Security in the Software Life Cycle

Joe Jarzombek
Department of Homeland Security

Karen Mercedes Goertzel
Booz Allen Hamilton

As a freely downloadable reference document, "Security in the Software Life Cycle: Making Application Development Processes – and Software Produced by Them – More Secure" presents key issues in the security of software and its development processes. It introduces a number of process improvement models, risk management and development methodologies, and sound practices and supporting tools that have been reported to help reduce the vulnerabilities and exploitable defects in software and diminish the possibility that malicious logic and trap doors may be surreptitiously introduced during its development. No single practice, process, or methodology offers the universal silver bullet for software security. "Security in the Software Life Cycle" has been compiled as a reference document with practical guidance intended to tie it together and inform software practitioners of a number of practices and methodologies from which they can evaluate and selectively adopt to reshape their development processes to increase not only the security but also the quality and reliability of their software applications, services, and systems, both in development and in deployment.

In an era riddled with asymmetric cyber attacks, claims about system reliability, integrity and safety must also include provisions for built-in security of the enabling software. The Department of Homeland Security (DHS) Software Assurance Program has undertaken to partner with software practitioners in industry, government, and academia to increase the availability and use of tools, knowledge, and guidance that will help improve the security and quality of the software they produce. In addition to its BuildSecurityIn Web portal [1] and Software Assurance Common Body of Knowledge [2], the DHS Software Assurance Program is publishing *Security in the Software Life Cycle: Making Application Development Processes – and Software Produced by Them – More Secure* [3] (freely downloadable from the DHS BuildSecurityIn portal).

Background

Software is ubiquitous. Many functions in the public and private sectors depend on software to perform correctly even during times of crisis despite attempts to subvert or compromise its functions. Software is relied on to handle sensitive and high-value data on which users' privacy, livelihoods, and lives depend. Our national security and homeland security increasingly depend on ever more complex, interconnected, software-intensive information systems that use Internet-exposed networks as their common data bus. Software enables and controls systems that run the nation's critical infrastructure: electrical power grids; water treatment and distribution systems; air traffic control and transportation signaling systems; nuclear, biological, and chemical laboratories and manufacturing plants; medical and emergency response systems; financial systems; and other critical functions, such as law enforcement, criminal justice, immigration, and, increasingly, voting. Software also protects other software. The majority of filtering routers, firewalls, encryption systems, and intrusion detection systems are implemented, at least in large part, through the use of software.

> *"The key to secure software is the development process used to conceive, implement, deploy, and update/sustain it."*

Perhaps because of this dependence, nation-state adversaries, terrorists, and criminals have joined malicious and recreational attackers in targeting this growing multiplicity of software-intensive systems. These new threats are both better resourced and highly motivated to discover and exploit vulnerabilities in software. The National Institute of Standards and Technology's (NIST) special publication 800-42, *Guideline on Network Security Testing* [4] sums up the challenge: Many successful attacks exploit errors (bugs) in the software code used on computers and networks. This is why software security matters. As more and more critical functions become increasingly dependent on software, that software becomes an extremely high-value target.

The objective of *Security in the Software Life Cycle* is to inform developers about existing processes, methods, and techniques that can help them to specify, design, implement, configure, update, and sustain software that is able to accomplish the following:

1. Resist or withstand many anticipated attacks.
2. Recover rapidly and mitigate damage from attacks that cannot be resisted or withstood.

The key to secure software is the development process used to conceive, implement, deploy, and update/sustain it. A *security-enhanced* software development life cycle process includes practices that not only help developers root out and remove exploitable defects (e.g., vulnerabilities) in the short term, but also, over time, increase the likelihood that such defects will not be introduced in the first place.

Security in the Software Life Cycle also addresses the risks associated with the software supply chain, including risks associated with selection and use of commercial off-the-shelf and open source software components, software pedigree (or more accurately the inability to determine software pedigree), and outsourcing of software development and support to offshore organizations and unvetted domestic suppliers.

Many security defects in software could be avoided if developers were better equipped to recognize the security implications of their design and implementation choices. Quality initiatives that reduce the number of overall defects in software are a good start, as many of those defects are security-related vulnerabilities. However, quality-based methods that focus only on improving the *correctness* of software seldom result in secure software. This is because in quality terms, *correctness* means correctness of the software's operation under anticipated, *normal* conditions.

Attacks on software and its environment typically create *un*anticipated, *ab*normal conditions. Moreover, rather than targeting single defects, attackers often leverage combinations of what seem to be correct functionalities and interactions in software. Individually, none of those functionalities or interactions might be problematical. Only when they are combined in an unexpected or unintended way does an exploitable vulnerability manifest. To a great extent, software security relies on the developer's ability to anticipate the unexpected.

A software security program must include training and educating developers to do exactly that, and to recognize the security implications not only of simple defects, but also of *abnormal* series and combinations of functions and interactions within their software, and between their software and other entities. Developers need to work within a development process framework that not only allows but also encourages them to analyze their software and recognize the security vulnerabilities that manifest from seemingly *minor* individual defects and coding errors, as well as their larger design and implementation choices.

Security Enhancement of the Software Development Life Cycle

Security enhancement of development life cycle processes and practices requires a shift in emphasis and an expansion of scope of existing development activities so that security is given equal importance as other desirable properties, such as usability, interoperability, reliability, safety, and performance. These shifts in emphasis and scope will affect the life cycle in the following ways:

1. **Requirements.** The review of the software's functional requirements will include a security vulnerability and risk assessment that forms the basis for capturing the set of general non-functional security requirements (often stated in terms of constraints on functionality) or *derived* security provisions (not explicitly specified in customer documents) that will mitigate the identified risks. These high-level, non-functional/constraint requirements will then be *translated* into more specific requirements for functionalities and functional parameters that otherwise may not have been included in the software specification.
2. **Design and implementation.** Design assumptions and choices that determine how the software will operate and how different modules/components will interact should be analyzed and adjusted to minimize the exposure of those functions and interfaces to attackers. Implementation choices will start with selection of only those languages (or language constructs), libraries, tools, and reusable components that are determined to be free of vulnerabilities, or for which an effective vulnerability mitigation can be realistically implemented. Bugs in source code must be flagged not just when they result in incorrect functionality, but when otherwise correct functionality is able to be corrupted by unintended and unusual inputs or configuration parameters.

> *"A key component of risk-driven software development occurs at the beginning of the development process: threat modeling."*

3. **Reviewing, evaluating, and testing.** Security criteria will be generated for every specification, design, and code review, as part of the software/system engineering evaluation and selection of acquired and reused components, and as part of the software's unit and integration test plans. Life cycle phase-appropriate security reviews and/or tests will establish whether all software artifacts (e.g., code, documentation) have satisfied their security exit criteria at the end of one life cycle phase before development is allowed to move into the next phase.
4. **Distribution, deployment, and support.** Distribution will be preceded by careful code clean-up to remove any residual security issues (e.g., debug hooks, hard-coded credentials). Preparation for distribution will also include documenting the secure configuration parameters that should be set when the software is installed; these include both parameters for the software itself and for any components of its execution environment (e.g., file system and Web server access controls, application firewall) that it will rely upon to protect it in deployment. *Secure by default* distribution configurations and trustworthy distribution mechanisms will be adopted. Ongoing, post-deployment, vulnerability and threat assessments, and forensic analyses of successful attacks will be performed for the software and its environment. New requirements to be satisfied in future releases will be formulated based on the assessment and analysis findings.

Risk-Driven Requirements Engineering

It is easier to produce software that can resist and recover from attacks when risk management activities and checkpoints are integrated throughout the software life cycle. A key component of risk-driven software development occurs at the beginning of the development process: threat modeling. To identify and prioritize mitigation steps that must be taken, developers first need to recognize and understand the threats to their software, including the threat agents, anticipated attack patterns, vulnerabilities likely to be exploited, and assets likely to be targeted, and they need to assess the level of risk that the threat will occur and its potential impact if it does. Several freely downloadable methodologies have emerged to support the developer in modeling threats to applications and other software-intensive systems, including the following:

- Microsoft's ACE (Application Consulting and Engineering) Threat Analysis and Modeling (also called Threat Modeling Version 2.0) [5].
- European Union Consultative Objective Risk Analysis System (CORAS) [6, 7, 8] and Research Council of Norway Model-Driven Development and Analysis of Secure Information Systems (SECURIS) [9, 10].
- Practical Threat Analysis (Technologies' Calculative Threat Modeling Methodology) [11].

- Trike: A Conceptual Framework for Threat Modeling [12].
- National Aeronautics and Space Administration Software Security Assessment Instrument [13, 14].
- Visa USA Payment Application Best Practices [15].

A number of other system-level methodologies and tools are also in use for risk analysis of software-intensive systems, including NIST's Automated Security Self Evaluation Tool, Carnegie Mellon University's Operationally Critical Threat Asset and Vulnerability Evaluation-Secure, and Siemens/Insight Consulting Central Computer and Telecommunications Agency Risk Analysis and Management Method. These methodologies are also introduced in *Security in the Software Life Cycle*.

Security-Enhanced Process Models and Development Methodologies

The use of repeatable process improvement models has been demonstrated to improve the efficiency and adaptability of software development life cycle activities and the overall quality of software by reducing the number and magnitude of errors. The Software Engineering Institute's Capability Maturity Model® (CMM®) and General Electric's Six Sigma are the most widely used process improvement models. A number of CMM variants have been tailored for specific communities or problem spaces, e.g., CMM Integration (CMMI®), integrated-CMM (iCMM), and System Security Engineering (SSE)-CMM, which is also an international standard [16].

Beyond the SSE-CMM, a number of efforts have been undertaken to adapt or extend existing maturity models or define new process improvement models that have security-enhancement as their main objective. These include the following:

- Safety and Security Extensions to CMMI/iCMM [17].
- Revised International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) Standard 15026 *System and Software Assurance*, which adds security assurance activities to ISO/IEC 15288 system life cycle and ISO/IEC 12207 software life cycle processes.
- Microsoft Security Development Lifecycle (SDL) [18, 19].
- Comprehensive, Lightweight Application Security Process (CLASP) [20].
- Carnegie Mellon University Software Engineering institute (SEI) Secure Team Software Process℠ (TSP℠ Secure) [21].

The CMM and ISO/IEC process models are defined at a higher level of abstraction than SDL and CLASP, which bridge the abstraction gap between CMM-level models and software development methodologies. It is quite possible to implement both a security-enhanced CMMI or iCMM as an overarching framework in which SDL or CLASP processes can be executed. SDL or CLASP could then itself provide a framework in which life cycle, phase-specific, security-enhanced methods such as Model-Driven Architecture (MDA) or Aspect-Oriented Programming (AOP) could be employed. Indeed, these higher-level models are intended to be *methodology-neutral* and to accommodate development using any of a variety of methodologies.

Using Familiar Development Methodologies in Ways that Improve Software Security

Unlike process improvement models, software development methodologies are specific in purpose and applicability. Efforts have been made to use existing methodologies in ways that are expressly intended to support the engineering of security functionality in software. In a few cases, efforts have been made to adapt these methodologies to expressly improve the security of the software produced.

MDA

Defined by the Object Management Group (OMG), MDA automatically transforms Unified Modeling Language (UML) models into platform-specific models and generates a significant portion of the application source code (the eventual goal is to generate whole applications). By using MDA, developers need to write less code, and the code they do write can be less complex. As a result, software contains fewer design and coding errors (including errors with security implications). Researchers outside OMG are looking at ways to add security to MDA by combining it with elements of Aspect-Oriented Modeling or by defining new UML-based security modeling languages to be used in producing MDA-based secure design models. Both Interactive Objects' ArcStyler [22] and IBM/Rational's Software Architect [23] support MDA-based security modeling, model checking, and automatic code generation from security models.

Object-Oriented Modeling With UML

Unlike security functions, security properties in object-oriented modeling are treated as generic nonfunctional requirements and thus do not require specific security artifacts. UML, which has become the *de facto* standard language for object-oriented modeling, lacks explicit syntax for modeling the misuse and abuse cases that can help developers predict the behavior of software in response to attacks. Recognizing these omissions, some UML profiles have been published that add expressions for modeling security functions, properties, threats and countermeasures, etc., in UML. SecureUML [24] provides extensions to support modeling of access controls and authorization. UMLSec [25] adds both access control/authorization modeling extensions and support for vulnerability assessment of UML models. The CORAS UML profile for security assessment [26], which has been adopted as a recommended standard by the OMG, is the most directly applicable to software security needs as it provides UML extensions for modeling threats and countermeasures.

Aspect-Oriented Software Development (AOSD)

Object-oriented development requires security properties and functions to be associated with each individual object in which that property/function must be exhibited. Security properties such as *nonsubvertability* and functions such as code signature validation are *crosscutting*, i.e., they are required in multiple objects. In object-oriented development, the developer would have to replicate and propagate the expressions of such cross-cutting security properties and functions to every object to which they pertain. This represents an unnecessarily high level of effort, both to initially specify and even more to make changes to cross-cutting security functions and properties, because such changes would also need to be replicated, propagated, and tracked.

AOM and Design extend the expressions possible in object-oriented modeling so that cross-cutting properties and functions are able to be expressed only once in a single modular component of the software model and design specification. This cross-cutting component is then referred to by all the components/

® Capability Maturity Model, CMM, and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.
℠ Team Software Process and TSP are service marks of Carnegie Mellon University.

objects to which the property/ function pertains. AOP allows the developer to write and maintain the cross-cutting component at a single location in the code. The AOP tools then automatically replicate and propagate that cross-cutting code to all appropriate locations throughout the code base. This automated approach reduces the potential that the developer may inadvertently omit the cross-cutting code from some components/objects.

Agile Methods and Secure Software

Can Agile development produce secure software? As with structured development methodologies, Agile methods seek to promote development of high-quality software. With the exception of Lean Development, the collection of methodologies that falls under the umbrella of *Agile Methods* all share a commitment to the core principles of the Agile Manifesto [27]. According to the Agile Manifesto, *Agility is enhanced by continuous attention to technical excellence and good design*. Proponents often cite key practices in many agile methods that help reduce the number of exploitable defects that are introduced into software practices such as enforced coding standards, simple designs, pair programming, continuous integration, and test-driven development (also known as *continuous testing*).

It has been suggested that Agile methods can support risk-driven software engineering *if* – and this a very big *if* – the requirements-based, functional test-driven development approach that underpins all Agile methods is extended to include the following:

- Redefine the customer (who drives all response to change) to include the stakeholders such as risk managers, certifiers, and accreditors responsible for enforcing security policy and preventing and responding to attacks on software.
- Expand agile testing to accommodate both requirements-based and risk-based tests (see Software Security Testing).

Security in the Software Life Cycle includes an extensive discussion of whether, and, if so, how and when Agile methods can be used for secure software development without violating the Agile Manifesto's core principles.

Formal Methods and Secure Software

Formal methods use mathematical proofs and precise specification and verification methods to clarify the expression and understanding of software requirements and design specifications. The strict mathematical foundations of formal methods create a basis for positive assurance that the software's specifications are consistent with its formally modeled properties and attributes.

Formal methods have been used successfully to specify and prove the correctness and internal consistency of security function specifications (e.g., authentication, secure input/output, mandatory access control) and security-related trace properties (e.g., secrecy). However, to date, none of the widely used formal languages or techniques are explicitly devoted to specification and verification of *non*-trace security properties, such as non-subvertability or correct, predictable behavior in the face of unanticipated changes in environment state.

> *"Security analysis and testing are most effective when a multifaceted approach is used that employs as wide a variety of techniques and technologies as time and resources allow."*

Software Security Testing

Software security testing verifies that the software produced is indeed secure. It does this by observing the way in which software systems and the components they contain behave in isolation and as they interact. The main objectives of software security testing are: 1) Detection of security defects, coding errors, and other vulnerabilities including those that manifest from complex relationships among functions, and those that exist in obscure areas of code, such as dormant functions; 2) demonstration of continued secure behavior when subjected to attack patterns; and 3) verification that the software consistently exhibits its required security properties and functional constraints under both normal and hostile conditions. However, a security requirements testing approach alone demonstrates only whether the stated security requirements have been satisfied, regardless of whether those requirements were adequate. Most software specifications do not include negative and constraint requirements such as *no failure may result in the software dumping core memory or the software must not write input to a buffer that is larger than the memory allocated for that buffer*.

Risk-based testing takes into account the fact that during the time between requirements capture and integration testing, the threats to which the software will be subject are likely to have changed. For this reason, risk-based testing includes subjecting the software to attack patterns that are likely to exist at time of deployment, not just those that were likely at time of requirements capture. In practical terms, many of the same tools and techniques used for functional testing will be useful during security testing. The key difference will be the test scenarios exercised in risk-based tests. The software's security test plan should include test cases (including cases based on abuse and misuse cases) that exercise areas and behaviors that may not be exercised by functional, requirements-based testing. These security test cases should attempt to demonstrate the following:

- The software behaves consistently and securely under all conditions, both expected and unexpected.
- If the software fails, the failure does not leave the software, its data, or its resources exposed to attack.
- Obscure areas of code and dormant functions cannot be exploited or compromised.
- Interfaces and interactions among components at the application, framework/middleware, and operating-system levels are consistently secure.
- Exception and error handling resolve all faults and errors in ways that do not leave the software, its resources, its data, or its environment vulnerable to unauthorized modification (disclosure) or denial of service.

Security analysis and testing are most effective when a multifaceted approach is used that employs as wide a variety of techniques and technologies as time and resources allow. These techniques may include the following:

- **White box security reviews and tests.** Performed on source code, white box testing techniques include code security review (direct code analysis, property-based testing); source code fault injection with fault propagation analysis; and automated compile-time detection.
- **Black box security test techniques.**

Targeting individual binary components and/or the software system as a whole, black box techniques are the only testing option when source code is not available. Black box techniques include software penetration testing, security fault injecting of binaries, fuzz testing, and automated vulnerability scanning.

- **Reverse engineering.** Disassembly and decompilation generate, in the former case, assembler code, and in the latter, source code – both of which can then be analyzed for security-relevant implementation errors and vulnerabilities. Decompiled source code can be subjected to standard white box security tests and tools. Reverse-engineering is often more difficult and time-consuming than other software security test techniques; many commercial software products use obfuscation techniques to deter reverse-engineering. Like formal methods, the level of effort required makes reverse-engineering practical only for the examination of high-consequence, high-confidence, or high-risk components. Even then, there is no 100 percent guarantee of success.

The DHS-funded NIST Software Assurance Metrics and Tool Evaluation program [28] has developed a taxonomy of security testing tool categories that include a database of profiles of commercial and open source tools within each category. This database is being used to capture the results of tool evaluations and measurements of tool effectiveness, and it is used to conduct a gap analyses of tool capabilities and methods.

Both DHS and the DoD continue to dedicate resources toward achieving software assurance [29], and individual organizations can implement practices today to contribute toward software assurance in the near future.

Sound Practices for Security Enhancing Life Cycle Activities

Practitioners and program managers need to first understand *what secure software is* [30]. Appendix G of *Security in the Software Life Cycle* presents a collection of security principles and sound practices that have been expounded on by respected practitioners of secure software development in the private, public, and academic sectors, in the U.S. and abroad. These principles and practices enable the insertion of security considerations into all phases of the software life cycle. Appendix G also describes practices that span life cycle phases such as secure configuration management, security-minded quality assurance, security training and education of developers, and selection and secure use of frameworks, platforms, development tools, libraries, and languages. Developers who start applying these practices today should start to see improvements in their software's security, as well as its quality. This is true even if the organizations they work for never commit to adopting a security-enhanced, structured development methodology or process improvement model.

Summary

With its increasing exposure and criticality, software has become a high-value target not just for malicious and recreational hackers, but for highly motivated, well-resourced cyber-terrorists, cyber-criminals, and information warfare adversaries. At the same time, user expectations (real or perceived) that new functionality can be delivered near-instantaneously has driven software suppliers to adopt ever-shortening release schedules and agile development methods that do not allow sufficient time for careful specification, design, coding, and testing. As a result, the software they produce is inordinately convoluted and complex, with seemingly infinite possible internal states and a multiplicity of flaws and defects. All of these factors make software increasingly vulnerable to the intensifying threats that surround it.

"... software has become a high-value target not just for malicious and recreational hackers, but for highly motivated, well-resourced cyber-terrorists, cyber-criminals, and information warfare adversaries."

Developers need to start questioning their assumptions about how software should be built. They need to understand that functional correctness must be exhibited not only when the software executes under anticipated conditions, but also when it is subjected to *un*anticipated, *hostile* conditions. *Security in the Software Life Cycle* provides developers with information that can help them achieve a two-phase security enhancement of their software processes. For the first phase, Appendix G describes sound practices and principles that developers can begin to apply immediately throughout the life cycle. These practices and principles are intended to *raise the floor*, enabling developers to achieve a basic level of security in their software processes. The security-enhanced process improvement models and life cycle methodologies in the rest of the document are intended to help developers *raise the ceiling* over the longer term by making additional, significant increases in the security of their processes and by adding structure and repeatability to further security-enhancement of those processes.◆

References

1. United States. Dept. of Homeland Security. BuildSecurityIn Portal. National Cyber Security Division <https://buildsecurityin.us-cert.gov/>.
2 Redwine, Jr., Samuel T. ed. Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software (DRAFT Version 1.1). Washington, D.C.: Department of Homeland Security, July 2006 <https://buildsecurityin.us-cert.gov/>.
3 Goertzel, K.M., et al. Security in the Software Life Cycle: Making Application Development Processes – and Software Produced by Them – More Secure (DRAFT Version 1.1). Washington, D.C.: Department of Homeland Security, July 2006 <https://buildsecurityin.us-cert.gov/>.
4. National Institute of Standards and Technology. Guideline on Network Security Testing. Special Publication 800-42. Oct. 2003 (intended solely as a source of information and guidance, not as a proposed standard, directive, or policy from DHS; descriptions of processes, methodologies, and technologies containing this document should not be interpreted as formal endorsements by DHS) <http://csrc.nist.gov/publications/nistpubs/800-42/NIST-SP800-42.pdf>.
5. Microsoft. Microsoft Threat Analysis & Modeling. v2.0 BETA2 <www.microsoft.com/downloads/details.asp

x?FamilyID=570dccd9-596a-44bc-bed7-1f6f0ad79e3d&DisplayLang=en>.
6. CORAS. A Platform for Risk Analysis of Security Critical Systems <www2.nr.no/coras/>.
7. CORAS. The CORAS Project <http://coras.sourceforge.net/>.
8. CORAS. A Tool-Supported Methodology for Model-Based Risk Analysis of Security Critical Systems <http://heim.ifi.uio.no/~ketils/coras/>.
9. SECURIS. Model-Driven Development and Analysis of Secure Information Systems <www.sintef.no/content/page1_1824.aspx>.
10. The SECURIS Project. Model-Driven Development and Analysis of Secure Information Systems <http://heim.ifi.uio.no/~ketils/securis/index.htm>.
11. PTA Technologies. Practical Threat Analysis for Securing Computerized Systems <www.ptatechnologies.com>.
12. Trike. A Conceptual Framework for Threat Modeling <http://dymaxion.org/trike/> and <www.octotrike.org/>.
13. National Aeronautics and Space Administration. Reducing Software Security Risk <http://rssr.jpl.nasa.gov>.
14. "Reducing Software Security Risk through an Integrated Approach." University of California-Davis <http://seclab.cs.ucdavis.edu/projects/testing/>.
15. "Visa USA Cardholder Information Security Program: Payment Applications." Visa <http://usa.visa.com/business/accepting_visa/ops_risk_management/cisp_payment_applications.html>.
16. International Standard ISO/IEC 21827, System Security Engineering (SSE)-CMM® <http://www.issea.org> and <http://www.SSE-CMM.org>.
17. Federal Aviation Administration Integrated Process Group. Safety and Security Extensions to Integrated Capability Maturity Models. Washington D.C.: Sept. 2004 <http://faa.gov/ipg/news/docs/SafetyandSecurity Ext-FINAL.pdf>.
18. Lipner, Steve, and Michael Howard. "The Trustworthy Computing Security Development Lifecycle." Microsoft <http://msdn.microsoft.com/security/sdl>.
19. Howard, Michael. "How Do They Do It?: A Look Inside the Security Development Lifecycle at Microsoft." MSDN Magazine: The Microsoft Journal for Developers. 20.11 (Nov. 2005) <http://msdn.microsoft.com/msdnmag/issues/05/11/SDL/default.aspx>.
20. "Comprehensive, Lightweight Application Security Process." Secure Software <https://securesoftware.custhelp.com/cgi-bin/securesoftware.cfg/php/enduser/doc_serve.php?2=CLASP>.
21. SEI Software Process (TSP) for Secure Systems Development <www.sei.cmu.edu/espltsp-secure.presentation/>.
22. "ArcStyler Overview." Interactive Objects <www.interactive-objects.com/products/arcstyler-overview>.
23. "Rational Software Architect." IBM <www.306.ibm.com/software/awdtools/architect/swarchitect/>.
24. Lodderstedt, Torsten. "Model Driven Security from UML Models to Access Control Architectures." Diss. Albert-Ludwigs-Universität, 2003 <http://deposit.ddb.de/cgi-bin/dokserv?idn=971069778&dok_var=d1&dok_ext=pdf&filename=971069778.pdf>.
25. UMLsec <http://www4.in.tum.de/~umlsec/>.
26. "The CORAS UML Profile." CORAS <http://coras.source-forge.net/uml_profile.html>.
27. Agile Alliance. "Manifesto for Agile Software Development." Agilemanifesto <http://agilemanifesto.org/>.
28. National Institute of Standards & Technology. "Software Assurance Metrics and Tool Evaluation." <http://samate.nist.gov/ index.php>.
29. "Software Assurance." Wikipedia <http://en.wikipedia.org/wiki/software-assurance>.
30. Goertzel, K.M. "What Is Secure Software?" IA Newsletter (Summer 2006) <http://iac.dtic.mil/iatac>.

About the Authors

Joe Jarzombek is the Director for Software Assurance in the Department of Homeland Security (DHS) National Cyber Security Division. He leads government interagency efforts with industry, academia, and standards organizations to shift the security paradigm away from patch management by addressing security needs in work force education and training, research and development (especially diagnostic tools), and development and acquisition practices. After retiring from the U.S. Air Force as a Lt. Col. in program management, Jarzombek worked in the cyber security industry as vice president for product and process engineering. He later served in two software-related positions within the Office of the Secretary of Defense prior to accepting his current DHS position. As a Project Management Professional, Jarzombek has spoken extensively on measurement, software assurance, and acquisition topics. He encourages further review of DHS-sponsored software assurance efforts via the BuildSecurityIn Web site.

National Cyber Security Division
Department of
Homeland Security
Phone: (703) 235-5126
Fax: (703) 235-5962
E-mail: joe.jarzombek@dhs.gov

Karen Mercedes Goertzel is a software security subject-matter expert supporting the director of the Department of Homeland Security's Software Assurance Program, and has provided support to the Department of Defense's Software Assurance Tiger Team. She was project manager for the Defense Information Systems Agency Application Security Support Task, and currently leads the team developing the National Institute of Standards and Technology's special publication 800-95, *Guide to Secure Web Services*. In addition to software assurance and application security, Goertzel has extensive experience in trusted systems and cross-domain information sharing solutions and architectures, information assurance (IA) architecture and cyber security architecture, risk management, and mission assurance. She has written and spoken extensively on security topics and IA topics both in the United States and abroad.

Booz Allen Hamilton
8283 Greensboro DR
H5061
McLean, VA 22102
Phone: (703) 902-6981
Fax: (703) 902-3537
E-mail: goertzel_karen@bah.com