SEI SERIES • A CERT® BOOK

SOFTWARE SECURITY SERIES

Software Security Engineering

A Guide for Project Managers

Julia H. Allen • Sean Barnum
Robert J. Ellison • Gary McGraw
Nancy R. Mead

Software Security Engineering

This page intentionally left blank

Software Security Engineering

A Guide for Project Managers

Julia H. Allen
Sean Barnum
Robert J. Ellison
Gary McGraw
Nancy R. Mead

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Carnegie Mellon
Software Engineering Institute
The SEI Series in Software Engineering

SOFTWARE SECURITY SERIES

The Addison-Wesley Software Security Series

Library of Congress Cataloging-in-Publication Data
Software security engineering : a guide for project managers / Julia H. Allen ... [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-50917-8 (pbk. : alk. paper) 1. Computer security. 2. Software engineering. 3. Computer networks—Security measures. I. Allen, Julia H.

QA76.9.A25S654 2008
005.8—dc22

2008007000

ISBN-13: 978-0-321-50917-8
ISBN-10: 0-321-50917-X
Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.
First printing, April 2008

Contents

This page intentionally left blank

Foreword

Everybody knows that software is riddled with security flaws. At first blush, this is surprising. We know how to write software in a way that provides a moderately high level of security and robustness. So why don't software developers practice these techniques?

This book deals with two of the myriad answers to this question. The first is the meaning of secure software. In fact, the term "secure software" is a misnomer. Security is a product of software *plus environment*. How a program is used, under what conditions it is used, and what security requirements it must meet determine whether the software is secure. A term like "security-enabled software" captures the idea that the software was designed and written to meet specific security requirements, but in other environments where the assumptions underlying the software—and any implied requirements—do not hold, the software may not be secure. In a way that is easy to understand, this book presents the need for accurate and meaningful security requirements, as well as approaches for developing them. Unlike many books on the subject of secure software, this book does not assume the requirements are given *a priori,* but instead discusses requirements derivation and analysis. Equally important, it describes their validation.

The second answer lies in the roles of the executives, managers, and technical leaders of projects. They must support the introduction of security enhancements in software, as well as robust coding practices (which is really a type of security enhancement). Moreover, they must understand the processes and make allowances for it in their scheduling, budgeting, and staffing plans. This book does an excellent job of laying out the process for the people in these roles, so they can realistically assess its impact. Additionally, the book points out where the state of the art is too new or lacks enough experience to have approaches that are proven to work, or are not generally accepted to work. In those cases, the authors suggest ways to think about the issues in order to develop effective approaches. Thus, executives, managers, and technical leaders can figure out what should work best in their environment.

An additional, and in fact crucial, benefit of designing and implementing security in software from the very beginning of the project is the increase in assurance that the software will meet its requirements. This will greatly reduce the need to patch the software to fix security holes—a process that is itself fraught with security problems, undercuts the reputation of the vendor, and adversely impacts the vendor financially. Loss of credibility, while intangible, has tangible repercussions. Paying the extra cost of developing software correctly from the start reduces the cost of fixing it after it is deployed—and produces a better, more robust, and more secure product.

This book discusses several ways to develop software in such a way that security considerations play a key role in its development. It speaks to executives, to managers at all levels, and to technical leaders, and in that way, it is unique. It also speaks to students and developers, so they can understand the process of developing software with security in mind and find resources to help them do so.

The underlying theme of this book is that the software we all use could be made much better. The information in this book provides a foundation for executives, project managers, and technical leaders to improve the software they create and to improve the quality and security of the software we all use.

Matt Bishop
Davis, California
March 2008

Preface

The Problem Addressed by This Book

Software is ubiquitous. Many of the products, services, and processes that organizations use and offer are highly dependent on software to handle the sensitive and high-value data on which people's privacy, livelihoods, and very lives depend. For instance, national security—and by extension citizens' personal safety—relies on increasingly complex, interconnected, software-intensive information systems that, in many cases, use the Internet or Internet-exposed private networks as their means for communication and transporting data.

This ubiquitous dependence on information technology makes software security a key element of business continuity, disaster recovery, incident response, and national security. Software vulnerabilities can jeopardize intellectual property, consumer trust, business operations and services, and a broad spectrum of critical applications and infrastructures, including everything from process control systems to commercial application products.

The integrity of critical digital assets (systems, networks, applications, and information) depends on the reliability and security of the software that enables and controls those assets. However, business leaders and informed consumers have growing concerns about the scarcity of practitioners with requisite competencies to address software security [Carey 2006]. Specifically, they have doubts about suppliers' capabilities to build and deliver secure software that they can use with confidence and without fear of compromise. Application software is the primary gateway to sensitive information. According to a Deloitte survey of 169 major global financial institutions, titled *2007 Global Security Survey: The Shifting Security Paradigm* [Deloitte 2007], current application software countermeasures are no longer adequate. In the survey, Gartner identifies application security as the number one issue for chief information officers (CIOs).

Selected content in this preface is summarized and excerpted from *Security in the Software Lifecycle: Making Software Development Processes—and Software Produced by Them—More Secure* [Goertzel 2006].

The absence of security discipline in today's software development practices often produces software with exploitable weaknesses. Security-enhanced processes and practices—and the skilled people to manage them and perform them—are required to build software that can be trusted to operate more securely than software being used today.

That said, there is an economic counter-argument, or at least the perception of one: Some business leaders and project managers believe that developing secure software slows the software development process and adds to the cost while not offering any apparent advantage. In many cases, when the decision reduces to "ship now" or "be secure and ship later," "ship now" is almost always the choice made by those who control the money but have no idea of the risks. The opposite side of this argument, including how software security can potentially reduce cost and schedule, is discussed in Chapter 1 (Section 1.6, "The Benefits of Detecting Software Security Defects Early") and Chapter 7 (Section 7.5.3, in the "Knowledge and Expertise" subsection discussing Microsoft's experience with its Security Development Lifecycle) in this book.

Software's Vulnerability to Attack

The number of threats specifically targeting software is increasing, and the majority of network- and system-level attacks now exploit vulnerabilities in application-level software. According to CERT analysts at Carnegie Mellon University,[1] most successful attacks result from targeting and exploiting known, unpatched software vulnerabilities and insecure software configurations, a significant number of which are introduced during software design and development.

These conditions contribute to the increased risks associated with software-enabled capabilities and exacerbate the threat of attack. Given this atmosphere of uncertainty, a broad range of stakeholders need justifiable confidence that the software that enables their core business operations can be trusted to perform as intended.

Why We Wrote This Book: Its Purpose, Goals, and Scope

The Challenge of Software Security Engineering

Software security engineering entails using practices, processes, tools, and techniques to address security issues in every phase of the software

1. CERT (www.cert.org) is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

development life cycle (SDLC). Software that is developed with security in mind is typically more resistant to both intentional attack and unintentional failures. One view of secure software is software that is engineered "so that it continues to function correctly under malicious attack" [McGraw 2006] and is able to recognize, resist, tolerate, and recover from events that intentionally threaten its dependability. Broader views that can overlap with software security (for example, software safety, reliability, and fault tolerance) include the notion of proper functioning in the face of unintentional failures or accidents and inadvertent misuse and abuse, as well as reducing software defects and weaknesses to the greatest extent possible regardless of their cause. This book addresses the narrower view.

The goal of software security engineering is to build better, defect-free software. Software-intensive systems that are constructed using more securely developed software are better able to do the following:

- Continue operating correctly in the presence of most attacks by either *resisting* the exploitation of weaknesses in the software by attackers or *tolerating* the failures that result from such exploits
- Limit the damage resulting from any failures caused by attack-triggered faults that the software was unable to resist or tolerate and recover as quickly as possible from those failures

No single practice offers a universal "silver bullet" for software security. With this caveat in mind, *Software Security Engineering: A Guide for Project Managers* provides software project managers with sound practices that they can evaluate and selectively adopt to help reshape their own development practices. The objective is to increase the security and dependability of the software produced by these practices, both during its development and during its operation.

What Readers Can Expect

Readers will increase their awareness and understanding of the security issues in the design and development of software. The book's content will help readers recognize how software development practices can either contribute to or detract from the security of software.

The book (and material referenced on the Build Security In Web site described later in this preface) will enable readers to identify and compare potential new practices that can be adapted to augment a project's current software development practices, thereby greatly increasing the likelihood of producing more secure software and meeting specified security requirements. As one example, assurance cases can be used to assert and specify desired security properties, including the extent to which security practices have been successful in satisfying security requirements. Assurance cases are discussed in Chapter 2 (Section 2.4, "How to Assert and Specify Desired Security Properties").

Software developed and assembled using the practices described in this book should contain significantly fewer exploitable weaknesses. Such software can then be relied on to more capably resist or tolerate and recover from attacks and, therefore, to function more securely in an operational environment. Project managers responsible for ensuring that software and systems adequately address their security requirements throughout the SDLC should review, select, and tailor guidance from this book, the Build Security In Web site, and the sources cited throughout this book as part of their normal project management activities.

The five key take-away messages for readers of this book are as follows:

1. Software security is about more than eliminating vulnerabilities and conducting penetration tests. Project managers need to take a systematic approach to incorporate the sound practices discussed in this book into their development processes (all chapters).
2. Network security mechanisms and IT infrastructure security services do not sufficiently protect application software from security risks (Chapters 1 and 2).
3. Software security initiatives should follow a risk management approach to identify priorities and determine what is "good enough," while understanding that software security risks will inevitably change throughout the SDLC (Chapters 1, 4, and 7).
4. Developing secure software depends on understanding the operational context in which it will be used (Chapter 6).
5. Project managers and software engineers need to learn to think like an attacker to address the range of things that software should *not* do and identify how software can better resist, tolerate, and recover when under attack (Chapters 2, 3, 4, and 5).

Who Should Read This Book

Software Security Engineering: A Guide for Project Managers is primarily intended for project managers who are responsible for software development and the development of software-intensive systems. Lead requirements analysts, experienced software and security architects and designers, system integrators, and their managers should also find this book useful. It provides guidance for those involved in the management of secure, software-intensive systems, either developed from scratch or through the assembly, integration, and evolution of acquired or reused software.

This book will help readers understand the security issues associated with the engineering of software and should help them identify practices that can be used to manage and develop software that is better able to withstand the threats to which it is increasingly subjected. It presumes that readers are familiar with good general systems and software engineering management methods, practices, and technologies.

How This Book Is Organized

This book is organized into two introductory chapters, four technical chapters, a chapter that describes governance and management considerations, and a concluding chapter on how to get started.

Chapter 1, Why Is Security a Software Issue?, identifies threats that target most software and the shortcomings of the software development process that can render software vulnerable to those threats. It describes the benefits of detecting software security defects early in the SDLC, including the current state of the practice for making the business case for software security. It closes by introducing some pragmatic solutions that are further elaborated in the chapters that follow.

Chapter 2, What Makes Software Secure?, examines the core and influential properties of software that make it secure and the defensive and attacker perspectives in addressing those properties, and discusses how desirable traits of software can contribute to its security. The chapter introduces and defines the key resources of attack patterns and assurance cases and explains how to use them throughout the SDLC.

Chapter 3, Requirements Engineering for Secure Software, describes practices for security requirements engineering, including processes that are specific to eliciting, specifying, analyzing, and validating security requirements. This chapter also explores the key practice of misuse/abuse cases.

Chapter 4, Secure Software Architecture and Design, presents the practice of architectural and risk analysis for reviewing, assessing, and validating the specification, architecture, and design of a software system with respect to software security, and reliability.

Chapter 5, Considerations for Secure Coding and Testing, summarizes key practices for performing code analysis to uncover errors in and improve the quality of source code, as well as practices for security testing, white-box testing, black-box testing, and penetration testing. Along the way, this chapter references recently published works on secure coding and testing for further details.

Chapter 6, Security and Complexity: System Assembly Challenges, describes the challenges and practices inherent in the design, assembly, integration, and evolution of trustworthy systems and systems of systems. It provides guidelines for project managers to consider, recognizing that most new or updated software components are typically integrated into an existing operational environment.

Chapter 7, Governance, and Managing for More Secure Software, describes how to motivate business leaders to treat software security as a governance and management concern. It includes actionable practices for risk management and project management and for establishing an enterprise security framework.

Chapter 8, Getting Started, summarizes all of the recommended practices discussed in the book and provides several aids for determining which practices are most relevant and for whom, and where to start.

The book closes with a comprehensive bibliography and glossary.

Notes to the Reader

Navigating the Book's Content

As an aid to the reader, we have added descriptive icons that mark the book's sections and key practices in two practical ways:

- Identifying the content's relative "maturity of practice":

  L1 The content provides guidance for how to think about a topic for which there is no proven or widely accepted approach. The intent of the description is to raise awareness and aid the reader in thinking about the problem and candidate solutions. The content may also describe promising research results that may have been demonstrated in a constrained setting.

L2 The content describes practices that are in early (pilot) use and are demonstrating some successful results.

L3 The content describes practices that are in limited use in industry or government organizations, perhaps for a particular market sector.

L4 The content describes practices that have been successfully deployed and are in widespread use. Readers can start using these practices today with confidence. Experience reports and case studies are typically available.

- Identifying the designated audiences for which each chapter section or practice is most relevant:

  E Executive and senior managers

  M Project and mid-level managers

  L Technical leaders, engineering managers, first-line managers, and supervisors

As the audience icons in the chapters show, we urge executive and senior managers to read all of Chapters 1 and 8, plus the following sections in other chapters: 2.1, 2.2, 2.5, 3.1, 3.7, 4.1, 5.1, 5.6, 6.1, 6.6, 7.1, 7.3, 7.4, 7.6, and 7.7.

Project and mid-level managers should be sure to read all of Chapters 1, 2, 4, 5, 6, 7, and 8, plus these sections in Chapter 3: 3.1, 3.3, and 3.7.

Technical leaders, engineering managers, first-line managers, and supervisors will find useful information and guidance throughout the entire book.

Build Security In: A Key Resource

Since 2004, the U.S. Department of Homeland Security Software Assurance Program has sponsored development of the Build Security In (BSI) Web site (https://buildsecurityin.us-cert.gov/), which was one of the significant resources used in writing this book. BSI content is based on the principle that software security is fundamentally a software engineering problem and must be managed in a systematic way throughout the SDLC.

BSI contains and links to a broad range of information about sound practices, tools, guidelines, rules, principles, and other knowledge to help project managers deploy software security practices and build secure and reliable software. Contributing authors to this book and the articles appearing on the BSI Web site include senior staff from the Carnegie Mellon Software Engineering Institute (SEI) and Cigital, Inc., as well as other experienced software and security professionals.

Several sections in the book were originally published as articles in *IEEE Security & Privacy* magazine and are reprinted here with the permission of IEEE Computer Society Press. Where an article occurs in the book, a statement such as the following appears in a footnote:

> This section was originally published as an article in *IEEE Security & Privacy* [citation]. It is reprinted here with permission from the publisher.

These articles are also available on the BSI Web site.

Articles on BSI are referenced throughout this book. Readers can consult BSI for additional details, book errata, and ongoing research results.

Start the Journey

A number of excellent books address secure systems and software engineering. *Software Security Engineering: A Guide for Project Managers* offers an engineering perspective that has been sorely needed in the software security community. It puts the entire SDLC in the context of an integrated set of sound software security engineering practices.

As part of its comprehensive coverage, this book captures both standard and emerging software security practices and explains why they are needed to develop more security-responsive and robust systems. The book is packed with reasons for taking action early and revisiting these actions frequently throughout the SDLC.

This is not a book for the faint of heart or the neophyte software project manager who is confronting software security for the first time. Readers need to understand the SDLC and the processes in use within their organizations to comprehend the implications of the various techniques presented and to choose among the recommended practices to determine the best fit for any given project.

Other books are available that discuss each phase of secure software engineering. Few, however, cover all of the SDLC phases in as concise and usable a format as we have attempted to do here. Enjoy the journey!

Acknowledgments

We are pleased to acknowledge the support of many people who helped us through the book development process. Our organizations, the CERT Program at the Software Engineering Institute (SEI) and Cigital, Inc., encouraged our authorship of the book and provided release time as well as other support to make it possible. Pamela Curtis, our SEI technical editor, diligently read and reread each word of the entire manuscript and provided many valuable suggestions for improvement, as well as helping with packaging questions and supervising development of figures for the book. Jan Vargas provided SEI management support, tracked schedules and action items, and helped with meeting agendas and management. In the early stages of the process, Petra Dilone provided SEI administrative support as well as configuration management for the various chapters and iterations of the manuscript.

We also appreciate the encouragement of Joe Jarzombek, the sponsor of the Department of Homeland Security Build Security In (BSI) Web site. The Build Security In Web site content is a key resource for this book.

Much of the material in this book is based on articles published with other authors on the BSI Web site and elsewhere. We greatly appreciated the opportunity to collaborate with these authors, and their names are listed in the individual sections that they contributed to, directly or indirectly.

We had many reviewers, whose input was extremely valuable and led to many improvements in the book. Internal reviewers included Carol Woody and Robert Ferguson of the SEI. We also appreciate the inputs and thoughtful comments of the Addison-Wesley reviewers: Chris Cleeland, Jeremy Epstein, Ronda R. Henning, Jeffrey A. Ingalsbe, Ron Lichty, Gabor Liptak, Donald Reifer, and David Strom. We would like to give special recognition to Steve Riley, one of the Addison-Wesley reviewers who reviewed our initial proposal and all iterations of the manuscript.

We would like to recognize the encouragement and support of our contacts at Addison-Wesley. These include Peter Gordon, publishing partner; Kim Boedigheimer, editorial assistant; Julie Nahil, full-service production manager; and Jill Hobbs, freelance copyeditor. We also appreciate the efforts of the Addison-Wesley and SEI artists and designers who assisted with the cover design, layout, and figures.

About the Authors

Julia H. Allen

Julia H. Allen is a Senior Member of the Technical Staff within the CERT Program at the Software Engineering Institute (SEI), a unit of Carnegie Mellon University in Pittsburgh, Pennsylvania. In addition to her work in software security and assurance, Allen is engaged in developing and transitioning executive outreach programs in enterprise security and governance. Prior to this technical assignment, Allen served as Acting Director of the SEI for an interim period of six months as well as Deputy Director/Chief Operating Officer for three years. She formalized the SEI's relationship with industry organizations and created the Customer Relations team.

Before joining the SEI, Allen was a Vice President at Science Applications International Corporation (SAIC), responsible for starting a new software division specializing in embedded systems software. Allen led SAIC's initial efforts in software process improvement. Allen also worked at TRW (now Northrop Grumman), tackling a range of assignments from systems integration, testing, and field site support to managing major software development programs.

Her degrees include a B.S. in Computer Science from the University of Michigan and an M.S. in Electrical Engineering from the University of Southern California. Allen is the author of *The CERT® Guide to System and Network Security Practices* (Addison-Wesley, 2001), *Governing for Enterprise Security* (CMU/SEI-2005-TN-023, 2005), and the CERT Podcast Series: Security for Business Leaders (2006–2008).

Sean Barnum

Sean Barnum is a Principal Consultant at Cigital, Inc., and is Technical Lead for Cigital's federal services practice. He has more than twenty years of experience in the software industry in the areas of development, software quality assurance, quality management, process architecture and improvement, knowledge management, and security. Barnum is a frequent contributor and speaker for regional and national software security and software quality publications, conferences, and events. He is very active in the software assurance community and is involved in numerous knowledge standards-defining efforts, including the Common Weakness Enumeration (CWE), the Common Attack Pattern Enumeration and Classification (CAPEC), and other elements of the Software Assurance Programs of the Department of Homeland Security and the Department of Defense. He is also the lead technical subject matter expert for the Air Force Application Software Assurance Center of Excellence.

Robert J. Ellison

As a member of the Survivable Systems Engineering Team within the CERT Program at the Software Engineering Institute, Robert J. Ellison has served in a number of technical and management roles. He was a project leader for the evaluation of software engineering development environments and associated software development tools. He was also a member of the Carnegie Mellon University team that wrote the proposal for the SEI; he joined the new FFRDC in 1985 as a founding member.

Before coming to Carnegie Mellon, Ellison taught mathematics at Brown University, Williams College, and Hamilton College. At Hamilton College, he directed the creation of the computer science curriculum. Ellison belongs to the Association for Computing Machinery (ACM) and the IEEE Computer Society.

Ellison regularly participates in the evaluation of software architectures and contributes from the perspective of security and reliability measures. His research draws on that experience to integrate security issues into the overall architecture design process. His current work explores developing reasoning frameworks to help architects select and refine design tactics to mitigate the impact of a class of cyberattacks. He continues to work on refinements to the Survivability Analysis Framework.

Gary McGraw

Gary McGraw is the Chief Technology Officer at Cigital, Inc., a software security and quality consulting firm with headquarters in the Washington, D.C., area. He is a globally recognized authority on software security and the author of six best-selling books on this topic. The latest book is *Exploiting Online Games* (Addison-Wesley, 2008). His other books include *Java Security*, *Building Secure Software*, *Exploiting Software*, and *Software Security*; he is also editor of the Addison-Wesley Software Security series. Dr. McGraw has written more than ninety peer-reviewed scientific publications, authors a monthly security column for darkreading.com, and is frequently quoted in the press as an expert on software security.

Besides serving as a strategic counselor for top business and IT executives, Dr. McGraw is on the advisory boards of Fortify Software and Raven White. He received a dual Ph.D. in Cognitive Science and Computer Science from Indiana University, where he serves on the Dean's Advisory Council for the School of Informatics. Dr. McGraw is also a member of the IEEE Computer Society Board of Governors and produces the monthly Silver Bullet Security Podcast for *IEEE Security & Privacy* magazine.

Nancy R. Mead

Nancy R. Mead is a Senior Member of the Technical Staff in the Survivable Systems Engineering Group, which is part of the CERT Program at the Software Engineering Institute. She is also a faculty member in the Master of Software Engineering and Master of Information Systems Management programs at Carnegie Mellon University. Her research interests are in the areas of information security, software requirements engineering, and software architectures.

Prior to joining the SEI, Mead was Senior Technical Staff Member at IBM Federal Systems, where she spent most of her career in the development and management of large real-time systems. She also worked in IBM's software engineering technology area and managed IBM Federal Systems' software engineering education department. She has developed and taught numerous courses on software engineering topics, both at universities and in professional education courses.

To date, Mead has more than one hundred publications and invited presentations. She is a fellow of the Institute of Electrical and Electronic Engineers (IEEE) and the IEEE Computer Society, and is also a member of the Association for Computing Machinery (ACM). Mead received her Ph.D. in Mathematics from the Polytechnic Institute of New York and received a B.A. and an M.S. in Mathematics from New York University.

This page intentionally left blank

Chapter 1

Why Is Security a Software Issue?

1.1 Introduction

Software is everywhere. It runs your car. It controls your cell phone. It's how you access your bank's financial services; how you receive electricity, water, and natural gas; and how you fly from coast to coast [McGraw 2006]. Whether we recognize it or not, we all rely on complex, interconnected, software-intensive information systems that use the Internet as their means for communicating and transporting information.

Building, deploying, operating, and using software that has not been developed with security in mind can be high risk—like walking a high wire without a net (Figure 1–1). The degree of risk can be compared to the distance you can fall and the potential impact (no pun intended).

This chapter discusses why security is increasingly a software problem. It defines the dimensions of software assurance and software security. It identifies threats that target most software and the shortcomings of the

Selected content in this chapter is summarized and excerpted from *Security in the Software Lifecycle: Making Software Development Processes—and Software Produced by Them—More Secure* [Goertzel 2006]. An earlier version of this material appeared in [Allen 2007].

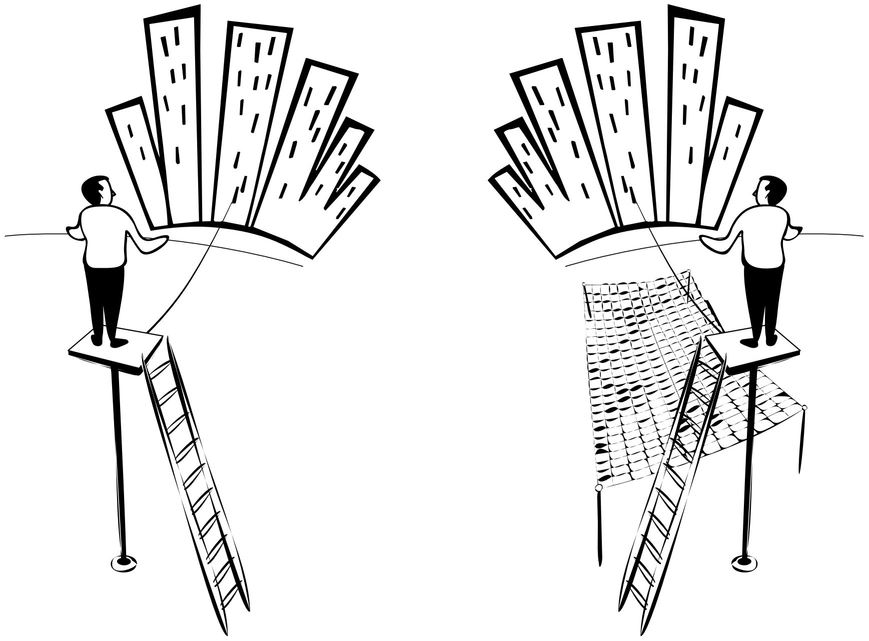

Figure 1–1: *Developing software without security in mind is like walking a high wire without a net*

software development process that can render software vulnerable to those threats. It closes by introducing some pragmatic solutions that are expanded in the chapters to follow. This entire chapter is relevant for executives (E), project managers (M), and technical leaders (L).

1.2 The Problem

Organizations increasingly store, process, and transmit their most sensitive information using software-intensive systems that are directly connected to the Internet. Private citizens' financial transactions are exposed via the Internet by software used to shop, bank, pay taxes, buy insurance, invest, register children for school, and join various organizations and social networks. The increased exposure that comes with global connectivity has made sensitive information and the software systems that handle it more vulnerable to unintentional and unauthorized use. In short, software-intensive systems and other software-enabled capabilities have provided more open, widespread access to sensitive information—including personal identities—than ever before.

Concurrently, the era of information warfare [Denning 1998], cyberterrorism, and computer crime is well under way. Terrorists, organized crime, and other criminals are targeting the entire gamut of software-intensive systems and, through human ingenuity gone awry, are being successful at gaining entry to these systems. Most such systems are not attack resistant or attack resilient enough to withstand them.

In a report to the U.S. president titled *Cyber Security: A Crisis of Prioritization* [PITAC 2005], the President's Information Technology Advisory Committee summed up the problem of nonsecure software as follows:

> Software development is not yet a science or a rigorous discipline, and the development process by and large is not controlled to minimize the vulnerabilities that attackers exploit. Today, as with cancer, vulnerable software can be invaded and modified to cause damage to previously healthy software, and infected software can replicate itself and be carried across networks to cause damage in other systems. Like cancer, these damaging processes may be invisible to the lay person even though experts recognize that their threat is growing.

Software defects with security ramifications—including coding bugs such as buffer overflows and design flaws such as inconsistent error handling—are ubiquitous. Malicious intruders, and the malicious code and botnets[1] they use to obtain unauthorized access and launch attacks, can compromise systems by taking advantage of software defects. Internet-enabled software applications are a commonly exploited target, with software's increasing complexity and extensibility making software security even more challenging [Hoglund 2004].

The security of computer systems and networks has become increasingly limited by the quality and security of their software. Security defects and vulnerabilities in software are commonplace and can pose serious risks when exploited by malicious attacks. Over the past six years, this problem has grown significantly. Figure 1–2 shows the number of vulnerabilities reported to CERT from 1997 through 2006. Given this trend, "[T]here is a clear and pressing need to change the

1. http://en.wikipedia.org/wiki/Botnet

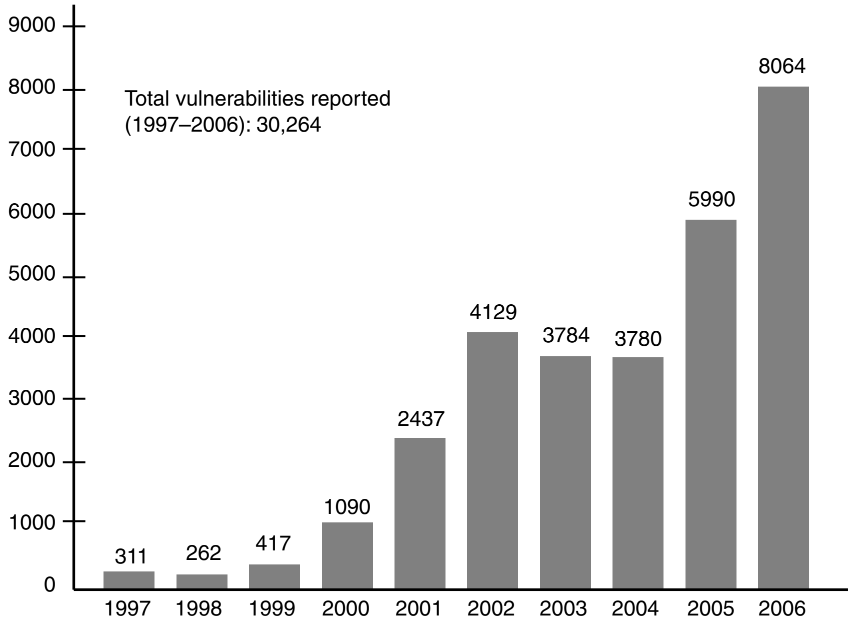

Figure 1–2: *Vulnerabilities reported to CERT*

way we (project managers and software engineers) approach computer security and to develop a disciplined approach to software security" [McGraw 2006].

In Deloitte's *2007 Global Security Survey*, 87 percent of survey respondents cited poor software development quality as a top threat in the next 12 months. "Application security means ensuring that there is secure code, integrated at the development stage, to prevent potential vulnerabilities and that steps such as vulnerability testing, application scanning, and penetration testing are part of an organization's software development life cycle [SDLC]" [Deloitte 2007].

The growing Internet connectivity of computers and networks and the corresponding user dependence on network-enabled services (such as email and Web-based transactions) have increased the number and sophistication of attack methods, as well as the ease with which an attack can be launched. This trend puts software at greater risk. Another risk area affecting software security is the degree to which systems accept updates and extensions for evolving capabilities. Extensible systems are attractive because they provide for the addition of new features and services, but each new extension adds new capabilities, new interfaces, and thus new risks. A final software security risk area is the unbridled growth in the size and complexity of software systems (such as the Microsoft Windows operating system). The unfortunate reality is that in general more lines of code produce more bugs and vulnerabilities [McGraw 2006].

1.2.1 System Complexity: The Context within Which Software Lives

Building a trustworthy software system can no longer be predicated on constructing and assembling discrete, isolated pieces that address static requirements within planned cost and schedule. Each new or updated software component joins an existing operational environment and must merge with that legacy to form an operational whole. Bolting new systems onto old systems and Web-enabling old systems creates systems of systems that are fraught with vulnerabilities. With the expanding scope and scale of systems, project managers need to reconsider a number of development assumptions that are generally applied to software security:

- Instead of centralized control, which was the norm for large stand-alone systems, project managers have to consider multiple and often independent control points for systems and systems of systems.
- Increased integration among systems has reduced the capability to make wide-scale changes quickly. In addition, for independently managed systems, upgrades are not necessarily synchronized. Project managers need to maintain operational capabilities with appropriate security as services are upgraded and new services are added.
- With the integration among independently developed and operated systems, project managers have to contend with a heterogeneous collection of components, multiple implementations of common interfaces, and inconsistencies among security policies.
- With the mismatches and errors introduced by independently developed and managed systems, failure in some form is more likely to be the norm than the exception and so further complicates meeting security requirements.

There are no known solutions for ensuring a specified level or degree of software security for complex systems and systems of systems, assuming these could even be defined. This said, Chapter 6, Security and Complexity: System Assembly Challenges, elaborates on these points and provides useful guidelines for project managers to consider in addressing the implications.

1.3 Software Assurance and Software Security

The increasing dependence on software to get critical jobs done means that software's value no longer lies solely in its ability to enhance or sustain productivity and efficiency. Instead, its value also derives from its ability to continue operating dependably even in the face of events that threaten it. The ability to trust that software will remain dependable under all circumstances, with a justified level of confidence, is the objective of software assurance.

Software assurance has become critical because dramatic increases in business and mission risks are now known to be attributable to exploitable software [DHS 2003]. The growing extent of the resulting risk exposure is rarely understood, as evidenced by these facts:

- Software is the weakest link in the successful execution of interdependent systems and software applications.
- Software size and complexity obscure intent and preclude exhaustive testing.
- Outsourcing and the use of unvetted software supply-chain components increase risk exposure.
- The sophistication and increasingly more stealthy nature of attacks facilitates exploitation.
- Reuse of legacy software with other applications introduces unintended consequences, increasing the number of vulnerable targets.
- Business leaders are unwilling to make risk-appropriate investments in software security.

According to the U.S. Committee on National Security Systems' "National Information Assurance (IA) Glossary" [CNSS 2006], software assurance is

> the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally

inserted at any time during its life cycle, and that the software functions in the intended manner.

Software assurance includes the disciplines of software reliability[2] (also known as software fault tolerance), software safety,[3] and software security. The focus of *Software Security Engineering: A Guide for Project Managers* is on the third of these, software security, which is the ability of software to resist, tolerate, and recover from events that intentionally threaten its dependability. The main objective of software security is to build more-robust, higher-quality, defect-free software that continues to function correctly under malicious attack [McGraw 2006].

Software security matters because so many critical functions are completely dependent on software. This makes software a very high-value target for attackers, whose motives may be malicious, criminal, adversarial, competitive, or terrorist in nature. What makes it so easy for attackers to target software is the virtually guaranteed presence of known vulnerabilities with known attack methods, which can be exploited to violate one or more of the software's security properties or to force the software into an insecure state. Secure software remains dependable (i.e., correct and predictable) despite intentional efforts to compromise that dependability.

The objective of software security is to field software-based systems that satisfy the following criteria:

- The system is as vulnerability and defect free as possible.
- The system limits the damage resulting from any failures caused by attack-triggered faults, ensuring that the effects of any attack are not propagated, and it recovers as quickly as possible from those failures.
- The system continues operating correctly in the presence of most attacks by either *resisting* the exploitation of weaknesses in the software by the attacker or *tolerating* the failures that result from such exploits.

2. Software reliability means the probability of failure-free (or otherwise satisfactory) software operation for a specified/expected period/interval of time, or for a specified/expected number of operations, in a specified/expected environment under specified/expected operating conditions. Sources for this definition can be found in [Goertzel 2006], appendix A.1.

3. Software safety means the persistence of dependability in the face of accidents or mishaps—that is, unplanned events that result in death, injury, illness, damage to or loss of property, or environmental harm. Sources for this definition can be found in [Goertzel 2006], appendix A.1.

Software that has been developed with security in mind generally reflects the following properties throughout its development life cycle:

- *Predictable execution.* There is justifiable confidence that the software, when executed, functions as intended. The ability of malicious input to alter the execution or outcome in a way favorable to the attacker is significantly reduced or eliminated.
- *Trustworthiness.* The number of exploitable vulnerabilities is intentionally minimized to the greatest extent possible. The goal is no exploitable vulnerabilities.
- *Conformance.* Planned, systematic, and multidisciplinary activities ensure that software components, products, and systems conform to requirements and applicable standards and procedures for specified uses.

These objectives and properties must be interpreted and constrained based on the practical realities that you face, such as what constitutes an adequate level of security, what is most critical to address, and which actions fit within the project's cost and schedule. These are risk management decisions.

In addition to predictable execution, trustworthiness, and conformance, secure software and systems should be as attack resistant, attack tolerant, and attack resilient as possible. To ensure that these criteria are satisfied, software engineers should design software components and systems to recognize both legitimate inputs and known attack patterns in the data or signals they receive from external entities (humans or processes) and reflect this recognition in the developed software to the extent possible and practical.

To achieve attack resilience, a software system should be able to recover from failures that result from successful attacks by resuming operation at or above some predefined minimum acceptable level of service in a timely manner. The system must eventually recover full service at the specified level of performance. These qualities and properties, as well as attack patterns, are described in more detail in Chapter 2, What Makes Software Secure?

1.3.1 The Role of Processes and Practices in Software Security

A number of factors influence how likely software is to be secure. For instance, software vulnerabilities can originate in the processes and practices used in its creation. These sources include the decisions made by software engineers, the flaws they introduce in specification and design, and the faults and other defects they include in developed code, inadvertently or intentionally. Other factors may include the choice of programming languages and development tools used to develop the software, and the configuration and behavior of software components in their development and operational environments. It is increasingly observed, however, that *the most critical difference between secure software and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software* [Goertzel 2006].

The return on investment when security analysis and secure engineering practices are introduced early in the development cycle ranges from 12 percent to 21 percent, with the highest rate of return occurring when the analysis is performed during application design [Berinato 2002; Soo Hoo 2001]. This return on investment occurs because there are fewer security defects in the released product and hence reduced labor costs for fixing defects that are discovered later.

A project that adopts a security-enhanced software development process is adopting a set of practices (such as those described in this book's chapters) that initially should reduce the number of exploitable faults and weaknesses. Over time, as these practices become more codified, they should decrease the likelihood that such vulnerabilities are introduced into the software in the first place. More and more, research results and real-world experiences indicate that *correcting potential vulnerabilities as early as possible in the software development life cycle, mainly through the adoption of security-enhanced processes and practices, is far more cost-effective* than the currently pervasive approach of developing and releasing frequent patches to operational software [Goertzel 2006].

1.4 Threats to Software Security

In information security, the threat—the source of danger—is often a person intending to do harm, using one or more malicious software agents. Software is subject to two general categories of threats:

- *Threats during development* (mainly insider threats). A software engineer can sabotage the software at any point in its development life cycle through intentional exclusions from, inclusions in, or modifications of the requirements specification, the threat models, the design documents, the source code, the assembly and integration framework, the test cases and test results, or the installation and configuration instructions and tools. The secure development practices described in this book are, in part, designed to help reduce the exposure of software to insider threats during its development process. For more information on this aspect, see "Insider Threats in the SDLC" [Cappelli 2006].

- *Threats during operation* (both insider and external threats). Any software system that runs on a network-connected platform is likely to have its vulnerabilities exposed to attackers during its operation. Attacks may take advantage of publicly known but unpatched vulnerabilities, leading to memory corruption, execution of arbitrary exploit scripts, remote code execution, and buffer overflows. Software flaws can be exploited to install spyware, adware, and other malware on users' systems that can lie dormant until it is triggered to execute.[4]

Weaknesses that are most likely to be targeted are those found in the software components' external interfaces, because those interfaces provide the attacker with a direct communication path to the software's vulnerabilities. A number of well-known attacks target software that incorporates interfaces, protocols, design features, or development faults that are well understood and widely publicized as harboring inherent weaknesses. That software includes Web applications (including browser and server components), Web services, database management systems, and operating systems. Misuse (or abuse) cases can help project managers and software engineers see their software from the perspective of an attacker by anticipating and defining unexpected or abnormal behavior through which a software feature could be unintentionally misused or intentionally abused [Hope 2004]. (See Section 3.2.)

Today, most project and IT managers responsible for system operation respond to the increasing number of Internet-based attacks by relying on operational controls at the operating system, network, and database or Web server levels while failing to directly address the insecurity

4. See the Common Weakness Enumeration [CWE 2007], for additional examples.