网络信息安全白皮书

（第一版）

石望湘

2005-09-23

【目　录】

危害网络信息安全的军事、商业信息失窃事件可能发生在任何时间，窃贼也可能来自于世界任何一个角落，因此，对于一个网络规划而言，没有信息安全规划的网络决不是一个值得信赖的网络。

本白皮书摒弃了传统学院派照本宣科式的空洞理论说教，而以娓娓的笔调从实际运用的角度提出网络信息安全规划应该考虑到的若干安全因素以及对应的解决方案，因此非常适合于作为网络信息安全从业人员和学院派讲师的必备参考教材。

【网络信息安全白皮书】
（第一版）

前言

随着网络技术的日新月异和网络应用的逐步普及，随着电子政务、电子商务、办公自动化等各类信息系统的深入应用，危害网络信息安全的攻击入侵案例日益增多，并且手段也日益复杂多样，这一方面导致了各级政府、军事以及科研、商业机构大量内部敏感数据失窃，另一方面也给我们的信息安全建设敲响了警钟。

危害网络信息安全的攻击入侵可能发生在任何时间，也可能来自于世界任何一个角落。毫不夸张地说，只要存在网络，网络信息安全问题就不可避免的存在。为了及时应对和解决这些突发的信息安全事件，网络信息安全作为一个新兴的课题日益受到重视。

不难想象，没有信息安全规划的方案决不是一个值得信赖的方案，也不难预计，缺乏安全规划的方案也绝对不能成为一个合格的解决方案。

有鉴于此，特制定此《网络信息安全白皮书》，方便每一位网络信息安全从业人员在进行网络信息安全规划时作为辅助参考。

第一章　危害信息安全的典型手段

通过分析典型网络信息安全入侵案例，我们知道，入侵和攻击的产生，客观上固然与系统平台安全漏洞的不断发现和黑客软件的泛滥有关，主观上也很大程度归结于简陋的信息安全规划以及普通用户甚至网络管理人员自身安全意识的薄弱和网络安全知识的贫乏。

入侵事件并不总是发生在军事、政府、大公司财团等大型科研商业机构或者金融行业，更多的数据安全失窃事件往往来源于疏于防范的中小型机构，因为那些重要行业往往非常重视安全问题，并且部署了更高安全级别的防火墙和更具专业水准的安全策略，而普通中小型机构即算没有国家机密，但是对

于同行业竞争者或者别有用心的人来说，往往也存在一系列值得窃取的重要敏感数据。更让人忧虑的是，这些缺乏足够防范的计算机还可能作为黑客扩大恶意入侵战果的跳板，而直接危害到其所能访问到的任何资源，最典型的案例是如果某集团公司 CEO 的计算机一旦被入侵，损失的将不仅仅是他个人的信息，其所管理的整个集团公司及其所属子机构的所有资源都可能遭到非法窃取。

通常，入侵事件往往首先从以下方面寻找突破：

- 通过物理接触直接获取资源。
- 利用自己（或者窃取）的合法账号越权操作。
- 寻找系统弱口令甚至空口令。
- 伪造受信任的 IP（MAC） 地址。
- 不恰当的防火墙配置以及糟糕的安全策略。
- 扫描目标机器并得到开放端口、服务商和版本以及漏洞列表等信息。
- 利用系统或者相关软件漏洞远程侵入并提升权限。
- 病毒木马以及其它貌似合法的后门程序植入。
- 基于关键路由的混杂模式敏感数据包窃听甚至隔墙的电磁辐射窃听。
- 收集目标机器必要信息(例如所属机构、管理人员姓名、联系方式以及个人喜好习惯等)，作为暴力破解的基础参照依据。
- 社会工程学诡计。
- 其它不可预知的新兴技术手段。

如果以上突破点有一个或者更多被满足，那么入侵事件就会紧接着发生！并且入侵企图更会得寸进尺和变本加厉。

因此，保证网络信息安全就必须满足以下前提：

- 严密的总体安全规划。
- 足以防范所有已知的入侵手段。
- 主观上足够的信息安全意识。

以下将针对这些手段分别加以描述信息安全对策和解决方案。

第二章　信息安全策略

信息安全的核心是预防，对于一个具体的应用而言，保证信息系统安全的最基本依据是首先具有一个设计周密的安全规划并且通过安全专家组的评审，这个规划应该在网络硬件结构和部署中设计必要的

硬件级别保护措施，以此作为信息安全的第一道物理防线，从物理访问逻辑上尽可能缩小可能遭受的被攻击面，阻止非法来源的访问路由。

其次，还需要基于足够强悍的安全策略和足够丰富的网络安全技术，从各个层面的软件技术实现上，检查和过滤大多数可能发生的入侵探测行为。

最后，应该具备一个尽可能完善的网络安全紧急处理机制和一个完整的备份及恢复策略。

下面，我们从各个不同的层面阐述信息安全对策。

您的网管值得信任吗?

假如您是公司老板，尤其您是一个信息产业类公司的老板，而恰恰您又很钟爱无纸化办公以及关注您的源码安全和知识产权保护，那么您会选择一个怎样的网络管理员呢？

至少而言，一名合格的网络管理员至少需要具备足够的网络信息安全意识并具备相当的网络安全从业水平，否则有何价值呢？除此之外，还有什么需要更加特别关心的吗？

您是否考虑过，网络管理员只要存在任何不良私欲或被竞争对手收买，就有足够的时间和权限来收集公司所有的资料（包括任何机密数据和核心源代码），同时也有足够的条件和理由来监听公司所有的内外通讯，而监听内容甚至包含了您的电子批文和个人私密邮件，且不说您的电子批文可能被恶意篡改和伪造，单单这些敏感信息的泄漏是否就已经构成了一种非常严重的信息安全威胁呢？

假如您聘用的网管是您的心腹之人倒也罢了，假如不是，那么赤裸裸的您还有什么信息是竞争对手所不知道的呢？进一步思考，您还会有什么核心技术和知识产权而言呢？

所以，我们把这个网管的信任问题列为首要的信息安全策略加以提醒，就是为了首先敲上一记警钟，希望引起您最充分的关注。

网管无耻的背叛或无情的倒戈无疑是在身后给我们狠狠地捅上最致命的一刀。

装机必读

当我们刚刚装好一台机器并且新装了操作系统以后，任何具有一丁点安全意识的人都应该会想到需要通过网络安装操作系统补丁程序和升级杀病毒软件，但是有一个隐患可能让我们还没有来得及修补系统，刚刚接入网络就被立即“黑”了！

这不是好莱坞黑客题材影片虚构的情节，而是真实存在于我们周围的事实。

究其根源，这是微软 Windows 系统 RPC（Remote Procedure Call）服务的系统漏洞所引起的。

“冲击波”以及其它基于相似原理的蠕虫或者黑客程序就是通过远程访问存在漏洞的 RPC 服务的 135 端口进行传染和扩散的，这也是为什么我们刚刚接入网络尚未来得及修改系统漏洞就中招的根源所在。

但是，RPC 服务是 Windows 系统很重要的一个服务，通常不允许关闭此服务，否则很多系统功能将丧失，因此我们只能通过防火墙策略对外屏蔽此服务，以最大限度保证我们新安装系统的安全。

针对此隐患的正确的防火墙策略应该是阻塞所有针对 135 端口的 TCP/UDP 数据包。

此外，默认的系统磁盘根共享和管理共享也是非常危险的隐患之一，正确的防护方法应该至少暂时默认阻塞所有针对 137/138/139/445 端口的 TCP/UDP 数据包。

系统漏洞是不断被发现的，基于系统漏洞的攻击手段也在不断翻新，请对系统安全保持随时的关注，及时添加正确的防火墙策略，以防范和阻塞未来的被攻击端口。

另外，在接入网络以前，请确保系统不存在空连接、空口令等隐患，并请确保系统没有开放任何不必要的共享和服务。

如果能够满足这些安全条件，那么欢迎开始网络接入。

物理安全

物理安全也就是保证重要设备不会丢失、被窃和损坏，并且保持相对的物理隔离，这些设备包括管理控制（监视）台、服务器、关键工作站、网络交换机以及路由器等。

物理隔离的必要性在于防止任何非法或者未经授权的本地操作。

无论防火墙以及系统安全配置多么的完美，都只能有限防范基于网络的攻击，而入侵者只要能够物理接触到目标设备，那么一切都是虚设，非法入侵者可以通过采用异种操作系统引导、合法账号登录、加载特种程序等许多手段盗取目标设备口令文件以及其它任何信息，并且可能修改系统配置或者植入特定的特洛依木马作为将来随意出入的后门。

很经典的 Windows 2000/XP/2003 本地入侵方式就是通过物理接触用引导光盘启动系统，然后修改甚至摧毁目标操作系统的帐号密码，重新引导目标操作系统之后，便可以用指定的新密码甚至不需要密码以 Administrator 帐号登录系统，获得系统的完全控制权。如果精心，还可以事后不露痕迹的清除访问记录和恢复原始帐号密码。

因此，物理隔离是必须的，而最安全的隔离是完全从物理上隔离网段，并且将所有服务器和工作站分别部署在受监管的、单独的房间。

在一个集中的办公室内，无法隔离各个工作站，这种场合下，为了不给任何其它人非法使用不属于他自己的计算机资源的任何机会，需要随时锁定工作站，这不仅仅是私人信息的保密要求，也是服务器远程身份信任的安全保证前提。

如果不能做到完全的物理隔离，那么至少需要完善以下安全保障措施：

- 划分必要的逻辑子网。
- 交换机和路由器的管理模块采用足够强度的口令保护。

- 通过可管理交换机和路由器将特定 IP、MAC 绑定到特定设备端口，并设置相应的 ACL 访问控制列表进行地址过滤。
- 采用更高安全级别的系统推荐分区格式，例如 Windows 采用 NTFS 分区。
- 禁止其它源于光盘、软盘以及其它分区的引导。
- 加载必要的系统设置口令和引导口令。
- 禁止除特定账号或系统管理员以外的任何合法账号的本地登录。
- 系统管理员密码足够复杂并且保持经常性的变更。
- 即使短暂离开，也要锁定设备。

物理接触情况下的数据安全

为了保证机要敏感数据的安全保密，所有服务器都隔离在单独受监管的房间，除了采取相应手段阻止基于网络的非法访问以外，服务器钥匙以及开机密码分别由不同职责人员保管，保证每一次软件安装部署和启动、配置变更、数据更新维护都能够在可监管的前提下进行。

在软件部署时，需要软件开发商（或委托商）、信息中心安全监督人员以及软件使用者三方共同实施，其余需要物理接触服务器的操作至少需要安全监督员以及操作者两方共同完成。

账号安全

首先请记住：再强悍的安全系统都会葬送在脆弱的账号和密码手里。

事实证明，很多的内部敏感资源泄漏案例都来自于内部合法账号的越权使用、恶意攻击或者弱口令高权限账号的被非法破解，因此我们需要完善账号安全管理以及权限分配。

账号管理原则

对于系统账号管理，应该基于以下原则：

- 所有验证系统必须加载口令保护或者其它硬件认证加密。
- 禁止所有来宾账号、可疑账号以及任何不需要或者已经过期的账号。
- 所有口令不能为空或者与用户名相同、相近和追加简单数字组合，并且也最好不要以生日、电话号码以及易于辨别和猜测的单词作为口令。理想的口令组合要求具有 8 位以上长度，并且同时包含字母、数字以及特殊符号。
- 所有口令要求经常保持变更以增大可能的、被暴力破解的难度。
- 严格限制每个账号的访问权限。

系统管理员/超级用户口令必须足够复杂和保持变更，切忌无口令或者弱口令，并且不允许将口令记录在任何未受严格保护的媒介。

Unix/Linux 账号安全

Linux 作为类 Unix 操作系统，几乎沿用了 Unix 的账号文件结构，即所有用户账号均默认存放在 /etc/passwd 文件中，如果采用了口令加密机制，那么将同时保留一份以密文形式存在的密码在 /etc/shadow 文件中。

Unix 口令验证的机制是系统将用户输入的口令经过一定的算法加密后与密码文件比较，如果符合则验证通过，因此密码文件的妥善存储是至关重要的。

目前有许多黑客软件可以暴力破解通过各种手段获取的 Unix 口令文件列表，所以我们必须保证口令文件不能被所有普通用户读取，也就是说我们需要用 Chmod 命令取消所有普通用户对此相关文件的所有访问权限。

由于 Unix 系统的口令认证是在系统超级权限下执行的，因此出于安全的考虑，如果没有其它守护进程需要使用到口令机制，那么我们完全可以 Chmod 000 取消这两个文件的所有访问权限。

Unix 采用基于用户的文件访问权限控制，因此对于系统内的每一个资源，都必须规划和综合使用 Chown 和 Chmod 命令指定哪些用户可以访问哪些资源，而哪些属于禁止访问。

Windows 账号安全

Windows 操作系统由于使用非常普及，因此发现的安全问题也特别多。

从账号安全的角度来说，Windows 9x 几乎没有安全可言，任何人都可以通过简单的删除口令文件来登录系统，只要保存了登录口令，还可以很轻易通过星号密码察看工具或者 PWL 密码文件察看工具获得密码原文，最糟糕的是其基于 FAT 的分区结构任何操作系统都能轻易读写（包括最原始的 DOS），并且可以绕开任何的系统口令验证。

Windows NT/2000/XP/2003 采用了更高安全级别的 NTFS 分区格式，不会轻易的被其它操作系统直接存取（请注意，Windows Preinstallation Environment 和 Winternals 的第三方超级工具包可以读写，这种情况下物理隔离和随时锁定才足够安全），但是 Windows NT/2000/XP/2003 作为可以远程管理的、基于用户身份和口令验证的操作系统，由于空连接的存在以及默认共享的自动开启，Windows NT/2000/XP/2003 的账号安全受到威胁。

如果采用系统默认的安装配置，那么非常不幸，我们可以利用“中华经典网络军刀”的“探测 NetBios 信息”功能通过 IPC 进程间通讯以用户名和密码均为空的匿名连接方式轻易刺探到远程 Windows NT/2000/XP/2003 的用户账号列表以及相关登录信息。

同时，Windows NT/2000/XP/2003 默认开启了 C$、D$、Admin$ 等具有完全控制权限的系统共享，即使取消，每次重新启动以后还会自动开启，这本来是为了方便系统管理员远程维护使用的，但是众所

周知，系统管理员身份的识别是依靠口令的强度保护的，我们完全有理由怀疑系统管理员口令被网络非法窃听或者被暴力破解而失密，因此系统管理员的远程维护可以定义为一个危险的操作。

为了预防危险的非法超级权限使用，同时也为了尽可能预防系统管理员工作的失误或者权力滥用而导致的系统问题，建议取消系统自动共享或者每次启动以后执行手动取消。

中华经典网络军刀(仅 20 余 K 的精巧网络信息刺探和调试工具)提供方便快捷的关闭方式，下载地址和完全使用手册在：http://www.ccview.net/software/NB.htm

除此之外，Windows NT/2000/XP/2003 的普通共享也是建立在用户信任的基础上，因此在保证每个共享资源用户权限分配的同时，还需要加强对各个用户账号的口令安全监控，以防止用户账号被冒名盗用而导致的资源泄漏和非法窃取。

特别警示：绝对不要在他人的机器上执行任何包含自己密码的操作，因为键盘事件、鼠标事件以及剪贴板事件都可能被暗中记录，同时通讯数据也可能被本地截获。

另外，对于那些确实有必要开放完全控制权限的共享文件夹，也需要加强监管，以防止病毒的传播扩散和由于泛滥而导致的网络带宽堵塞。

用户身份认证

信息安全很大程度上依赖于登录用户身份认证的可靠程度。如果一个貌似受信任用户的连接来自于一个非法的未经授权入侵者，那么数据失窃事件就会发生。

因此，对于用户身份认证，可以通过如下手段强化管理：

所有网络工作站都设置足够强度的密码，杜绝空口令和弱口令，本机登录时除了要求核对启用密码以外，还可以通过智能 IC 卡、USB 指纹盘等附加硬件加密设备加强认证。如果操作员短暂离开，也需要锁定电脑，锁定办法包括但不仅限于系统热键锁定、USB 指纹设备锁定等。

服务器关闭所有不需要的服务，并且采用系统内置的权限策略以及软件防火墙过滤所有非法的访问，以辅助实现用户身份的甄别和存取权限分配。

同时，所有认证的敏感数据通讯采用加密传输，减小被窃听的可能。

特别值得注意的是，我们可以考虑采用数字签名确定身份，但是却不能迷信这种身份甄别方式，毕竟所有的电子数据都可能是并且从技术的角度而言也的确是始终可以伪造的。

正确配置防火墙以及安全策略

为了保证网络信息安全，建议在配置适当子网基础上，再部署硬件防火墙，并且同时在需要受保护的服务器内部安装软件防火墙，此外，网络节点的所有工作站都需要部署杀病毒软件以及特洛依木马

检测系统。

软硬防火墙兼备的目的是尽可能保证服务器数据的存取安全。

硬件防火墙主要用于阻挡绝大部分的已知攻击企图和流量拦截，但是由于其可能被穿透，所以需要软件防火墙作为后备安全策略，软件防火墙主要用于过滤来自硬件防火墙后端的内部入侵企图以及防范一些可能从硬件防火墙漏网的关键安全策略。

应该指出，部署了防火墙并不意味着可以从此高枕无忧，随着网络技术的发展以及系统最新安全漏洞的公布，防火墙安全策略需要随时进行维护和添加，同时，有了周密的防火墙策略也仍然不能说明信息系统已经安全，只有安全意识真正普及，全员重视安全问题，才能将安全策略真正落到实处，信息系统的安全才真正用了实质意义上的保障。

安装防火墙并启用安全策略的目的是要阻止任何未经授权的访问，因此安全策略的合理制定至关重要。

制定防火墙安全策略通常都不外乎采用以下两种截然不同的原则：

- 保守性原则：即首先假设所有的服务都可能是有害的，因此该原则的操作方式是首先关闭所有的服务，然后再根据需要逐一打开相应的服务。
- 乐观性原则：即首先假设所有的服务都是必须和有安全保障的，因此该原则的操作方式是开放所有的服务，然后再根据需要逐一关闭危险的服务。

以上两种原则的最终效果是绝对不同的，而且安全系数也大相径庭。

严格说来，乐观性原则虽然易于操作，但是绝对是不够安全的，原因在于虽然可以预先关闭已知的危险系统服务例如 Telnet 和 FTP，但是却不能阻止非法植入的基于非标准端口的危险服务，例如将 Telnet 和 FTP 端口分别开在 1023 和 1024，防火墙将如何防范？

基于同样的理由，乐观性原则也不能阻止任何未知或者安全人员不熟悉的特洛依木马的监听端口，要保证安全，网络管理人员必须随时监控系统服务以发现新的不明端口，所以站着纯安全的角度，对于防火墙策略的设置建议采用保守性原则。

不过，保守性原则也存在一个普及和应用的难度问题，那就是对于网络安全人员的素质要求相对更高，因为安全人员必须熟悉所有可能的系统服务的运作机制和监听交互方式。

对于实际运用来说，将根据操作系统和安全级别要求的不同而灵活处理。

千万不能迷信防火墙，绝对的安全只有拔掉网卡并物理隔离。

防火墙不是绝对安全的，在以下情况下防火墙将被穿透：

- 精心伪造的非常规 ICMP、IGMP 载体包。

- 会话劫持和 IP、MAC 地址欺骗。
- 遭遇拒绝服务（Denial of Service，简称 DoS）攻击并试图恢复正常工作的某个瞬间。
- 不严密的安全策略。
- 不可预知的基于合法端口的不合法访问请求。

另外，安装了防火墙也不能高枕无忧，因为防火墙根本无法有效阻止以下攻击企图：

- 携带 ARP 路由欺骗的伪造 IP、MAC 地址信息包。
- 拒绝服务攻击。
- 各种来自防火墙内部受信任局域子网的攻击。
- 拨号进入而绕开了防火墙网关安全机制的网络连接入侵。

特别警示：绝对不能让软件防火墙策略盲目信任本地地址如 127.0.0.1，因为存在被本地植入了包转发工具的可能(详见中华经典网络军刀完全使用手册)。

关闭所有不需要的服务

入侵企图的得逞往往归功于系统漏洞的发现和重要配置的失误，这一切的挖掘又全归功于入侵扫描软件的反馈信息。

通常说来，任何一款入侵扫描软件均能迅速发现目标机器上打开的所有服务或者可能带来安全问题的端口，一旦获得这个服务的端口，就可以有的放矢地根据相关信息进行刺探，如果确实存在问题，则最终可以入侵成功，即使具有密码保护，也可以被远程暴力破解。

因此，请遵照网络安全界的一条至理名言，那就是“关闭所有不需要的服务”，服务越少，可能带来的隐患也越少，同时系统性能也会得到提升。

关注局域网安全

绝大多数数据失窃等安全事件的发生并非来源于外部黑客的侵入，而主要是来自于局域网内部，这个内部不仅仅包含对公司心存不满或者具有其它不良意图的内部员工，同时更有可能是那些借机来公司并借口需要上网而接入内部网络的、带有信息情报搜集和核心技术窃取意图的、由竞争对手精心安排并有备而来的外部访客，这些所谓的访客通常都会以最快的速度首先扫描出网络拓扑结构，然后分析排查出关键节点，并进行重点端口的空口令尝试、弱口令猜测以及颇有胜算的暴力破解，进而获取极具价值的商业情报，因此除了指定专人进行重点陪同（其实是为了监管）之外，还需要充分认识到局域网安全绝对是一个不容忽视的安全防范环节。

如果局域网服务器部署了软硬件防火墙，那么这个防火墙同样应该部署很周密的安全策略。

如果公司内部存在哪怕短期更不用说长期的外来人员驻留办公，那么单独隔离其网段并且主动从

路由严格过滤其网络访问权限是十分必要的，因为如果不是这样，局域网内部总会有一些弱智的用户机器会给他们以可乘之机进而作为跳板侵害到整个局域网的信息安全。

如果局域网必须开放数据共享，那么共享权限和安全属性的设置应该是绝对恰当和毫不多余的，例如：

- 文件共享必须严格按文件级别分门别类建立文件夹并设置相关文件夹的安全属性，杜绝非法访问。
- VSS（Visual SourceSafe）源码管理如果采用文件共享渠道，其项目文件夹应该首先设置安全属性，保证仅项目组成员才有访问权限，其次必须强制要求项目组成员采用强密码，同时建议最好不要在 VSS 客户端图一时方便而直接 View 相关文件，以避免在 VSS 服务端的 Temp 文件夹中遗留 View 文件副本而导致文件泄漏。附带提一句，如果 VSS 基于 HTTP 及其它通道访问，则相关配套软件如 SOS（SourceOffSite）必须支持并设置为加密传输，否则安全性更加令人不快。

关注安全漏洞公告

任何操作系统和应用软件都有可能存在安全漏洞或者可能导致负面影响扩大的功能缺陷，对于这些问题，我们唯一所能做的就是关注相关厂商以及互联网上的各类安全漏洞公告，并根据自身的具体情况通过各个应用提供商的官方下载站点（非官方下载可能包含基于各种企图的恶性窜改）下载安装相应的安全更新补丁对系统进行升级。

很具讽刺意义的是，这些官方更新站点一定是安全的吗？曾经发生过的安全事件提醒我们，我们仍然需要警惕被劫持了的或者被精心放置了恶意代码的更新站点，虽然这是官方的站点！

无论如何，安全漏洞却总是需要及时弥补的，例如 Windows 操作系统下的基于 IIS 的 Unicode 漏洞，可以让所有远程访问者通过简单的几条命令数分钟以内获得整个计算机乃至整个物理网络内的完全控制权，而解决这个漏洞的办法也很简单，不需要任何技术支撑，仅仅需要的就是轻点鼠标，下载安装微软的最新安全补丁。

安装病毒、木马防御系统

病毒木马的出现是十几年前开始的事情，当时几乎都需要高深的汇编语言功底才能编写。

近年来，随着 VBS 等脚本程序以及 Javascript 有害代码的源文件随处可读，导致病毒木马越来越依靠这些低门槛的编程手段和系统漏洞的简单捆绑泛滥成灾，加之互联网的日益普及，病毒木马通过 Email 传播的速度令人咋舌，而我们的疏于防范更是助纣为虐，因此病毒木马防御系统的安装是绝对需要的。

同时我们也应该看到，网络上的病毒木马如同 SARS 生物病毒，是不断变异和被改造的，并且每

天也有数种甚至数十种新病毒问世，因此为了保证我们的病毒木马监测数据库保持最新，我们必须随时关注这些软件的最新升级更新公告以获取最新的检测数据库，并获得最新的病毒木马防御能力。

过时的病毒木马防御系统比没有安装任何防御系统更加令人担忧，因为这只会令人麻痹大意甚至彻底松懈。

另外需要指出的是，很多病毒木马程序就潜伏在一些非正规的个人网站或者所谓的黑客站点软件之中，更多的也许还存在于一些貌似合法的邮件附件中，当没有经验的用户轻易相信并点击之后，噩梦将从此开始。

软件审查

为了尽可能预防由于应用软件的缺陷或者被有意无意间嵌入的恶意代码对整个网络的信息安全造成威胁，网络上安装的所有软件（尤其是服务器软件）都有必要通过安全审查，审查手段包括操作日志登记、软件来源追溯、病毒木马检测以及更高安全级别的源代码审查等。

请特别留意我们自己内部的程序员编写的软件和小程序，经验欠缺和时间仓促可能带来客观上的不足，但是更令人愤慨的是，某些程序员出于种种目的，刻意在某些隐蔽的角落预留了一些极不光彩的后门、逻辑锁、逻辑炸弹甚至是更为恶劣的有害代码。

防止窃听

窃听是指利用网卡或者其它数据探头进行的基于网络混杂模式的非法信息包截获分析，通常发生在数据传输必须流经的网关或者关键路由上，也普遍运用于广播发送数据交换方式的中央集线器（俗称 Hub）网段内，随着 ARP 路由欺骗技术的出现，端到端通讯的 Switch 交换机也无法逃脱遭窃听的厄运，因此只要接入网络，通讯就存在被窃听的危险。

还有一种窃听方式是绝大多数人不愿意相信的，这就是隔墙有耳式的探针甚至电磁辐射窃听，通过特种的电子装备，我们可以进行通讯线路电磁截听，也可以在一定距离之外直接探听到正在工作中的计算机设备发出的所有电磁讯息，并且可以对其还原再现，这也就是高度机密部门为什么需要采取电磁屏蔽的原因所在。

光缆没有电磁辐射，通常我们会认为是无法无损窃听的，但是非常不幸，光缆存在长度限制，超过一定长度就必须进行信号放大，然后再传入另一条光缆，而这个信号放大的过程就是一个从光信号转换成电信号，再从电信号还原到光信号的过程，电信号难道不是可以窃听的吗？

在无线通讯网络不断普及的今天，无线通讯电波（含红外通讯）的非法截收也是一个值得特别关注的安全隐患。

对于我们的日常通讯而言，非常不幸的现实是，Telnet、FTP、Email 等通讯工具按照相关 RFC 协

议标准均通过明文传输密码，最普遍和最令人不安的是，我们日常通过网页提交的用户名和认证口令也总是通过明文传输，这些敏感信息完全有可能在途中遭到非法窃听而失密，因此，千万不能通过任何不受信任的网络、网关或者代理服务器传输这些数据。

对于合法网关节点的使用，我们也要遵循一个尽量减少密码传输次数的原则，举例来说，由于 Telnet 是一个非常危险的操作，为了防止密码失窃，我们最好一次登录服务器，一直保持连接直到确实不再需要才退出（当然我们必须保证我们的登录工作站在此期间不会遭到劫持）。

通过减少密码明文的传输次数可以极大降低密码被窃取的可能性。

如果可能，对于这些明文协议，强烈建议采用诸如 SSL 或者 VPN 等加密通讯的传输方式，而且每次通讯均应该采用加、解密双方动态协商约定的可变密钥进行加解密（简单演示请参照中华经典网络军刀的完全使用手册）。

另外，传输敏感数据时依据特定规则填充大量垃圾数据也是一个减轻窃听风险而采取的不得已而为之的、不是办法的办法。

警惕社会工程学诡计

社会工程学（Act of Social Engineering）是一门诱骗他人达到自己意愿的艺术，可以帮助企图入侵者侦测到足够多的信息或者入侵机会。

典型的诡计包括但不仅限于：

- 冒充主管领导或者上级机关以及其它调查机构，索取用户资料，或者通过各种联系方式套用到用户个人的有用信息，以此作为密码暴力破解的字典依据。
- 冒充设备或服务提供商工作维护人员，远程或者直接上门骗取敏感信息。
- 通过发送声称很有趣的软件或者其它欺骗性程序诱导潜在受害对象主动植入后门程序以达到完全控制的目的。
- 伪造某受信任的知名公司电子邮件，以种种名义进行商务诈骗。这是非常值得警惕的，因为伪造一封电子邮件实在太容易，而且即算是编写一个伪造邮件软件，也非难事。
- 收集整理垃圾箱里的任何有潜在商业价值的信息。这是一个最容易忽略的严重问题，因为很少有员工会关注扔往垃圾箱的纸张或者报废软盘会导致信息泄漏，而这却千真万确的存在，所以办公室的任何废弃资源都不能直接扔掉，应该采用碎纸机和其它粉碎手段处理信息垃圾。

综上所述，不要轻易相信任何人的身份，不能轻易透露任何信息，也尽量不要在互联网上或者公开场合泄漏自己的任何个人资料。对于常见的用户注册资料、在线调查等的填写也尽可能不要提供真实

信息，尤其不能填入公司规模以及常用操作系统等信息。

网络安全紧急处理机制

考虑过一旦发生网络安全侵害事件，我们应该如何进行紧急处理吗？

首先，有特定部门有专人负责处理吗？能够第一时间进行联系并能第一现场开始处理吗？

是拔掉网线停止服务还是在线处理呢？

是重装机器呢还是选择就近恢复？

是求助于专业安全公司呢还是自行处理？

如果自行处理并且在线处理，能够制伏对手而不被对手玩弄甚至被一气之下格式化硬盘吗？

如果自行处理，能够找到入侵突破口以及漏洞根源并实施修补吗？

如果自行处理，能够提取足够的证据来支持起诉或者举报吗？

如果自行处理，能够圆满解除威胁并清理现场吗？

考虑过以上问题吗？如果没有，到了该考虑的时候了，越早越好，趁着入侵伤害还没有发生。

系统备份及恢复

无论多么完善的信息安全防范，都无法完全避免由于自然灾害或者人为因素造成系统软硬件损坏而导致的数据损失，因此系统备份是最后的防线。

系统及时备份可以在进行系统恢复时最大限度地缩短系统故障时间，并最大程度减少由于系统故障或者入侵破坏而导致的各种损失。

不可否认，结合 RAID5（Redundant Array of Inexpensive/Independent Disks Level5）冗余磁盘阵列可以获得更高的系统容错性能，虽然这种容错也具有在线备份的作用，但是容错对象却仅能限于磁盘阵列本身的、在允许范围内的单个磁盘损坏，而根本无法保证由于数据被恶意篡改或者入侵破坏导致系统故障的数据恢复。因此，系统备份是需要稍微额外付出并能在关键时候获得丰厚回报的工作。

系统备份一般可以考虑在定期进行系统完全备份的前提下进行一定时间间隔的增量或者特定数据备份并且这些备份数据应该存放在具有安全保障的异地，可以采用安装完整的备份服务器或者将数据分布式备份到不同备份机的策略，或者可以考虑分类刻录成若干 CDROM/DVD，这些备份方式均可以很方便的根据需要随时进行。

系统备份不仅仅是系统数据的物理备份，同时也是网络安全策略组合中的最后一个灾难恢复策略。

提高信息安全意识

无论多么完美的信息安全规划设计，如果没有从上至下的层层信息安全意识，那么一切都将是纸上

谈兵而毫无任何意义，因此，普及信息安全教育、培养敏锐的信息安全意识是信息安全规划能否落到实处、能否真正发挥应有防护作用的最基本素质前提。

由于信息安全意识决定着信息安全部署的成败，因此应该提供针对特定人员的网络信息安全相关知识以及数据备份策略的培训，并且辅助撰写相关信息安全管理条例，保证受培训人员足以胜任日后的信息安全维护工作，确保信息安全工作的深入开展。

第三章　服务器安全指南

服务器作为提供公共服务的角色而存在，这种服务可能仅部署在 Intranet 企业内部网，也可能面向整个 Internet，但是无论如何，服务器总是需要开放某些服务，并且总是需要面向一定的目标用户，尤其当我们的服务基于移动办公或者面向世界每一个角落的时候，谁能担保访问我们服务的用户一定没有恶意吗？谁能担保看上去合法的用户一定就是合法的吗？如果无法作出肯定的答复，那么就需要关注服务器的安全了。

本篇虽然主要面向公共发布服务器，但是也同样适合于所有采用任何服务器类操作系统的任何客户端计算机。

服务器安全部署

通常而言，如果服务器提供对外服务，那么这台服务器必须与内部网络进行逻辑隔离，甚至最好是物理的隔离，只有这样才能避免由于服务器不经意间的陷落而沦为跳板，进而对内部网络构成安全威胁。

出于安全考虑，在这台服务器和公网之间应该安装硬件防火墙，并精心规划和配置恰到好处的安全策略，以尽可能从外围阻止非法的入侵探测企图。

如果这台服务器必须与内部网络相连，那么这台服务器和内网之间还应该安装一台硬件防火墙，并精心规划和配置恰到好处的安全策略，以尽可能减小由于服务器沦陷而可能给内网带来的安全风险。

服务器安全安装

很显然，服务器首先应该部署在一个物理安全的环境。

然后，服务器操作系统和相关服务程序必须是来源合法并且经过安全验证的，市面上的盗版软件光盘往往可能正是病毒木马的发源地，因此服务器的安装建议采用正版软件，以免成为他人的猎物。

在系统安装的全过程，除非必要，应该是从网络断开的，以免在安装过程中由于原始系统未经修补的安全脆弱性而遭受不必要的攻击。

对于磁盘分区格式的规划，FAT32 及以下格式是绝对不可取的，因为这种分区格式毫无任何安全

性可言，建议至少采用 NTFS 或更高安全级别的格式，只有这些高安全级别格式才能自定义磁盘访问的安全策略。

对于系统服务而言，应该仅安装仅需要的服务。例如我们仅需要提供 HTTP 网页服务，那么在 IIS 安装过程中默认附带安装的 SMTP 和 FTP 服务就是多余的并且可能是有害的。

如果系统要求初始化一个密码，一定要在第一次就设置一个 8 位以上的、符合系统口令复杂度要求的口令，而不能贪图安装方便而将密码置空或者很弱，因为给自己方便的同时也给不怀好意的第三者伺机从事不怀好意的入侵行为提供了便利。

如果没有资金限制，尽可能保证每台服务器仅提供一个服务，这便于为每个服务器及相关服务设置最严密的安全策略，同时也可以最大限度避免“一篮子鸡蛋全部打碎”的全面沦陷甚至是全盘崩溃。

服务器安全配置

当服务器安装完成并且相关服务正常启动以后，并不能急于将服务器立即对外发布，而应该首先密切关注厂商的默认配置，这些默认配置可能是最易于部署使用的，但是却极有可能是最不安全的，因此还有很多建议的工作要做，这些工作按先后顺序分别为：

指派防火墙策略

每当一个服务器安装成功，首要的工作应该是安装一个值得信赖的防火墙，一个优秀的、基于本机部署的软件防火墙很多时候甚至比外接的硬件防火墙更为重要和可靠，因为硬件防火墙通常不会仅仅只为这一台服务器提供防护，而这类部署的硬件防火墙根本无法阻挡来自后院的非法访问，另外，即使硬件防火墙仅仅就是为这台服务器提供服务，安装一个基于本机的软件防火墙总是可以互相取长补短，防范于意料之外的种种未然。

一个好的防火墙至少应该提供 IP 地址和端口过滤的功能，否则没有任何意义。

如果无法部署第三方防火墙，那么 Windows 2000 以上系统内置提供的 IPSec 将是最佳的选择。

通常而言，我们总是应该按照如下原则指派防火墙安全策略：

- 阻塞所有基于 TCP/UDP 135 端口的通讯，以避免“冲击波”及其类似变种蠕虫的攻击。
- 阻塞所有 ICMP/IGMP 数据包，以防范非法的数据渗透和被拒绝服务。
- 严格过滤对相关服务的远程访问。例如，假设我们必须开放 3389 终端服务，那么应该仅允许特定 IP 地址进行访问，其余 IP 地址应该是被禁止并且记录在案的，否则自己的桌面可能就会变成别人恣意玩弄的战利品；再如，假设我们必须开放 1433 SQL Server 服务，那么也应该仅允许来自特定地址的访问，如果访问程序部署在提供服务的本机，则只需要信任 127.0.0.1 即可，否则可能遭致暴力破解和远程溢出攻击。

- 在系统管理员指导下谨慎地阻塞所有不熟悉的或者未使用的端口如 443/SSL 等。
- 监测或者阻塞所有未经显式明确允许的端口通讯，以尽早充分察觉可能的入侵企图。

关闭不必要的服务

虽然我们在安装时仅选择安装了我们必须的服务，系统仍然会自动安装一些我们无法事先可选定置的服务，这时我们就有必要手工检查并关闭和禁用这些不需要的并且可能会带来隐患的服务。

这些多余的服务通常包括 Messenger、Remote Registry Service、RunAs Service、TCP/IP NetBIOS Helper Service、Windows Time 等，这些服务有的存在严重安全隐患，有的可能遭致网络骚扰和拒绝服务，因此应该关闭和禁用。

如果没有特殊协同处理要求，还应该关闭和禁用 DTC（Distributed Transaction Coordinator）服务，因为这个服务存在被远程拒绝服务甚至溢出的风险。

除非特殊需要，文件及打印机的共享服务最好是关闭的，尤其应该杜绝可写共享。

当所有的服务调整完毕，请备份服务列表，以备日后随时对比和查验。

操作系统升级

人无完人，因此任何软件都不可能是完美和安全的，操作系统也是一样，因此操作系统需要不断修补和升级，Windows 98 以上操作系统均可以通过简单点击“Windows Update”获得微软官方的免费升级支持。

每个升级补丁安装完成，在修补旧漏洞的同时，可能又引入了新的漏洞，因此需要反复查验和安装升级，直到再没有任何安全补丁可以安装为止。

当然，在进行操作系统升级的同时，也应该升级相关的服务程序，以修补最新发现的安全漏洞。

如果操作系统升级以后，由于种种原因又重新开放了一些不必要的服务，请仔细检查并及时予以关闭和禁用。

请仅在官方网站在线更新或下载升级程序，以免被植入极度危险的非法代码。

安装杀病毒软件

全世界每天都会有数十种计算机病毒或者特洛伊木马程序被制造和散布，而这些有害程序是很难被及时手工检测和发现的，因此需要安装技术专业的、升级勤奋的杀病毒软件。

同时，由于新病毒木马或者变种的出现，将导致病毒库过期而失去防御作用，因此需要随时保持杀病毒软件的病毒资料库为最新，这可以通过随时手工查看或者设置每日定时自动调度来保持更新。

谨慎分配帐号和权限

通常，请记住以下原则：

- 禁止除特定账号或系统管理员以外的任何合法账号的本地登录。
- 最好禁止系统管理员来自网络的访问权限，避免被恶意的远程控制。
- 如果没有特殊需要，不要额外分配帐号。
- 如果没有特殊需要，不要额外分配权限，也不要无故提升权限。
- 如果没有特殊需要，不要开启任何来宾帐号。
- 如果必要，重命名系统管理员帐号，如果愿意，再创建一个貌似的、口令很复杂的管理员帐号但不分配任何权限以便实施诱捕性监控。
- 如果一定要创建帐号，请确保满足口令复杂度要求，并且尽可能分配最小权限。
- 经常检查帐号和权限，防止非法帐号植入以及权限提升，尤其应该警惕那些貌似主管领导或者系统服务的帐号（特别是当这些帐号分配了管理员权限的时候）。

当所有帐号和权限分配完毕，请备份清单，以备日后随时对比和查验。

设置磁盘访问的安全权限

如果我们的磁盘分区采用了 NTFS 格式，那么我们可以设置该 NTFS 格式磁盘的用户访问安全权限，以最大限度保证我们的系统数据不会被非法访问甚至篡改。

默认情况下，系统分区对于所有用户都赋予了完全的控制权限（通过鼠标右键单击 C: 盘属性中的安全可以得到证实），这是没有必要的而且也是不提倡的，正确的做法应该是删除 Everyone 用户，然后仅分配 System/Administrator 具有完全控制权限，其余用户均没有任何访问权。

如果 HTTP/WWW 及其它公众服务安装在了这个具有访问权限控制的磁盘上，那么为了保证这些服务的正常工作，还必须为这些服务的所属账号添加相应的读权限（请慎重开放写权限，因为对系统分区的写入可能极具毁灭性危害），当然，我们强烈建议将公众服务安装在其它分区。

典型的 HTTP/WWW 服务可以作这样的访问权限控制，即仅允许 IUSR_XXX（Internet 来宾帐号）具有“读取和运行 / 列出文件夹目录 / 读取”权限，其余任何用户包括系统管理员和系统均不具备任何权限，如此一来，任何本地用户均无法轻易对该文件夹实施阅读、删除和更改操作（当然这不是绝对的），而不会影响远程访问的用户正常浏览网页。

如果具有更高的对文件 / 文件夹数据加密的需要，请考虑额外的专业加密算法。

系统备份

当所有的安装配置完成，建议采用磁盘或者分区备份工具对系统安装分区进行全面的压缩备份，以保证日后能够在灾难恢复时短短几分钟之内将系统恢复到目前状态。

重要的系统备份数据最好异地保存并且备份数量不仅仅只是一份。

随时进行系统完全备份或者增量备份是一个十分良好的、值得推荐的习惯。

编写安全的服务端运行代码

操作系统和服务系统再安全，也仅仅只能表明服务支撑环境的相对安全，自己开发的、存在安全隐患的服务端代码将会让自己十分脆弱，最终导致远端的致命一击。同时，这些不安全的代码还将造成包括公司商业产品丢失、经济损失、用户信任度降低以及股市震荡等严重后果。

编写安全的代码（Secure Code）是开发高信度软件的前提条件，虽然软件开发始终是一门不完美的科学，所有的软件都存在 Bug，而其中有一些就会成为或轻微或严重的安全隐患，但是我们的目标却始终应该是在基于充分安全设计的基础上进行尽可能安全的编码。

如何编写安全代码是一门专业的学科，在此仅针对个别最重要的问题给出相应安全警示，未涉及的更详细资料请参阅附录提供的《开发必知的、保证代码安全的十个最重要安全警示》以及微软发行的《编写安全的代码》等其它更专业的书籍。不过，本篇阐述的某些知识却是实践的产物，而这些技巧偏偏是官方的资料所学不到的。

编写安全 Web 服务代码

首先，我们需要特别警告的是，Web 服务不像其它诸如 FTP/Telnet/SMTP/POP3 服务可以有 IP 地址很有针对性的小范围开放并且拥有严密的防火墙策略保护，HTTP 服务往往需要对 Internet 完全开放，防火墙也必须完全信任来自任何地址的 Web 服务请求，稍有不慎，该服务在安全体系中将最容易首先遭致灾难性撕裂，从而成为更严重和更大范围入侵事件的突破口和不幸而无辜的跳板，因此，该服务也最需要得到最高关注。

经验表明，Web 服务本身就存在最多的漏洞，但是我们除了经常关注安全公告和更新补丁，再也无能为力，因为这不属于我们的可控因素，只有我们编写的服务端运行代码才是本篇关心的话题。

首先而言，Web 服务的身份验证就是一个头疼的问题。

以 .NET 开发平台为例，Web 应用的身份验证方式大体上可以分为 Forms/Windows/Passport/None 等几类，其中：

- Forms 认证采用 HTTP 明文口令传输，除非运行在 HTTPS 或加密通道环境，否则毫无安全可言。

- Windows 集成散列认证在跨网段、跨防火墙以及跨平台应用中多有不便，难以部署在 Internet 应用。
- 微软的 Passport 服务提供跨站点的整合安全机制，支持用户在一个 Web Session 过程中仅登陆一次，就可以从一个使用 Passport 的网站切换到另一个使用 Passport 的网站而不需要再次登陆。这个听起来很方便，但是我们且不说这个第三方服务仍然令人或多或少心存顾忌，至少用户还是必须在一个 HTML 表单中录入一个用户名和密码提交给 Passport 服务器，而这个又是明文。

如此算来，我们好像就只能破罐子破摔选择 None 不验证了，不过不用担心，我们可以采用 Digest 摘要验证来弥补我们的安全需求。

所谓 Digest 摘要认证，是一个遵循 RFC 2617 协议规范的 IETF（Internet Engineering Task Force）标准认证机制。在这个机制中，客户端浏览器（可能仅 IE 浏览器支持）在把包含用户名和密码的认证凭据发送到 HTTP 服务器之前对数据进行 MD5 散列加密，如果认证算法和后台验证代码设计得足够好，还可以混合高强度种子和时间戳参与 MD5 运算并且指定一个极短的过期时间，如此一来，即算再高明的黑客截获了当次通讯数据并破译了加密算法，也需要随时同步这个 MD5 散列，而这个并非易事，因此 Digest 认证相对我们的日常应用而言加密强度应该是足够的。

如果我们的 Web 服务决定采用 Digest 认证，那么请务必让我们的 Web 服务器允许匿名访问并取消其它验证方法如基本验证和集成 Windows 验证等，因为 Digest 认证是需要靠我们自己进行编码处理的。

如果采用 Digest 认证，服务端 Web.Config 文件 <configuration> 的 <system.web> 节中需要作如下设置：

```
<authentication mode="None" />
<authorization>
    <deny users="?"/>
</authorization>
```

为了获得 Web 服务全面的防护，强烈推荐采用 HttpModule 进行全局统一入口验证，而要获得这种支持，需要在节中再增加如下定义：

```
<httpModules>
    <add type="classname,assemblyname" name="modulename"/>
</httpModules>
```

然后，我们开始创建这个基于 System.Web.IhttpModule 的 HttpModule 模块，并在 AuthenticateRequest 事件中进行全局统一的入口请求身份验证。

如果有必要，我们还可以在 HttpModule 模块中参照防火墙的标准融入各种入侵检测机制，从入口侦测并阻止任何非法的入侵意图，同时，还可以引入 ACL（Access Control List）访问控制列表实现基于 IP 地址、时间段和用户角色及组策略的 URL（Uniform Resource Locator） 资源访问权限控制，而这一切，更可以记录在案。

具体实现以上整体安全策略的 HttpModule 解决方案源码由于篇幅太长太复杂，并不适合在本白皮书中详细讲解，因此一笔带过。

熟练掌握 HttpModule 是编写安全 Web 服务代码所必需的基本技能之一。

总体上而言，我们在编写 Web 服务端代码和提供配置运行环境时，还应该遵守以下原则：

- 防止通过协议头泄露任何敏感的系统配置信息如操作系统型号版本、服务程序名称版本、服务安装物理路径以及缓存架构和缓存策略等（这是环境配置提供人员的首要职责）。
- 尽量避免直接 URL 地址访问，保证所有的访问都通过了身份验证并仅能通过应用程序进行导航，尤其应该精心配置站点访问权限，杜绝后台数据等敏感资料文件被猜中并在授权外被非法直接下载。
- 如果采用 Session/Cookie 保存身份认证信息，应尽可能防范 Session/Cookie 未过期时的非法窃取冒用（中华经典网络军刀可以演示如何进行 Session/Cookie 冒用），应对策略可能需要通过缩短 Session/Cookie 有效生存期并采用更为强悍的身份认证体系来减轻这种风险。
- 严格杜绝在隐藏域字段或者其他潜在的可能被篡改的地方保存敏感数据。很多人喜欢并习惯于在网页中设置隐藏字段，并在其中保存甚至是验证口令、商品折扣之类的敏感信息，这一方面导致了敏感信息被非法偷窥，另一方面也会遭到客户端出于种种攻击目的的、最恶意的肆意篡改，因此应该严格杜绝。
- 严格采用正则表达式过滤来自 Cookie/QueryString/Form/ViewState 的数据以及任何文本控件的 Text 属性数据，并且保证这种过滤总是在客户端和服务端同时得到有效执行，避免遭受跨站脚本攻击以及语句注射等的威胁。
- 警惕提交的参数被篡改。我们无法绝对保证客户端参数不会被伪造和篡改，因此我们也不能给予客户端提交参数过多的信任，除了进行严格过滤和有效性校验之外，我们还不能直接将参数组合为可执行指令如 SQL 语句等。
- 参数提交时尽量采用 POST 而不是 GET 方法，避免信用卡号以及密码等敏感信息在浏览器地址栏可见和在服务器访问日志中的明文显示(SSL 无法掩盖日志中的明文，因为 SSL

仅保证对传输中的数据进行加密）。

- 不能依赖基于 Referer 的认证，因为该 HTTP 协议头内容是可以轻易伪造的。
- 尽量不提供文件或附件上传，如果一定要提供，则从客户端和服务端两方面均严格过滤上传文件的类型，尤其是对于那些绝对危险的、后台解析的文件类型如 ASP/PHP/JSP 以及任何 CGI 程序等。
- 谨慎开放自定义签名和头像等花哨功能，以避免遭致精心构造的本地或者跨站脚本的信息收集甚至攻击。

辅助采用 Web 服务器 IP 和身份限制以及防火墙安全策略是十分明智的选择。

编写安全的 .NET 代码

相对而言，Microsoft Visual Studio .NET 为我们编写安全代码提供了更加完备的支持，基于 CLR（Common Language Runtime，通用语言运行库）的托管代码有助于减少一些常见的安全隐患如缓冲区溢出（buffer overrun）等，但是这些都还仅仅只是保证代码安全的一小部分，更多的安全性依然需要我们自己来保证。

.NET Web 应用可选支持对提交参数进行数据有害性检查（如检查危险的<script>脚本字符），这可以通过在 Web.config 文件中配置下列语句来实现：

```
<pages validateRequest="true" />
```

不过，这对于一个那怕最基本的 Web 应用而言仍然是不够的，我们还需要对提交参数进行严格过滤，过滤的字符原则上应当包含“`'"<>;|&%#\r\n”以及连续两个减号（SQL 注释符号）等等，基于正则表达式的过滤语句如下：

```
System.Text.RegularExpressions.Regex.Replace(srcString, "[`'\"<>;|&%#\r\n]|-{2}", "");
```

典型的应用是，我们可以采取如下方式安全获取用户提交的帐号和密码：

```
Regex.Replace(Request.Form["user"].Trim(), "[`'\"<>;|&%#\r\n]|-{2}", "");
Regex.Replace(Request.Form["password"].Trim(), "[`'\"<>;|&%#\r\n]|-{2}", "");
```

为了避免泄露过多信息，在面向用户的正式版本中，应当禁用调试和跟踪，这在 Aspx 页面中可以通过下面语句实现：

```
<%@ Page language="c#" Trace="False" Debug="False" %>
```

在 Web.config 文件中，可以通过下列语句达到具有项目全局约束性的同样目的：

```
<compilation defaultLanguage="c#" debug="false" />
<trace enabled="false" />
```

此外，进行 .NET 编码还至少应该在以下方面加以高度关注：

- 保证序列化（Serialize）和反序列化（Deserialize）的安全性。数据的序列化可以使攻击更加隐蔽，因此，如果我们对二进制序列化数据进行了反序列化还原，那么还原以后需要进行过滤处理，尤其这些数据可能包含或组合为一些可执行条件的时候。
- .NET 让我们直接操作通讯协议头更加方便，不过如果我们决定自己处理协议头，那么各种畸形数据需要进行有效识别和处理。
- .NET 也给我们直接使用 SOAP 服务提供了方便，不过如果我们决定采用 SOAP 服务，那么各种非法 SOAP 请求需要进行有效识别和处理。
- .NET 同样也为我们编写更加安全的身份认证系统提供了支持，不过如果我们决定自己编写安全认证机制，那么如何保证采用足够随机的随机种子和协商机制显得尤为重要。

SOAP 通讯安全

SOAP（Simple Object Access Protocol）即简单对象访问协议，是一种基于 XML 的、用于在 Web 上交换结构化和类型信息的、简单的轻量协议，目前已广泛应用于跨平台的 Web Service 服务。

然而 SOAP 协议也是基于 XML 的、明文传输的协议，也就是说，所有客户端提交的 SOAP 请求报文（如用户名和密码）以及服务端返回的 SOAP 数据内容都可能遭到轻易地偷窥。

幸好，我们可以选择采用微软的 Web 服务安全套件 WSE（Web Services Enhancements） 来加密 SOAP 报文。当然，我们也可以自主编程，实现恰到好处的、最轻量级的、最能体现程序员成就感的解决方案，自主编程的要点如下：

首先，为了通知服务端项目用我们指定的类进行 SOAP 扩展处理，我们必须在服务端 Web.Config 文件 <configuration> 的 <system.web> 节中增加类似如下设置：

```
<webServices>
    <soapExtensionTypes>
        <add type="soap extension class" priority="1" group="0" />
    </soapExtensionTypes>
</webServices>
```

然后，我们在指定的、继承自 System.Web.Services.Protocols.SoapExtension 类的自定义 SOAP 扩展类中重写 `ChainStream` 以及 `ProcessMessage` 两个关键方法（其余必须实现的次要方法仅需要 Return 一个框架即可），关键代码如下（限于篇幅略去完整源代码）：

```
/// <summary>
/// 保存原始数据流。
/// </summary>
```

```
private Stream oldStream;
/// <summary>
/// 存放用于内存处理的数据流。
/// </summary>
private MemoryStream newStream;
/// <summary>
/// 加密选项枚举(不加密/加密/解密)。
/// </summary>
private enum Encrypt {None, Encode, Decode}

/// <summary>
/// 截获 SOAP 数据流。
/// </summary>
/// <param name="stream">原始数据流</param>
/// <returns>新数据流</returns>
public override Stream ChainStream(Stream stream)
{
    // 保存原始数据流并返回一个新的内存流。
    oldStream = stream;
    newStream = new MemoryStream();
    return newStream;
}

/// <summary>
/// SOAP 消息处理方法。
/// </summary>
/// <param name="message">SOAP 消息</param>
public override void ProcessMessage(SoapMessage message)
{
    // 根据不同消息处理阶段进行相应的 SOAP 加解密操作
    switch (message.Stage)
    {
        // 序列化之后的加密处理
        case SoapMessageStage.AfterSerialize:
            newStream.Position = 0;
            Copy(newStream, oldStream,(如果需要加密)?Encrypt.Encode:Encrypt.None);
            break;
        // 反序列化之前的解密处理
        case SoapMessageStage.BeforeDeserialize:
            Copy(oldStream, newStream,(如果需要解密)?Encrypt.Decode:Encrypt.None);
```

```
                newStream.Position = 0;
                break;
            }
        }

        /// <summary>
        /// 流拷贝。
        /// </summary>
        /// <param name="from">拷贝原始流</param>
        /// <param name="to">拷贝目标流</param>
        /// <param name="flag">加密标志</param>
        private void Copy(Stream from, Stream to, Encrypt flag)
        {
            // 创建原始和目标字符流
            TextReader reader = new StreamReader(from);
            TextWriter writer = new StreamWriter(to);
            // 读入原始数据流
            string str = reader.ReadToEnd();
            // 如果指定加密则应用加解密规则
            if (flag!=Encrypt.None)
            {
                // 根据指定的不同加解密算法进行编解码操作
                // 支持 Base64 编解码、自定义种子加密 以及 Rijndael 等任何可逆加密算法
                switch (encryptArithmetic) { …… }
            }
            // 写入目标数据流
            writer.WriteLine(str);
            writer.Flush();
        }
```

至于可逆的加解密或编解码算法选择以及指定哪些关键 Web Serivice 方法需要进行加解密保护则可以在 Web.Config 的 <appSettings> 节进行详细配置，从而实现不必修改源代码即可灵活改变的 SOAP 安全通讯策略机制。

举例进行比较，没有采用任何加密策略的服务端 SOAP 原始响应报文类似如下内容：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 22 Sep 2005 12:42:24 GMT
X-Powered-By: ASP.NET
```

```
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 381

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><soap:Body><GetResultResponse
xmlns="http://www.ccview.net/"><GetResultResult>---&gt;AsD123456789!@&lt;---</GetResultResult></GetResu
ltResponse></soap:Body></soap:Envelope>
```

而采用加密策略的服务端 SOAP 报文则会呈现为如下模样（请注意： XML 的明文描述已被编码）：

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 22 Sep 2005 12:43:56 GMT
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 546

P123aD94bWwgdmVyc2lvbj0iMS4 ……
```

很明显，经过服务端加密的 SOAP 报文内容不再是明文，通讯传输安全性得到了保证，但是客户端也无法解析这个加密报文了，因此，客户端也需要配置该 SOAP 扩展类进行算法匹配的 SOAP 解密操作。

反之，如果客户端采用加密 SOAP 提交，那么服务端也必须配置与之加密算法相匹配的 SOAP 解密才能识别。

XML 安全性

XML（Extensible Markup Language）以其基于文本方式的、严谨的可自描述特点，业已成为跨系统、跨平台数据交换的首选格式，正在日益成为我们软件中不可获取的数据存储和数据交换格式，然而我们不应该忽视的是，XML 安全性也日益成为安全方案中一个很容易忽视的环节，毫不夸张的说，XML 正在引发一个新的安全威胁。

如果我们决定采用 XML 作为数据存储和数据交换源，那么有谁真正关注过 XML 外部引用甚至是 XSLT（Extensible Stylesheet Language Transformations）样式表的安全性呢？

一旦我们决定采用 XML 作为数据源，那么我们应该首先保证其所有的外部引用都是安全的和可

控制的，同时还应该保证以下数据安全性：

- 后台 XML 配置信息或数据文件不能被 URL 直接浏览或下载。
- 关键或敏感的 XML 数据应该是加密存储的。
- XML 数据在读写操作时均应该进行严格过滤，绝对杜绝畸形数据处理，并避免包含可执行指令或者脚本。

其它

除了以上安全代码编写原则，我们还应该注意以下的普遍性问题：

- 严格杜绝代码中存在程序员有意无意遗留的后门或放置的有害代码以及各种调试选项，避免任何形式的非法访问和调试信息泄露。
- 清除代码中的任何特权帐号和特殊权限分配，避免被授权外的非法利用。
- 进行必要的参数检查，充分保证对 Null、零长度数据、错误类型数据处理的高度关注和容错处理。
- 如果处理一个外部输入，需要高效率、低资源分配的尽快完成，尤其是处理无效输入的时候，因此在代码入口一开始就进行过滤和有效性检验总是比分配了一大堆资源和进行了一系列类型申明等初始化操作以后再来判断性能要好，这是避免拒绝服务攻击的有效方法之一，也是一个众所周知的、良好的编码习惯，但是偏偏在现实中很多人却并非总是这样做。
- 用户身份认证通过前不要保存任何数据，因为这极有可能会保存到成千上万的虚假数据而引发系统资源耗尽，并遭致拒绝服务攻击。
- 尽量避免将帐号密码或者关键配置信息以磁盘文件的形式保存，虽然可以采用甚至如 MD5 的不可逆加密方式，但是仍然可能被远程访问和改写，从而成为系统全面入侵的突破口，因此一个好的建议是将这些信息以加密的形式无规律可循的散乱保存在系统注册表中。
- 尽量避免任何接口上的缓冲区溢出，防止远程通过溢出手段创建匿名管道并进而获取系统控制权。
- 尽量避免提供服务器文件上传或者文件写操作功能。如果一定要提供，那么请保证这些操作是可控的。
- 永远不要信任用户输入，针对用户输入的第一条黄金法则是：在证明其无害之前，所有输入数据都是有害的；第二条黄金法则是：当数据跨越不可信任环境和可信任边界时，必须对其进行验证。而可信任环境是指可以完全控制的、明确可信任的数据，应该注意

的是，一个可信任的用户也并不总是可以提交值得信任的数据。

检查用户的输入很少会引起性能问题，即使有可能，也不会比遭受黑客攻击而导致崩溃更为严重。

- 过滤任何可能带来隐患的非法字符如单双引号、HTML 标记、脚本语言标记、注释符号以及各种内部指令等，避免 SQL 或其它语言环境的语句注射。
- 保证关键数据库操作的可追溯和可核查，避免数据损失或将损失降低到最小限度。
- 采用高强度加密敏感数据如连接帐号等，预防在源码泄漏的情况下被轻易窃取敏感情报。
- 加密传输敏感关键数据，必要时还可以采用填充毫无意义的垃圾信息或者夹杂貌似敏感数据的迷惑信息等方式来掩饰真实的关键敏感数据。
- 在不同接口和介质上同时记录安全日志和访问日志，以备随时交叉审计查验。
- 不建议在软件窗口中包含公司联系人及其详细通讯地址或联系方式之类的私有内容，因为这极有可能泄露某种信息并导致社会工程学欺骗。例如，我们假设获知了某人的详细信息，那么我们极有可能猜中其个人邮箱密码，至少经过精心的排练，也可以冒充他的身份进行第三方欺诈。

此外，安全的代码同时也应该是健壮的代码（Robust Code），经常弹出意外错误信息或者导致系统崩溃的软件显然我们不会对其安全性表示信赖。

安全性测试

软件测试的重要性众所周知，普通软件测试人员可能非常熟练于寻找软件的功能性和应用性缺陷，但是软件安全性检测却是很多测试人员所不具备的技能，而这恰恰应该是最需要得到强化的产品检验环节。

没有 100% 安全的代码是一个不争的事实，因此我们需要不断进行安全性测试以发现并修补代码安全漏洞。

安全性测试至少应该包括以下方面：

- 缓冲区溢出测试。
- 授权外的直接 URL 地址访问或下载测试。
- 非法或者伪造参数（Lyin or Cheatin）测试。
- 执行代码注射测试。
- 客户端安全机制绕开测试。
- Session/Cookie 欺骗盗用测试。

- 敏感数据传输保存的保密性测试。
- XML 或其他外部数据源攻击性篡改测试。
- 抗 DoS 能力测试。
- ……

应当指出的是，所有在安全测试中发现的 Bug 在得到修正后，仍然需要重头开始进行重新测试，这一方面是为了验证是否已经满足了安全性测试要求，同时也为了避免因为修改了一个 Bug 而引入了另外一个甚至更多的 Bug。

服务器安全维护

一旦服务器投入运行，将时刻面临来自世界任何角落的网络安全挑战，因此我们必须时刻保持足够的警惕以防止安全事件的发生。

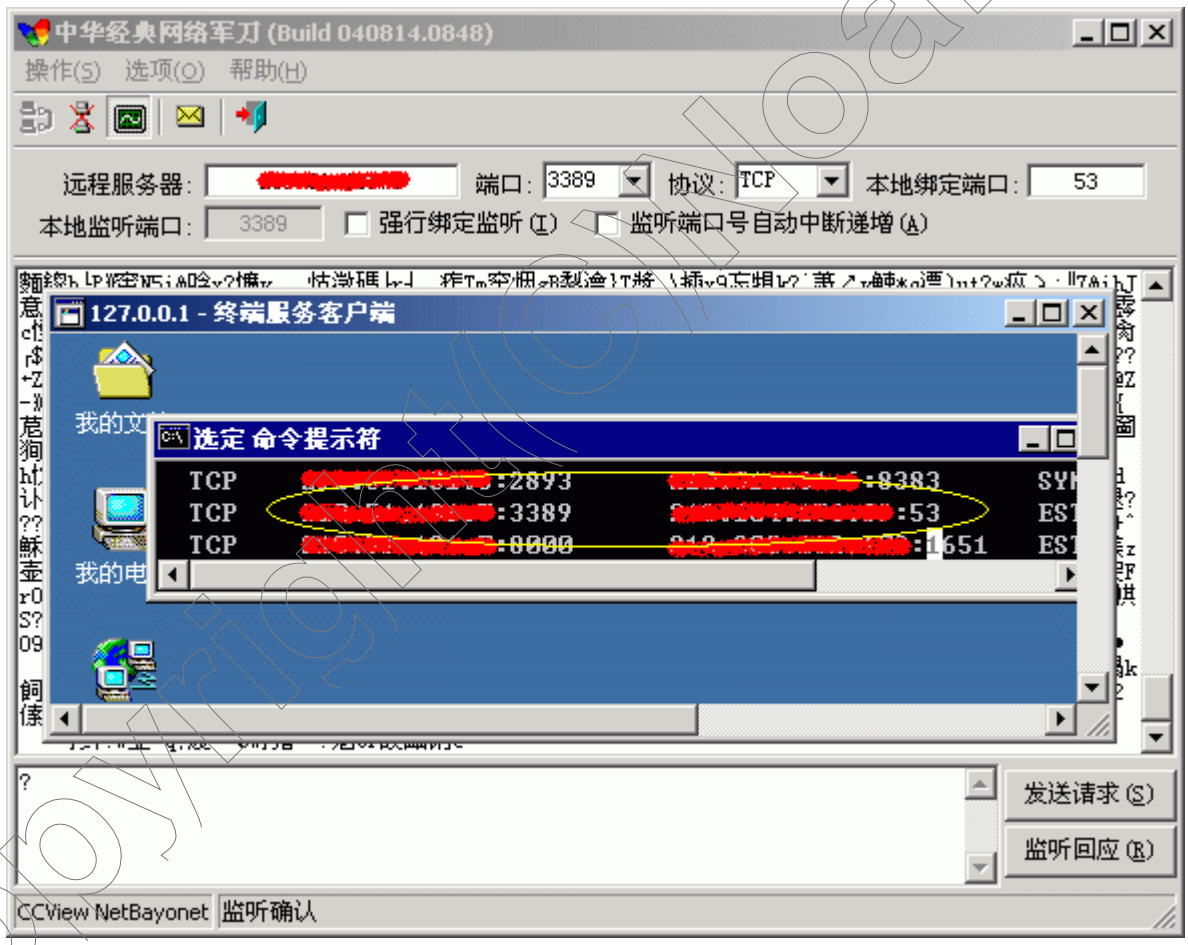
服务器日常安全维护应该注意以下原则：

- 随时保持操作系统和病毒升级库为最新。
- 精心构建访问权限控制，尽量保证所有系统和应用日志的不可抹除。
- 如果采用第三方服务软件如 FTP、论坛程序、聊天室等，随时关注其官方安全更新。
- 不将服务器任何操作权限交给第三者，如果一定需要授权，则必须保证第三者的任何操作均在可监控和可管理的范围之内。
- 不在服务器上安装任何不需要的或者类似的多余软件，如除非必要，安装了 IIS 通常就没有必要再安装 Apache，反之亦然，尽可能减少软件冲突以及由各软件自身造成的安全风险。
- 不直接在服务器上执行网页浏览以及其它通讯操作，杜绝由于各种原因引发的、任何可能的直接感染渠道。
- 不要随意运行来历不明或者未经安全审查的软件，避免被释放植入有害程序。
- 警惕集成在合法共享软件中的广告程序和间谍程序，它们在后台可能会收集系统最敏感的信息。
- 警惕那些捆绑在合法软件中的木马程序（通常这些软件不是从官方站点下载，但极端情况下可能官方下载站点也已被黑客攻陷并下套），它们可能会利用一个完全相同的登录界面来截获用户名和密码，然后假装提示密码错误而要求重新输入，而这个时候就切换到正常的登录界面，但是用户名和密码已经被盗了。
- 不要随意在光驱或其它可能允许自动运行的媒体中插入来历不明的光盘或其它媒介，防止由于自动运行而导致的安全灾难，最简单的预防方法就是禁止所有介质的自动运行。

- 随时仔细查验系统事件查看器中的事件日志，及时察觉任何哪怕最细微的可疑痕迹。
- 定期或不定期采用最新升级的、功能可靠的各类扫描（如补丁扫描、漏洞扫描、端口扫描、木马扫描、间谍软件扫描等）软件对系统进行全面的安全风险评估，防患于未然。
- 安装入侵检测系统并随时进行日志审计，虽然这是被动的和事后的监测分析，但如果足够细心和耐心，仍然可以从中发现很多非常隐蔽的、入侵探测留下的种种无法抹除的蛛丝马迹。
- 主动采用蜜罐系统（如中华经典网络军刀的简易蜜罐部署）来诱捕非法入侵企图，同时也可以第一时间、第一现场迅速掌握国际国内第一手的最新入侵技术手段。

日常安全维护时，我们通常会审计安全日志或者查看当前网络连接以发现可疑情况。

现在我们举一个安全审查范例，让我们看一看下面终端服务的 DOS 窗口内容：



假如我们是这台终端服务的管理人员，我们应该如何看待这个从本地 3389 到远程 53 端口的连接？是我们在启动终端服务时系统查询某个网络 DNS 吗？

答案是完全否定的。

很明显，我们从中华经典网络军刀的用户界面可以清楚的知道，这是一个精心构造的、绑定远端 53 端口的、极具迷惑性的、针对我们 3389 终端服务的非法访问，但是很遗憾的是，很多网管却经常轻易

的忽略了这个连接信息，因为这太具迷惑性了。

举一反三，假如我们发现一个指向远程 80 端口的连接，这一定就是我们在访问某个网站吗？我们在配置防火墙策略时是不是应该多考虑些什么？

从上面的案例可以知道，服务器的安全维护绝对不是一件容易的事情，这需要具有相当丰富的、涵盖各个安全领域的网络安全知识和从业经验。

此外，我们还应该注意一个问题，就是网络安全的重任并不能完全压在一个领军人物的肩头，毕竟一个人很难是完美的，也许其可以精通一个或多个安全领域，但是却并不一定可以精通安全链中的所有环节，而即算其精通了所有环节，也并不能保证可以每次都及时处理所有突发的安全事件，因为一个人的精力也是有限的，如果维护一个安全系统领域尚有足够的实力和精力可以保证足够安全性的话，那么维护由多个系统安全领域组成的安全链却极有可能由于疲劳、判断失误和响应不及时而犯错误，而这个错误可能是致命的并导致整个安全链的断裂进而带来整个安全系统的全线崩溃。

安全意识

无论如何，请首先记住一个原则，没有什么是值得完全信赖的。只有依靠不断提高自身的安全意识并且具备相当的网络安全功底，才能拥有片刻的相对安全，正如只有保持足够的军队战斗力才能赢得暂时的和平环境一样。

另外，不要自欺欺人的以为安装了所有操作系统补丁并且安装了最新的病毒软件库，同时也配置了高规格高安全的防火墙策略就可以从此高枕无忧，谁都永远不可能挑战并战胜全球的黑客，因此不要以为服务器今天没有遭到攻击就会永远安全，且不说攻击可能是被精心隐盖了的，世界上永远也不可能有一劳永逸的、永远的安全，正如没有永远的世界和平一样。

如果我们关注服务器安全，请保持以下两个关注：

- 关注操作系统和相关软件的安全漏洞资讯，做到预防为主、有备无患。
- 关注黑客技术和相关工具的发展动向，做到知己知彼、百战不殆。

最后，祝各位好运！

石望湘

http://www.ccview.net

2005-09-23

附录一：开发必知的、保证代码安全的十个最重要安全警示

本附录原则译自微软 MSDN 官方网站的相关英文文档，并根据实际情况进行了必要阐述、补充和删节。
英文原文网址：http://msdn.microsoft.com/msdnmag/issues/02/09/securitytips/

很多原因都将让你的代码陷入不安全困境，这些因素包括但不仅限于：

- 过于信任在网络上运行的代码。
- 用户可以自由访问重要文件。
- 从不检查电脑里面没有改变过的代码。
- 代码缺乏安全考虑，没有杀病毒软件以及防火墙的保护。
- 给不必要的多余账号分配了不必要的多余权限。
- 服务器端口不加监控。
- ...

很显然，导致不安全的因素正在日益增加，并且增长速度和涉及环节甚至将可能超出我们目前的想象，因此，如何保护我们的系统和代码显得尤为重要。

微软官方从多层次、多渠道传达了这样一种思想，就是“本公司将安全问题作为最优先课题来处理。用户也应该采取导入安全软件及更新旧电脑等措施，在安全方面承担起相应的责任”，这是美国微软首席安全战略官(Chief Security Strategist）Scott Charney 提出的安全方针。

微软董事长比尔·盖茨于 2002 年 1 月宣布了微软的“Trustworthy Computing”可信赖计算方针，强调与产品供货相比更为优先考虑安全对策，提倡“Secure by design, secure by default, and secure by deployment”，为此，微软斥资数亿美金，暂停所有研发工作，历时两月，全员彻查代码安全问题。

对于我们而言，安全问题直接牵涉到产品质量以及可维护性，同时，安全设计也是每一个开发人员最基本的素质要求。

安全隐患源自多方面，安全风险存在于任何角落，安全专家提出了如下保证代码安全的十条安全警示：

不要信任用户输入（Don't trust user input）

不要指望用户输入都是符合格式并且无害的，风险来自于任何客观因素甚至人为的故意，这些因素包括缓冲区溢出、跨站脚本攻击、SQL 语句注射以及其它各种甚至是未知的攻击形式。

防范缓冲区溢出（Protect Against Buffer Overruns）

未经检查和妥善处理的缓冲区溢出将导致覆盖后续内存，而这种覆盖可能是被精心设计的、可执行的汇编语言代码所取代，这些植入代码将可能拥有系统最高权限并且可以执行任何操作，这意味着什么？

防止交叉跨站脚本（Prevent Cross-site Scripting）

跨站脚本是 Web 应用所特有的安全问题，仅仅一个简单的 Web 页就可能危及客户端信息安全，下面这句 ASP.NET 代码想必使用得十分频繁和不足为奇：

```
Response.Write("Hello, " + Request.QueryString("name"));
```

然而，这却是一句 Bug 成灾的代码，因为没有采取任何手段来过滤用户输入！

如果用户提交的是一个正规的姓名倒也无妨，但是如果提交了一个精心构造的包含 HTML 或者 Javascript 脚本的字串，那么客户端本地 cookie 甚至其他敏感信息如电子商务数据将可能被发送到某个隐含的被黑客控制的非法站点。

因此，需要通过正则表达式或者其它经过严密设计的数据处理逻辑，严格过滤用户提交的输入参数（尤其是单双引号、分号以及任何标记语言元素如“<>”符号等）。

另外，如果确实需要输出包含 HTML 的输入，那么在传递之前必须至少通过一种编码以减少风险，ASP.NET 提供 HttpServerUtility.HtmlEncode 编码方法（ASP 为 Server.HTMLEncode）。

防止授权外的 SQL 语句注射（Don't Require sa Permissions）

许多开发人员都习惯于组合用户输入为最终的 Select 查询串，并且从来未曾意识到会有什么样的严重后果，例如下列代码使用得十分普遍：

```
sqlstring="Select * From Table Where ID='"+ID+"'";
```

首先，Select 语句运行采用 sa 角色权限，因此拥有至高的操作权力，假如这个用户输入的 ID 又是一个精心酝酿的、两端封闭嵌套的另一个 Select 类语句或者扩展查询条件（如" 1' or '1'='1 ",），那么，这将导致执行额外的操作，这种不受欢迎和不曾预料的额外操作可能是灾难性的，例如导致身份验证失效、获得系统最高权限以及获取甚至删除关键敏感数据。

SQL 语句注射安全风险普遍存在于所有支持 Select 查询的数据库系统，包括但不仅限于 SQL Server、Oracle、mySQL 等，消除这个风险的方法是采用正则表达式过滤提交字符串，同时尽量采用支持参数的存贮过程（出于安全和性能的双重考虑）。

注意自己的加密算法（Watch that Crypto Code）

我们每天都在同身边很亲近的人谈论一些很机密至少是很私密的信息，我们的应用也一样需要进行

类似敏感数据的通讯传递，不要以为自定义的算法别人就无法破解，且不说自定义算法可能设计得永远都不是那么完善，黑客也完全有时间和足够的知识来研究这种业余的算法，并通过反编译工具实施跟踪和破解，甚至这还是很轻易的。

因此，对于加密算法，应该使用系统提供的 Win32 CryptoAPI 或者 System.Security.Cryptography 命名空间来构架更加完善的加密机制。

尽量减少被攻击的可能性（Reduce Your Attack Profile）

如果一个功能没有 90% 被客户端使用的可能，默认就不要安装它（包括对系统组件和服务的安装以及应用程序的部署设计）。

扩展说来，不要安装不必要的服务，不要在应用程序里滥用 Administrator 权限去处理一些低级别事务，也不要提供任何不必要的功能和操作。

采用最小权限分配原则（Employ the Principle of Least Privilege）

操作系统和公共语言运行时(CLR)提供安全策略的原因有很多，人们普遍认为多数原因是为了阻止内部用户蓄意的越权操作，如存取不应该访问的文件、按照他们的个人需要修改网络配置等种种卑鄙行为。

这种观点固然不错，但还有其它更重要的外部原因需要保持安全系统的严密，这就是来自外部网络的、频频发生的、普通用户所不能察觉的、可能导致系统和网络瘫痪的意外事件，例如很典型的就是伪装在邮件附件中的病毒（Virus）或者特洛依木马（Trojan horse）可能在不知不觉中被不期望的后台运行，从而导致内部资源泄漏，甚至导致系统和网络的崩溃。

一个好的安全策略可以将各种危害降低到尽可能最低的程度。

在设计、构造配置和部署应用服务的时候，不能一厢情愿的假定所有请求都来自合法用户的合法需求，除了用户可能因为好奇而试图去越权访问，同时还因为一个下级节点受信用户的网络通讯可能遭到了不法劫持，甚至这种劫持也可能发生在网络的上一级节点！另外，在设计构造的时候还需要预防代码被非法溢出和突破。

基于以上思想，采用最小权限分配原则意味着仅仅应该赋予特定操作必要的、恰好的权限并且保证仅在被执行时才分配这样的权限，执行完毕应该立即取消。也就是说，要时时刻刻尽可能保持防范，

如果的确需要请求更高权限，则应该尽可能分离代码，即仅仅允许确实必要的代码执行更高权限。如果必要，可以考虑采用基于 COM 或者 Microsoft .NET remoting 的通讯封装，并保证接口的通讯回路最小，以减少被泄露和劫持的几率。

如果使用 .NET Framework 分布式部署，仔细分析每个部分的权限级别，将具有更高权限的请求单独部署，以保证绝大多数的代码运行在尽可能低的权限。

许多人喜欢为他们的应用程序提供插件支持，但是这种易于扩充的模式却很难防范由于插件的 Bug 以及安全漏洞而带来的风险，因此，如果确实需要提供插件支持，则应该明确规划哪些操作是允许的，而哪些是应该限制的。

充分关注错误处理（Pay Attention to Failure Modes）

必须承认，你的下一任开发人员和你一样痛恨错误百出的代码，相信任何人都不愿意听到这样对自己的评价。

大多数程序员，包括我们自己，都更关注于正常的代码执行流程，而忽略错误处理。

也许业务流程的设计会是十分周详和完美的，但是错误捕获的异常处理却可能非常欠缺和糟糕。可以回忆一下，你是否像设计严谨的业务逻辑一样设计过错误处理机制，还可以想一想，是否像逐行审查和调试业务逻辑代码一样检查过错误处理代码？

未经充分测试的代码往往可能带来安全漏洞，可以采用四种方案降低这种风险。

- 首先，像关注业务逻辑代码一样重视每一个细微的错误处理环节，保证系统始终运行在正确和安全的状态。
- 每完成一个函数，总是进行反复跟踪和测试，包括提交一些非法的参数或者更改系统配置而故意引入错误，直到尽可能涵盖了所有可能（至少是已知）的错误处理（虽然我们客观上可能仍然无法发现一些很细微的错误）。总之，需要反复不断地、一步一步地调试每一行代码，因为每一次仔细的、不同角度的逐行分析都有可能发现一些程序缺陷，从而及时消除问题。
- 每一次重新编译都需要重新审查代码，因为每一次对旧错误的修改都极有可能引入新的错误。
- 每一次运行错误都可能发生在任何不可预知的位置，因此我们不应该很乐观地信任被检测条件或者对象，而应该总是很悲观的假设这个条件或者对象是不能信任的，直到这个条件或者对象确实已经通过验证并且值得信任。举例说明，假如我们需要打开一个文件并且返回这个操作成功与否的标志，那么是首先假设能够成功打开，一旦捕获到失败再设置失败标志好呢，还是首先假设失败直到成功打开才设置成功标志好？很显然，我们应该首先假设系统可能失败，这才符合设计安全的要求。

扮演模式是脆弱的（Impersonation is Fragile）

扮演模式通常运用于服务代理，即接受远程客户端请求，然后在本地执行实际操作并将操作数据返回给客户端，这种方式存在一个问题，即虽然客户端角色的权限很低，但是本地进程有可能是基于系统最高权限，也就是说存在被客户端恶意缓冲区溢出从而导致客户端权限在本地提升，进而对服务器系统

构成严重威胁的可能。

要解决这个问题，需要谨记预防缓冲区溢出以及最小权限分配原则，尽可能保证服务安全。

编写非管理员权限可以执行的应用（Write Apps that Non-admins Can Actually Use）

很显然，这条原则是基于前面第七条安全法则的必然结果，也只有这样，才能充分利用操作系统的安全验证机制，将安全风险控制到最小。

版本修订

【2004-10-07】 第一版定稿。

【2005-09-23】 第一版第一次修改，主要在《编写安全 Web 服务代码》节增加了 Digest 身份验证相关内容，以及增加《SOAP 通讯安全》节，其余部分进行了少量的必要补充和完善。