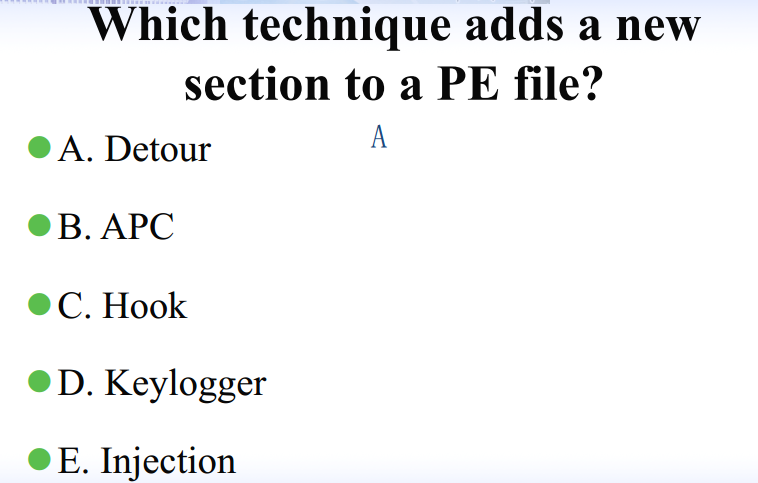


哪种技术可以处理从进程到自身的消息？

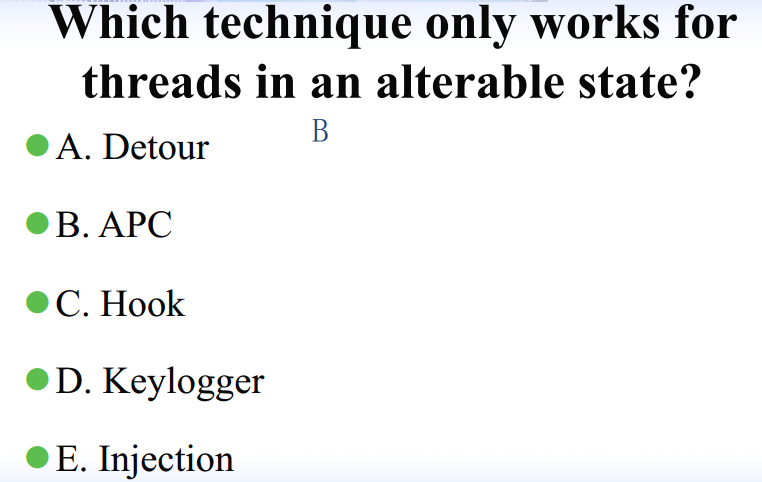
A



B



A

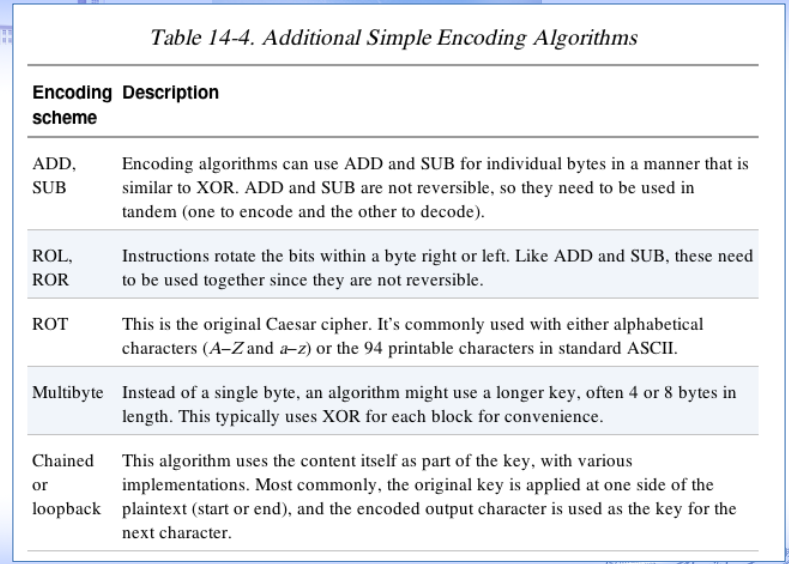


APC针对**可警告等待状态**的线程

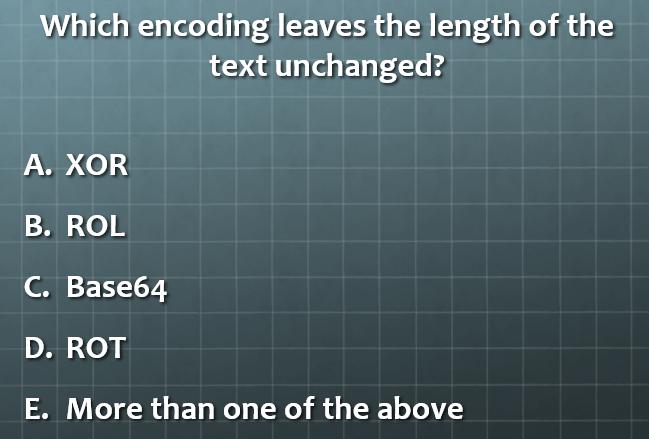
选B

**第13章 Data Encoding 数据加密**

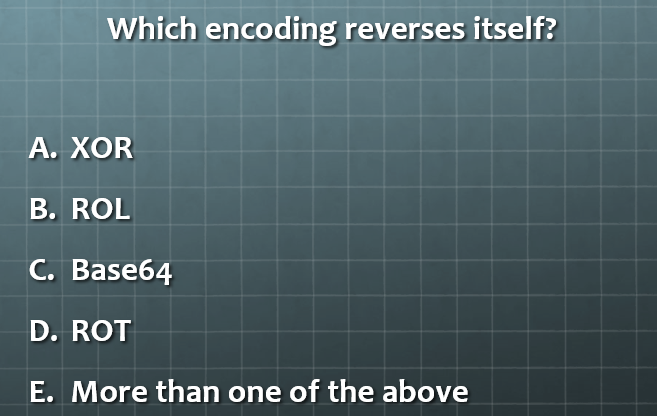
* **Cipher**
* **XOR**
* **Other Simple Cipher**



* **Base64 ：加密结果全是可见字符，不会因为密文中的不可见字符，导致传输密文时出现歧义**
* **不标准的Base64**
* **Cryptography 能对抗蛮力攻击的叫 Cryptography**
* **Entropy 熵 熵值越大，混沌程度越高，说明越有可能是密文**
* **块大小64字节，熵阈值5.95**
* **块大小256字节，熵阈值7.9**
* **Self-Decoding 自解密，类似脱壳时的dump**



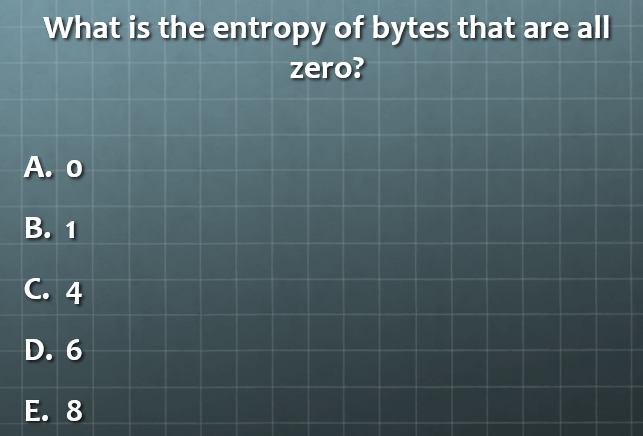
ABD（E）



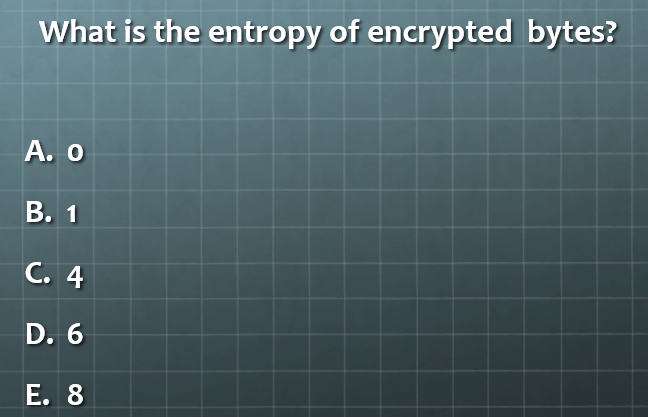
A



C



A



E？

10道单选

5道多选

8道简答

2道大题