安全技术经典译丛

24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them

软件安全的24宗罪

——编程缺陷与修复之道

Michael Howard
(美) David LeBlanc 著
John Viega
董艳 包战 程文俊 译

- 由著名软件安全专家编写的安全漏洞专业书籍
- 发现和修复各种安全漏洞的最佳指导
- 丰富的安全漏洞示例以及修复措施
- 作者长期实践经验的总结

McGraw Hill Education

清华大学出版社

安全技术经典译丛

24 Deadly Sins of Software Security:
Programming Flaws and How to Fix Them

软件安全的24宗罪

——编程缺陷与修复之道

Michael Howard
(美) David LeBlanc 著
John Viega
董艳 包战 程文俊 译

清华大学出版社
北京

Michael Howard, David LeBlanc, John Viega
24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them
EISBN：978-0-07-162675-0

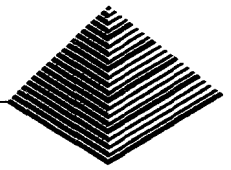
北京市版权局著作权合同登记号　图字：01-2010-0580

图书在版编目(CIP)数据
软件安全的 24 宗罪——编程缺陷与修复之道/(美)霍华德(Howard, M.)，(美)勒布朗(LeBlanc, D.)，(美)维佳(Viega, J.) 著；董艳，包战，程文俊 译. —北京：清华大学出版社，2010.6
书名原文：24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them
ISBN 978-7-302-22634-5
Ⅰ. 软… Ⅱ.①霍… ②勒… ③维… ④董… ⑤包… ⑥程… Ⅲ. 软件开发—安全技术 Ⅳ.TP311.52
中国版本图书馆 CIP 数据核字(2010)第 082322 号

责任编辑：王　军　于　平　　**封面设计**：久久度文化
版式设计：康　博　　**责任校对**：胡雁翎
责任印制：李红英
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京鑫丰华彩印有限公司
装 订 者：三河市金元印装有限公司
经　　销：全国新华书店
开　　本：185×260　**印　张**：20.5　**字　数**：499 千字
版　　次：2010 年 6 月第 1 版　　**印　　次**：2010 年 6 月第1 次印刷
印　　数：1～4000
定　　价：48.00 元

产品编号：033607-01

评　论



“我们一直在为过去编写的软件中的安全漏洞付出代价，如果我们不从这些糟糕的软件中吸取教训，就一定会面临失败。这本专门针对软件安全漏洞的书是由业界一些最受人尊重的作者编写的，是所有软件开发人员和热衷软件安全的人员的必备图书。绝不要再出这样的纰漏！”

——George Kurtz

Hacking Exposed 一书的所有 6 个版本的作者之一，McAfee Security 公司风险和适应业务组的高级副总裁和总经理

“本书的精华是建议如何在您的程序中避免出现 24 个严重问题，以及如何在其他人的程序中检查是否也出现了这些问题。本书的阐述简单、直接而全面，解释了为什么这些问题是漏洞，应如何处理它们。本书适合于每一位程序员，无论使用什么编程语言。我也把本书放在自己的书架上，并作为我的一本教学资料。写得非常棒！”

——Matt Bishop

位于戴维斯的加利福尼亚大学，计算机科学系

“作者再次证明了为什么他们是软件安全专家。本书适合于开发人员、安全人员、项目经理，以及正在开发高质量、可靠和安全代码的任何利益关系人。本书图文并茂，列举了多种语言(C++、C#、Java、Ruby、Python、Perl、PHP 等)中最常见、最危险的错误，以及如何减轻这些错误的影响、修复过去漏洞的各种有效方法。其内容切合实际，指出了可导致代码受漏洞影响的模式(从高级应用函数到代码级的字符串搜索)、软件测试方法、改进易受攻击的元素的策略，以及已实施的攻击实例。本书提出的建议和意见非常符合实际，是由资深从业人员编写的，这些人员已经编写了坚固而有效的软件，供许多不同的用户使用，包括开源社区的用户和大型商业公司。有了这本软件安全宝典，就不会再陷入漏洞中了。”

——Joel Scambray

Consciere 的 CEO，*Hacking Exposed* 系列图书的作者之一

作者简介

Michael Howard 是 Microsoft 公司 Trustworthy Computing(TwC) Group(可信赖计算组)下属安全工程组的高级安全项目经理，负责管理整个公司的安全设计、编程和测试技术。Howard 是一位 Security Development Lifecycle (SDL)构建师，SDL 是一个提高微软软件安全性的过程。

Howard 于 1992 年开始在微软公司工作，那时他在微软公司的新西兰分部，刚开始的前两年在产品支持服务小组为 Windows 和编译器提供技术支持，接着为 Microsoft Consulting Services 提供技术支持，在此阶段，他为客户提供安全基础架构支持，并帮助设计定制的解决方案和软件开发。1997 年，Howard 调到美国，为微软的 Web 服务程序 Internet Information Services 的 Windows 分部工作，2000 年开始担任目前的工作。

Howard 是 *IEEE Security & Privacy* 一书的编辑，经常在与安全相关的会议上发言，定期发表安全编码和设计方面的文章。Howard 与他人一起编写了 6 本安全图书，包括获奖书籍 *Writing Secure Code*(第二版，Microsoft Press，2003 年)、*19 Deadly Sins of Software Security* (McGraw-Hill Professional 出版社，2005 年)、*The Security Development Lifecycle* (Microsoft Press，2006 年)，最近出版的图书 *Writing Secure Code for Windows Vista* (Microsoft Press，2007 年)。

David LeBlanc 博士目前是 Microsoft Office Trustworthy Computing 工作组的一位主要软件开发工程师，负责设计和实现 Microsoft Office 中的安全技术。他还给其他开发人员提供安全编程技术方面的建议。自从 1999 年加入微软公司以来，他一直负责操作网络安全，还是可信赖主动计算(Trustworthy computing Initiative)的创始人之一。

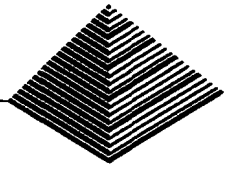
David 与他人合著了获奖书籍 *Writing Secure Code*(第二版，Microsoft Press，2003 年)、*19 Deadly Sins of Software Security* (McGraw-Hill Professional 出版社，2005 年)、*Writing Secure Code for Windows Vista* (Microsoft Press，2007 年)，还发表了许多文章。

John Viega 是 McAfee 的 SaaS Business Unit 的 CTO，是 *19 deadly programming flaws* 一书的作者，这本书引起了出版社和媒体的极大关注。本书就是以该书为基础的。他和其他人共同编写了许多其他关于软件安全的图书，包括 *Building Secure Software* (Addison-Wesley Press，2001 年), *Network Security with OpenSSL* (O'Reilly Press，2002 年)，以及 *Myths of Security* (O'Reilly Press，2009 年)。他负责许多软件安全工具，是 Mailman(GNU 邮件列表管理器)的第一作者，他为 IEEE 和 IETF 中的标准化做了大量的工作，还与他人一起开发了 GCM(NIST 已标准化的一种加密算法)。John 还是几家安全公司的安全顾问，包括 Fortify 和 Bit9 公司。他拥有 Virginia 大学的硕士和学士学位。

技术编辑

Alan Krassowski 是 McAfee 公司的首席软件安全工程师，他领导设计了新一代安全保护产品，并获奖。在此之前，Alan 领导了 Symantec 公司的产品安全小组，帮助其产品团队发布更安全的安全和存储产品。在过去的 25 年里，Alan 参与了众多的商业软件项目的开发。他曾是一名开发主管、软件工程师，同时也是许多在行业中领先的公司(比如 Microsoft、IBM、Tektronix、Step Technologies、Screenplay Systems、Quark 以及 Continental Insurance)的顾问。他获得了纽约罗彻斯特技术学院的计算机工程系学士学位。他目前住在美国俄勒冈州的波特兰市。

序



在应用计算机工程领域中，使安全工作切实可行是我们面临的最大挑战。

所有的工程系统都有指导性需求——它们已成为可测量的要素，如果达不到这些要求，系统就会失败。例如，大楼必须是安全的(不能倒塌！)，但这还不够，大楼还必须是可用的(里面的空间可以使用)、能盖得起来、可以维护(建筑和维护的成本必须使大楼在投入使用后可以盈利)，最后，大楼还应有一定的吸引力(大楼的外观与其居住者的状态以及该属性的价值相关)。每个需求都有自己的优先级，但它们必须都得到满足。

在许多应用计算机工程领域中，人们不大重视安全。一些项目只是轻描淡写地提到了安全，这使安全无法成为真正的工程实践要求。这样很糟糕。软件的复杂性是毋庸置疑的——现代操作系统，甚至是现代网络浏览器，都比航天飞机复杂得多。航天飞机可以杀人，但带有极少的几个异常的软件却不会杀人。所以，安全的基本核心——“正确”，从来都没有成为软件的明确设计规则，更不用说成为根本的设计规则了。于是，软件中对安全的这种漠视使我们不得不忍受大量的重复设计(往好听了说)或错误(往难听了说)。毕竟，无论软件编写得多糟糕，在几乎所有的情况下，都不会有人因此丢掉性命。

破产则完全是另一回事，并不是只有人才会死，公司也会消亡。

计算机安全研究已经持续了数十年，但直到 2000 年以后，不安全软件的后果才最终为外界所知。2003 年“夏虫”(the Summer of Worms)肆虐——简言之，几个恶意操作使整个商业界的 IT 资源完全不可靠达到 3 个月之久，2006 年出了 TJX 事件——攻击者利用无线天线光顾了 T.J.Maxx，盗走了大量的信用卡账户。2008 年，攻击率再创新高，据 Verizon Business 报告，2008 年受到威胁的个人财务记录超过了 2004、2005、2006 和 2007 年的总和。

人们仍没有醒悟。“正确”还是没有受到人们的重视，却得到寄生虫的青睐——这些坏家伙远程闯入系统，利用不正确的代码绕过安全设施、盗取财物。对于用户和公司来说，这都是一个非常显著的问题。

事情这么糟糕，而工程师看到了什么？

一天结束时，地位低的开发人员必须把他听到的所有命令都转换为代码。软件有许多工程要求：性能、可用性、可靠性等。这些要求都有一个非常重要的特点：如果这些要求没有满足，它们就会变成非常明显的问题。可是，安全问题却没有那么明显。

考虑下面的情形：

假定软件有一个性能问题。甚至未受过培训的工程师都会注意到，某个操作需要执行很长时间。为了解决这个问题，工程师可以使用标准的数据集，找出运行过于频繁的代码块。为了检查修复的代码块是否有效，可以在应用修复代码块的前后使用已知的数据集，

很容易看出执行过程需要的时间减少了。

假定软件有一个可用性问题，这比较难修复——工程师一般知道如何管理他们自己的系统，但用户不知道，用户只能立即让技术支持人员和销售人员知道系统出问题了。为了解决这个问题，工程师可以建立新的部署指南，对用户难以手工维护的组件自动化。为了检查修复的部分是否有效，可以给用户发送一个测试版，用户可以报告该测试版是否有效。

最后，假定软件有一个可靠性问题。它崩溃了！很难找出比这个更明显的问题。崩溃报告可以在开发期间发出，如果不在开发期间发出崩溃报告，则在软件发布后，崩溃报告可能由某个愤怒的用户那里手工发出，或者通过Windows错误报告自动收集和排序。

而用户告诉工程师，希望软件运行得更快、更可靠或更稳定时，用户可能并不确切地知道工程师会如何修复问题，但至少知道要工程师做什么。

但是，如果用户要求工程师编写出更安全的代码时，知道工程师会做什么吗？

这并不像是不言而喻的。实际上，除了偶尔的数据被破坏而导致可见的崩溃之外，大多数安全漏洞对可靠性都没有影响。更糟糕的是，不堵住这些漏洞对性能和可用性有正面的影响。实际上，世界上大多数不安全的系统都实现了它们的诺言——只要所有的事情都是按照工程师的设计来发生的。

但现实并没有这么友好，部署环境也不是工程师可以完全控制的——计算机工程师、土木工程师和机械工程师都不能。土木工程师和机械工程师都要考虑如何应对地球上对安全有直接威胁的一些意外事件。但即使是土木工程师和机械工程师所设计的系统，也很容易测试一些比较规范的问题：地震？每个人都很熟悉摇动某件东西的场景。火灾？可以划着一根火柴。水灾？把系统放在浴缸里，看看会发生什么。

安全？这需要请一位世界级的黑客来，问问他看到了什么。一般的计算机工程师更了解使办公大楼倒塌的原因，而不清楚自己编写的代码是如何被滥用的。毕竟，他的父母是告诉过他不要玩火，那么告诉过他不要玩格式字符串吗？

我们有很长的路要走。

本书非常重要的原因是，它出自两位业界经验最丰富的专家之手，工程师通过本书能理解编写安全代码的具体含义。本书反映了在代码发布后多年，Michael Howard 和 David LeBlanc 与开发人员一起奋战，告诉他们出了什么问题，并解决这些问题所积累的经验。

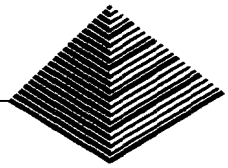
修复发布之后的代码的代价，无论怎样强调都不过分。NIST研究表明，从头开始编写新代码要比修复发布之后的代码便宜许多。修复发布之后的代码的过程很痛苦：必须切换到旧的代码基，重复所有的旧测试（记住，所有的代码仍必须执行良好、可用、稳定），再发布新代码，并确保新发布的代码部署到合适的位置上，等等。

为了使软件安全的成本可以接受，最终使软件能成功发布，我们必须从一开始就考虑到软件的安全。工程师应对安全要求做出合适的反应，这需要他们理解安全要求的含义以及满足安全要求意味着什么。本书的主要任务是讲述这些知识，将上述安全要求表达清楚，而不是让工程师“雇佣一位黑客”或者“像黑客那样想问题”。本书提供了这方面的指导。

——Dan Kaminsky

IOActive 渗透测试的主管

前　言



今天的软件工程人员必须理解建立安全软件的基本规则，这不是因为“这是一个好主意”，或者我们想卖出更多的书，而是因为互联网的本质和一小撮卑劣的家伙想控制互联网。这不是说安全是如何特殊，它其实就是可靠性的另一面。我们都想编写出可靠的软件，而不安全的软件肯定是不可靠的。

软件工程人员不可能花许多时间学习似乎没有什么回报的新规则，这就是我们编写本书的原因：读者不需要翻阅上千页来了解能应用于手头工作的某个新规则，而只需阅读可应用于当前建构的软件的一章或几章内容，以确保不会构建出不安全的软件。

本书是 *The 19 Deadly Sins of Software Security*(《一个都不能有——软件的 19 个致命安全漏洞》)的第 2 版。在开始编写本书时，我们其实并不知道读者对本书的期望。第 1 版卖得很好，大多数与我们谈论该书的人都喜欢它，这主要是因为第 1 版短小精悍、中肯、可操作性很强。与我们交谈的每个软件开发人员和设计人员都说，他们喜欢第 1 版的易于阅读，不需要学习软件安全的几乎所有内容，只要查看可应用于当前构建的软件的那几章即可。一些公司还把本书用作临时培训教材，在其员工开始设计或编写产品之前，必须阅读本书的相应章节。

类似这样的评论让我们很高兴，因为我们在开始编写第 1 版时，就希望它简明扼要、语言流畅、可操作性强。

但第一版是 4 年前编写的，而软件安全是一个不断变化的主题，不仅出现了新的漏洞类型，还出现了漏洞的变体，当然人们也想出了新的防范措施和缓解手段，以应对不断演变的各种威胁。由于安全领域的变化非常快，所以涉足软件开发的每个人都必须知道，有哪些安全问题、如何找出它们、如何解决它们。

我们第一次开始思考本书的第 2 版时，面对的问题是如何把软件安全致命漏洞的数量限制在可管理、切实有效的范围内。讨论软件界的安全问题时很容易跑题，还会描述对建立更安全的软件无关紧要的细节。这些细节可能在学术上和智力上很有刺激性，但我们只希望读者建立更安全的软件，而不是开始一次大脑探险！

如果您熟悉关系数据库，就应知道 Ted Codd 提出的 12 Rules，这 13 条法则(它们从 0 到 12 编号)定义了关系数据库。许多数据库界的人都逐字引用这 13 条法则，因为它们非常简单，能应用于他们的工作上。我们想使本书比较短小，就像 Codd 法则那样。我们可不希望把 19 个致命漏洞扩展到 100 个致命漏洞，其中包含了罕见的、奇异的、浅显的、不相

关的安全漏洞。这是一个两难问题：如何提高本书的价值，而又不使书过厚？

我们花了很长时间研究软件业在过去 4 年中的变化，最后编写出了 24 个致命漏洞。本书有许多新章节，删除了一些章节，还合并了一些章节。

我们对自己的成果很满意，认为本书反映了目前的大多数软件安全问题。我们也达到了之前的目标：短小、可操作性强、扼要。

本书读者和应读的章节

如果您是在设计、编写和测试软件，就是本书的核心读者。您不需要阅读本书的每一页，除非您喜欢这么做。

本书分为 4 个主要部分：

- Web 应用程序漏洞
- 实现漏洞
- 加密漏洞
- 联网漏洞

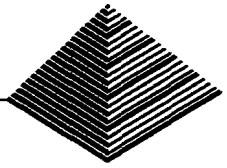
显然，如果您在建立任意类型的 Web 应用程序（客户端或服务器端），就需要阅读第 I 部分。第 II 部分最长，包含许多与语言相关的实现问题，稍后将讨论这一部分。如果应用程序涉及到加密，就应阅读第 III 部分。最后，如果应用程序进行了任意形式的网络通信，就应阅读最后一部分。

现在，看看第 II 部分的问题：

- 所有的开发人员都应阅读第 10、11、12 和 14 章；
- 需要频繁更新应用程序的开发人员应阅读第 15 章；
- 如果使用支持异常的语言，就应阅读第 9 章；
- 如果应用程序是用 C 或 C++编写的，就应阅读第 5、6、7 和 8 章。

如前所述，一些开发人员把上一版看作“即时”培训素材。我们认为，本版仍扮演这一角色，尤其是使用快捷软件开发方法进行软件开发的人员。在每个快速开发工作的开始，都应先确定要构建什么功能，并确保设计人员、开发人员和测试人员阅读了相关的章节。

目　录



第Ⅰ部分　Web 应用程序漏洞

第II部分 实 现 漏 洞