**1.1知识点：计算机病毒的定义和类型**

**1.计算机病毒的定义**

计算机病毒，是指编制或者在计算机程序中插入的破坏计算机功能或者毁坏数据，影响计算机使用，并能自我复制的一组计算机指令或者程序代码。

广义的计算机病毒定义，也称为恶意代码

任何以某种方式对用户、计算机或网络造成破坏的软件，都可以被认为是恶意代码，包括计算机病毒、木马、蠕虫、Rootkit、勒索病毒、间谍软件，等等。

**2.计算机病毒的功能分类**

**根据功能分类：**  
    1、**后门：恶意代码将自身安装到一台计算机来允许攻击者访问**。后门程序通常让攻击者只需很少认证甚至无须认证，便可连接到远程计算机上，并可以在本地系统执行命令。

2、**僵尸网络：与后门类似，也允许攻击者访问系统**。但是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相同的命令。

3、**下载器：这是一类只是用来下载其他恶意代码的恶意代码**。下载器通常是在攻击者获得系统的访问时首先进行安装的。下载器程序会下载和安装其他的恶意代码。

4、**间谍软件：这是一类从受害者计算机上收集信息并发送给攻击者的恶意代码**。比如嗅探器、密码哈希采集器、键盘记录器等。这类恶意代码通常用来获取E-mail、在线网银等账号的访问信息

5、**启动器：用来启动其他恶意程序的恶意代码**。通常情况下，启动器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。

6、**内核套件：设计用来隐藏其他恶意代码的恶意代码**。内核嵌套通常是与其他恶意代码（如后门）组合成工具套装，来允许攻击者提供远程访问，并且使代码很难被受害者发现。

7、**勒索软件：设计成吓唬受感染的用户，来勒索他们购买某些东西的恶意代码**。这类软件通常有一个用户界面，使它看起来像是一个杀毒软件或者其他安全程序。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是购买他们的“软件“，而事实上，他们所卖软件的全部功能只不过将勒索软件进行移除而已。

8、**发送垃圾邮件的恶意代码：这类恶意代码在感染用户计算机之后，便会使用系统与网络资源来发送大量的垃圾邮件**。这类恶意代码通过为攻击者出售垃圾邮件发送服务而获得收益。

9、**蠕虫或计算机病毒：可以自我复制和感染其他计算机的恶意代码**。

恶意代码还经常跨越多个类别。

**3.计算机病毒的攻击目标分类**

**根据攻击目标分类：**

**1、大众性的恶意代码**：比如勒索软件，采用的是一种撒网捞鱼的方法，设计为影响到尽可能多的机器。这类恶意代码比较普遍，通常不会太过复杂，容易被检测和防御，因为安全软件以这类恶意代码为防御目标。

**2、针对性恶意代码**：比如特制后门，是针对特定组织而研制的。针对性恶意代码在网络上是比大众性恶意代码更大的安全威胁，因为他们不是广泛传播的，而你的安全产品很可能不会帮你们防御它们。针对性恶意代码通常是非常复杂的，而你对它的分析往往要借助于一些高级分析技巧。

Types of Malware

•Backdoor: remote access

•Botnet: a army

•Downloader: install other malware

•Lancher: run other malware

•Rootkit: conceal malware

•Worm or Virus: recruit new machines

•Trojan or Ransomware: make money

**1.2知识点：计算机病毒分析的目标**

**1.计算机病毒分析**

计算机病毒分析，通常是为一起计算机病毒事件的应急响应提供所需信息，包括以下的分析内容：

1、系统到底发生了什么？

2、定位被感染的计算机和可疑的程序。

3、对可疑文件进行分析。

4、提取出可以在系统和网络上检测病毒的特征码。

5、衡量并消除计算机病毒所带来的损害。

**2.可疑文件分析**

对病毒文件进行分析是对计算机病毒进行应急响应的关键。

计算机病毒是如何工作的

如何识别计算机病毒

如何消除计算机病毒

**3.计算机病毒特征码**

特征码可用于查杀系统中的病毒文件、拦截病毒网络传播

1、基于主机的特征码（感染迹象）：用于在受感染主机上检测出恶意代码。这些感染迹   象经常是恶意代码所创建或修改的文件，或是它们对注册表的特定修改。

2、病毒特征码：反病毒软件所使用的病毒特征码关注恶意代码本身的特性。

3、网络特征码：通过监测网络流量来检测恶意代码的。僵尸网络的通信数据包、与恶意IP地址的连接等。

主机特征码与反病毒软件所使用的病毒特征码不同，恶意代码感染迹象关注的是恶意代码对系统做了什么，而不是恶意代码本身的特性。这使得有时它们较反病毒软件特征码会更加有效，比如检测那些经常变化自身形态的多态性恶意代码（多态、变形、加壳、花指令），甚至是恶意代码已经将自身文件从硬盘中删除。

网络特征码：

通过分析计算机病毒的网络通信数据提取出的特征码。

网络特征码包括恶意的IP地址、URL、邮件、攻击数据包、计算机病毒间的通信协议等。

可以和主机特征码相结合，提供更高的检测率和更少的误报。

在没有进行恶意代码分析时创建，但在恶意代码分析帮助下提取的特征码往往更加有效的，可以提供更高的检测率和更少的误报

**1.3知识点：计算机病毒分析技术概述**

静态分析：静态分析方法是在没有运行计算机病毒时，

对其进行分析的相关技术。常用的分析工具有VirusTotal、strings、IDA pro等。

动态分析：动态分析方法则需要运行计算机病毒。常用的分析工具有RegShot、Process Monitor、OllyDbg等。

这两类技术又进一步分为基础分析和高级分析。

**1. 静态分析基础技术**

静态分析基础技术包括分析病毒可执行文件，但不查看具体CPU指令的分析技术。常用的分析工具有VirusTotal、strings等。非常快速、简单。难以分析复杂的计算机病毒，而且可能会漏掉一些重要的行为。

**2. 动态分析基础技术**

动态分析基础技术通过运行计算机病毒，分析病毒在系统和网络上的行为，提取病毒的特征码。需要建立安全分析环境。快速、简单。也漏掉一些重要病毒功能。

**3. 静态分析高级技术**

对计算机病毒内部机制的逆向工程，通过将可执行文件装载到反汇编器中，查看程序CPU指令，来分析病毒到底做了什么。深入分析计算机病毒。需要掌握汇编语言、Windows系统编程等知识。易受代码混淆技术的干扰，例如加壳、花指令等。

**4. 动态分析高级技术**

使用调试器来分析一个病毒运行时刻的内部状态。动态执行每一条指令，验证静态高级分析的结果。可以缓解代码混淆的干扰。只能覆盖一条计算机病毒的执行轨迹。

**5. 恶意代码分析通用规则**

首先，不要过于陷入细节，关注主要功能。你应该在进入细节之前有一个概要性的理解。

其次，尝试多从不同角度，多使用不同工具和方法来分析恶意代码。

最后，恶意代码分析就像是猫抓老鼠的游戏，应该能够快速地应对恶意代码的新变化。

**2.1知识点：杀毒软件**

**1.杀毒软件**

病毒特征库是已知计算机病毒样本的特征数据库。

启发式规则是对特征库扫描技术的补充在已知计算机病毒分析经验的基础上，总结出来的规则可以检测到未知的计算机病毒变种。

**2.杀毒软件的挑战**

\*计算机病毒的编写者可以很容易的修改自己的代码，从而改变这些病毒的各种特征，来躲避杀毒软件的检测。

垃圾指令

多态技术：语义不变，语法混淆

变形技术：功能不变，语义混淆

单向执行技术：未解数学猜想、哈希值

\*新的病毒家族

新出现的计算机病毒家族，通常不会被杀毒软件识别，因为病毒特征库里面没有这些新的样本。启发式检测也可以被新的病毒家族绕过去，因为新的病毒家族的行为与已知家族的不一样。

**3.多杀毒软件的交叉检测**

（1）不同的杀毒软件使用了不同的特征库和启发式检测方法

（2）增加计算机病毒躲避的难度

**2.2知识点：哈希值：恶意代码指纹**

**2.4知识点：加壳与混淆**

**1.加壳与混淆**

\*目的

使计算机病毒更难被检测和分析

\*混淆

隐藏计算机病毒程序的信息

\*加壳

压缩计算机病毒的文件大小

加密

\*正常程序

Strings指令可以搜索出来很多字符串

\*加壳或加密后的程序

Strings指令搜索出来的字符串少

缺少对分析有用的字符串

**2.5知识点：PE文件格式**

**1.PE文件格式**

\*Portable Executable File Format

\*Windows可执行程序和动态链接库的文件格式

\*包含Windows将文件从硬盘装载到内存中执行的必要信息

**2.PE文件头**

可执行代码信息

程序的类型，exe、dll

依赖的动态链接库

需要的内存空间

**3.LoadPE工具**

**4.PE文件中的分节**

.text:  包含了CPU的执行指令，所有其他节存储数据和支持性的信息。一般来说，这是唯一可以执行的节，也是唯一包含代码的节。

.rdata:  通常包含导入导出函数信息，这个节中还可以存储程序所使用的其他只读数据。

.data:  包含了程序的全局数据。

.rsrc:  该节包含由可执行文件所使用的资源，而这些内容并不是可执行的，比如图标、图片、菜单项和字符串。

**5.PEview工具**

**6.Resource Hacker**

可以利用该工具查看.rsrc资源节。当单击Resource Hacker工具中分析获得的条目时，就会看到字符串、图标和菜单。

// 检查文件大小和是否为PE文件

private global rule FileSizeAndIsPE {

condition:

filesize < 150KB and

uint16(0)== 0x5A4D and // MZ 头

uint32(uint32(0x3C))== 0x00004550 // PE 头

}

**3.1知识点：虚拟机的结构**

1.1未知的计算机病毒充满了不确定性

1.2动态分析

•       运行计算机病毒，监控病毒的行为

•       需要一个安全、可控的运行环境

•       传染性： 计算机病毒会感染网络中的其它计算机

•       隔离： 阻断病毒与Internet和其它电脑的连接

•       难以清除

1.3隔离的物理机器

•       缺点

•       没有Internet连接，部分病毒的行为就不会展现出来

•       有些病毒难以从系统中清除，分析结束需要给机器重装系统

•       优点

•       有些病毒会探测执行环境，在虚拟机中无法正常执行

**3.3知识点：使用恶意代码分析机**

**1．文件传输**

再接下来，你需要安装VMware tools。从VMware菜单中，选择虚拟机（VM）→安装VMware tools（Install Vmware Tools）。VMware tools会让鼠标和键盘的响应变得更敏捷。还允许访问共享文件夹，进行拖放式文件传输。

**2.连接和断开外围设备**

大多数外部设备要么连接到宿主机上，要么连接到虚拟机上，不能同时连接两者。

VMware允许你连接和断开外部设备。如果你将一个usb设备连接到一台电脑，而同时有一个虚拟机窗口正处于活跃状态，VMware将会把usb设备连接到虚拟机中，而不是宿主主机。可以在VM→Settings→USB Controller菜单项取消选中 “Automatically connect new USB devices”复选框，防止USB设备被自动连接到虚拟机。

**3.拍摄快照**

在安装完操作系统和恶意代码分析工具，并且完成配置后，做一个快照，作为一个初始干净的基础镜像。

VMware的虚拟机快照允许你保存一个计算机的当前状态，并可以回滚到以前的一个状态。与Windows的还原功能类似。

VMware快照管理器允许你在任何时间返回任何快照，无论你是否做了其他额外的快照，或是对虚拟机进行任何操作。也可以对快照进行分支，让他们跟踪不同的分析路径。

例如，

1.     在分析恶意样本1的时候，项进行别的样本的分析

2.     在样本1的现场做一个快照。

3.     回滚到基础镜像快照

4.     你开始分析恶意样本2.

5.     你拍摄了样本2现场的快照

**4.1知识点：沙箱分析**

**1．常见的沙箱**

沙箱包含一个虚拟环境，通过某种方式模拟网络服务，以确保被测试的软件和恶意代码能正常执行，来进行基本的动态分析。

许多分析恶意代码的沙箱，诸如Norman沙箱、GFI沙箱、Anubis、Joe沙箱、ThreatExpert、BitBlaze和Comodo恶意代码分析，等等，都提供免费分析恶意代码的服务。其中，Norman沙箱、GFI沙箱最受欢迎。

**2．GIF沙箱分析报告**

**GFI对win32XYZ.exe样本的分析报告**

GFI沙箱生成的报告所包含的小节数量也会根据分析结果不同而存在差异：

1.分析摘要，列出了静态分析信息，以及对动态分析结果的一个高层摘要总结。

2. 文件活动，列出了恶意代码影响的每个进程所打开、创建和删除的文件列表。

3. 互斥量创建，列出了恶意代码创建的互斥量列表。

4. 注册表，列出了对注册表的修改活动列表。

5. 网络行为，包含了恶意代码实施的各种网络行为，包括创建监听端口、执行DNS请求，等等。

6. VirusTotal结果，列出了对恶意代码进行VirusTotal反病毒引擎扫描的结果。

**3．沙箱的挑战**

1. 沙箱只能简单地运行可执行程序，不能带有命令行选项。

2. 沙箱也不能记录所有事件，因为不管是你还是沙箱，都不会等待足够长的时间让所有事件发生。

3. 恶意代码经常检测它是否运行在一个虚拟机里，并且如果检测到了虚拟机，恶意代码将会停止运行，或者表现异常。不是所有沙箱都能完善地考虑这一问题。

4. 一些恶意代码需要系统上拥有特定地注册表项或者文件才会执行，而这些在沙箱内是找不到的。这些就可能需要包含一些合法数据，比如控制命令或者加密密钥。

5. 如果恶意代码是一个DLL，一些导出函数可能不会被恰当地调用，因为一个DLL文件不像可执行文件那样容易运行。

6. 沙箱环境地操作系统对恶意代码来说可能不正确。

7. 沙箱不能告诉你恶意代码做了什么。他能报告基本功能，但它不能告诉你恶意代码具体是一个什么程序，需要你自己总结。

**4.2知识点：运行病毒和进程监视**

**1．DLL格式文件的运行**

1、在所有的Windows版本中都包含rundll32.exe程序，它提供了一个运行DLL的平台

Export值必须是一个DLL文件导出函数表中的函数名或者序号。

2、DLL形态的恶意代码也可能需要被安装成一个服务，有时就像下面这个例子中的一样，在ipr32x.dll中导出了InstallService函数,ServiceName参数必须提供给恶意代码，让它能被安装起来并运行起来。而在Windows系统中启动指定的服务，可以使用net start命令。

**2．进程监视器的介绍**  
进程监视器（Process Monitor）是Windows系统下的高级监视工具，它提供一种方式来监控注册表、文件系统、网络、进程和线程行为。它结合并增强了两种工具的功能：文件监视器FileMon和注册表监视器RegMon。

**3．进程监视器的显示**

进程监视器显示了一个可配置栏，包含了每个事件的不同信息，如事件序列号、时间戳、引发事件的进程名、事件操作、事件使用的路径、事件结果等。

**进程监视器监视mm32.exe实例**

图展示了在一台机器上运行mm32.exe恶意代码所监视的事件列表。通过阅读事件操作栏，能很快告诉你mm32.exe在系统上做了什么操作，包括注册表和文件访问。

**4. 进程监视器的过滤**

程序过滤：只监控系统上运行的一个可执行程序，这个功能对病毒分析特别有用，因为可以对你运行的恶意代码设置一个过滤器。

调用过滤：针对特定的系统调用设置过滤，比如RegSetValue、CreateFile、WriteFile,或者其他可疑或具有破坏性的调用。

**设置进程监视器的过滤器**

正如你在图中前两行看到的，我们通过进程名和操作进行过滤。我们添加了一个进程mm32.exe,以及操作RegSetValue的过滤器。

注意：过滤器仅仅用于显示过滤，当你运行恶意代码时，所有产生的系统调用都会被捕获，包括原始可执行程序释放恶意程序的系统调用。

**5、进程监视器的自动过滤功能**

进程监视器提供默认下面四种过滤功能

**注册表**：通过检查注册表操作，你能辨别一个恶意程序是怎样把自己添加到注册表中的。

**文件系统**：检查文件系统能显示恶意程序创建的所有文件，或者它使用的配置文件。

**进程行为**：检查进程行为能告诉你恶意程序是否启动了其他的进程。

**网络**：识别网络能向你展示恶意程序监听的任意端口。

**4.4知识点：网络模拟**

**1. 网络模拟**

恶意代码经常会连接到命令与控制服务器，你需要采用一些模拟网络的基本步骤，用来快速获得网络资源，而不需要实际地连接互联网。这些资源包括DNS域名系统、IP地址和数据包记录器。

**2. 使用ApateDNS**

ApateDNS是一款来自Mandiant公司的免费软件，是用来查看恶意代码发出DNS请求最快速的方式。ApdateDNS在本机上通过监听UDP的53端口，对用户指定的IP地址做给出虚假的DNS响应。它用你指定的IP地址去响应DNS查询请求。ApateDNS可以使用十六进制和ASCII码来显示所有接收到的请求。

**3. 使用Netcat进行监视**

Netcat被称为“TCP/IP协议栈瑞士军刀”，可以被用在支持端口扫描、隧道、代理、端口转发等的对内对外连接上。在监听模式下，Netcat充当一个服务器，而在连接模式下作为一个客户端。Netcat从标准输入得到数据进行网络传输，而它得到的数据，又可以通过标准输出显示到屏幕上。

Netct的执行命令，显示了监听一个端口所需要的选项。-l选项意思是监听，-p选项指明了监听的端口。正如你在表中看到的，RShell是一个反向的shell，它不会立即提供一个shell。它的网络连接首先看起来像是一个向[www.google.com的HTTP](https://www.google.xn--comhttp-yk1r/" \t "_blank) POST请求，而这个POST数据是假冒的，RShell很可能插入了一个混淆之后反向的shell，因为网络分析师经常只看到一个会话的开始部分。

**4. 使用Wireshark进行数据包监听**

Winshark是一个开源的嗅探器，是一个截获并记录网络数据包的工具。Wireshark提供可视化、包一流分析和单个数据包的深入分析。

Wireshark的界面有四部分，如图所示。

1. 过滤栏用来过滤显示的数据包。

2. 数据包列表显示了所有满足过滤条件的数据包。

3. 数据包细节窗口显示了当前选中包的详细内容。

4. 十六进制窗口显示了当前数据包的十六进制内容，十六进制窗口和数据包细节窗口紧密关联，会高亮显示你所选择的任何域。

**5. 使用INetSim**

INetSim是一款基于Linux模拟常见网络服务的软件。INetSim是一款提供模拟服务最好的免费工具，它通过模拟服务（如HTTP、HTTPS、FTP、IRC、DNS、SMTP等），允许你在隔离的动态分析系统里面，构造一个虚拟的网络环境来分析未知恶意代码的网络行为。代码清单列举了INetSim默认模拟的所有服务，它们都随程序一起启动。

**使用INetSim欺骗NMAP扫描器**

1、INetSim一些最好的特性被融进了HTTP和HTTPS服务模拟中，INetSim几乎能提供所有的文件请求服务。

2、INetSim可以记录所有的连入请求与连接。如 ：使用端口号的修改、是否连接到一个标准服务、设置请求后返回一个页面或者一个项目。

3、INetSim的Dummy服务可以记录恶意代码连接的所有端口和发送数据。

**5.1知识点：逆向工程**

**1.计算机系统的抽象层次**

解释型语言Interpreted Languages

**硬件**：硬件层是唯一一个物理层，由电子电路组成。

XOR门、AND门、和NOT门等逻辑运算器的复杂组合，称为数字逻辑。

由于物理特性，硬件很难被软件操作。

**微指令：**微指令层又称为固件。微指令只能在为它设计的特定电路上执行。这层由一些微指令构成，它们从更高的机器码层翻译而来，提供了访问硬件的的接口。

病毒分析时，通常不关心微指令，因为微指令是为特定的计算机硬件设计的。

**机器码：**机器码层由操作码组成，操作码是一些十六进制形式的数字，用于告诉处理器你想它做什么。

机器码一般由多条微指令实现，这样底层硬件就能实际执行代码了。而机器码本身又由高级语言编写的程序编译而来。

**高级语言：**大部分程序员使用高级语言。

高级语言对机器层做了很强的抽象，从而可以轻松地使用程序逻辑和流控制机制。

高级语言包括C、C++等。它们被一个编译器经过称为编译的过程转化为机器码。

**解释型语言：**解释型语言位于最高层。

很多程序员使用诸如C#、Perl、NET、Java等解释语言。

这一层的代码不会被编译为机器码。而是被翻译为了字节码。

**低级语言：**低级语言是计算机体系结构指令集的人类易读版本，主要是汇编语言。

恶意代码分析师使用这一层，因为对人来说，机器码太难理解了。

我们使用反汇编器来生成低级语言的文本，这些文本由一些简单的助记符组成，如mov和jmp。汇编语言存在一些不同的语法。

**2.逆向工程**

恶意代码存储在磁盘上时，通常是机器码层的二进制形式。前面提到，机器码是一种计算机可以高效执行的代码形式。而我们的反汇编恶意代码，就是使用反汇编器，将恶意代码二进制文件作为输入，输出汇编语言代码。

\*不同的CPU，汇编指令是不一致的

x86 – 32-bit Intel 最常用的指令集

x64 – 64-bit Intel

SPARC, PowerPC, MIPS, ARM

Windows 、Linux操作系统运行在x86 or x64指令集上

x64 系统可以运行x86 的程序

\*大部分计算机病毒是针对的x86指令集

• BasicStaticAnalysis • Looks at malware from the outside

• BasicDynamicAnalysis • Only shows you how the malware operates in one case

.text --instructions for the CPU to execute

• .rdata--imports & exports

• .data –global data

• .rsrc–strings, icons, images, menus

**5.3知识点：cpu寄存器**

**1.寄存器**

寄存器是可以被CPU直接访问的少量数据存储器

访问其中内容的速度会比访问其他的存储器要快

可以将它们归为4类

**2.寄存器的分解**

所有的通用寄存器的大小都是32位，可以在汇编代码中以32位或16位引用。

有四个寄存器（EAX、EBX、ECX、EDX）还可以8位值的方式引用，从而使用其最低的8位。

例如，AL指向EAX寄存器的最低8位，而AH指向它的次低8位。

**3.通用寄存器**

通用寄存器一般用于存储数据或者内存地址，而且经常交换着使用以完成程序。

除了指令的定义，通用寄存器还被程序一致特性使用。

EAX 累加寄存器

EBX 基址寄存器

ECX 计数寄存器

EDX 数据寄存器

EBP 基地址指针 basic pointer

ESP 栈顶指针 stack pointer

ESI 源索引 source index

EDI 目标索引 destination index

• EAX extended accumulator register • EBX extended base register • ECX extended counter register • EDX extended data register • EBP extended basic pointer • ESP extended stack pointer • ESI extended source index • EDI extended destination index

• ZF Zero flag • Set when the result of an operation is zero • CF Carry flag • Set when result is too large or small for destination • SF Sign Flag • Set when result is negative, or when most significant bit is set after arithmetic • TF Trap Flag • Used for debugging—if set, processor executes only one instruction at a time

**4.标志寄存器**

EFLAGS寄存器是一个标志寄存器。

在x86架构中，EFLAGS是32位的，每一位是一个标志。

在执行期间，每一位要么是置为位（值为1），要么是清除（值为0），并由这些值来控制CPU的运算，或者给出某些CPU运算的值。

ZF：当一个运算的结果等于0时，ZF被置位，否则被清除。

CF:当一个运算的结果相对于目标操作数太大或者太小的时候，CF被置位，否则清除。

SF：当一个运算的结果为负数，SF被置位；若结果为正数，SF被清除。对算术运算，当运算结果的最高位值为1时，SF也会被置位。

TF：TF用于调试。当它被置位时，x86处理器每次只执行一条指令。

**5.EIP指令指针**

在x86构架中，EIP寄存器，又称为指令指针或程序计数器，保存了程序将要执行的下一条指令在内存中的地址，EIP的唯一作用就是告诉处理器接下来执行哪条指令。

**5.4知识点:汇编指令**

**1.mov指令**

用于将一个数据从一个位置移动到另一个位置。换言之，它是用于读写内存的指令。

mov指令可以将数据移动到寄存器或内存，其格式是：mov destination, source。

**2.lea指令**

“load effective address”的缩写，格式是lea destination,source

lea指令是用来将一个内存地址赋给目的操作数。

例如，lea eax,[ebx+8]就将EBX+8的值给EAX。

相反的，mov eax,[ebx+8]则加载内存中地址为EBX+8处的数据。

lea eax,[ebx+8]就和mov eax,ebx+8指令实际上是等价的。

**3.算数运算指令**

加法指令的格式是add destination, value

减法指令是sub destination, value

sub指令会修改两个重要的标志：ZF和CF。如果结果为零，ZF被置位；如果目标操作数比要减去的值小，则CF被置位。

**4.算数运算**

乘法和除法都需要预先规定存储被乘数、被除数和结果的寄存器

mul指令的格式是mul value;

div指令的格式是div value。

mul或div指令要操作的寄存器一般会在之前许多条指令的地方被赋值，因此可能需要在程序上下文中来寻找。

**5.逻辑运算符**

OR、AND和XOR可以对源操作数和目的操作数做相应的逻辑操作，并将结果保存在目的操作数中。

shr和shl指令用于对寄存器做位移操作。

循环位移指令ror和rol与移位指令类似，但移出的那一位会被填到另一端空出来的位上。

**6.NOP指令**

最后一个简单的指令nop什么事情都不做。

0x90

当它出现时，直接执行下一条指令。

nop指令实际上是xchg eax, eax的一个伪名字。不过EAX与它自身的交换等于什么事情都不做，所以直接就说NOP（无运算）了。

**5.5知识点：栈操作**

**1.栈操作**

栈是一种后入先出的数据存储结构

函数的参数、局部变量、流控制结构等被存储在栈中

ESP (Extended Stack Pointer) 存储栈顶的内存地址

EBP (Extended Base Pointer) 存储栈底的内存地址

PUSH指令将数据压入栈顶

POP指令从栈顶取出数据

**2.函数调用的实现流程**

1.使用push指令将参数压入栈中。

2.使用call memory\_location 来调用函数。此时，当前指令地址被压入栈中。这个地址会在函数结束后，被用于返回到主代码。当函数开始执行时，EIP的值被设为memory\_ location 。

3.通过函数的序言部分，分配栈中用于局部变量的空间，EBP也被压入栈中。

4.函数开始它的工作。

5.通过函数的结语部分，恢复栈。

6.函数通过ret指令返回。

7.调整栈，以移除此前压入的参数，除非它们在后面还要被使用。

**3.条件指令**

最常见的两个条件判断指令是test和cmp。

test指令和and指令的功能一样，但是并不会修改其使用的操作数，只设置标志位。

cmp指令与sub指令的功能一样，但它不影响其操作数，只用于设置标志位。

**4.分支指令**

分支指令可以控制CPU指令的执行顺序，类似于C语言的if语句。

**5.重复操作指令**

重复指令是一组操作数据缓冲区的指令。数据缓冲区通常是一个字节数组的形式，也可以是单字或者汉字。

在x86下，使用重复的前缀来做多字节操作。rep指令会增加ESI和EDI这两个偏移，减少ECX寄存器。rep前缀会不断重复,直至ECX=0。repe/repz和repne/repnz前缀则不断重复，直至ECX=0或直至ZF=1或0。