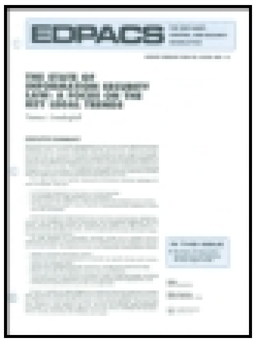


Why is Security a Software Issue?

Julia Allen

EDPACS
THE EDP AUDIT, CONTROL, AND SECURITY NEWSLETTER

JULY 2007 VOL. XXXVI, NO. 1

WHY IS SECURITY A SOFTWARE ISSUE?[1]

Julia Allen

INTRODUCTION

Software is everywhere. It runs your car. It controls your cell phone. It's how you access your bank's financial services, receive electricity and natural gas, and fly from coast to coast (McGraw, 2006). Many products, services, and capabilities within both the public and private sectors are highly dependent on software to handle the sensitive and high-value data on which people's privacy, livelihoods, health, and very lives depend. National security—and by extension citizens' personal safety—relies on complex, interconnected, software-intensive information systems—systems that in many cases use the uncontrolled Internet or Internet-exposed private networks as their means for communicating and transporting information.

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

This article 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 software development process that can render software vulnerable to those threats. It closes by introducing some pragmatic solutions.

Organizations rely heavily on application and IT software to conduct business. Much of today's operational software is vulnerable to attack, given it has not been developed with security in mind. Incorporating security practices during development reduces cost (associated with patching and incident response), reduces operational risk, enhances operational continuity and resiliency, and may assist in meeting compliance requirements that rely on software and IT.

THE PROBLEM

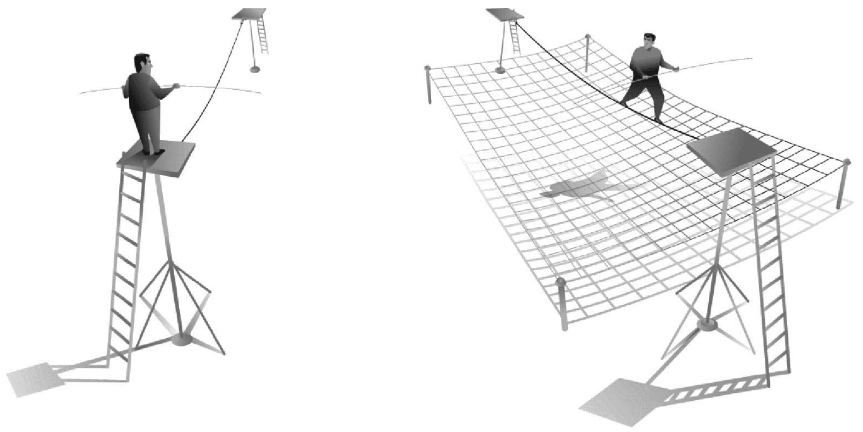
Organizations often store, process, and transmit their most sensitive information using software-intensive systems that are directly connected to the Internet. These systems are increasingly being exposed through the use of, for example, Web services that enable this sensitive information to be accessed and manipulated

Editor
DAN SWANSON

Editor Emeritus
BELDEN MENKUS, CISA

Taylor & Francis
Taylor & Francis Group

Figure 1 ***Building software without security in mind is like walking a high wire without a net***



ORGANIZATIONS RELY HEAVILY ON APPLICATION AND IT SOFTWARE TO CONDUCT BUSINESS.

by other Web services, which are themselves software systems—all without human intervention. This increased exposure has made sensitive business information and the software systems that handle it more visible.

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. In short, software-intensive systems and other software-enabled capabilities have provided open, widespread access as never before.

The era of information warfare (Denning 1998), cyberterrorism, and computer crime is well underway. Terrorists, organized crime, and other criminals are all targeting the entire gamut of software-intensive systems. Successful attacks on software systems result from human ingenuity. And by and large, these systems are not attack-resistant or attack-resilient enough to prevent the more determined efforts from succeeding.

In their report to the United States president titled *Cyber Security: A Crisis of Prioritization* (PITAC, 2005), the President's Information Technology Advisory Committee summed up the problem of non-secure 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

If you have information of interest to *EDPACS*, contact Dan Swanson (dswanson_2005@yahoo.com). *EDPACS* (Print ISSN 0736-6981/Online ISSN 1936-1009) is published monthly by Taylor & Francis Group, LLC., 325 Chestnut Street, Suite 800, Philadelphia, PA 19106. Periodicals postage is paid at Philadelphia, PA and additional mailing offices. The subscription rate is $264/year in the U.S. Prices elsewhere vary. Printed in USA. Copyright 2007. *EDPACS* is a registered trademark owned by Taylor & Francis Group, LLC. All rights reserved. No part of this newsletter may be reproduced in any form — by microfilm, xerography, or otherwise — or incorporated into any information retrieval system without the written permission of the copyright owner. Requests to publish material or to incorporate material into computerized databases or any other electronic form, or for other than individual or internal distribution, should be addressed to Editorial Services, 325 Chestnut Street, Suite 800, Philadelphia, PA 19106. All rights, including translation into other languages, reserved by the publisher in the U.S., Great Britain, Mexico, and all countries participating in the International Copyright Convention and the Pan American Copyright Convention. Authorization to photocopy items for internal or personal use, or the personal or internal use of specific clients may be granted by Taylor & Francis, provided that $20.00 per article photocopied is paid directly to Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923 USA. The fee code for users of the Transactional Reporting Service is ISSN 0736-6981/06/$20.00+$0.00. The fee is subject to change without notice. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged. Product or corporate names may be trademarks or registered trademarks, and are only used for identification and explanation, without intent to infringe. POSTMASTER: Send address change to *EDPACS*, Taylor & Francis Group, LLC., 325 Chestnut Street, Suite 800, Philadelphia, PA 19106

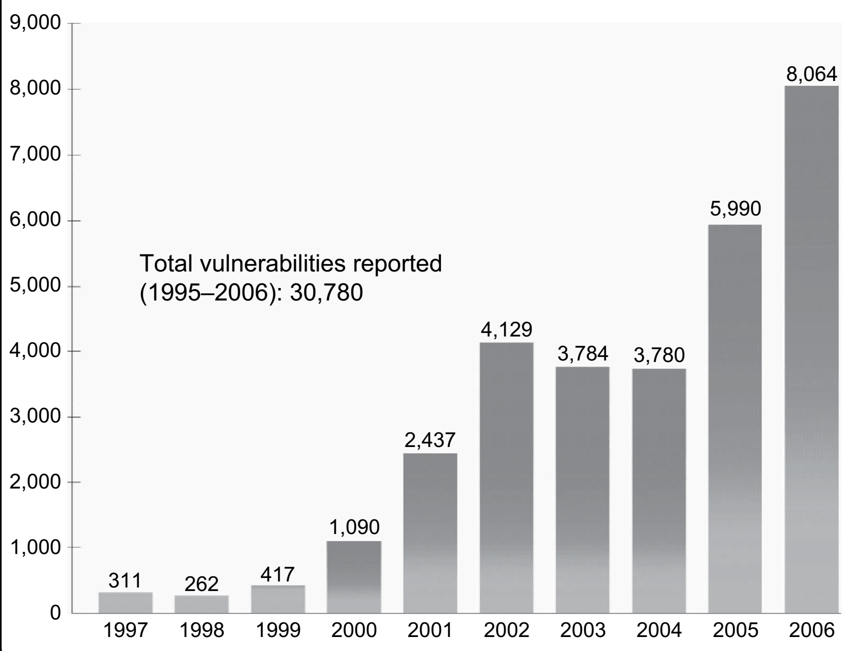
attackers exploit. Today, as with cancer, vulnerable software can be invaded and modified to cause damage to previously healthy software, and infected software can replicate itself and be carried across networks to cause damage in other systems. Like cancer, these damaging processes may be invisible 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[2] they use to obtain unauthorized access and launch attacks, can compromise systems by exploiting software defects. Internet-enabled software applications are a commonly exploited target, with software's increasing complexity and extensibility making this even more challenging (McGraw, 2006).

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 common. Over the past six years, the problem has grown significantly. Figure 2 shows the number of software vulnerabilities reported to CERT®[3] from 1995 through 2006. "There is a clear and pressing need to change the way we approach computer security and to develop a disciplined approach to software security" (McGraw, 2006, p. 4).

The growing Internet connectivity of computers and networks and the corresponding user dependence on network-enabled

Figure 2 *Software vulnerabilities reported to CERT*



services (such as e-mail and Web-based transactions) have increased the number of attack vectors and the ease with which an attack can be made. This puts software at greater risk. Another risk area affecting software security is the degree to which systems accept updates and extensions so that the capabilities that the system offers can evolve incrementally. Extensible systems are attractive because they provide for the addition of new features and services. 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).

SYSTEM COMPLEXITY—THE CONTEXT WITHIN WHICH SOFTWARE LIVES

Building a trustworthy software system can no longer be predicated on constructing 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. Today's technology must support an operating environment that is driven by business goals and organizational needs. The operating environment can be geographically and managerially distributed and constantly changing. Few businesses can stop to make changes and then restart.

With the expanding scope and scale of systems, we need to reconsider the development assumptions that are generally applied to software security. A number of trends influence how we address 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 likely more the norm than the exception and hence further complicates meeting security requirements.

There are no known solutions for ensuring a specified level of software security for complex systems and systems of systems.

SOFTWARE ASSURANCE AND SOFTWARE SECURITY

This 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 but also in 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. The growing extent of the resulting risk exposure is rarely understood, as evidenced by these facts:

1. System interdependence and software dependence have software as the weakest link.
2. Software size and complexity obscure intent and preclude exhaustive test.
3. Outsourcing and the use of unvetted software supply-chain components increase risk exposure.
4. Attack sophistication, and its increasingly more stealthy nature, eases exploitation.
5. Reuse of legacy software with other applications introduces unintended consequences, increasing the number of vulnerable targets.

According to the U.S. Committee on National Security Systems' "National Information Assurance (IA) Glossary" (CNSS, 2006, p. 56) 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 (also known as software fault tolerance), software safety, and software security. The focus of this article is on the third of these: software security, which is the ability of software to recognize, resist, tolerate, and recover from events that intentionally threaten its dependability. The main objective of software security is to build better, defect-free software so that it 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 vulnerabilities, 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 cannot be intentionally forced to fail. It is, in short, software that remains dependable (i.e., correct and predictable) in spite of intentional efforts to compromise that dependability.

The objective of software security is to field software-based systems that

- are as vulnerability- and defect-free as possible, including minimizing the introduction of unintended vulnerabilities
- limit the damage resulting from any failures caused by attack-triggered faults, ensuring that the effects of any attack are not propagated, and recover as quickly as possible from those failures
- continue 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

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 software, when executed, functions as intended.
- Trustworthiness: The number of exploitable vulnerabilities is intentionally minimized (to the greatest extent possible). The goal is no exploitable vulnerabilities.
- Conformance: There are planned, systematic, and multidisciplinary activities that 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 a project manager faces, such as what constitutes an adequate level of security, what is most critical to address, and what 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 strive to be as attack resistant, attack tolerant, and attack resilient as possible. Software components and systems need to be designed to recognize known attack patterns (Hoglund & McGraw, 2004) in the input 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.

A number of factors can influence how likely software is to be secure. Software vulnerabilities originate in the process and practices used by project managers to create that software. These include the decisions made by software engineers, the flaws they introduce in its specification and design, and the faults and other defects they include in its 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. But it is increasingly observed 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*.

A project that adopts a security-enhanced software development process is adopting a set of practices that should initially reduce the number of exploitable faults and weaknesses. One suggested set of practices is described on the U.S. Department of Homeland Security's Build Security In website.[4] 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.

THREATS TO SOFTWARE SECURITY

Threat refers to the source of danger; in the case of information security, the source of danger can be a person intending to do harm in combination with one or more malicious software agents that carry out the attacker's intent (McGraw, 2006). Software is subject to two general categories of threats:

1. 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. Secure development practices are, in part, designed to help reduce the exposure of software to insider threats during its development process.
2. 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 triggered to execute.[5]

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; such 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 & McGraw, 2004).

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 of the application-level software that is being compromised. There are two critical shortcomings with this approach:

1. The security of the application then depends completely on the robustness of operational protections that surround it.
2. Many of the software-based protection mechanisms (controls) can easily be misconfigured or misapplied. In addition, they are as likely to contain exploitable vulnerabilities as the application software they are (ostensibly) protecting.

The wide publicity about the literally thousands of successful attacks on software accessible from the Internet has only made the attacker's job easier. Attackers can study numerous reports of security vulnerabilities in a wide range of commercial and open source software programs and access publicly available exploit scripts. More experienced attackers often develop (and share) sophisticated, targeted attacks that exploit specific vulnerabilities. In addition, the nature of the threats is changing more rapidly than the software can be adapted to counteract those threats, regardless of the software development process and practices used. To be 100% effective, defenders must anticipate *all* possible vulnerabilities, while attackers need find only *one*.

SOURCES OF SOFTWARE INSECURITY

Threats to software exist whether project managers recognize them or not. Most commercial and open source applications, middleware systems, and operating systems are extremely large and complex. In normal execution, these software-intensive systems can transition through a vast number of different states. These characteristics make it particularly difficult to develop and operate software that is consistently correct, let alone consistently secure. The unavoidable presence of security threats and risks means that at least a minimal amount of software security is needed even if explicit requirements for it have not been captured in the software's specification.

A large percentage of security weaknesses in software could be avoided if project managers and software engineers were routinely trained in how to systematically and consistently address them. They are seldom taught how to design and develop secure applications and conduct quality assurance to test for insecure coding errors and the use of poor development techniques. They do not generally understand what practices are effective in recognizing and removing faults and defects or in handling vulnerabilities when software is exploited by attackers. They are often unfamiliar with the security implications of certain software requirements or their absence. Nor do they usually learn the security implications of how software is architected, designed, developed, deployed, and operated. The absence of this knowledge means that security requirements are likely inadequate and, in addition, the software is likely to deviate from specified (and unspecified) security requirements. Finally, this lack of knowledge prevents the manager and engineer from recognizing and understanding how mistakes can manifest as exploitable weaknesses and vulnerabilities in the software when it becomes operational.

Software—especially networked, application-level software—is most often compromised by exploiting weaknesses that result from

- complexities, inadequacies, and/or changes in the software's processing model (e.g., Web or service-oriented architecture model)
- incorrect engineer assumptions, including assumptions about the capabilities, outputs, and behavioral states of the software's execution environment or about expected inputs from external entities (users, software processes)
- flawed specification or design, or defective implementation of
  - the software's interfaces with external entities. Development mistakes of this type include inadequate (or non-existent) input validation, error handling, and exception handling.
  - the components of the software's execution environment (from middleware-level and operating-system-level to firmware- and hardware-level components)

Mistakes are unavoidable. Even if they are avoided during requirements engineering and design (e.g., through the use of formal methods) and development (e.g., through comprehensive code reviews and extensive testing), vulnerabilities may still be introduced into software during its assembly, integration, deployment, and operation. No matter how faithfully a security-enhanced life cycle is followed, as long as software continues to grow in size and complexity, some number of exploitable faults and other weaknesses are sure to exist.

The ability to determine whether a software fault or weakness can be exploited is an inexact science at best. Even if extreme caution is applied when developing software from scratch, most software systems are composed from a combination of acquired, reused, legacy, and newly developed components. Although it may be possible, through careful, extensive code review, to identify weaknesses in new source code and even to determine their potential exploitability, identifying comparable weaknesses and their exploitability in open source, acquired, or legacy components is for all intents and purposes impractical.

USING SECURE SOFTWARE DEVELOPMENT PRACTICES

Managers and software engineers should treat all software faults and weaknesses as potentially exploitable. Reducing exploitable weaknesses begins with the specification of software security requirements, along with considering requirements that may have been overlooked. Software that includes security requirements (such as security constraints on process behaviors and the handling of inputs, and resistance to and tolerance of intentional failures) is more likely to be engineered to remain dependable and secure in the face of an attack. In addition, exercising misuse and abuse cases that anticipate abnormal and unexpected behavior can aid project managers and software engineers in gaining a better understanding of how to create secure and reliable software.

Developing software from the beginning with security in mind is more effective by orders of magnitude than trying to validate, through testing and verification, that the software is secure. For example, attempting to demonstrate that an implemented system will never accept an unsafe input (that is, proving a negative) is impossible. You *can* prove, however, using formal methods or function abstraction, that the software you are designing will never accept an unsafe input. In addition, it is easier to design and implement the system so that input validation routines check *every* input that the software receives against a set of pre-defined constraints. Testing the input validation function to demonstrate that it is consistently invoked and correctly performed every time input enters the system is then included in the system's functional testing.

Analysis and modeling can serve to better protect your software against the more subtle, complex attack patterns involving externally forced sequences of interactions among components or processes that were never intended to interact during normal software execution. Analysis and modeling can help strengthen the security of the software's interfaces with external entities and increase its tolerance of all faults.

If your development organization's time and resource constraints prevent secure development practices from being applied to the entire software system, the software components that should be given highest priority in terms of secure engineering are

1. components that require a high degree of trust, such as components that implement security functions and other high-consequence functions. (Federal Information Processing Standard (FIPS) 199 (NIST, 2003) describes a process for categorizing a software system's impact level. This process can be adapted to prioritize the consequence of software components within a software system.)
2. interfaces between components (modules, processes, etc.) within the software system

3. interfaces between the software system and (a) databases and data storage locations that it accesses, (b) its environment, and (c) its users

A security-enhanced life-cycle process should (at least to some extent) compensate for security inadequacies in the software's requirements by adding risk-driven practices and checks for the adequacy of those practices during all software life-cycle phases.

Security controls in the software's life cycle should not be limited to the requirements, design, code, and test phases. It is important to continue performing code reviews, security tests, strict configuration control, and quality assurance during deployment and operations to ensure that updates and patches do not add security weaknesses or malicious logic to production software.[6]

SUMMARY

It is a fact of life that software faults, defects, and other weaknesses affect the ability of software to function securely. These vulnerabilities can be exploited to violate software's security properties and force the software into an insecure, exploitable state. Dealing with this is a particularly daunting challenge given the ubiquitous connectivity and explosive growth and complexity of software-based systems.

The goals of using secure software practices are as follows:

- Exploitable faults and other weaknesses are eliminated to the greatest extent possible by well-intentioned engineers.
- The likelihood is greatly reduced or eliminated that malicious engineers can intentionally implant exploitable faults and weaknesses or malicious logic or well-meaning backdoors into the software.
- The software is attack resistant, attack tolerant, and attack resilient to the extent possible and practical in support of fulfilling the organization's mission

Project managers responsible for ensuring that software and systems meet their security requirements throughout the development life cycle should review, select, and tailor guidance from the Build Security In website and the references listed in this article as part of normal project management activities.

ACKNOWLEDGMENTS

The author acknowledges the thought leadership and critical review comments from Nancy Mead and Robert Ellison of Carnegie Mellon's Software Engineering Institute (CERT) and Gary McGraw, Chief Technology Officer at Cigital, Inc. and the author of numerous books on software security. Nancy, Robert, and Gary serve as the Management Review Board for the Build Security In website. Pamela Curtis (Carnegie Mellon, SEI) performed a key role as this article's editor and the editor-in-chief for all Build Security In website content.

Notes

1. Selected content in this article is summarized and excerpted from Security in the Software Lifecycle: Making Software Development Processes—and Software Produced by Them—More Secure (Goertzel et al., 2006).
2. http://en.wikipedia.org/wiki/Botnet
3. CERT is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
4. https://buildsecurityin.us-cert.gov
5. Refer to the SANS Top-20 Internet Security Attack Targets (http://www.sans.org/top20/) and the OWASP Top 10 critical web application security flaws (http://www.owasp.org/index.php/OWASP_Top_Ten_Project) for additional examples.
6. Refer to the Build Security In Deployment & Operations content area for more information; https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/deployment.html.

References

Committee on National Security Systems (CNSS) (2006, June). "National Information Assurance (IA) Glossary, Instruction No. 4009." http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf.

Denning, Dorothy E. (1998, December). *Information Warfare and Security*. Boston, MA: Addison-Wesley.

Goertzel, Karen Mercedes, Winograd, Theodore, McKinley, Holly Lynne, & Holley, Patrick. (2006, August). *Security in the Software Lifecycle: Making Software Development Processes—and Software Produced by Them—More Secure*, Draft version 1.2. U.S. Department of Homeland Security. https://buildsecurityin.us-cert.gov/daisy/bsi/ resources/dhs/87.html.

Hoglund, Greg, & McGraw, Gary. (2004). *Exploiting Software: How to Break Code*. Boston, MA: Addison-Wesley.

Hope, Paco & McGraw, Gary. (2004, May/June) "Misuse and Abuse Cases: Getting Past the Positive." *IEEE Security & Privacy Magazine*, IEEE Computer Society. pp. 32–34

McGraw, Gary. (2006). *Software Security: Building Security In*. Boston, MA: Addison-Wesley.

National Institute of Standards and Technology (NIST) (2003, December). *Standards for Security Categorization of Federal Information and Information Systems* (FIPS PUB 199). Federal Information Processing Standards Publication, NIS. http://csrc.nist.gov/publications/fips/.

President's Information Technology Advisory Committee (PITAC). (2005, February). *Cyber Security: A Crisis of Prioritization.* National Coordination Office for Information Technology Research and Development. http://www.nitrd.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf.

JULIA ALLEN is a senior researcher within the CERT© Program at the Software Engineering Institute (SEI), a unit of Carnegie Mellon University in Pittsburgh, PA. Allen is engaged in developing and transitioning executive outreach programs in enterprise security and governance, and works extensively with the IT operations and audit communities. Prior to this tecnical assignment, Allen served as acting Director of the SEI for an interim period

of 6 months as well as Deputy Director/Chief Operating Officer for 3 years. Her degrees include a B.Sci. in Computer Science (University of Michigan) and an M.S. in Electrical Engineering (University of Southern California). She is the author of The CERT Guide to System and Network Security Practices (Addison-Wesley, June 2001) and Governing for Enterprise Security (CMU/SEI-2005-TN-023, 2005).