Preface

INTRODUCTION

The popularity of the first collection of the *Advances in Engineering Secure Software series*, entitled, “Issues and Challenges in Security-Aware Software Development” (Khan, 2012), has prompted us to compile this second collection of the series. Our first collection emphasizes on two objectives of security-aware software development, namely, software assurance and security assurance in the light of the development process. It also provides a detailed account on these two aspects. However, this book focuses mainly on three types of major ingredients for security-aware software development. It looks back some of the development of various processes, tools, techniques and security functions that took place last one decade since the inception of security-aware software engineering paradigm. The paradigm basically started in late 1990s with the goal of developing appropriate software engineering process and techniques, in addition to standard security functions, that could ensure constructing secure software products.

The two terminologies *security-aware software* and *secure software* are often used interchangeably. The core idea of this paradigm is to integrate security concerns with all phases in the software development process from requirements to testing and deployment. Such a process not only deals with technical aspects of secure software engineering, but the management aspects of software security should also be addressed. The development of security-aware software is mainly based on software engineering principles focusing on technical as well as managerial aspects of software security along with systems functionalities. The US department of Homeland Security initiated this with the title, ‘Build Security in’ that essentially advocates this idea (Homeland security).

The issue of security-aware software development was raised by researchers as well practitioners in late 1990s because security has been often treated as an afterthought, delayed to post-deployment phases of the software development process. Worst, security issues are delegated to systems administrators at the deployment as well as maintenance stages. Security-aware software development has emerged as a fundamental concern. Standard security mechanisms such as authentication, access control and encryption are definitely necessary for protecting information and software, but they do not provide end-to-end assurances. Incorporating security into the entire software development process is definitely challenging. It is proposed in (Erl, 2005) that services software should be developed in three ways in order to ensure security: top-down approach, bottom-up approach, and agile approach -- one has to consider security throughout all these three approaches.
The process of developing security-aware software involves many issues, from security policy formulation, security requirements modeling, developing security architecture, integrating security requirements with functionalities, testing verification and validation, and finally assurances that the end product is secure. If we classify these into broader categories, we find that the main research thrust on security-aware software systems revolves around the following three major areas:

- Software development process
- Formal techniques and tool
- Standard security functions

The relationship among the above three is summarized in Figure 1. This figure shows that a security-aware software development process should be based on software development process, formal techniques, and standard security functions. Secure software can be produced by integrating security functions, appropriate formal techniques into various phases of the development lifecycle. Various phases in the development process are associated with different specific techniques and tools. For example, the requirements analysis phase is supported by logic based techniques, the coding is supported by almost all formal techniques. The list of the techniques outlined in the figure is just an example. Similarly, requirements analysis, design, coding and deployment phases should be integrated with the standard security functions. Again, the list of the standard security functions is not complete, it just shows some examples of security functions. More research work is needed in order to identify the appropriate tools and techniques, standard security functions, and their relationships with various phases in the software development process. We now briefly discuss each of the three ingredients in the following sections.

Figure 1. Major ingredients of security-aware software development process
Integration with Software Development Process

The secure software development process involves the distinct tasks and procedures in the development cycle that are used to analyze, design and construct a software product. It has been argued that to achieve security-aware software, we need to address software security throughout the entire lifecycle of the software development process (Khan, 2012). A single task or procedure would unlikely produce security-aware software. Modern software systems are mainly intended to support highly complex business processes and interactions with other systems. While capturing, modeling and reflecting these complex processes in software systems is a challenging task, addressing security concerns of businesses in the software poses even greater challenges. In this context, security-aware software modeling definitely requires a holistic and collaborative approach. At the process level, considerable research activities have been reported in the literature. More notably, the integration of security requirements into UML for secure software modeling and analysis has been developed by several researchers. All these works are centered around UML notations.

Most of these attempts to extend UML in order to incorporate security concerns in the functionality. Probably for the first time, a Framework for Network Enterprise utilizing UML notations to describe a role-based access control (RBAC) model was proposed in (Epstein & Sandhu, 1999). It uses UML to represent RBAC. A similar work, the representation of access control such as MAC and RBAC using UML, was also proposed in (Shin & Ahn, 2000; Ray et al., 2003). An extension of UML, called UMLsec proposed in (Jurjens, 2002a; Jurjens, 2002b) focuses more on multi-level security of messages in UML interaction diagrams (namely sequence and state diagrams). Similarly, in another work called SecureUML (Lodderstedt et al., 2002), the authors introduce new meta-model components and authorization constraints expressed for RBAC.

The main objective of most of these attempts is to extend UML in order to incorporate security concerns in the functionality. Similar work on the extension of UML use cases to include security requirements is reported in (Siponen et al., 2006). Another extension of UML was proposed in (Basin et al., 2009) in order to model security properties such a way that could be evaluated against hypothetical run-time instances such as static analysis regarding users and permissions. A proposal for systematically verifying and validating non-temporal and authorization constraints via UML's Object Constraint Language (OCL) is proposed in (Sohr, K. et al., 2008). The work reported in (Doan et al., 2010) also proposes an approach to integrate access control into UML for modeling and analysis of secure software systems.

Another interesting area of addressing security at the process level is the business process model. The business process model could capture security functions during the modeling phase such as reported in (Baskerville, 1988; Herrmann & Pernul, 1999; Backes et al. 2003; Mana et al. 2003; D’Aubeterre et al. 2008). In most of these works, security requirements are incorporated into the business process model. Although, several UML based process models have been proceed, unfortunately, we do not have many reports on how successful these are in practice, in real world software development. It would be much interesting to see that these are being used in the industry in order to develop security-aware software.

Formal Techniques and Tools

This area involves research in finding tools and techniques such as programming level constructs, formal approach, application of artificial intelligence, logic based approach, vulnerability detections and mitigations, etc. The enforcement and verification of security policies at the low level software components is
definitely a challenging task. There are several techniques used to ensure security of software systems. At the technique level, several research fronts are active in accommodating security issues at the low level programming units and logic of the program flows. Researchers have been exploring ways in which programming languages can be used to ensure that application software correctly enforces its security policies. This type of research work typically focuses on the enforcement of access control policies, data provenance tracking, information disclosure policies, and various forms of information flow policies in the program (Swamy et al., 2008b). Programming languages enable programmers to specify security policies and reason about that these policies are properly enforced in the program (Swamy et al., 2008a). One of the programming approaches to verify the correct enforcement of policies is to encode them as information flow policies for programs.

Research on programming languages demonstrates that security concerns can be dealt with by using both program analysis and program rewriting as powerful and flexible enforcement mechanisms. The security typed programming languages can allow programmers to specify confidentiality and integrity constraints on the data used in a program, and the compiler could verify that the program satisfies the security constraints (Zdancewic, 2002).

Another technique used for security-aware software is logic based approach that provides powerful expressiveness for the specification of security properties and the reason about security functions such as reported in (Fernandez, et al. 1989; Das, 1992; Fagin et al., 1995; Jajodia et al. 2001; Bertino et al. 2003). The logic based approach also provides reasonable flexibility for capturing security requirements. The predicates and rules of logic based language are used for expressing and evaluating authorization policies. A logic formalism has enough capability to specify, reason about and enforce access control polices in software systems.

Answer set programming, a logic based approach, is also being used to express and verify access control polices of a system. The applicability of the knowledge representation and reasoning languages such as logic programming for the design and implementation of a distributed authorization system is becoming promising. Researchers use answer set programming to deal with many complex issues associated with the distributed authorization along with the trust management approach (Wang, 2005). Formal authorization language based on semantics of answer set programming can express non-monotonic delegation policies unambiguously. The expressive power of such languages can represent the delegation with depth, separation of duty, partial authorization, and positive-negative authorizations. The logic based approaches could be used in security requirements analysis and specification during the systems development process. A design decision on security associated with systems functionality could also be reasoned about using logic based programming.

The application of logic based solution as well as programming features to the development of security-aware software looks promising.

**Standard Security Functions**

Standard security functions such as encryption, access control, authentication etc. are also required to ensure security-aware software systems. These functions ensure that security design and policies are adequately implemented and enforced as planned. Security functions are essential for the development of security software products. More research activities are required in order to identify ways how various security functions could be integrated or deployed at the micro level computing units in the software. It
includes finding techniques on how various sophisticated security functions could be employed at data level as well as method level. It is also equally important to ensure that employing various security functions at the low level programming units would not degrade the usability of the system. There is a need for a balance between usability and the use of excessive visible security functions in a software system.

**CHALLENGES**

The research community has not come yet with any particular methods, tools, techniques and a process that will guarantee secure software construction. The following challenges still remain to be tackled:

- Security-aware software development needs a standard process agreed by all stakeholders such as software engineers, security experts and tool developers. There is no standard process that can be used as a de facto standard. The research communities as well as professional bodies such as IEEE Computer Society, NIST, etc. have not agreed on any single process for the development of secure software. Perhaps, there is none. It is also not realistic to expect that a single silver bullet will enable us to develop security-aware software without obstacles.

- We also need to catalogue various tools and techniques developed so far for security software. The catalog should clearly specify which tool or technique solves which specific problem, their performance rating, etc. An integration framework is required in order to show which tools and techniques are appropriate for which phases in the development process. A clear mapping of integration between tools/techniques and development phases is desperately needed.

- It is still an open question on how to measure the strength of software security in a continuous scale as opposed to a binary ‘secure’ ‘not secure’ values. How could we ascertain that system A is more secure than system B? We can envision a security metrics based approach that could forecast the relative security strength (or weakness) of a software product. A very few research work has been reported in this area so far. In other words, how do users know that a piece of software or its units are secure?

- An important security requirement of service oriented software such as cloud computing is that the software learns nothing about the input data, or the computed results of clients. The software can process clients’ data without seeing them, as well as without comprehending the meaning of the output data. This goal should be achieved without employing extensive cryptographic techniques.

- We also need more techniques on how to compose different security policies during the integration of two software units in order to achieve a functionality. The composition of security policies in a dynamic software integration time is a challenging research task. It involves identifying contradictions between two participating security policies in a service composition, resolving these conflicts automatically, and so on.

- Another challenging issue still remains unsolved – certification of software from security bugs. We need techniques that would enable software developers to specify achieved security in the software. The claimed security properties could be independently verified by third party certifying authorities. Once certified, software security properties could not be altered and the certificate cannot be tampered.

- Finally, addressing the issue of trust in a software system is a big challenge. We need to find ways that could enable users to decide if software is trustworthy, if it is, how much etc.
ORGANIZATION OF THE BOOK

The chapters of this book are grouped around the three major ingredients of secure software engineering process that have already been discussed earlier. This collection presents total fifteen chapters, and they are grouped into three parts:

- **Section 1: Software Development Process**
  - Chapters 1-6
- **Section 2: Formal Techniques and Tools**
  - Chapters 7-10
- **Section 3: Standard Security Functions**
  - Chapters 11-15

CHAPTER ABSTRACTS

Chapter 1 reports the results on an experience of developing a secure web based system for a financial service industry using the Secure Tropos methodology. The findings of the experiment demonstrate the effectiveness of the methodology for the construction of secure web based systems. It further supports the notion that incorporating security considerations from the early stages of the software development process promotes the idea of secure by design. The developed system was tested by an independent entity using penetration testing. The testing results indicate no presence of any major security problems. This chapter basically demonstrates how the application of a software engineering methodology could support the notion of secure by design. The chapter also shows that the proposed methodology is flexible enough to apply in other application domains.

Chapter 2 presents a method for evaluating the security properties of service-oriented systems at the architectural level. The method attempts to recover security properties of the system by using reverse engineering techniques. It also provides automated support for further security analysis at the structural level. The proposed method is based on system-independent indicators and a knowledge base. It is independent form any specific programming language or technology. The method is flexible enough for adapting in various application systems. The chapter also describes a prototype tool that implements the methods, and the associated knowledge base. It argues that the knowledge base allow flexible refinement and adaptation of existing evaluation rules, and addition of new security aspects to the analysis. It also supports reusability of security expertise obtained from past evaluations, and offers fine-tuning capabilities using its weighting scheme.

Chapter 3 proposes a process framework, called IRIS (Integrating Requirements and Information Security), which provides guidance for the selection of technique when specifying security requirements in order to achieve secure software systems. The chapter demonstrates the applicability of the proposed framework by presenting a case study where the process framework was used to derive missing security requirements for an information security policy for a UK water company. It concludes with three lessons informing future efforts to integrate Security, Usability, and Requirements Engineering techniques for secure system design. The chapter argues that without a better understanding of how to deal with security and usability concerns at an early stage of the systems development, the design process may not achieve the security goals of the system.
Chapter 4 reports an experience with a European Union project in designing security and dependability patterns. It uses SI* framework, an agent-goal-oriented modeling framework in order to analyze organizational settings together with technical functionalities. This chapter demonstrates how a framework could assist domain experts in designing security and dependability patterns, validating them by proof-of-concept implementations, and applying them to increase the security level of the system. It also shows how SI* framework has been used by industries for capturing security and dependability patterns, and how patterns can be applied to achieve a sufficient level of security in the system. The proposed patterns have been used in various application contexts such as air traffic management, e-Health smart systems, etc.

Chapter 5 presents the results of a systematic survey to identify the state-of-the-art model driven development (MDD) focusing on security. The survey was carried out in order to find out how research communities deal with security in model driven development. The chapter addresses three issues: what are the major scientific initiatives describing automatic code generation from design models within the context of security in MDD, what empirical studies exist on the topic, and what are the strengths of the evidence that security aspects can be modeled as an inherent property and transformed to more secure code. It provides an introduction to the major secure model driven development initiatives and suggests that there is a lack of empirical work on the topic. It calls for standardization and more empirical research in this area.

Chapter 6 proposes a methodology based on a comprehensive list of security mechanisms for detecting database security gaps. The security mechanisms are derived from widely accepted security best practices. The methodology is intended to be used for gap analysis of security features which have been extracted from seven software packages composed by widely used applications, including four DBMS engines and two Operating Systems (OS). The main objective of the study is to find how each software package actually helps developers and administrators to achieve their systems security goals. The chapter concludes with interesting results showing that while there is a common set of security mechanisms that are supported by most software packages, however, there are some important security issues that are not supported at all by any of these packages.

Chapter 7 reports on a research effort that uses executable slicing as a pre-processing aid to improve the prediction performance of rogue software detection. The prediction technique used in this work is an information retrieval classifier that applies the feature extraction technique of randomized projection in order to detect previously unknown, known or variances of known rogue software. This approach extracts particular sections or slices from potentially rogue software, and uses these slices to make a prediction. It demonstrates optimal results of a 4% increase in prediction performance and a five-fold decrease in processing time when compared to using the entire application for the prediction. It concludes that a better malicious software classifier can be predicted by applying an executable slicing technique as a pre-processing step to the technique of randomized projection.

Chapter 8 propose a modal logic based approach to protect information system. The logic is used to specify and enforce security policies. The chapter provides the reasoning technique in response to the system access request especially in the situation where the grant access request is incomplete. The semantic of the language is provided by translating the language into epistemic logic program. The chapter demonstrates its applicability with a good number of realistic examples. It is expected that the proposed mechanism could be able to prevent unauthorized access to information systems more effectively. The chapter argues that the language has a strong expressive power to describe a variety of complex security requirements and policies.
Chapter 9 proposes a formal language which can provide role-based access control to information stored in XML documents. The proposed language has the capacity to reason about whether access to an XML document should be allowed or not. The language constructs allows systems designers to specify authorisations on XML documents. The inclusion of temporal interval reasoning and the XPath query language in the language constructs is rich enough to support the specification of authorization features in XML formatted documents. The language basically can be used to define a security policy base capable of specifying all access rights in the scope of an XML environment. The semantics of the language is provided through its translation into a logic program, namely Answer Set Programming (ASP). The chapter shows the application and expressive power of the language with a case study.

Chapter 10 discusses the integration of XP with security functions based on the CLASP (Comprehensive Lightweight Application Security Process) methodology in order to enable software developers using XP to construct secure software. CLASP provides a structured way to address security issues throughout the software development lifecycle. The proposed integration is basically based on interweaving the XP core practices with CLASP security best practices. The chapter argues that security measures should be applied in all phases in the development process. The claims are that this integration could possibly minimize the security vulnerabilities exploited by attackers. It is expected that the integration of XP with security functions enables software developers build more secure software. The proposed approach for extending XP with security complements with CLASP security activities.

Chapter 11 analyzes access control mechanisms of the Enterprise Java Beans (EJB) architecture and defines a configuration of the EJB protection system. It claims that the proposed configuration is less ambiguous than the EJB 3.0 standard. It presents an algorithm that formally specifies the semantics of authorization decisions in EJB. The chapter examines the level of support for the American National Standard Institute’s (ANSI) specification of Role-Based Access Control (RBAC) components and functional specification in EJB. It concludes that the EJB specification does not adequately supports the Core ANSI RBAC. This paper finally proposes a framework for assessing implementations of ANSI RBAC for EJB systems.

Chapter 12 attempts to describe the process of development and testing of a new encryption key management protocol for highly dynamic and ad hoc networks. The proposed protocol is called Secure Key Deployment and Exchange (SKYE). It also discusses the simulation software design paradigms used to evaluate the performance of the proposed protocol. The research work demonstrates that the network security can be increased by requiring more servers to collaborate in order to generate a certificate for the new member, or by requiring a higher trust threshold along with the certificate request chain. Three different locations for servers relative to other nodes have been experimented with. It concludes with experimental evidence that the location of nodes designated as servers within the network has an impact on the likelihood of a successful issuance of a certificate.

Chapter 13 primarily presents a model-driven development framework, namely JavaSPI, that allows the user to reliably develop security protocol implementations in Java. It supports a range of modeling activities ranging from the development of abstract models to the verification of models formally. Java is used as both a modeling language and the implementation language. The proposed JavaSPI framework is validated for its applicability by implementing a scenario of the SSL protocol. The chapter also claims that the framework could be successfully interoperated with OpenSSL, and has comparable execution time with the standard Java JSSE library.

Chapter 14 presents a systematic study describing various attempts made to forge fingerprints to fool biometric systems. The chapter reviews a comprehensive number of relevant publications on forging
fingerprints to fool sensors. It introduces the basics of biometrics in order to define the meaning of the term security along with the state of the art of biometric systems. It focuses on security of fingerprint scanners. It reports that a series of more than 30,000 experiments have been conducted to fool scanners. The experiments demonstrate that fingerprint scanners are not that easy to be deceived as the other research works suggest. The chapter provides a good deal of data to convince the reader that fingerprint scanners are secure, and cannot be fooled that easily. This work is believed to store public confidence on finger print scanners.

Chapter 15 explores the consequences of integrating patient electronic consents in e-health access control by examining requirements. Based on the analysis, it proposes an architecture incorporating patient consent in the access control service of an e-health system. The work examines various options for representing patient consents, and investigates how to incorporate their directives into the access control decision process. The legal requirements concerning patient are introduced. The chapter proposes a format for representing patient consents, and suggests on how to govern them in their lifecycle. It also presents a policy evaluation algorithm and a reference authorization architecture that incorporates patient consents. The reference architecture is based on the extension of the XACML authorization model. An prototype of the proposed architecture and its evaluation with a case study is also presented.

CONCLUSION

The collection of these chapters embodies a wealth of research outputs and ideas in secure software development. The book is suitable for researchers, graduate students, as well as practitioners. It is expected that this collection promotes the concepts of security-aware software development to a wider community comprising software engineers, security designers, researchers and beyond. I believe that the above line up of chapters with various flavors would generate enough interests in this topic. These chapters are expected to cover the three major ingredients of security-aware software development that have been outlined in this preface. The focus of each chapter varies from chapter to chapter ranging from hard technology to high level description of different ideas. Some chapters are based on complex mathematics, some are on logic, and few are based on operating system products. I am sure this book will satisfy readers from all areas of security-aware software development.

Khaled M. Khan
Qatar University, Qatar

REFERENCES


