2009 33rd Annual IEEE International Computer Software and Applications Conference

# On Selecting Appropriate Development Processes and Requirements Engineering Methods for Secure Software

Muhammad Umair Ahmed Khan and Mohammed Zulkernine
School of Computing, Queen's University
Kingston, Ontario, Canada, K7L3N6
{umair, mzulker}@cs.queensu.ca

*Abstract* — **To avoid security vulnerabilities, there are many secure software development efforts in the directions of secure software development life cycle processes, security specification languages, and security requirements engineering processes. In this paper, we compare and contrast various secure software development processes based on a number of characteristics that such processes should have. We also analyze security specification languages with respect to desirable properties of such languages. Furthermore, we identify activities that should be performed in a security requirements engineering process to derive comprehensive security requirements. We compare different security requirements engineering processes based on these activities. Our analysis shows that many of the secure software requirements engineering methods lack some of the desired properties. The comparative study presented in this paper will provide guidelines to software developers for selecting specific methods that will fulfill their needs in building secure software applications.**

***Keywords – Software security; secure software development process; software security requirements engineering.***

## I. INTRODUCTION

Traditionally, security of software is not considered from the very beginning of a software development life cycle (SDLC), and it is only incorporated in the later stages of development as an afterthought. As a consequence, there are increased risks of security vulnerabilities that are introduced into software in various stages of development [1]. Secure software engineering (or software security) aims to avoid security vulnerabilities in software by considering security aspects from the very beginning and throughout the SDLC. Secure software engineering is the process of designing, building, and testing software so that it becomes secure. Software security includes secure software development life cycle (SSDLC) processes and secure software development (SSD) methods. A SSDLC process considers security aspects of the software during the development life cycle by using SSD methods. SSD methods include, among others, security specification languages, security requirements engineering processes, and software security assurance methods. It should be noted that software security concerns are different from "application security" issues. Application security is about protecting software after it is developed and deployed. It usually includes input filters, intrusion detection systems, firewalls, and other protection mechanisms [1].

It is important to select an appropriate SSDLC process that provides ways to address security issues throughout development life cycle. Moreover, developing and deploying a patch to remove an error in the requirements (that may cause a security vulnerability) can be up to 200 times more expensive [2] than if the error had been removed as soon as it was introduced. Given that, it is necessary that suitable secure requirements engineering methods are used.

In this paper, we analyze the existing SSDLC processes and requirements engineering methods for secure software development based on their strengths and weaknesses. We first identify a number of characteristics of a SSDLC process that make it complete and effective. These characteristics are then used to compare and contrast various SSDLC processes. We also present a detailed comparison of security specification languages based on a number of desirable properties. This comparison focuses on identifying languages that can effectively specify security requirements. Moreover, we identify activities that should be performed as part of any security requirements engineering process. Based on these activities, we compare various security requirements engineering processes and identify their strengths and weaknesses.

The analysis presented in this paper can be useful in a number of ways. The comparison of various SSD methods will help software developers in selecting a particular SSDLC process, security specification language, or a security requirements engineering process for their particular development scenarios. The identified properties of software security specification languages can be useful to translate one specification language into another. Such a translation is particularly useful when a user of one language intends to use security tools developed for other languages.

The rest of this paper is organized as follows. Section II presents an in-depth critical analysis of SSDLC models or processes. Section III analyzes methods for software security requirements engineering. Section IV concludes this paper by summarizing our findings and identifying the corresponding open research issues.

## II. SSDLC PROCESSES

As the idea of incorporating security into software from the very beginning of development has gained acceptance, various SSD methods were suggested. However, software are still being developed by following the conventional SDLC models or processes, and various SSD methods are

0730-3157/09 $25.00 © 2009 IEEE
DOI 10.1109/COMPSAC.2009.206

used at different stages as deemed fit to the developers. Many SSDLC processes have been proposed that use different SSD methods during development. Each of these SSDLC processes has its strengths and weaknesses. Gregoire et al. [3] have compared Trustworthy Computing Security Development Life Cycle or Microsoft Software Development Life cycle (MS SDL) [4] and Comprehensive, Lightweight Application Security Process (CLASP) [5] based on the structure of the process, the resources provided by the process, and the SSD activities performed during different phases of the software life cycle (project inception, education, analysis, design, implementation, testing, verification, deployment, and support phases). Modifying the comparison criteria used by Gregoire et al. [3], we propose that an SSDLC process should have the following characteristics.

*Specification of SSD activities for the requirements engineering phase:* Our analysis shows that majority of the SSDLC processes propose a similar set of activities to be performed in the requirements engineering phase.

*Specification of SSD activities for the design phase:* Most of the processes pay little or no attention to SSD activities in the design phase. Only MS SDL and CLASP provide a comprehensive set of SSD methods for the design phase.

*Specification of SSD activities for the implementation phase:* MS SDL proposes a more comprehensive set of activities as compared to other processes.

*Specification of SSD activities for the security assurance phase:* With the exception of Appropriate and Effective Guidance for Information Security (AEGIS) [6], Secure Software Development Model (SSDM) [7], and Secure Software Development Model (SecSDM) [8], all the processes recommend using multiple security assurance methods such as security testing, code reviews, static code analysis, and so on.

*Resources available to the developers:* As is evident from Table I, only CLASP, Software Security Assessment Instrument (SSAI) [9], Hadawi's set of activities [10], MS SDL, and McGraw [1] provide developers with resources. However, resources provided by CLASP are more extensive.

*Use of artifacts of an SSD activity in the later phases of development:* Both AEGIS and McGraw's SSDLC process specify abuse cases without explicitly mentioning their future use within the SSDLC. Nevertheless, abuse cases have been used to generate test cases [11]. CLASP uses security requirements to guide testing.

*Usage in the industry:* Only MS SDL, McGraw's SSDLC process, and CLASP have been reportedly used in industry. MS SDL has an edge over CLASP and McGraw's SSDLC process as it was developed based on the experiences gained from previously developed commercial software.

In a nutshell, MS SDL and CLASP are more comprehensive with respect to the aforementioned characteristics. Table I presents a summary of the comparison of various SSDLC processes [1, 4-14].

## III. Software Security Requirements Engineering

Requirements engineering is the foundation of developing quality software. The main objective of software security requirements is to specify that the confidentiality, integrity, and availability of the software should be preserved. Usually, these requirements specify the security features needed for the software system. However, such requirements are not precise enough and need to be more explicit about who can do what and when [15].

Research reported in the field of security requirements engineering are primarily on security specification languages and security requirements engineering processes. In the following subsections, we first compare existing security specification languages based on the desirable properties of these languages and then analyze various security requirements engineering processes based on a set of activities that should be part of such a process.

### A. Software Security Specification Languages

Many different languages have been proposed in the literature to represent security specifications (security requirements or attack specifications). Some authors use the term security requirement as a synonym to attack specification [16, 17]. However, a security requirement [1, 18, 19] is different from an attack specification in that a security requirement is a control or constraint which if not implemented may lead to a vulnerability. On the other hand, an attack specification represents the sequence of the steps of an attack. Nevertheless, it is true that attack specifications can form a basis to identify vulnerabilities in software. Moreover, attack scenarios are also helpful in developing intrusion-aware software.

A security specification language can be a software specification language used to specify attacks (AsmL [20] and UML state charts [20]), an extension of a software specification language to represent attacks (Misuse Cases [21], Abuse Cases [22], AsmLSec [17], and UMLintr [23]) and security requirements (UMLsec [2], SecureUML [24], SecureTropos [25], and Misuse Cases), or an attack specification language (e.g., STATL [26] and Snort Rules [27]). Here we discuss the software specification languages and their extensions that have been used in the context of security. Similarly, from the various attack specification languages reported in the literature [16], we analyze only those that have been employed with software specification languages. Different security specification languages have different properties. In the following paragraphs, we identify desired properties of these languages and then use these properties to compare these languages in Table II.

*Mandatory requirements for a security requirements specification language [2]:* Only UMLsec and SecureTropos fulfill all three requirements. Misuse cases only lack low level details.

*Similarity with software specification languages:* All the languages based on UML (such as Abuse cases, Misuse cases, UMLsec, SecureUML, and UMLintr) would be much

TABLE I. COMPARISON OF SSDLC PROCESSES

| SDLC Processes / Characteristics | McGraw SDL [1] | MS SDL [4] | AEGIS [6] | SSDM [7] | Apvrille & Pourzandi [12] | SecSDM [8] | SSAI [9] | Hadawi [10] | CLASP [5] | S2D-ProM [13] | TSP Secure [14] |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **SSD Activities for Requirements Engineering** | Specification of abuse cases and security requirement. Risk analysis. | Identification of interfaces, security objectives and required security features. Definition of exit criteria. | Risk analysis. Identification of assets, abuse case and high level security requirements. | Threat modeling. Specification of security policy. | Identification of high level security objectives. Threat modeling. Specification of security requirements. Risk analysis. | Risk analysis. Identification of security objectives. | Modeling using flexible modeling framework (FMF). Model checking for security. | None | Risk analysis. Threat Modeling. Identification of attackers and attack surface. Specification of misuse cases with their mitigations, security features. | Following security standards. Risk analysis. Identification of errors. Specification of security features. | Threat modeling. Specification of abuse cases. Risk Analysis. |
| **SSD Activities for Design** | Risk analysis. | Identification of critical components, attack surface, design methods, and completion criteria. Threat modeling. Risk analysis. Definition of secure architecture. | Design decisions based on security requirements. | Following the security policy. | Use of UMLsec and security patterns. | Identification of security features to fulfill security objectives. | None | Following design guidelines. | Following design guidelines. Annotation of class diagrams with security information. Threat modeling. Risk analysis. | Using security modeling language, security patterns. Risk analysis. Design reviews. Model checking. | Design patterns. State machine design and verification. |
| **SSD Activities for Implementation** | None | Following secure coding standards. Code reviews. Use of testing and static code analysis tools. | None | None | Use of a secure programming language. Use of established security algorithms. Avoiding buffer overflow and format string vulnerabilities. | Following secure coding standards. | None | Use of a secure programming language. Following secure coding standards and guidelines. | Following secure coding guidelines. | Following secure coding standards. Use of a secure programming language. | Following secure programming standards and guidelines. |
| **SSD Activities for Security Assurance** | Risk-based and penetration testing. Use of static code analysis tools. | Code reviews. Security testing. | None | Penetration testing. | Code reviews. Ad-hoc unit and system security testing. Fuzz testing. Use of static code analysis tools. | None | Use of static code analysis tools. Property-based testing using the PBT tool. | Code reviews. Use of static code analysis tools. | Code reviews. Use of static code analysis tools. Security testing. | Risk analysis. Code reviews. Use of static code analysis tools. | Fuzz testing. Penetration testing. Use of static code analysis tools. Code reviews. |
| **Resources** | Secure design guidelines. | Secure design guidelines. | None | None | None | None | Vulnerabilities and their mitigations, a list of security code scanning tools, and potential items for a security checklist to guide development. | Common vulnerabilities to avoid and secure design and implementation guidelines. | Secure design and implementation guidelines. A comprehensive list of more than 200 types of vulnerabilities with their possible mitigations. | None | None |
| **Use of Artifacts of an SSD Activity in the Later Phases of Development** | Abuse cases can be used to derive test cases. | None | Abuse cases can be used to derive test cases. | None | None | None | None | None | Security testing is based on requirements. | None | None |
| **Usage in the Industry** | Reported | Reported | Not Reported | Not Reported | Not Reported | Not Reported | Not Reported | Not Reported | Reported | Not Reported | Not reported |

TABLE II. COMPARISON OF SECURITY SPECIFICATION LANGUAGES

| Desired Properties / Security Specification Languages | Mandatory Requirements by Jurjens [2] | | | Similarity with Software Specification Language (Yes/No) | Used as an Effective Secure Design Language (Yes/No) | Use of Specified Security Requirements and Attack Specifications in a Later Phase (Name of the Phase) | Tool Support (Yes/No) |
|---|---|---|---|---|---|---|---|
| | Ability to Formulate Basic Security Requirements (Yes/No) | Ability to Represent Usage Scenarios (Yes/No) | Ability to Represent Security Mechanisms and Low Level Security Requirements (Yes/No) | | | | |
| **UMLsec [2]** | Yes | Yes | Yes | Yes | Yes | Testing [2] | Yes [2] |
| **SecureUML [24]** | No | No | Yes (RBAC Policy Only) | Yes | Yes (RBAC Mechanism Only) | Model Checking and Testing [30] | Yes [28] |
| **Secure Tropos [25]** | Yes | Yes | Yes (Constraints) | Yes | Yes | No | Yes [29] |
| **Misuse Cases [21]** | Yes | Yes | No | Yes | No | Testing [11] | No |
| **Abuse Cases [22]** | No | Yes | No | Yes | No | Testing [11] | No |
| **UMLintr [23]** | No | Yes | No | Yes | No | Intrusion Detection [27] | Yes [31] |
| **AsmL [20]** | No | Yes | No | Yes | No | Intrusion Detection [20] | Yes [20] |
| **AsmLSec [17]** | No | Yes | No | Yes | No | Intrusion Detection [17] | Yes [17] |
| **UML State Charts for Security [20]** | No | Yes | No | Yes | No | Intrusion Detection [20] | Yes [20] |
| **Snort Rules [27]** | No | Yes | No | No | No | Intrusion Detection [27] | Yes [27] |
| **STATL [26]** | No | Yes | No | No | No | Intrusion Detection [26] | Yes [26] |

easier to learn for the software practitioners who know UML. The languages based on the notion of extended finite state machine and the ones with syntax close to existing programming languages (e.g., STATL) may also gain popularity more quickly among developers.

*Language used as an Effective Secure Design Language:* UMLsec, SecureUML, and SecureTropos can be used to represent design (including secure design decisions). However, SecureUML is limited to representing only role-based access control notions in a UML class diagram. Static structure of the software cannot be represented in SecureTropos. Our analysis shows that UMLsec can be used more effectively to represent a secure design.

*Use of specified security requirements or attack specifications in a later phase:* Abuse cases [11], Misuse cases [11], UMLsec [23], and SecureUML [28] have been reported to be helpful in guiding testing.

*Tool Support:* Nearly all of the languages have some kind of tool support. This can be a design environment (for UMLintr 23], UMLsec [2], SecureUML [28], UML state charts [20], and SecureTropos [29]) or an intrusion detection system for AsmL [20], UML state charts [20], Snort rules [27], AsmLSec [17], and STATL [26]).

### B. Security Requirements Engineering Processes

Security requirements specification languages are used to specify security requirements. However, a process is needed to derive these requirements. Such a process is usually called a security requirements engineering process and has multiple activities. The main objective of these activities is to identify security requirements that can reasonably ensure security of the developed software. A security requirements engineering process should have the following activities (A1 to A23). This set of activities is based on the activities suggested by various security requirements engineering processes [5, 15, 25, 31-33]. Each of these activities results in information that can help in identifying security requirements. Performing all of these activities might not be absolutely essential and the sequence of these activities is also flexible. We propose that activities A17 to A23 should be performed iteratively till a satisfactory level of security for the requirement specifications is attained.

A1. Specification of high level functional requirements (services provided by the software).
A2. Identification of the deployment environment of the software (including interactions with other software).
A3. Identification of assets and resources of the software.
A4. Valuation of assets and resources of the software.
A5. Identification of users of the software.
A6. Identification of potential attackers of the software.
A7. Identification of attacker's interest in the resources/assets of a piece of software.
A8. Identification of attacker's capabilities.
A9. Specification of use cases.
A10. Specification of misuse scenarios.
A11. Identification of potential threats.
A12. Identification of security goals (derived through discussion with the stakeholders).
A13. Specification of high level security requirements such as security mechanisms to be incorporated.
A14. Specification of security policy and constraints on the working of the software derived by discussing with the stakeholders (e.g., access control policy).
A15. Specification of low level functional requirements (including requirements of security mechanisms e.g., password length).
A16. Definition of exit criteria depending on the security state of the requirement specifications (the security state can be calculated by using security index [4]).
A17. Requirements inspection to identify security errors.
A18. Risk analysis.
A19. Performing security assessment of the requirement specifications (if the assessment meets the exit criteria then exit).

TABLE III. COMPARISON OF SECURITY REQUIREMENTS ENGINEERING PROCESSES

| Security Requirements Engineering Processes | Activities | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 | A22 | A23 |
| Secure Tropos [25] | X | X | | | X | | | | X | | | X | X | X | | | | | | | | | |
| SQUARE [32] | X | X | X | X | | | | | X | X | X | X | X | | | X | X | X | X | | X | X | X |
| CLASP [5, 31] | | X | X | X | X | X | X | X | | | | | X | | | | | | | | | | |
| Haley *et al.* [15] | X | | X | | | | | | | | X | X | X | X | X | | | X | | | | | |
| SREP [33] | | | X | | | | | | | | X | X | X | X | X | | X | X | | | | X | |

A20. Specification of low level security requirements to remove security errors.
A21. Cost/benefit analysis.
A22. Categorization and prioritization of low level security requirements.
A23. Inclusion of selected low level security requirements in the requirement specifications.

These activities can also serve as a comparison criteria for different processes. Table III compares various processes on the basis of these activities. As we can see from Table III, there are many differences among the existing processes with respect to the activities that need to be performed. None of the existing processes perform all the activities. Based on Table III, Security Quality Requirements Engineering (SQUARE) [32] and Security Requirements Engineering Process (SREP) [33] are more comprehensive than other methods with respect to the identified activities that should be performed.

## IV. SUMMARY

Software security has become an important concern in today's software dependent society. Secure software is achieved by employing various secure software development methods. This paper analyzes the current SSDLC processes and SSD methods for requirements (security specification languages and security requirements engineering processes).

The major contributions of this paper are as follows. First, this paper presents a comparison of various SSDLC processes and SSD methods for requirements engineering. Such a comparison can be helpful for software developers in selecting an appropriate SSDLC process, security specification language, and security requirements engineering process according to their needs. The properties identified to compare various software security requirements specification languages can be used to choose a particular language depending on the application requirements. Second, the identified properties can guide software developers in designing a mechanism for translating one language into another. Translating one specification language to another language is useful as the tools developed for the target language can also be used to for the source language (after translation). The following paragraphs summarize our analysis on various SSDLC processes and SSD methods for requirements along with the related open issues.

This paper first proposes seven desirable characteristics of a SSDLC process. These characteristics are then used to compare and contrast the existing SSDLC processes. The analysis of SSDLC processes shows that many of them do not incorporate SSD methods to address security concerns throughout the development cycles (only in some phases). Among the processes, MS SDL and CLASP cover more aspects of secure software development.

We also propose properties that should be present in any effective security specification language. These properties are used to analyze security specification languages. Our findings indicate that UMLsec has most of the identified desirable properties of a security specification language. We observe that many security requirements specification languages have also been used for secure software design. However, there is a need to further develop security requirements specification languages.

As mentioned before, a security requirements engineering process specifies the activities that need to be followed to derive security requirements. We identify 23 activities that are essential to engineer complete and detailed security requirements. We use these 23 activities as a basis to compare five different requirements engineering processes. Our analysis shows that SQUARE incorporates more of these activities than other processes. However, none of the existing processes includes all the activities. It is true that many of these activities are sometimes implicitly performed, and their results are incorporated in the process. However, as with other SSD methods, clear guidance should be provided to the developers, most of whom are not well versed in security issues. As a future work, we recommend to design improved security requirements engineering process based on the 23 proposed activities. This would ensure that all the relevant information has been considered and precise security requirements have been defined (both high and low level).

## ACKNOWLEDGMENT

This research is partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES


[1] G. McGraw, Software Security: *Building Security In*, Addison Wesley, 2006.

[2] J. Juerjens, *Secure Systems Development with UML*, Springer, 2005.

[3] J. Gregoire, K. Buyens, B. De Win, R. Scandariato, and W. Joosen, "On the Secure Software Development Process: CLASP and SDL Compared," In *Proceedings of the 3rd International Workshop on Software Engineering for Secure Systems (SESS'07)*, Minneapolis, MN, USA, IEEE CS Press, 2007, pp. 1-1.

[4] M. Howard and S. Lipner, *The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software*, Microsoft Press, 2006.

[5] OWASP CLASP Project, http://www.owasp.org/index.php/Category:OWASP_CLASP_Project. Last Accessed March 2009.

[6] I. Flechais, C. Mascolo, and M.A. Sasse, "Integrating Security and Usability into the Requirements and Design Process," *International Journal of Electronic Security and Digital Forensics*, Inderscience Publishers, Geneva, Switzerland, 2007, vol. 1, no. 1, pp. 12-26.

[7] A.S. Sodiya, S.A. Onashoga, and O.B. Ajayi, "Towards Building Secure Software Systems," *Issues in Informing Science and Information Technology*, Informing Science Institute, CA, USA, 2006, vol. 3, pp. 635-646.

[8] L. Futcher and R.v. Solms, "SecSDM: A Model for Integrating Security into the Software Development Life Cycle," In *Proceedings of the 5th World Conference on Information Security Education*, Springer, 2007, pp. 41-48.

[9] D. Gilliam, J. Powell, E. Haugh, and M. Bishop, "Addressing Software Security Risk and Mitigations in the Life Cycle," In *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop (SEW'03)*, Greenbelt, MD, USA, 2003, pp. 201-206.

[10] M.A. Hadawi, "Vulnerability Prevention in Software Development Process," In *Proceedings of the 10th International Conference on Computer & Information Technology (ICCIT'07)*, Dhaka, Bangladesh, 2007.

[11] S.d. Vries, "Software Testing for Security," *Network Security*, ScienceDirect, 2007, vol. 3, pp. 11-15.

[12] A. Apvrille and M. Pourzandi, "Secure Software Development by Example," *IEEE Security and Privacy*, IEEE CS Press, 2005, vol. 3, no. 4, pp. 10-17.

[13] M. Essafi, L. Labed, and H.B. Ghezala, "S2D-ProM: A Strategy Oriented Process Model for Secure Software Development," In *Proceedings of the 2nd International Conference on Software Engineering Advances (ICSEA'07)*, Cap Esterel, French Riviera, France, 2007, p. 24.

[14] N. Davis, "Secure Software Development Life Cycle Processes: A Technology Scouting Report", Technical Note CMU/SEI-2005-TN-024, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[15] C.B. Haley, J.D. Moffett, R. Laney, and B. Nuseibeh, "A Framework for Security Requirements Engineering," In *Proceedings of the International Workshop on Software Engineering for Secure Software (SESS'06)*, Shanghai, China, ACM Press, 2006, pp. 35-41.

[16] M. Hussein, M. Raihan, & M. Zulkernine, "Software Specification and Attack Languages," In *Advances in Enterprise Information Technology Security*, D. Khadraoui and F. Herrmann, Eds. IGI Global, 2007.

[17] M. Raihan and M. Zulkernine, "AsmLSec: An Extension of Abstract State Machine Language for Attack Scenario Specification," In *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*, Vienna, Austria, IEEE CS Press, 2007, pp. 775-782.

[18] M.U.A. Khan and M. Zulkernine, "Quantifying Security in Secure Software Development Phases," In *Proceedings of the 2nd IEEE International Workshop on Secure Software Engineering (IWSSE'08)*, Turku, Finland, IEEE CS Press, 2008, pp. 955-960, 2008.

[19] I.V. Krsul, "Software Vulnerability Analysis," Doctoral Dissertation, Department of Computer Sciences, Purdue University, West Lafayette, IN, USA, 1998.

[20] M. Zulkernine, M. Graves, & M.U.A. Khan, "Integrating Software Specification into Intrusion Detection," *International Journal on Information Security*, Springer, 2007, vol. 6, pp. 345-357.

[21] G. Sindre, A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases," In *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'00)*, Sydney, Australia, IEEE CS Press, 2000, pp. 120-131.

[22] J. McDermott and C. Fox, "Using Abuse Case Models for Security Requirements Analysis," In *Proceedings of the 15th Computer Security Applications Conference (ACSAC'99)*, Pheonix, AZ, USA, IEEE CS Press, 1999, pp. 55-64.

[23] M. Hussein and M. Zulkernine, "Intrusion Detection Aware Component-Based System: A Specification-Based Framework," *Journal of System and Software*, Elsevier Science, 2007, vol. 80, no 5, pp. 700-710.

[24] T. Lodderstedt, D.A. Basin, and J. Doser, "SecureUML: A UML-Based Modeling Language for Model Driven Security," In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML'02)*, Dresden, Germany, Springer, 2002, LNCS 2460/2002, pp. 426-441.

[25] H. Mouratidis, P. Giorgini, and G. Manson, "When Security Meets Software Engineering: A Case of Modeling Secure Information Systems," *Journal of Information Systems*, Elsevier Science, 2005, vol. 30, no. 8, pp. 609-629.

[26] S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-Based Intrusion Detection," *Journal of Computer Security*, IOS Press, Amsterdam, 2002, vol. 10, no. 1/2, pp. 71-104.

[27] Snort, www.snort.org. Last Accessed March 2009.

[28] SecureUML Tool, http://www.foundstone.com/us/resources/proddesc /secureumltemplate.htm. Last Accessed March 2009.

[29] Si, http://sesa.dit.unitn.it/sistar_tool/home.php?7. Last Accessed March 2009.

[30] Model-Driven Security with SecureUML, http://www.infsec.ethz.ch/people/doserj/mds. Last Accessed March 2009.

[31] J Viega, "Building Security Requirements with CLASP," In *Proceedings of the 2005 International Workshop on Software Engineering for Secure Systems (SESS'05)*, St. Louis, MO, USA, 2005, pp. 1-7.

[32] N.R. Mead, E. Hough, and T. Stehney, Security Quality Requirements Engineering (SQUARE) Methodology, Technical Report CMU/SEI-2005-TR-009, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[33] D. Mellado, E. Fernandez-Medina, and M. Piattni, "A Common Criteria-Based Security Requirements Engineering Process for the Development of Secure Information Systems," Computer Standards and Interfaces, Elsevier Science, 2007, vol. 29, pp. 244-253.