缓冲区溢出防范策略

缓冲区溢出防范策略

软件是一种典型的以有效的方式操纵数据的书面形式的产品。这些数据可以是文本，图像，视频或声音；但是，就程序而言，本质上它只是一串数据——通常以字节（8 位数据）的形式代表某些意义（例如，颜色或文本字符）。当一个程序员申明一个缓冲区时，非常容易申请一个可能不适合程序后来的指针使用的缓冲区，或者可能会在没有很充分地验证输入时接受超出缓冲区大小的数据。这就涉及到缓冲区溢出的问题，本手册将围绕这一问题展开讲解。

初识缓冲区溢出

许多流行的语言，例如 C 和 C ＋ ＋，留给程序员很大的自由度——比如让程序员明确申明缓冲区为一定的大小。那么什么是缓冲区溢出呢？它对我们的软件程序应用会带来什么样的危害呢？

- 是什么导致网络应用中的缓冲区溢出和内存泄露？
- 缓冲区溢出攻击是如何发生的

深入理解缓冲区溢出

了解了缓冲区溢出之后，你是否想更进一步的深入理解它的发生原理呢？目前的软件市场上这种情况呈现怎样的趋势呢？

- 如何破坏代码之缓冲区溢出
- 淹没于缓冲区溢出漏洞

- 缓冲区溢出攻击：它们是如何进行的？

缓冲区溢出的应对策略

现在我们完全理解了缓冲区溢出进行的原理，那么我们该怎样在自己的程序中避免它的发生呢？还有，我们应该怎样防御来自外界的缓冲区溢出的攻击呢？下面将向你介绍相关的应对策略。

- 理解和阻止缓冲区溢出
- 防止缓冲区溢出攻击的策略

是什么导致网络应用中的缓冲区溢出和内存泄露？

问：我们所说的在网络应用程序中的内存泄漏和缓冲区溢出是什么意思呢？你可以用一个例子说明如何检测它们吗？

答：这两个问题一直困扰着网络应用程序。缓冲区溢出攻击（如 Code Red 蠕虫病毒），会导致重要的数据泄露，危及系统的安全。不过，首先让我们来看看内存泄漏，因为这将帮助你理解缓冲区溢出。如果程序员为某种类型的变量动态分配了内存空间，而在程序结束之前又没有释放出这部分空间，此时内存泄露就发生了。这将使系统可用的内存更少，而重复运行这样内存泄露的程序或函数，则最终会导致系统崩溃或拒绝服务（denial of service）。下面是一个会导致内存泄露的简单的代码示例：

```
int main(int argc, char **argv)
{
char * memoryA = new char[10];
memoryA [0] = 'A';
printf("%cn", memoryA [0]);
}
```

该程序给数组 memoryA 分配了 10 个字符的存储空间，但该程序最后并没有显式释放这些空间。

为了防止内存泄漏，当变量不再使用时，程序员应该总是释放所有动态分配的空间。对于简单的应用程序，你可以检查你的代码，以确保每一个新的申请内存的操作都有一个对应的释放操作，或者是采用语言近似配对（language-equivalent pairing）的办法。而对于更复杂的项目，你需要运行可以检测内存错误的应用程序诊断工具，如 Purify 或 LeakTracer。此外，还可以对你的应用程序进行压力测试（stress-test），并监视其内存的消耗情况。

缓冲区溢出可以导致应用程序或系统的崩溃，甚至还会使黑客攻击系统，并启动未经认可的进程。当程序或进程试图存储超过缓冲区设定大小的数据到缓冲区时，缓冲区溢出就发生了。例如，黑客故意给出一个比分配给它的内存缓冲区大的数据，以表（form）的形式将数据提交到网络应用程序。这种额外的数据可以覆盖邻近的内存，重写其中的任何有效的数据，往往是重写函数结束时的返回地址。通过编写一个新的返回地址，黑客可以欺骗系统去执行自己的代码。

你应该知道，有些语言比其他语言更易产生缓冲区溢出和其他内存泄漏错误。C 和 C++没有提供内置的检测，因此不能确保写入到缓冲区内的数据是在该缓冲区的范围内。Windows 程序员应该使用微软 Visual C++.NET2003 提供的 strsafe.lib 和 strsafe.h，从而使用那些能够安全处理字符串的函数库。然而，用 Java 编写的应用程序并没有真正受到缓冲区溢出和内存泄漏的威胁，因为 Java 是一种强类型（strongly typed）语言。但是，请注意，当和其它语言编写的服务和库交互时，用 Java 和其他“安全”语言编写的应用程序仍然可能受缓冲区溢出影响。

防止缓冲区溢出问题的最好办法是验证所有应用程序收到的输入数据。你甚至可以编写你自己的字符串复制函数，从而可以同时加入数据过滤检测的功能。

（作者：Michael Cobb　　译者：Sean　来源：TechTarget 中国）

缓冲区溢出攻击是如何发生的

用户提问：

我不确信我完全理解缓冲区溢出。我知道在向目标机器插入代码时会导致严重的缓冲区管理缺陷，但我不知道黑客是怎样来使这些代码得以执行的——很可能是在他或她获得了目标机器的控制权的时候发生的。黑客能够保证代码被调用吗？亦或者这其实仅仅是一个概率问题？

专家回答：

当程序或进程试图在其分配的数据存储区域、或缓冲区中存储超出预计的数据时，缓冲区溢出就会发生。由于创建的缓冲区中只能存储有限的数据，额外的信息会溢出到邻近的缓冲区中。当这种情况发生时，原来存储在这些缓冲区中的有效数据会被破坏或覆盖。函数中的局部变量的缓冲区溢出可能会覆盖该函数的返回地址。（返回地址指向该函数执行完成后应该执行的下一条指令）。这可能会导致段错误，而段错误（segmentation fault）则可以使程序崩溃。在某些特定情形下，程序崩溃后黑客会收到一个 shell 提示（shell prompt：提示用户输入命令行），这就把计算机的控制权交给了黑客。更复杂的攻击是：黑客用期望执行的代码的指针重写返回地址，而不仅仅是让计算机崩溃。

基于堆栈的缓冲区溢出攻击是最常见的，但让我们看看在 JPEG 处理（GDI +）开发中基于堆的缓冲区侵占，见识缓冲区溢出攻击的巧妙之处。

微软的动态链接库文件称为 GDIPlus.dll，其中包含了图形设备接口以及（GDI +）应用程序编程接口（API）库。它允许程序员表示图形对象，并传输到输出设备--如显示器和打印机。这些 DLL 有处理 JPEG 图像文件的能力，但它在检查其实际的值之前允许 JPEG 文件声明注释区域的长度，。这时就可能会导致基于堆的缓冲区溢出。基于堆的缓冲区攻击发生在数据副本被写入到位于堆内的缓冲区的时候。这意味着非可执行堆栈保护机制可以被绕过，最终导致系统十分脆弱，并允许黑客指向他们希望运行的下一个进程的代码。讽刺的是，黑客可以在 JPEG 文件的注释区域存储这些代码。现在，只要被攻击者查看已经被篡改的图像，黑客就可以利用这个漏洞了。

缓冲区溢出攻击如此常见是因为编写程序使用的都是级别相对较低的编程语言，如汇编语言，C 和 C + +，它们没有自动给缓存定限的功能。因此程序检查到数组或指针时，要求程序员手动管理分配内存的大小。虽然黑客不能保证每一次溢出攻击代码都能成功运行，但鉴于各种病毒以及蠕虫的成功案例，可以说他们仍然可以有很高的成功率。要查看缓冲区溢出是如何进行的，请访问以下网站的 Java 程序演示：
http://nsfsecurity.pr.erau.edu/bom_docs/Demos/script.html。另外一个很好的初学者学习缓冲区溢出攻击的网站是：
http://www.securiteam.com/securityreviews/5OP0B006UQ.html。

（作者：Michael Cobb　　译者：Sean　来源：TechTarget 中国）

如何破坏代码之缓冲区溢出

缓冲区溢出就像是个给软件安全捣蛋的男孩。围绕缓冲区溢出的讨论一直无所不在的主要原因是：缓冲区溢出是在利用软件时向目标远程注入恶意代码的主要方法。虽然缓冲区溢出的技术在很多地方公布了，这篇文章仍然是必要的。缓冲区溢出攻击已经进化了多年，和其他的攻击手段一样，新的更强大的缓冲区溢出攻击已经出现。即使不为别的，本章也将作为你更细致地掌握缓冲区溢出攻击的基础。

缓冲区溢出 101

缓冲区溢出攻击仍然享有恶意攻击之王的桂冠，并很可能在接下去的几年中一直如此。那些广泛存在的导致缓冲区溢出的漏洞是部分原因。只要这些漏洞存在，他们就将被利用。那些仍使用已经过时的内存管理功能的语言，如 C 和 C++，助长了缓冲区溢出攻击。只要开发商仍然意识不到使用一些常用的数据库功能和系统调用的安全隐患，缓冲区溢出攻击仍将普遍存在。

控制流和内存漏洞可以有多种形式。在谷歌上搜索关键词"缓冲区溢出"会返回 176000 个结果。显然，曾经神秘难解的技术现在都普及了。然而，大多数攻击者（和维护者）无论对缓冲区溢出以及对他们能造成的破坏都只有最基本的了解。大多数人对安全有点兴趣的人（那些阅读信息安全文件并出席各种安全保障会议和商业活动的人）都知道，缓冲区溢出将允许远程代码注入到系统，然后运行。结果适当蠕虫和其它类型的恶意移动代码有了明确的攻击系统的路径，并留下 rootkit 等后门。在许多情况下，通过缓冲区溢出进行远程代码注入是可行的，后门也可以很容易地安装。

缓冲区溢出是一种内存使用上的漏洞。这个漏洞是和计算机科学的发展史密切相关的。内存曾经是一种昂贵的资源，因此，内存管理是至关重要的。在一些较旧的系统，如 Voyager spacecraft，内存非常珍贵，所以一旦某些部分机器代码已不再需要，它们会马上从内存中被移除，以腾出空间作其他用途。这有力地促成了创建那些自释放的，只运行一次的程序。相反的，现代系统都有大容量的内存，几乎从来不用释放。今天大多数的连接到网络的软件系统

都有可恶的内存问题，特别是当直接连接到不可靠的环境，如互联网时。内存已经很便宜，但低效的内存管理的代价是非常昂贵的。对内存的不良使用可能会导致程序内部崩溃（尤其是对控制流的引用），以及拒绝服务问题，甚至缓冲区溢出等这些远程攻击问题。

讽刺的是，虽然业界已经知道如何避免缓冲区溢出问题；但是，这些年来，能够解决这个问题的知识，对阻止网络环境下猖獗的缓冲区溢出问题几乎没有发挥任何作用。事实上，在现有的技术下能够解决这个问题，但从社会学上来说我们还有相当长的路要走。主要问题是，绝大多数的开发商仍然没有意识到这个问题。很可能在未来五到十年的时间，各种类型的缓冲区溢出问题将继续困扰着软件业。

最常见的缓冲区溢出——堆栈溢出——可以很容易地被程序员杜绝。其实一些更高深的内存崩溃的形式，比如堆溢出，才是更难避免的。一句话，在现代内存管理方案的现代编程语言被广泛使用之前，内存利用漏洞还将一直是攻击软件的主要源头。

（作者：Greg Hoglund and Gary McGraw　译者：Sean　来源：TechTarget 中国）

淹没于缓冲区溢出漏洞

用碰撞时会爆炸的油箱去制造汽车可不是一个好主意，所以汽车厂商都不会在他们的产品中使用有这样的部件。软件厂商可以从中汲取教训，并用于解决他们自己内部的安全炸弹：缓冲区溢出。

Gartner 公司的研究指出，缓冲区溢出是程序的最常见的。不幸的是，尽管在 1999 年就已经得出了这项研究成果，这一安全漏洞问题却没有得到多少改善。往往一个漏洞刚刚修复，不到一个星期，新的漏洞就又出现了。

溢出发生在程序试图存储比分配的内存容量更多的数据。多出的数据会溢出到邻近的内存区域，覆盖掉已经存储在那里的数据和指令。不怀好意的黑客们已经能十分熟练地利用这种溢出方式把他们自己的代码植入到计算机程序中，从而有效地劫持计算机。

如果在存储数据之前没有检测该数据块大小的代码，溢出同样也会发生。在一些编程语言中，溢出很难或几乎不可能发生，因为它们能自动扩大内存空间来接收到来的数据。而包括 C 语言在内的一些其他语言，溢出几乎则不可避免，它们通常都缺乏任何自动检测容量大小的机制，并会十分乐意地向 5 磅的内存区中塞入“10 磅的数据”。

如果程序员不对溢出条件作专门的测试，这样的缺陷就会出现在应用软件之中。开发时限带来的压力更加重了这一问题：问题往往不是被开发人员或测试人员发现，而是在数以百万计最终用户那里才被发现。

这些溢出问题有多么普遍呢？一些软件供应商是比其他人更恶劣的违规者？这可能正如您所预测，因为不同的公司在编写自己的应用软件时，使用的是不同的编程标准。那么这个问题是变得好了呢，还是变得更糟了？

一份 IT 安全公司 Secunia 对咨询建议数据库的期望报告，可以为溢出问题的地位提供一个评估。这篇报告的评估标准包括：严重的漏洞攻击，远程利用和毁坏（服务的允许和拒绝、

敏感信息或系统信息的暴露、系统劫持、ID 欺骗，操纵数据、权限升级、安全旁路或系统访问）。

非常明显，由溢出而引发的漏洞攻击是很广泛的。2002 年 9 月至 2004 年 3 月的调查报告中显示，三分之一的漏洞攻击与溢出相关（ 224/659 ） 。在这段时间内，每个月至少有 4 个与溢出有关的咨询，而其中的一些月份则达到了 26 个，平均每个月有 11 个。

不仅仅小型的、资源紧缺的公司目前正为这一问题而烦恼。相反，产品中严重的溢出问题更加困扰着许多大型的、知名的软件供应商。

各厂商出现的漏洞中，由于溢出导致的漏洞数各有不同。苹果、Opera、甲骨文、赛门铁克和雅虎所出现的几乎所有的警告都与溢出有关。而思科、微软和 Sun 则只有不到一半的警告与溢出有关。当然，这样差异可以有很多不同的解释方式。也许后者是更擅长于在代码中避免发生溢出，也许前者更有效地消除了所有其他类型的漏洞。

（作者：Edmund X. DeJesus　　译者：Sean　来源：TechTarget 中国）

缓冲区溢出攻击：它们是如何进行的？

缓冲区溢出是黑客们喜欢利用的手段之一。绝大多数微软提供的补丁修复程序并未检查缓冲区问题，而应用程序的内部开发是怎样的呢？它们和商业应用软件一样，对缓冲区溢出攻击敏感。因此，在使用本地应用程序之前，知道应用软件如何工作和怎样进行漏洞测试是很重要的。

缓冲区溢出攻击利用了程序等待用户输入的缺点。目前，主要有两种类型的缓冲区溢出攻击：基于堆栈和基于堆。基于堆的攻击是对程序预留内存空间进行溢出，但由于难以加入可执行的指令，这种攻击现在很罕见。目前最常见的是基于堆栈的缓冲区溢出攻击。

在基于堆栈的缓冲区溢出攻击中，被攻击者利用的程序使用一个名叫堆栈的内存对象来存储用户的输入。通常情况下，在程序没有用户输入以前，堆栈是空的。有用户输入时，程序先写入一个返回内存的地址到堆栈，然后把用户的输入数据存储在返回地址的上方。当执行堆栈时，用户的输入数据就被传送到程序指定的返回地址中。

然而，堆栈的大小都是有限的。开发代码的程序员必须给堆栈保留一定数量的空间。如果用户的输入超出了在堆栈里为它预留的空间大小，那么堆栈会溢出。这本来不是一个很大的问题，但如果遇到恶意输入时，它就变成了一个巨大的安全漏洞。

例如，假设某个程序正等待用户输入他或她的姓名，黑客可能会输入长度超过堆栈大小的可执行命令，而不是输入姓名。而这条命令通常很简短。例如，在 Linux 环境下，典型的如 EXEC（“sh”）命令，它要求系统打开一个命令提示符窗口，使用 Linux 的人称它为 root shell.。

然而有可执行指令的缓冲区溢出并不意味着指令就会被执行。攻击者必须为恶意指令指定一个返回地址。由于堆栈溢出，程序会部分崩溃。然后程序会试图通过指定返回地址进行恢复，但是返回地址已经被黑客改变为指向恶意指令。当然，这意味着黑客必须知道恶意指令将

存储的地址。为了省去寻找精确地址的麻烦，攻击者可以通过对恶意指令的两头进行 NOP 指令（一种指针）填充。在不知道内存精确范围时，对堆栈两头进行填充是常见的技术。因此，如果黑客指定的地址中落在填充的范围之内，恶意命令将可能会被执行。

最后一部分是可执行程序的权限。如你所知，大多现代化的操作系统都采用某种机制来控制当前登录用户的访问级别权限，而执行程序通常需要更高级别的权限。因此，这些程序以内核模式或以葱服务帐户继承的权限模式运行。当堆栈溢出攻击执行新的返回地址上的命令时，程序认为它仍然在运行。这意味着，命令提示符窗口同被攻击的应用程序一样具有相同的权限。通常，这意味着攻击者已经完全取得了操作系统的控制权。

（作者：Brien M. Posey　译者：Sean　来源：TechTarget 中国）

理解和阻止缓冲区溢出

什么是缓冲区溢出

软件是一种典型的以有效的方式操纵数据的书面形式的产品。这些数据可以是文本，图像，视频或声音；但是，就程序而言，本质上它只是一串数据——通常以字节（8 位数据）的形式代表某些意义（例如，颜色或文本字符）。许多流行的语言，例如 C 和 C + +，留给程序员很大的自由度——比如让程序员明确申明缓冲区为一定的大小。其结果是，当一个程序员申明一个缓冲区时，非常容易申请一个可能不适合程序后来的指针使用的缓冲区，或者可能会在没有很充分地验证输入时接受超出缓冲区大小的数据。

当你把超出缓冲区存储范围的数据放入缓冲区时会成为一个问题，因为这些数据后来往往终结在执行代码栈。常常攻击者会通过覆盖函数的返回值，使他们中断程序流程而运行自定义的代码来实现攻击。即使在有限的空间（不到 100 个字节的可执行代码）大多数攻击者可以插入足够的代码威胁系统。这是尤其危险的，因为为了访问用户文件和其他敏感的业务，许多应用程序，特别是网络服务程序，以较高的权限运行——因此攻击者的代码也运行在一个很高的权限。

为了避免缓冲区溢出，程序员在接受数据时必须牢记缓冲区大小，检查所有的输入数据的规模，以确保它不是太大。不幸的是，并不是所有的缓冲区都是可访问或可知的。往往和程序互动的第三方应用程序和库将会在没有充分检查输入数据的情况下调用缓冲区。所有这些都超出了程序员的控制范围。

防止缓冲区溢出

现在你知道什么是缓冲区溢出了。但你很可能像我一样，没有足够的时间去审核每一行源代码并重新编译应用程序——前提是您可以访问源代码。幸运的是，有多种软件包和操作系统可用于防止缓冲区溢出。当然，即使这安装了这些软件包，攻击者通常仍然可以执行拒绝这一服务。大多数操作系统可以通过检测缓冲区溢出并停止任何进一步的操作来有效地扼杀攻击。

有三个主要的保护方案：修改系统内核（从而保护所有的应用程序）；修改编译器或源代码（从而保护某一应用程序，并要求重新编译）；增加一个中间设备层（以保护所有的应用程序）。您对源代码的权限和您能修改您的系统的程度将决定您采用哪种方案。

Windows

Windows 本身没有能力来检测和防止缓冲区溢出。有一个来自 SecureWave 的叫做 SecureStack 的第三方软件包可以检测并阻止很多缓冲区溢出，但需要提醒的是它需要一个强劲的 CPU 才能有效地工作，而且不便宜，每套软件 249 美元。

Solaris

Solaris 通过设置不可执行的堆栈而内置了检测和阻止某些种类缓冲区泄露的功能。但是请记住，有些程序依赖可执行的堆栈，设置这样的选项可能破坏这些程序。你可以加入这一行代码来检测和记录缓冲区溢出：

```
set noexec_user_stack_log=1
to /etc/system and reboot.
```

这样可以报告缓冲区溢出。

加入这一行来阻止缓冲区溢出

```
set noexec_user_stack=1
to /etc/system and reboot.
```

这将会有效的阻止很多缓冲区溢出。

OpenBSD on Solaris

OpenBSD 的内置了检测和防止某些缓冲区溢出功能。在 Sparc 平台下的 StackGhost 有一个硬件上的特性，只允许“cookie”或小块的数据经过 XOR 处理后才存放到返回地址。它

在缓冲区被使用过后马上清空，让其对攻击者不可预测，这就打防止了许多攻击。欲了解更多信息请参阅：stackghost.cerias.purdue.edu/

Linux -- libsafe

Libsafe 作为操作系统的接口来保护并防止一系列易被攻击的函数受缓冲区溢出的影响，以及避免字符串格式泄露问题。这是特别方便的，因为不需要为了缓冲区溢出而重新编译内核或者应用程序。欲了解更多信息请参阅：www.research.avayalabs.com/project/libsafe/

Linux -- Openwall 内核补丁

Linux 下的 Openwall 内核补丁需要重新编译内核，而且只适用于较老的 2.2.x 内核；幸好，2.4.x 的测试补丁已经发布。它增加了一个非可执行堆栈功能——只适用于 x86 系统——这将防止许多缓冲区溢出。欲了解更多信息请参阅：www.openwall.com/linux/

Linux -- StackGuard

StackGuard 通过在缓冲区末尾放置一个标志位防止缓冲区溢出；如果这个标志位由于缓冲区溢出而改变，系统就会检测到这个问题并使程序停止。StackGuard 是 GCC 的增强版。但不幸的是，程序必须重新编译才能得到保护。 www.immunix.org/stackguard.html

（作者：Kurt Seifried 译者：Sean 来源：TechTarget 中国）

防止缓冲区溢出攻击的策略

了解了缓冲区溢出是如何进行的以后，如何防止黑客利用缓冲区溢出攻击并控制你的本地应用程序就是一个很重要的问题了。

避免使用编译器中自带的库文件

编程语言通常都要带有库文件。如果一个库文件具有某些漏洞，任何包括该库文件的应用程序就都会有这些漏洞。因此，黑客往往会先试图利用常用的库文件中已知的漏洞来达到攻击本地应用程序的目的。

库文件本身也不可靠。虽然最新的编译器都开始加入大量可靠的库文件，但长期以来库文件为了提供了快速、简单的方式来完成任务，几乎没有考虑到安全编码的问题。C + +编程语言就是这种形式的最典型代表。而用 C + +编写的程序中依赖的标准库就很容易在运行时产生错误，这也为希望利用缓冲区溢出进行攻击的黑客们提供了实现他们想法的机会。

验证所有的用户输入

要在本地应用程序上验证所有的用户输入，首先要确保输入字符串的长度是有效长度。举个例子，假设你的程序设计的是接受 50 个文本字符的输入，并将它们添加到数据库里。如果用户输入 75 个字符，那么他们就输入了超出数据库可以容纳的字符，这样以来谁都不能预测程序接下来的运行状况。因此，用户的输入应该这样设计：在用户输入文本字符串时，先将该字符串的长度同最大允许长度进行比较，在字符串超过最大允许长度时能对其进行必要的拦截。

过滤掉潜在的恶意输入

过滤是另一个很好的防御措施。先看下面例子中的 ASP 代码：

这是从用户的输入中过滤掉 HTML 代码，撇号和引号的代码。

```
strNewString = Request.Form("Review")
strNewString = Replace(strNewString, "&", "& amp;")
strNewString = Replace(strNewString, "<", "& lt;")
strNewString = Replace(strNewString, ">", "& gt;")
strNewString = Replace(strNewString, "'", "``")
strNewString = Replace(strNewString, chr(34), "``")
```

上面的代码用于目前我正在开发的电子商务网站中。这样做的目的是为了过滤掉可能会导致数据库出现问题的 HTML 代码和符号。在 HTML 代码中，使用"<"和">"的符号来命名一个 HTML 标签。为了防止用户可能会在他们的输入里嵌入 HTML 代码，因此程序过滤掉了"<"和">"符号。

在 ASP 代码中，撇号，引号和连字符都是保留符号。这些保留的符号不可以包括在用户的输入中，否则它们会导致应用程序崩溃。例如，如果用户输入一个文本行中只使用了一个撇号，之后登陆数据库时，这个命令将会失败，因为 ASP 需要利用成对的撇号将文本括起来提交到数据库里；ASP 不知道如何处理用户的输入中的撇号。为了防止这种情况发生，以上的代码可以寻找到输入字符串中的撇号，并以"``"替代它。

测试应用程序

为了保护程序免受缓冲区溢出的攻击，验证和过滤用户的输入已经实施很久了。但在部署应用程序之前，你仍然需要对它进行全面彻底的测试。应当有专门的人来仔细地审查应用程序，并试图使它们崩溃。让他们尝试输入长的字符串或保留字符。如果你的应用程序在编写中已经做了足够的工作，它应该能应付各种各样的情况。如果程序崩溃了，最好马上把问题找出来，而不要等到已经应用之后。

（作者：Brien M. Posey 译者：Sean 来源：TechTarget 中国）