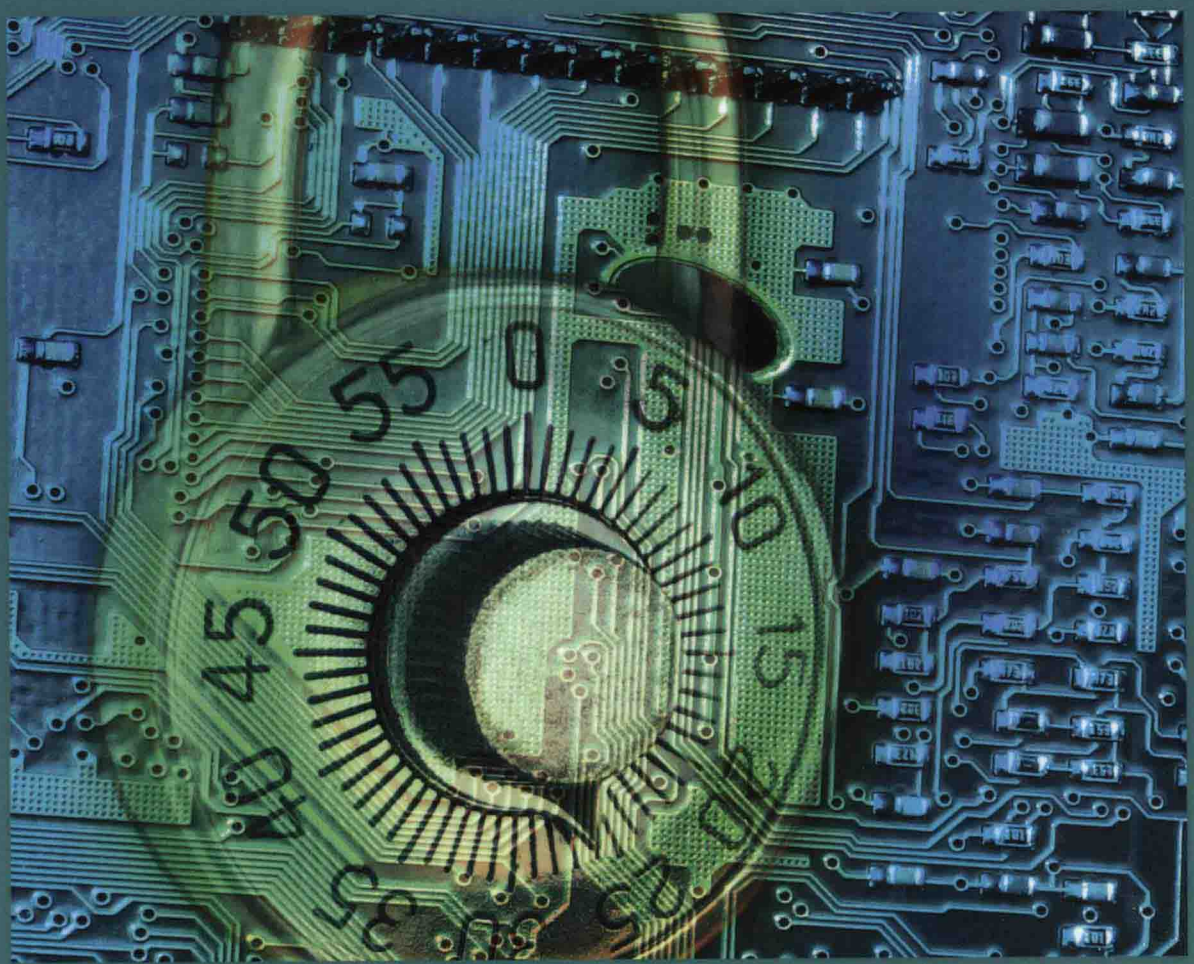
PREMIER REFERENCE SOURCE

Security-Aware Systems Applications and Software Development Methods



Khaled M. Khan

Security-Aware Systems Applications and Software Development Methods

Khaled M. Khan
Qatar University, Qatar

Information Science
REFERENCE

Managing Director: Lindsay Johnston
Senior Editorial Director: Heather A. Probst
Book Production Manager: Sean Woznicki
Development Manager: Joel Gamon
Acquisitions Editor: Erika Gallagher
Typesetter: Jennifer Romanchak
Cover Design: Nick Newcomer, Lisandro Gonzalez

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com

Library of Congress Cataloging-in-Publication Data

Security-aware systems applications and software development methods / Khaled M. Khan, editor.
p. cm.
Includes bibliographical references and index.
ISBN 978-1-4666-1580-9 (hardcover) -- ISBN 978-1-4666-1581-6 (ebook) -- ISBN 978-1-4666-1582-3 (print & perpetual access) 1. Computer networks--Security measures. 2. Computer software--Development. 3. Computer security. I. Khan, Khaled M., 1959-
TK5105.59.S43924 2012
005.8--dc23

2012002105

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Editorial Advisory Board

Associate Editors

Yun Bai, *University of Western Sydney, Australia*
Konstantin Beznosov, *University of British Columbia, Canada*
Kendra Cooper, *University of Texas at Dallas, USA*
Frédéric Cuppens, *ENST-Bretagne, France*
Martin Gilje Jaatun, *SINTEF, Norway*
Jan Jürjens, *University of Dortmund, Germany*
Florian Kerschbaum, *SAP, Germany*
Fabio Martinelli, *National Research Council, Italy*
Raimundas Matulevicius, *University of Tartu, Estonia*
Per Håkon Meland, *SINTEF, Norway*
Frank Piessens, *Katholieke Universiteit Leuven, Belgium*
Riccardo Scandariato, *Katholieke Universiteit Leuven, Belgium*
Hossain Mohammad Shahriar, *Queen's University, Kingston, Canada*
George Yee, *Carleton University, Canada*
Yan Zhang, *University of Western Sydney, Australia*

Preface

ISSUES AND CHALLENGES IN SECURITY-AWARE SOFTWARE DEVELOPMENT

Introduction

This is the first collection of the *Advances in Engineering Secure Software* series. This book addresses the paradigm of security-aware software development which is increasingly becoming an important area of software engineering, and steadily gaining solid ground as the computing technologies evolve and new security threats emerge. The paradigm deals with the problem of development duality between constructing a functional software system, and at the same time creating a secure system; that is, security concerns are integrated with the analysis and design of the systems functionality during the software development process. In this practice, security is considered as an integral part in all phases of software development. The engineering of secure software systems emphasizes security from a software engineering perspective, and deals with technical, as well as managerial aspects of software security. The process includes all aspects of software security in the development, deployment, and management processes of software systems. The "Build Security In" initiative by Homeland Security essentially advocates for secure software engineering that would provide adequate practices, tools, guidelines, rules, principles, and other resources that software development stakeholder can use to build security into software (Homeland security).

In this book, we use *security-aware software development* and *secure software engineering* interchangeably. The process of security-aware software development has two distinct but related objectives: *software assurance* and *security assurance* of the software. Various definitions of these are available in literature. In the context of this book, the former focuses more on an operational software that is free from defects, reliable, and provides a level of confidence. Software assurance includes the development process that ensures reliable functions and execution of software products as expected. Whereas, security assurance guarantees that security requirements of the software are adequately met and software can withstand security attacks. These two objectives are claimed to be achieved only when software continues to be operational correctly even under attacks. To achieve these, we need to address security throughout the entire lifecycle of the software development process.

In an another view point (Goertzel et al., 2008), a secure software must exhibit three properties: dependability, trustworthiness, and survivability. Dependability refers to the notion of software assurance, that is, the software executes correctly under attack or even can run on a malicious host. Software is trustworthy if it contains little vulnerability, or no malicious logic. Survivability is the ability of software

to recover as quickly as possible with negligible damage from attacks. In this perspective, secure software must provide justifiable confidence that it is free of vulnerabilities, executes as expected, does not compromise any of its required security properties, and can be trusted to continue executing under attacks.

Security-aware software development is not a single task that would produce secure software products; rather, it involves corporate policies, entire software life cycle, development culture of team members, budgeting, scheduling, etc. It may affect the way software is developed, the procedures are used, and the practices are adopted. It requires organizations to define or modify their existing development process in order to tailor towards security-aware development. Merely adopting security aspects into the traditional development process may not be very useful unless an evaluation and monitoring of the effectiveness of the new process is carried out.

Focus Areas

The conventional approach of bolting-in security functions on the top of software product after the implementation of software is definitely not a good practice. It is often argued that security of software has two binary values, either secure or not secure; and a software could only be secure by implementing security functions such as cryptography, firewalls, access control mechanisms etc. However, this view of better access control and cryptographic protocols and firewalls would keep a software secure does not hold anymore. Security functions such as cryptography or firewalls alone cannot solve the problem of software security. In contrast, security vulnerabilities in the software cause most of the security problems, and vulnerabilities are only present in bad software. Security functions such as encryptions, access control cannot change bad software into a good one. It is stated rightly in (Viega & McGraw, 2001) that bad software is the main reason for every computer security problem and malicious attacks. Thus, the main goal of security-aware software development is how to design and build good software. This leads us to identify the following focus areas that security-aware software development should include:

- A software development process that is well defined, and integrates the security issues from the early phases of development.
- Sound requirements analysis and modeling techniques that support and address security requirements along with the functional requirements as opposed to modeling security in isolation
- Tools and techniques to detect vulnerabilities and monitor attacks at the systems functional level.
- Development of adequate protections against the identified vulnerabilities and attack scenarios
- Availability of supporting tools to aid the security requirements modeling, testing, and secure integration of software, and
- Establishment of adequate training and practices for security-aware software development.

We discuss each of these briefly in the following sections.

Secure Software Development Process

In order to ensure good software production, there is a need for the establishment of a sound foundation of knowledge and capabilities in the software development process. We need a secure software development process that begins early with the analysis and modeling of functional and security requirements, identification of vulnerabilities, planning for protection, and finally designing the software for security.

The development process should include required activities, methods, tools, and procedures to ensure that the software product resulted of the process has necessary assurances in terms of operational and security objectives of the software.

Failing to recognize the importance of security aspects up front may cause many security problems and could be overlooked and left for post-development solutions. Secure software can be produced by integrating security strategies, designs, and procedures into each phase of the development lifecycle. Traditionally, security requirements in the software have often been considered as an afterthought, and as a consequence, security is delayed to the end of the development.

The idea of integrating security requirements and objectives from the early stages of the software development process began in late 1990s. In 2001, the US President's then special advisor for cyberspace security pointed out it clearly that members of IT Industry must build information security into their products at the point of development, and not treat security as an afterthought (Ghosh et al., 2001). The need for security-aware software was important and urgent that the concern was raised at the highest level of the US administration. By not addressing security along with the systems functionality at an earlier development stage, software engineers consequently may end up with bad software (Viega & McGraw, 2001). The Software Engineering Institute's (SEI) CERT program, and Team Software Process (TSP) (Humphrey, 2000) initiatives launched a joint effort called TSP-Secure to develop secure software. Main objective of this effort is to develop software that ensures quality and security. This process has typical tasks such as identifying security risks, eliciting and defining security requirements, secure design, secure code reviews, fuzz testing etc. The Office of Homeland Security also initiated the approach of "Build Security In" in mid 2000s. The International Secure Software Engineering Council (ISSECO), founded in 2008, focuses on the production of secure software, and its goal is to establish a secure computing. The challenge is how to fit the 'security aspects' into the 'functional aspects' of the software development process. Can the existing development methodologies be mixed with the rigid and formal processes associated with software security, and how?

Security Requirements Analysis And Modeling

Security requirements engineering deals with the protection of assets from potential attacks that may cause the software dis-functional (Haley et al. 2008). The current approaches have limited capability for identifying and modeling security requirements of a software system at the beginning of the development process. Security requirements analysis after the completion of a software system has created a negative effect on ensuring a secure system. Security requirements need to be analyzed along with the systems functional requirements at the early stages of the development process. Security requirements cannot be analyzed in isolation from the functional requirements of the system. There is a need for a balance between the conventional way of doing security requirements analysis and the iteration-centric, feedback-driven systems functional requirements.

There are considerable research works done in this direction. To represent multilevel access control (MAC) and role based access control (RBAC), UML notations have been used in (Shin & Ahn, 2000) and (Ray et al., 2003) rather than extending the UML notations. In contrast, UMLsec proposed in (Jurjens, 2002) is an extension of UML. It allows software engineers to perform security analysis of the system and verify if the model satisfies security requirements of a functionality. It also focuses on MAC of message in sequence/state diagrams. Another work reported in (Alghathbar & Wijesekera, 2003) proposes a framework called AuthUML for addressing security in use cases. The main idea is that the software engineer can specify a list of functionalities, and the possible security issues of each of the functionalities.

Other approaches to analysing and modeling security requirements include KAOS (van Lamsweerde 2004), Secure Tropos (Mouratidis et al. 2003; Giorgini et al. 2005; Mouratidis et al. 2006), and Secure i*(Liu et al. 2003). KAOS is used for reasoning about confidentiality requirements of a software system (de Landtsheer and van Lamsweerde 2005). Secure Tropos is used to model security concerns of a system such as security constraints, trust issues, and delegation of permission. Secure i* analyses security requirements by analyzing the relationships between various system stakeholders and potential attackers. SecureUML proposed in (Lodderstedt et al. 2002), an another extension of UML, is used to model access control policies and how the policies could be integrated into the software development.

The approach to elicit, specify and analyze security requirements proposed in (Haley et al. 2008) addresses both systems requirements and security requirements in the development process. It uses functional requirements with security constraints in a security engineering perspective. The approach in abuse frames (Lin et al. 2003) is used to analyze security requirements to determine security vulnerabilities. It is based on the notion of an anti-requirement -the requirement of a malicious user that can subvert the systems security requirement.

The process of understanding and modeling threats and security concerns at the atomic functional level helps drive the analysis and design of the software towards more robust secure systems. This early analysis of security requirements helps find potential security holes at the design level rather than wait after the implementation is complete. This practice of identifying security design flaws from the start will not only ensure design for security, it will also safe significant resources typically needed for post development patching. Addressing the post-development security design flaws requires a major upgrade and patching effort to minimize the resulting security problems which are usually too expensive. This approach definitely calls for a concerted efforts from software engineers and security experts to work together during the development process. They should strive together to mitigate the identified security problems at the analysis and design level before coding.

Detecting Vulnerabilities and Attacks

Another important issue in secure software development needs attention, that is, identifying as well as predicting attack vulnerabilities in software design. One of the pre requisite for secure software is the guarantee of absence of vulnerabilities in the software. Vulnerabilities can be introduced in the software at any point in the software development process. Vulnerabilities are exploited by any entity to launch attacks. Most common code-level vulnerabilities include buffer overflow, format string bugs, SQL injection, cross site scripting, and cross site request forgery. These vulnerabilities are directly exploitable by attackers. A vulnerable software can be exploited at runtime by providing specially crafted data to overflow the buffer of the program, or SQL injection attack.. Buffer overflow (BOF) is one of the worst and oldest vulnerabilities in software. It allows attackers to overflow data buffers in order to execute arbitrary code. A vulnerable program can be exploited by specially crafted inputs to overflow data buffers in order to execute malicious code or execute denial of service attacks by the attacker. In SQL injection attacks, a malicious entity may bypass systems authentication, change privileges, launch a denial-of-service attack, or run remote commands to install malicious software in the application. To address buffer overflow vulnerabilities, various approaches are proposed such as static analysis, testing, and fixing of vulnerable code. Static analysis is an approach to address this. It is the examination of code in order to identify patterns that indicate potential design errors and/or security problems. The technique is a very useful tool in detecting vulnerable code in programs. More severe form of buffer

overflow vulnerability attacks might not be identified until the program is operational. One of the main objectives of security-aware software development is to reduce vulnerabilities as much as possible, and improve protections to potential attacks in the software.

Software Protection

There is a need for "programmatic" protection techniques to address security problems related to software. These may include language based approach, better memory management, efficient, hot and static patching, or logic based reasoning techniques. Programmers could use secure coding practices such as avoiding coding errors, the awareness of bugs, guarding variables from exploitations etc. Developing supportive tools for secure integration and testing software is also essential to aid the protection of software. Defensive coding practices require training of developers and modification of the legacy applications to assure the correctness of validation routines and completeness of the coverage for all sources of input. Adequate training and good software security practices can help ensure that a software is secure.

Education and Training

The awareness, caution, intention, and adherence are the important elements that need to be integrated in training and practices for secure software development. It is important that awareness of security is reflected in every phases of the process. There is a strong demand for skilled professionals who can build security from the ground up. The IEEE Computer Society (IEEE-CS) and Association for Computing Machinery (ACM) have recently recognized the Master of Software Assurance (MSwA) Reference Curriculum for a master's program in software assurance (http://www.cert.org/mswa/). This recognition sends signals to the educational community that software assurance is an important part in computing education.

Challenges Ahead

To achieve the goals of security software development, several challenges need to be addressed.

- There is an urgent need for closed collaborations between security experts and software engineers during all phases of the development process. Security personnel should be integrated in the software development team. Calling in security experts only after the completion of software development would not bring much benefits. The practice of thinking security as a post development phase would not help secure software engineering much.
- We need a complete software development life cycle that integrates security aspects in all phases in the development process. It is difficult to identify how and which ways security could be dealt with in different phases of the development process. In other words, security requirements should be analyzed along with the functional units of the software.
- The development of a culture of "think security" in the development environment might be difficult to achieve. Changing the mindset of all team members of the software development team needs a different approach. The development projects should motivate their team members not to think security just as an afterthought. This paradigm calls for the change of corporate culture, and there is a chance that it may face resistance within the organization.

- The development of automatic tools that could aid the process of modeling, designing, and testing of security along with the functional requirements of the system. Without automatic or semi-automatic tool support, it would be difficult to achieve the objectives of secure software engineering. Development of new tools and languages need more research.
- There is a need for assurances that the process of security-aware development would not undermine other non-functional attributes of the process as well as the finished software products. In other words, a secure software, for example, would not degrade or ignore the usability of the system due to the inclusion of security as a necessary quality property of the software. The challenge is how to make a balance between security and other quality properties of the system.
- Enough care should be needed so that the process of secure software development does not become clumsy and too complicated for the stakeholder. Imposing an overly complex and unrealistic process would not be very beneficial for the achievement of the main goals of this new paradigm, rather, this may be counter-productive in software development.
- Controlling and monitoring budgets and schedules of secure software development project could be problematic if proper process is not adopted. There is a possibility for overblown budgets and delayed delivery schedules.
- Finally, convincing senior management in software development organizations could be difficult for some managers. This challenge is important because any changes to the development processes and practices involve resources, and these resources need to be approved by the top managerial body of the organization. This requires organizations to modify their organizational as well as management policies and activities.

Organization of the Book

The materials of this book are selected around six related themes of secure software engineering process that have already been discussed earlier. This collection catalogues total eighteen chapters, and they are grouped into six parts corresponding to six broad themes:

- Section 1: Secure Software Development Process
  - Chapters 1, 2 and 3
- Section 2: Security Requirements Analysis and Modeling
  - 4, 5, 6 and 7
- Section 3: Vulnerability Detection
  - Chapters 8 and 9
- Section 4: Protection Mechanisms
  - Chapters 10, 11, 12, 13
- Section 5: Tools for Security-Aware Development
  - Chapter 14 and 15
- Section 6: Secure Software Education and Training
  - Chapters 16, 17, and 18.

Chapter Abstracts

Chapter 1 focuses upon software security as the resistance against misuse and/or attacks. This chapter argues that secure code features are important, aiming at making the code un-exploitable, preventing attacks like buffer overflow. It presents an empirical study on how agile software developers include security in their software projects. The chapter also presents a case study showing that software development without a persistent focus on security results in software with a number of vulnerabilities. Finally, the chapter presents two possible extensions to agile methodologies, intended to increase developers' awareness of software security.

Chapter 2 presents a framework and step-wise approach towards achieving and optimizing assurance by infusing security knowledge, techniques, and methodologies into each phase of the Software Development Lifecycle (SDLC). This chapter outlines a progression of techniques and procedures that help promote and optimize security assurances. By infusing security into the SDLC from inception to implementation, security is proactively considered while preserving customer priorities and mitigating threat agent's opportunities to negatively impact those goals and assets. The detailed approaches provide a basis for understanding software assurance techniques and methodologies that could be used throughout the project development process.

Chapter 3 proposes a design approach that incorporates privacy risk analysis using UML diagrams to minimize privacy risks in the final design. The approach iterates between the risk analysis and design modifications to eliminate the risks until a design is risk free. The objective of this chapter is to propose an e-services design approach that incorporates privacy risk analysis to obtain designs that are more likely to preserve privacy. The final design is obtained as the culmination of a series of alternative designs where each alternative design is obtained by re-design to avoid or lessen privacy risks identified through a privacy risk analysis on the last design.

Chapter 4 introduces a method for security embedded business process modeling that captures security functions during the business process modeling phase. The proposed method draws on two well-tested theoretical foundations – enterprise ontology and organizational semiotics. The proposed method results in a security-embedded business process model that is completely based on formal semantics. That is, the models can be automatically analyzed or simulated to study the impact of the incorporated security requirements and whether the security requirements compromise business performance in any way. The resultant models can be straightforwardly simulated in order to observe how the security functions are executed.

Chapter 5 presents an approach to integrate RBAC and MAC into use-case, class, and sequence diagrams of the unified modeling language (UML), providing a cohesive approach to secure software modeling that elevates security to a first-class citizen in the process. This chapter details a practical approach that integrates RBAC and MAC into UML for secure software modeling and analysis with a two-fold emphasis. These access control extensions and security analyses have been prototyped within a UML tool.

Chapter 6 discusses the role of pilot case studies in security requirements engineering and their impact on method refinement, student projects, and technology transition. The chapter starts by providing some general background on the importance of requirements engineering and some specifics on the problems encountered in security requirements engineering. It then introduces the SQUARE and SQUARE-Lite methods, which were the research models used in the case studies. The chapter discusses both benefits and challenges to the underlying research, education, and technology transition effort.

Chapter 7 reviews current approaches to security requirements engineering and conclude that they lack explicit support for managing the effects of software evolution. It then suggests that a cross fertilization of the areas of software evolution and security engineering would address the problem of maintaining compliance to security requirements of software systems as they evolve. The chapter suggests that one approach to addressing this problem of preserving security properties is a cross fertilization of approaches to managing software evolution in security engineering

Chapter 8 classifies runtime Buffer overflow (BOF) attack monitoring and prevention approaches based on seven major characteristics. It then compares these approaches for attack detection coverage based on a set of BOF attack types. The classification is expected to enable researchers and practitioners to select an appropriate BOF monitoring approach or provide guidelines to build a new one.

Chapter 9 proposes a new testing methodology called Configuration Fuzzing. As the application runs in the deployment environment, this testing technique continuously fuzzes the configuration and checks "security invariants" that, if violated, indicates vulnerability. The chapter discusses the approach and introduces a prototype framework called ConFu (CONfiguration FUzzing testing framework) for implementation. It also presents the results of case studies that demonstrate the approach's feasibility and evaluate its performance.

Chapter 10 presents an approach for retrofitting existing web applications with run-time protection against known as well as unseen SQL injection attacks (SQLIAs) without the involvement of application developers. The precision of the approach in this chapter is also enhanced with a method for reducing the rate of false positives in the SQLIA detection logic, via runtime discovery of the developers' intention for individual SQL statements made by web applications. The proposed approach is implemented in the form of protection mechanisms for J2EE, ASP.NET, and ASP applications. The AMNESIA test bed is extended to contain false-positive testing traces, and is used to evaluate SQLPrevent.

Chapter 11 describes several vulnerabilities for C and C++, and how these could be remedied by modifying the management information of a representative manual memory allocator and garbage collector. The chapter examines the security of several memory allocators and discusses how they could be exploited.

Chapter 12 presents a method for hot patching executable and linkable format (ELF), formerly called Extensible Linking Format binaries that supports synchronized global data and code updates; and reasoning about the results of applying the hot patch. It develops a format, which is called Patch Object, for encoding patches as a special type of ELF relocatable object file. It then builds a tool called Katana that automatically creates these patch objects as a by-product of the standard source build process.

Chapter 13 primarily investigates security issues of the XML documents, and discusses a protection mechanism, and presents a formal approach to ensure the security of web-based XML documents. The proposed approach starts by introducing a high level language to specify an XML document and its protection authorizations. The chapter also examines the syntax and semantics of the language.

Chapter 14 presents a prototype tool for the integration of security-aware services based applications. The tool is constructed on the principles of security characterization of individual software services. It uses the technique of reasoning between the ensured security properties of the services and the security requirements of the user's system. Rather than reporting the research outcomes, this chapter describes the architecture and capabilities of the tool for secure software integration. It demonstrates the applicability of the tool with an example.

Chapter 15 details CAIRIS (Computer Aided Integration of Requirements and Information Security), a step towards tool-support for usable secure requirements engineering. The chapter claims CAIRIS not

only manages the elements associated with task, requirements, and risk analysis, it also supports subsequent analysis using novel approaches for analysing and visualising security and usability. The chapter illustrates an application of CAIRIS by describing how it was used to support requirements analysis in a critical infrastructure case study.

Chapter 16 proposes a context as well as a set of models used to develop and apply a secure software production pedagogy. A secure adaptive response model is discussed in the chapter to provide an analytical tool to assess risk associated with the development and deployment of secure information systems. A pedagogical model for information assurance curriculum development is then established in the context of the developed secure information system models. The relevance of secure coding techniques to the production of secure systems, architectures and organizational operations is also discussed.

Chapter 17 presents an overview of the Master of Software Assurance curriculum project, including its history, student prerequisites, and outcomes, a core body of knowledge, and a curriculum architecture from which to create such a degree program. The chapter also provides recommendations for implementing a Master of Software Assurance program.

Chapter 18 reports on the rigorous and scientific participatory approach for producing the adequate learning program meeting requirements elicited from the professional association members. It presents the skills card that has been elaborated for capturing these requirements and the program, called Master in Information System Security Management, which has been built together with the University of Luxembourg for matching these requirements. This program proposes a holistic approach to information security management by including organization, human and technical security risks within the context of regulations and norms.

CONCLUSION

As one can see from the above abstracts that all these chapters are timely in terms of their importance, coverage and new ideas. It is not an easy task to select chapters that are appropriate for various aspects of secure software engineering, and are in the correct sequence in terms of their focus. Carefully selected, these chapters reasonably cover major aspects of security-aware software development that have been discussed in this preface. I am sure that the selected chapters in this book address some of the challenges pointed out here. This collection of research work not only serves the hard technological aspects, but rather some of these also address other areas such as e-commerce and educational issues of secure software engineering. The contributing authors are drawn from different parts of the world ranging from Norway to Australia to Gulf and North America. This collection is a fine blending of various researchers with different focus and writing styles. I am very positive that our readers with different backgrounds will enjoy this collection.

Khaled M. Khan
Qatar University, Qatar

REFERENCES

Alghathbar, K., & Wijesekera, D. (2003). AuthUML: A three-phased framework to model secure use cases. Proceedings of the Workshop on Formal Methods in Security Engineering: From Specifications to Code, pp. 77-87

de Landtsheer, R., & van Lamsweerde, A. (2005). Reasoning about confidentiality at requirements engineering time. In Proceedings of the 10th European software engineering conference, Lisbon, Portugal: ACM. pp. 41-49.

Ghosh, A., Howell, C., & Whittaker, J. (2002). Building software securely from the ground up. *IEEE Software*, *19*(1), 14–16. doi:10.1109/MS.2002.976936

Goertzel, K. (2008). *Enhancing the Development Life Cycle to Produce Secure Software. A reference Guidebook on Software Assurance*. Department of Homeland Security.

Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2005). Modeling security requirements through ownership, permission and delegation. in Proceedings of 13th IEEE International Conference on Requirements Engineering, Paris, France, pp. 167-176

Haley, C. B., Laney, R., Moffett, J., & Nuseibeh, B. (2008). Security Requirements Engineering: A Framework for Representation and Analysis. *IEEE Transactions on Software Engineering*, *34*(1), 133–153. doi:10.1109/TSE.2007.70754

Security, H. Build Security In – Setting a high standard for software assurance. https://buildsecurityin.us-cert.gov/bsi/home.html (Extracted on January 29, 2012)

Humphrey, W. (1990). *Introduction to the Team Software Process*. Addison Wesley.

Jurjens, J. (2002). UMLsec: extending UML for secure systems development. Proceedings of UML, Springer LNCS, Vol. 2460, pp. 1-9.

Lin, L., Nuseibeh, B., Ince, D., Jackson, M., & Moffett, J. (2003). Introducing abuse frames for analysing security requirements. in Proceedings of 11th IEEE International Requirements Engineering Conference, pp. 371-372

Liu, L., Yu, E., & Mylopoulos, J. (2003). Security and Privacy Requirements Analysis within a Social Setting, In Proceedings of the 11th International Requirements Engineering Conference, IEEE CS Press, pp. 151-161

Lodderstedt, T., et al. (2002). SecureUML: A UML-based modeling language for model-driven security. In Proceedings of UML, Springer LNCS, Vol. 2460, pp. 426-441.

Mouratidis, H., Giorgini, P., & Manson, G. (2003). Modelling secure multiagent systems. In Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems Melbourne, Australia:ACM, pp. 859-866.

Mouratidis, H., Jurjens, J., & Fox, J. (2006). Towards a Comprehensive Framework for Secure Systems Development. In Advanced Information Systems Engineering, pp. 48-62.

Ray, I., et al. (2003). Using parameterized UML to specify and compose access control models. Proceedings of the 6th IFIP Working Conference on Integrity and Internal Control in Information Systems, ACM Press, pp. 115-124.

Shin, M., & Ahn, G. (2000). UML-based representation of role-based access control. Proceedings of the 9th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society, pp. 195-200.

van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. in 26th International Conference on Software Engineering, pp. 148-157

Viega, J., & McGraw, G. (2001). *Building Secure Software - How to avoid security problems the right way*. Addison-Wesley.

Section 1
Secure Software Development Process