Security Requirements for the Rest of Us: A Survey

Inger Anne Tøndel Martin Gilje Jaatun Per Håkon Meland

SINTEF ICT
{first.middle-initial.last@sintef.no}

Abstract
Information security requirements are important in all software engineering projects, not only to ensure the correct level of security in the end product, but also to avoid implementing security solutions that turn out to be a bad fit. This paper presents a comparative study of existing methods for eliciting and describing security requirements in software development projects, from the viewpoint of developers without extensive security skills. We sketch the outline of a new approach, and conclude with our most important conviction: The need for a well-balanced amount of security awareness in all software development projects right from the beginning.

1 Introduction

Most software developers are not primarily interested in (or knowledgeable about) security [1, 2]; for decades the focus has been on implementing as much functionality as possible before the deadline, patching the inevitable bugs when it's time for the next release or hotfix [3]. However, it is slowly beginning to dawn on the software engineering community that information security is important also for software whose primary function is not related to security. Security features or mechanisms will typically not have a prominent place in the user interface of such software [4], but a persistent search of menus and configuration options normally uncovers a number of security-relevant items.

The latest CSI/FBI survey [5] reports that 131 respondents estimate their losses due to computer security incidents in 2006 to a total in excess of $50 million. If we eliminate causes such as laptop theft and run-of-the-mill virus infections, we're left with about $30 million; calculating a naïve average, this tells us that the average organization experiences a setback in the region of $230 000 a year. Some of this is probably due to configuration error, but our claim is that a large part of this cost is caused by security flaws in software.

Since security thus is important also for "ordinary" software development projects, we need mechanisms for security requirements elicitation that will be palatable to "regular" software developers and suitable for use in all software development. These mechanisms must be both easy to understand and easy to use! Although formal methods undoubtedly have their merits [6], their use is precluded in this context.

There are numerous publications (e.g.,[7-9]) that underline the importance of security requirements; however, few provide advice that is concrete and specific enough for immediate deployment. Our impression is that a lot is being said about security requirements, but in reality there is still a significant "under-elicitation" in actual software development projects.

We have performed a literature survey in order to identify and describe concrete techniques for eliciting security requirements. For our purposes, a *technique* is a series of well-defined steps that collectively lead to the elicitation of security requirements.

The remainder of this paper is organized as follows: The next section presents our survey, followed by a discussion which sketches a new approach. We then identify further work, and finally offer a conclusion.

2 Security requirements in the literature

In the following, we focus on how security requirements are described and elicited, and on typical artifacts that are produced along the way. We have deliberately left out formal methods because of our aim for a lightweight approach.

2.1 What is a security requirement?

Firesmith [7] claims that most requirements engineers are poorly trained to elicit, analyze, and specify security requirements. This results in security requirements often being confused with architectural security mechanisms that are traditionally used to fulfill requirements, and one ends up making architecture and design decisions. The same problem is recognized by Haley et al.[8], who show that several recognized standards (e.g. the Common Criteria and the NIST Computer Security Handbook) suggest describing security requirements in terms of security mechanisms. However, as they point out, "Defining requirements in terms of function leaves out key information: *what* objects need protecting and, more importantly, *why* the objects need protecting."

Haley et al. define security requirements as "constraints on the functions [that] [...] operationalize one or more security goals." They take issue with those who specify security requirements as high–level security goals, arguing that this makes it difficult to make the requirements specific enough to guide designers and to verify that the requirements are met. For the same reason Haley et al. recommend security requirements that "express what is to happen in a given situation, as opposed to what is not ever to happen in any situation".

The Comprehensive Lightweight Application Security Process (CLASP) [10] states that all requirements should be SMART+ requirements: S̲pecific, m̲easurable, a̲ppropriate, r̲easonable and t̲raceable. No examples are however given as to what a typical security requirement should look like. Such examples are given by Firesmith [7], who suggests dividing security requirements into different categories, like identification, integrity and privacy requirements. A security requirement is defined as "a detailed requirement that implements an overriding security policy", e.g.:

- "The application shall identify all of its client applications before allowing them to use its capabilities." (Identification)
- "The application shall not allow unauthorized individuals or programs access to any communications." (Privacy)

Examples of security requirements are also given by Haley et al.[8]: "The system shall provide Personnel Information only to members of Human Resources Dept." By expressing security requirements in relation to specific functional requirements they claim that they are able to achieve enough specificity to guide designers and make it possible to verify that the requirements are actually fulfilled.

These examples notwithstanding, we have not found a universally accepted definition of "security requirement" in the literature. We will return to this point in the following sections.

2.2 How to proceed

Below we present some of the major approaches when it comes to security requirements engineering. A birds-eye-view of what is included in the different approaches can be found in Table 1.

Table 1: Comparison of different approaches

	Definitions	Objectives	Misuse/threats	Assets	Coding standards	Categorize/prioritize	Inspect/validate	Process planning
SQUARE [11]	x	x	x			x	x	
Haley et al. [8]		x	x	x			x	
Boström et al. [12]			x	x	x	x		
CLASP [10]		x	[1]	x			x	
Microsoft [3, 13, 14]		x	(x)[2]	(x)[2]				x
Apvrille et al. [15]		x	x			x		
Fernandez [16]			x					
van Wyk et al. [17]			x					
Peterson [9]			x					

2.2.1 Categories for comparison of approaches

The surveyed approaches have been presented in Table 1 based on what tasks are recommended as part of the requirements phase. The division into task categories has been made based on the survey, with a level of detail appropriate to include all recommended tasks, while at the same time avoiding complexity. For a task to be part of an approach, it has to be considered important enough to be explicitly included in the description of the approach. The task categories are as follows:

- **Definitions:** Central concepts like attack, asset and authentication are defined as part of the requirements phase.
- **Objectives:** High-level business-centric goals are identified.
- **Misuse/threats**: Identification of misuse or threat scenarios are part of the process.
- **Assets:** Assets are identified and their value estimated.
- **Coding standards:** Requirements to the coding standard are identified, i.e. what language to use or what type of functions to avoid.
- **Categorize/prioritize:** Categorization and prioritization of security requirements are performed.
- **Inspect/validate:** Requirements are inspected/validated for completeness, lack of conflicts or other matters.
- **Process planning:** Planning of security activities is part of the requirements phase.

[1] Not part of CLASP green field roadmap
[2] Part of threat modelling specified for the design phase.

2.2.2 Description of security requirement approaches

The Software Engineering Institute (SEI) has presented SQUARE (Secure Quality Requirements Engineering) [11], consisting of nine main steps:

1. Agree on definitions
2. Identify security goals
3. Develop artifacts
4. Perform risk assessment
5. Select elicitation technique
6. Elicit security requirements
7. Categorize requirements
8. Prioritize requirements
9. Requirements inspection

SQUARE is based on interaction between requirements engineers and the stakeholders of an IT project, where facilitation by a requirements engineering team is of major importance. Although SQUARE is a methodology intended to be used in the early phases of software development, it should be noted that specifically steps 3 and 4 require that some sort of design activity has already taken place, since the artifacts that are to be developed/identified include system architecture diagrams in addition to use case and misuse case diagrams and attack trees, and since all vulnerabilities and threats are expected to be already identified.

As seen from the number of steps included, SQUARE covers most of the tasks used for comparison of initiatives (see Table 1). Stating what is included and not included in SQUARE is however not straightforward, since several techniques can be chosen for the different steps. Identification of assets is not included in the main SQUARE methodology, but one of the case studies [11] employs Survivable System Analysis, which includes identification of essential assets and services.

Haley et al. present a framework consisting of four main steps [8]:

1. Identify functional requirements
2. Identify security goals – includes identification of assets, threats, management principles and business goals
3. Identify security requirements
4. Verification of the system

The authors suggest using Jackson’s problem diagrams [18], a notation that is not well known by most developers. Although they also suggest including formal argumentation in the verification step, it is likely that the main methodology would work also if using more informal techniques. It is also worth noting that iteration between requirement and design activities are an important part of the framework. Fulfilling a security requirement may lead to new assets – resulting in new security requirements.

Boström et al. [12] consider security requirements engineering in a different context – Agile development with a focus on XP practices. They suggest a seven-step approach to identify and handle security requirements:

1. Identification of critical assets
2. Formulation of abuser stories
3. Abuser story risk assessment
4. Abuser story and user story negotiation
5. Definition of security-related user stories
6. Definition of security-related coding standards

7. Abuser story – Countermeasure cross-checking

This process is based on XP user stories and extends them to include security requirements. The main ideas should however be relevant also when using other types of development processes.

One of the major initiatives when it comes to secure software development lifecycles is CLASP [10] which specifies a set of process pieces that can be integrated into any software development process. Maybe because of CLASP's general nature, the steps outlined are not as concrete as e.g. those described by Boström et al. [12]. The steps that are placed in the requirements phase in the CLASP green field roadmap (the steps recommended for new software development) are:

- Document security relevant requirements – Includes identifying business requirements, functional security requirements and dependencies, determine risk mitigations and resolve deficiencies and conflicts
- Identify resources and trust boundaries – Includes network level design and data resources

Several authors suggest use cases as a starting point for identifying security requirements. Though not included in the CLASP green field roadmap, CLASP describes "detail misuse cases" as a possible requirements activity. Fernandez [16] points out that use cases can be used to determine the needed rights for each actor, and that based on the use cases one can consider possible attacks. Peterson [9] suggests use cases and misuse cases as a basis for security requirements, though he states that additional non-functional requirements may be needed. van Wyk and McGraw [17] suggest using abuse cases. Abuse cases is also one of McGraw's "touchpoints" [19] for the "requirements and use cases" phase, together with "security requirements". Little detail is however given on the touchpoint "security requirements".

Lipner and Howard describe the Microsoft Trustworthy Computing Security Development Lifecycle [14], focusing on planning security activities that are to take place later in the software development lifecycle. They also state that key security objectives should be identified together with security feature requirements that are based on customer demand and compliance with standards. Other security feature requirements will however be identified as part of threat modeling which is performed in the design phase. This corresponds with Meier's (also from Microsoft) description of web application security engineering [3], which describes identification of security objectives as an important activity before threat modeling takes place. Threat modeling [13], though part of the design phase, also has some steps that may be considered for the requirements phase:

- Identification of use scenarios
- Identification of assets
- Identification of threats
- Identify dependencies

Apvrille and Pourzandi [15] suggest the following steps for the phase "Security requirements and analysis":

- Security environment and objectives
- Threat model
- Security policy - includes prioritizing according to the information's sensitivity
- Risk evaluation

The steps are intended to be performed by the developers themselves, but are not explained in much detail.

2.2.3 What has been left out?

Because of our aim for a lightweight approach, we have deliberately left out formal methods in our survey. Unfortunately, this also means that the work by e.g. Chivers [20] on automated risk analysis falls outside the scope of our paper, since this approach relies on formal modeling. The same goes for the work of van Lamsweerde on intentional anti-models [21].

2.3 Security requirements artifacts

This subsection presents typical artifacts that are produced when eliciting security requirements.

2.3.1 Misuse cases

Misuse cases have been suggested by Sindre and Opdahl [22], and have been used in several research and industrial projects. They extend the regular UML use case diagrams with negative use cases (misuse cases) that specify behavior not wanted in the system. Use cases can *mitigate*[3] misuse cases, meaning that a use case can be a countermeasure against a misuse case - thereby reducing the chances that the misuse case succeeds. Misuse cases can *threaten*[3] a use case meaning that the use case is exploited or hindered by a misuse case.

McDermott and Fox [23] have suggested abuse cases for expressing threats using the standard UML use case notation. In their approach, abuse cases are kept in separate models. Firesmith [24] has discussed the concept of security use cases. Security use cases are similar to ordinary use cases, but represent specific security functionality (countermeasures). The differences between misuse cases, abuse cases and security use cases are further discussed by Sindre and Opdahl [22] and by Røstad [25]. Røstad has also suggested extending the notation for misuse cases to be able to represent vulnerable functions and discern between insiders and outside attackers.

2.3.2 Abuser stories

Abuser stories were first introduced by Peeters [26] as an agile counterpart to abuse cases. An abuser story is a brief and informal description of how an attacker may abuse the system at hand. Abuser stories are not yet widely used, but Boström et al. [12] demonstrate how abuser stories can be used to extend the XP “planning game”, and Kongsli [27] reports how “misuse stories” have been used successfully as an element in addressing security in a software development organisation.

2.3.3 Attack trees and threat trees

Attack trees [28] represent attacks against a system in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes. The diagrams can be used both in the requirements and design phases. Trees can be represented graphically or can be written in outline form. It is also possible to add information on e.g. cost of attack.

Microsoft uses threat trees in their threat modeling activity. Threat trees are quite similar to attack trees, but also add mitigated conditions to the identified threats [29].

2.3.4 Softgoal Interdependency Graphs

The NFR (Non-Functional Requirements) framework [30, 31] and its associated Softgoal Interdependency Graphs (SIGs) provide a systematic approach to identify system goals, decomposing them into lower level subgoals, and ultimately into potential solutions (operationalizations) [32]. In comparison to goals which are satisfied absolutely when their subgoals are identified, softgoals are “goals that do not have a clear-cut criterion for their satisfaction” [33], i.e. they are satisfied when there is “sufficient positive and little negative evidence for this claim” [33]. The NFR framework and SIGs are not yet broadly adopted in industry, but their usefulness has been demonstrated in several complex software systems [32].

[3] *Mitigate* and *threaten* are relationships between misuse cases and use cases defined by Sindre and Opdahl

Cleland-Huang et al. [32] have suggested extending SIGs to represent security requirements and threats to these requirements. In their approach it is possible to add information used to estimate potential loss, and use this to find which safeguards are most efficient (calculate annual loss savings).

2.4 Comparisons

Our survey reveals that there is no common agreement on what a security requirement is. More specifically, the various approaches do not agree to what extent concrete security measures should be stated in the requirements. SQUARE requires some design work as background material for security requirement elicitation. Microsoft considers some of the other approaches' requirement activities as part of the design phase. CLASP considers determining risk mitigations to be a requirements activity, while others consider this part of design. One of the reasons for these differences may be that the focus on iteration is different. Agile development, being the focus of Boström et al., is very much iterative in nature, in contrast to what seems to be the case for e.g. SQUARE.

The approaches also differ on the level of detail provided as to how to perform the tasks. As an example, CLASP's and Apvrille and Pourzandi's s suggestions are more general and thereby less concrete than those of Boström et al. who focus on the XP development process. The approaches also differ when it comes to the level of expert knowledge required. SQUARE relies on a requirements team that facilitates the process. Haley et al. suggest artifacts that probably are too complex to be used by the regular developer. The approaches of e.g. Boström et al., Microsoft and CLASP are more lightweight.

The surveyed approaches are presented in Table 1 based on the tasks that they recommend. The underlying assumption is that the tasks that are often recommended are the most important ones. We are aware of methodical limitations: The activities may be weighted differently in the different approaches, some approaches are more accepted than others, and the comparison criteria may be inadequate. Table 1 shows that identification of misuse/threats is commonly recommended. This task is also supported by many of the typical artifacts used for security requirements engineering. Many of the approaches also recommend identifying objectives and assets. Our impression is therefore that these three tasks are of high importance to security requirements engineering, though there are differences in recommendations as to how to perform them.

3 Discussion

Our aim is to identify a set of security requirement techniques that are convenient enough to be adopted by the average software developer. The resulting approach will probably not be able to identify *all* security requirements – our focus being the most critical ones. In an iterated development process it will however be possible to revise and add further requirements at later stages.

In our survey we found that identification of threats, assets and security objectives are commonly recommended security requirement tasks. We therefore direct our main focus here. Additionally, the lack of common understanding on what a security requirement is indicates that developers need concrete examples as to how to describe and document such requirements. In the following, we sketch a lightweight method for security requirements engineering. In our recommendations we focus on easy-to-do tasks based on own experience and recommendations in the surveyed literature.

3.1 Security objectives

With security objectives we mean the high-level requirements/goals that are most important to customers, and the requirements that must be met to comply with relevant legislation, policies and standards. Knowledge of these is necessary to decide where to put the main effort. A security objective will typically be achieved by one or more detailed security requirements, in accordance with Haley et al. [8]. We suggest dividing this activity into two main steps: Identification of the customer's need for security, and identification of relevant legislation, policies, standards and best practices that apply to the system/module. Step-by-step guides should be given on how to prepare for customer meetings that focus on security issues, and where to look for requirements from legislation, policies and standards, based on e.g. work done by Verdon [34].

3.2 Asset identification

To be able to elicit and prioritize security requirements, it is important to identify *which* properties of *what* assets are most important to protect. We suggest to look at the value of the assets both from the customer's and the system owner's point of view, and from the view of an attacker - since, as also pointed out by Haley et al. [8], different actors' view of an asset are not directly related. To identify assets one can use functional requirements[4] of the system, possibly combined with brainstorming techniques. For each asset one should then make a judgment regarding the different stakeholders' priorities of the confidentiality, integrity and availability of this asset. This could be formalized by calculating the risk to an asset in the conventional manner (risk = impact × probability), but since both impact and probability are frequently difficult to estimate, it may be just as useful to choose one of a predefined set of qualitative categories directly, e.g. high, medium or low.

3.3 Threat analysis

We recommend focusing on the most important assets and using predefined threat categories such as STRIDE (<u>S</u>poofing, <u>T</u>ampering, <u>R</u>epudiation, <u>I</u>nformation disclosure, <u>D</u>enial of service, <u>E</u>levation of privilege) recommended by Microsoft [13]. The most important threats identified should then be further elaborated using e.g. attack trees. This way the number of threats analyzed, and thereby the effort needed, will be reduced – but always with the risk of neglecting something. We recommend attack trees, since they are not closely coupled to any specific development process, while still being quite common and well known. However, if developers have already modeled the system using UML use cases, misuse cases may be a better alternative. In the same way, agile development projects may find abuser stories best suited. The attack trees (or any other artifact) created at this stage should not make assumptions regarding design decisions that have not been made yet. Their level of detail can however be increased later in the development process.

3.4 Documentation of security requirements

There is a need for good descriptions of what a security requirement is. We recommend describing requirements that are focused on *what* should be achieved – not *how*. This corresponds to the recommendations and examples made by Firesmith [7]. Security requirements are however different from high-level security goals, as pointed out by Haley et al. [8] since they provide more details regarding e.g. "*who* can do *what*, *when*", without specifying exactly which mechanisms to use. When it comes to our techniques, "security objectives" and "asset identification" will typically result in "policy level" requirements such as: "The privacy of the users of the service must be protected". This may then result in a number of security requirements like "The personal

[4] One should be aware that using functional requirements as a starting point comes with the risk of not covering assets such as reputation. We however believe that for this lightweight approach, such assets can be indirectly covered by looking at the different actors' value of the assets.

profile of users must be protected in transit"[5]. The mechanism(s) used to achieve this, e.g. an encryption algorithm, will then be selected during design.

Our goal is creating a method where developers, by following step-by-step guides, "automatically" elicit security requirements of good quality. These requirements should then be documented to ensure visibility, show which security requirements have high priority, and arrange for traceability and follow-up of requirements. We suggest describing all security requirements in one place, preferably as part of a general requirements document, in order to be able to retain an overview of all requirements. This also makes it natural to "tie" security requirements to the relevant functional requirements, as suggested by Haley et al. [8].

3.5 Does "lightweight" equal "worthless"?

As already mentioned, we are aware of the fact that a lightweight approach will not be bullet-proof, but if "normal" applications constitute (say) 90% of the global development effort, even a 5% reduction in security vulnerabilities will have a significant impact. If developers can be given assistance in choosing which techniques to use, and then get detailed, step-by-step guidance in applying these techniques, we are a step closer to the holy grail of "Secure Software" – if for no other reason, for making developers at least *think* about security requirements in their development projects.

4 Further work

No matter how formally pleasing or academically correct a method may be; if it is not being used by the intended audience, it is not good enough. Instead of creating "yet another methodology", we have looked at what is currently considered good practice in security requirements engineering, and selected a low-threshold set of techniques that should be palatable to the average software developer.

We realize that it is not possible to alter development methodologies overnight, and thus a success criterion will be education of software developers. Teaching old dogs new tricks is always difficult, which is why we advocate introducing *secure software engineering*[6] as a mandatory curriculum component for all software engineering students. Especially in this context we are convinced that a lightweight approach is the only way to go: Unless techniques are easy to understand and easy to use, even impressionable students will shy away from them.

We plan to refine the ideas presented above into a full-fledged lightweight security requirement elicitation method with step-by-step guidance. We will then validate and improve this method by performing more practical studies, e.g. use it in development performed by students, in software research projects and in industrial software development projects.

5 Conclusion

Security must be considered from the very beginning of every software development project. However, creating the right security requirements for a project is not an easy task, and in many cases, requirements of this type are difficult to trace and test during the development process because of their "non-functional" nature. Several initiatives to aid the process of specifying security requirements exist, but we have little knowledge about their usefulness, and there is a

[5] This requirement could probably be further detailed by stating e.g. when this would happen – placing it in context with the functional requirements of the system.
[6] Note that this is something quite different from the "Computer Security" courses that already are a part of many undergraduate CS/CEng programs.

lack of detailed description on how they are to be used in practice. Also, most of them have been created with traditional security-critical projects in mind, which means that these methods are not always appropriate, especially for developers without the proper security background. We have highlighted the need for a lightweight security requirements elicitation approach that should be used in *every* project, without consuming more resources than necessary.

Acknowledgements

The authors would like to thank Prof. Nuseibeh of the Open University, SINTEF colleagues, and the anonymous reviewers for useful comments.

References

[1] P. Coffee, "Security Onus Is on Developers," eWeek, 2006, http://www.eweek.com/article2/0,1895,1972593,00.asp.

[2] H. Mouratidis, P. Giorgini, and G. Manson, "When security meets software engineering: a case of modelling secure information systems," *Information Systems*, vol. 30(8), pp. 609-629, 2005.

[3] J. D. Meier, "Web application security engineering," *Security & Privacy Magazine, IEEE*, vol. 4(4), pp. 16-24, 2006.

[4] S. Furnell, "Why users cannot use security," *Computers & Security*, vol. 24(4), pp. 274-279, 2005.

[5] L. A. Gordon, et al., "CSI/FBI Computer Crime and Security Survey," CSI, 2006, http://www.gocsi.com/forms/fbi/csi_fbi_survey.jhtml.

[6] J. F. Davis, "The affordable application of formal methods to software engineering," Atlanta, GA, United States, 2005.

[7] D. G. Firesmith, "Engineering Security Requirements " *Journal of Object Technology*, vol. 2(1), pp. 53-68, 2003.

[8] C. B. Haley, et al., "Security Requirements Engineering: A Framework for Representation and Analysis," *(to appear in) IEEE Transactions on Software Engineering*, 2007.

[9] G. Peterson, "Collaboration in a Secure Development Process Part 1," *Information Security Bulletin*, vol. 9(June), pp. 165-172, 2004.

[10] "The CLASP Application Security Process," Secure Software Inc., 2005, http://www.securesoftware.com/solutions/clasp.html.

[11] N. R. Mead, E. D. Houg, and T. R. Stehney, "Security Quality Requirements Engineering (SQUARE) Methodology," CMU/SEI CMU/SEI-2005-TR-009, 2005.

[12] G. Boström, et al., "Extending XP practices to support security requirements engineering," presented at Proceedings of the 2006 international workshop on Software engineering for secure systems (SESS '06), Shanghai, China, 2006

[13] P. Torr, "Demystifying the threat modeling process," *Security & Privacy Magazine, IEEE*, vol. 3(5), pp. 66-70, 2005.

[14] S. Lipner and M. Howard, "The Trustworthy Computing Security Development Lifecycle," Updated version of paper presented at the 2004 Annual Computer Security Applications Conference, 2005, http://msdn2.microsoft.com/en-us/library/ms995349.aspx.

[15] A. Apvrille and M. Pourzandi, "Secure Software Development by Example," *Security & Privacy Magazine, IEEE*, vol. 3(4), pp. 10-17, 2005.

[16] E. B. Fernandez, "A methodology for secure software design," presented at Intl. Symposium on Web Services and Applications (ISWS'04), Las Vegas, NV, 2004

[17] K. R. van Wyk and G. McGraw, "Bridging the gap between software development and information security," *Security & Privacy Magazine, IEEE*, vol. 3(5), pp. 75-79, 2005.

[18] J. G. Hall, L. Rapanotti, and M. Jackson, "Problem frame semantics for software development," *Software and Systems Modeling*, vol. 4(2), pp. 189-198, 2005.

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

[20] H. Chivers, "Information modeling for automated risk analysis," in *Communications and Multimedia Security, Proceedings*, vol. 4237, *Lecture Notes in Computer Science*. Berlin: Springer-Verlag Berlin, 2006, pp. 228-239.

[21] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," presented at 26th International Conference on Software Engineering, 2004.

[22] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," *Requirements Engineering*, vol. 10(1), pp. 34-44, 2005.

[23] J. McDermott and C. Fox, "Using abuse case models for security requirements analysis," presented at Computer Security Applications Conference, Phoenix, AZ, USA, 1999.

[24] D. G. Firesmith, "Security Use Cases," *Journal of Object Technology*, vol. 2 (3), pp. 53-64, 2003.

[25] L. Røstad, "An extended misuse case notation: Including vulnerabilities and the insider threat," presented at The Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06), Luxembourg, 2006

[26] J. Peeters, "Agile Security Requirements Engineering," presented at Symposium on Requirements Engineering for Information Security, Paris, France, 2005

[27] V. Kongsli, "Towards Agile Security in Web Applications," presented at OOPSLA 2006 Conference Companion, 2006

[28] B. Schneier, "Attack Trees - Modeling security threats," *Dr. Dobb's Journal*(July), 2001.

[29] F. Swiderski and W. Snyder, *Threat Modeling*: Microsoft Professional, 2004.

[30] L. Chung and B. A. Nixon, "Dealing with non-functional requirements: Three experimental studies of a process-oriented approach," presented at International Conference on Software Engineering, 1995

[31] L. Chung, "Dealing with Security Requirements During the Development of Information Systems," presented at 5th Int. Conf. Advanced Information Systems Engineering (CAiSE'93), 1993

[32] J. Cleland-Huang, et al., "A Goal-Oriented Approach to Identifying and Mitigating Security Risks," presented at International Symposium on Secure Software Engineering, Washington, DC., 2006

[33] J. Mylopoulos, L. Chung, and E. Yu, "From object-oriented to goal-oriented requirements analysis," *Communications of the ACM*, vol. 42(1), pp. 31-37, 1999.

[34] D. Verdon, "Security policies and the software developer," *IEEE Security & Privacy*, vol. 4(4), pp. 42-49, 2006.