2010 International Conference on Availability, Reliability and Security

Security modeling and tool support advantages

Egil Trygve Baadshaug*, Gencer Erdogan* and Per Håkon Meland†
*The Norwegian University of Science and Technology
7491 Trondheim, Norway
{egiltryg,erdogan}@stud.ntnu.no
†SINTEF ICT, System development and security
7465 Trondheim, Norway
per.h.meland@sintef.no

***Abstract*—Security modeling is an important part of software security, especially when it comes to making security knowledge more easily accessible. The purpose of this paper is to give an overview of some of the current approaches to graphical security modeling and present an initial study related to benefits of tool support. Our working hypothesis is that specialized security modeling tools will substantially outperform more general, prevailing tools, and we have sought indications of evidence for this claim. The study consisted of the following steps; (1) Investigate state-of-the-art security modeling formalisms and tools, (2) Select a security modeling formalism for further analysis and implement dedicated tool support for it, (3) Perform testing related to usability and performance aspects, comparing the tool to a general purpose drawing/modeling tool, and (4) Compare and analyze the results. The study included ten test subjects with a similar background and education, and we got clear indications that our hypothesis is valid.**

***Index Terms*—security modeling; tool support; experimental study;**

I. Introduction

Security modeling is a collective term for modeling techniques that are used for identifying threats, vulnerabilities and/or countermeasures, which further leads to prevention of security problems at an early stage of software development. This kind of modeling is a part of the software security field, where the overall motivation is to engineer software so that it continues to function correctly under malicious attack. The software security field is quite new, and some of the first books and academic classes about software security appeared in 2001, which presented systematic ways for creating secure software [1]. Since then, there has been proposed several software security practices, such as the Security Development Lifecycle (SDL) by Microsoft [2], the Risk Management Framework [3], and the CORAS methodology [4]. All of these security practices include elements of security modeling.

According to Meland et al. [5], security modeling can be handled from three distinct viewpoints; vulnerabilities and causes, threats and attacks, and countermeasures. The first viewpoint, vulnerabilities and causes, describes the underlying issues of what causes the vulnerabilities. The second viewpoint, threats and attacks, describes the system and the vulnerabilities from an attackers point of view. This means looking at how a system can be attacked and the vulnerabilities exploited. The third and final viewpoint, countermeasures, describes how attacks can be mitigated. Today there are various modeling languages and formalism that can be used to represent these viewpoints, ranging from informal text descriptions to standardized notation. We believe that the greatest benefit comes from the ones with a clearly defined graphical syntax, purpose and usage. In most cases, a corresponding textual syntax is needed in order to explain what cannot be explained through the graphical model.

An assumption mentioned in [6] is that *there must be good tool support that enhances security during development, preferably integrated into the current development tools.* Ardi et al. [7] claim that *security modeling is usually done with general-purpose drawing tools* and that *standardized modeling methods and tools are needed*. This will again encourage sharing of security models between security experts and developers, thus contribute to a stronger security awareness and improved knowledge management.

The purpose of this paper is to give an overview of some the current approaches to graphical security modeling and present an initial study related to benefits of tool support. Our working hypothesis is that specialized security modeling tools will substantially outperform more general, prevailing tools, and we have sought indications of evidence for this claim. The study consisted of the following steps; (1) Investigate state-of-the-art security modeling formalisms and tools, (2) Select a security modeling formalism for further analysis and implement dedicated tool support for it, (3) Perform testing related to usability and performance aspects, comparing the tool to a general purpose drawing/modeling tool, and (4) Compare and analyze the results. The study included ten test subjects with a similar background and education, and we got clear indications that our hypothesis is valid.

II. Security modeling languages and formalisms

This section gives a state-of-the-art overview and shows the diversity of graphical security modeling languages and formalisms. It has been used a basis for selecting a formalism for the rest of the study.

One of the earliest and perhaps most well-known formalism related to software security is the attack tree [8]. The attack tree is a formal way of representing the security of a system based on attacks. The attacks are represented in graphs or tree structures.

978-0-7695-3965-2/10 $26.00 © 2010 IEEE
DOI 10.1109/ARES.2010.11

Related to root-cause analysis, we have Vulnerability Cause Graphs (VCGs) [9], which are visual representations of causes that leads to vulnerabilities. VCGs are mainly used for educational purposes, and not performed during development. Security activity graphs (SAGs) are derived from VCGs via a formal process described by Ardi et al. [10]. SAGs are used to discover the activities needed in a development process in order to mitigate specific vulnerabilities. In addition, the use if SAGs is a helpful modeling technique when looking for potential tradeoffs between different security activities [11], [12].

Security Goal Indicator Trees (SGITs) [13], [14], [15] are used for goal-driven software security inspection. SGITs are said to be a "top-down" security inspection method since it starts by looking at a security goal that an object, i.e. a software artifact, under inspection needs to achieve.

An Indicator Specialisation Tree [13], [14], [15] contains specific information on how and where to look for a general indicator, thus, it is a specialization of an indicator (an indicator is either a positive or a negative indicator. An indicator specialization tree can be expanded (further specialized) independently from the SGIT(s) it belongs to. In this way, the complexity of an SGIT is reduced and modularity is introduced into an SGIT.

Guided checklists are derived from SGITs to inspect software artifacts for defects that may hinder the fulfillment of given security goals. Guided checklists differ from traditional checklists (standard checklists and focused checklists) by having a strict inspection process and by having much more detailed guidance for an inspector. In addition, they are shown as flowcharts to emphasize in which order the questions have to be answered [15].

Vulnerability Inspection Diagrams (VIDs) [16], [14], [15] are a formal and structured way of modeling procedural instructions on how to carry out a software security inspection for detecting the presence or absence of a specific vulnerability class. This vulnerability-directed inspection is executed on software artifacts; e.g. source code, system architecture or requirements. In contrast to SGITs, VIDs are used as a "bottom up" approach for security inspection.

The misuse case [17] is a well-known technique to bring forth and elicit security requirements and threats. It is an extension of the UML use case diagram, which is a popular language used to capture, document, specify and communicate the functional requirements of a system [18], [19].

UMLsec [20], [21] extends UML, but does not add any new kind of graphical model notation, but simply adapts all the graphical notation directly from UML having a clear software security focus.

The CORAS graphical modeling language [22] has five different diagrams; asset diagrams (also known as asset overview diagrams), threat diagrams, risk diagrams, treatment diagrams and treatment overview diagrams. These languages are part of the CORAS method [4], which is used for conducting security risk analysis, specially developed to support structured brainstorming for risk identification, risk estimation and risk treatment [23].

Softgoal Interdependency Graphs (SIGs) are a part of the Non-Functional Requirement Framework (NFR Framework) defined in Chung et al. [24]. SIGs are used to graphically model the requirement activities initiated by the NFR Framework.

Secure Tropos makes it a notation used to analyze the security needs of the stakeholders and the system. This notation is an extension to Tropos [25], which is an agent oriented software development methodology, by adding elements related to "Secure Goal", "Secure Task", "Secure Resource" and "Security Constraint".

III. Security modeling tools

For each of the various security modeling languages and formalisms mentioned in the previous section, there is in most cases at least one tool that supports it. Here, we briefly show examples of such tools that are currently either commercial or freely available.

Secur*IT*ree [26] is a risk modeling software that is mainly based on attack trees and attack tree analysis. According to Secur*IT*ree's website [26] it can, among other things, identify which vulnerabilities require mitigation and which do not, and validate that the proposed countermeasures will be effective and cost effective.

AttackTree+ [27] is used to model the threats against a system in a graphical manner by using attack trees. AttackTree+ also allows users to model the probability of a successful attack, to define indicators that quantify the cost of an attack, the operational difficulty in mounting the attack and any other relevant quantifiable measure that may be of interest.

SeaMonster [5] is a graphical security modeling tool based on a set of Eclipse frameworks. It can be used to model attack trees, misuse case diagrams, security activity graphs and vulnerability cause graphs. This software can be downloaded from http://sourceforge.net/projects/seamonster/.

Goat is another graphical modeling tool supporting vulnerability cause graphs and security activity graphs. Though it is currently not publicly available, it will be in the future from http://www.shields-project.eu/.

CORAS language editor is fully based on the graphical modeling language defined in CORAS [22]. This software can be downloaded from http://coras.sourceforge.net.

Non-Functional Requirements Modeling Tool [28] is a modeling tool that supports NFR [24] modeling using the the softgoal interdependency graph (SIG) notation.

Si*-tool [29] is used for drawing Secure Tropos models and for performing formal analysis of Secure Tropos specifications. The tool is provided as an Eclipse plugin and can be downloaded from http://sesa.dit.unitn.it/sttool/.

UMLsec Tool(s): There are not any modeling tools that are specifically created for UMLsec. However, there are some guidelines of how to extend a UML editor to include UMLsec, and there are some tools that have implemented some of the UMLsec characteristics [30].

IV. Selecting a security modeling language

In our study, we wanted to implement tool support for a language which did not already have that. By comparing sections II and III, you can see that there is currently no dedicated tool support for SGITs, Indicator Specialisation Trees, Guided checklists and VIDs. Having these candidates, we defined the following criteria which to select from:

C.01 - Reducing complexity: Does the security modeling technique reduce the complexity of the underlying activity? (i.e. the activity which the security modeling technique initiates)?

C.02 - Increasing knowledge: Does the security modeling technique contribute to increase the user's security-specific knowledge?

C.03 - Increasing efficiency: Does the security modeling technique increase the efficiency (time spent) of the underlying activity of improving software security?

C.04 - Reusability: Are the models (i.e. the diagrams), which are created by the security modeling technique, suitable for reuse?

C.05 - Easy legend: Does the security modeling technique have an easy-to-understand legend?

C.06 - Independence: Can the security modeling technique be used independently from other security modeling techniques (e.g. indicator specialisation trees and guided checklist are derived from SGITs)?

The results in table I shows that VIDs support all the above-mentioned criteria, and was therefore selected for implementation based on an evaluation by the authors:

- VID fulfills criteria C.01 by firstly, having the ability to include other VIDs as *procedure calls*, which by itself is another VID. This makes referring to other VIDs easier, and makes the diagram less complex. Secondly, it provides a short and a detailed version of procedure calls, which may be used by a security expert and a security novice respectively. The reduction of complexity in this context is primarily for the security expert that will not need to go through all the details.
- VID fulfills criteria C.02 by providing detailed descriptions of a VID through procedure calls. This directs the user to go deeper into each VID that is indicated by the procedure call, and thus gain security-specific knowledge along the process.
- VID fulfills criteria C.03 by, as mentioned in the first point, providing a short and detailed version of a procedure call (VID). By e.g. distributing the short version of a procedure call to security experts, and the detailed version of a procedure call to security novices, prevents unnecessary lag time of the process.
- VID fulfills criteria C.04 by providing a formal, generic and modular syntax. The first two properties makes it possible to inspect for a given vulnerability, independently from the environment in which the vulnerability may be in. By environment, it is meant e.g. the programming language (given that nothing else has been specified), the operative system, etc. The last property, which is modularity, supports the reuse of the diagrams by making sure that an update or a modification in a procedure call, do not affect the underlying VID.
- VID fulfills criteria C.05 by, simply, providing an easy to understand legend that does not consist of many different items.
- VID does not rely on other security modeling techniques, thus it fulfills criteria C.06.

The new VID modeling tool was created based on SeaMonster mentioned in section III. The reason for doing so was that SeaMonster is freely available and open source, and so that VID support can be easily added to SeaMonster instead of creating yet-another-tool.

V. Execution of the experiment

The test subjects were ten master level computer science students familiar to graphical modeling tools. Participation was voluntary and the experiment itself consisted of two parts:

Task 1: The purpose of this task was to introduce the test subjects to VIDs and their purpose, and get feedback on the usability of the VID tool. This was done by showing the test subjects examples of VIDs and the palette. They were then instructed to create the VID shown in figure 1, and then answer a few questions on usability and desired functionality. The test subjects were encouraged to speak loud about the actions he/she executes during the task. This makes it possible for the observer to hear how the user thinks and to note where and why the user gets frustrated.

Task 2: The goal of the second part (task 2) is to measure the efficiency of VID tool compared to a general purpose drawing/modeling tool. This was done by first instructing the tester to create the VID that is shown in figure 2 using a tool they were familiar with. After that, they were to use the VID tool, performing the same task.

In both cases, a stopwatch was used to measure the time it took for the test subjects to complete the tasks.

During the preparation of the case study, it was assumed that most of the users would choose to use Microsoft Visio (hereby Visio) in task 2. With this in mind, the authors created a palette in Visio containing the VID elements. This was done for two reasons: Firstly, to make the execution of task 2 easier, given that the tester would choose to use Visio. Secondly, to have more or less equal level of difficulty in task 1 and task 2. Otherwise, the test subjects would either have to make the palette, or draw each VID element using a drawing tool. This would complicate the task and possibly demotivate the test subjects in completing the task.

VI. Result and discussion

Task 1: The feedback in this task showed that there were some issues related to text breaking the boundaries of the VID elements, but this issue was present in both tools. Fixing these issues would probably increase the efficiency since the test subjects spent some time trying to manually fix this. Another typical thing that was pointed out was that the drag-and-drop

TABLE I
COMPARISON TABLE OF DIFFERENT SECURITY MODELING TECHNIQUES

Criteria / Modeling technique	Reducing complexity	Increasing knowledge	Increasing efficiency	Re-usable	Easy legend	Independent
VID	✓	✓	✓	✓	✓	✓
SGIT		✓	✓	✓		✓
Indicator specialisation tree		✓	✓	✓		
Guided checklist		✓	✓	✓		

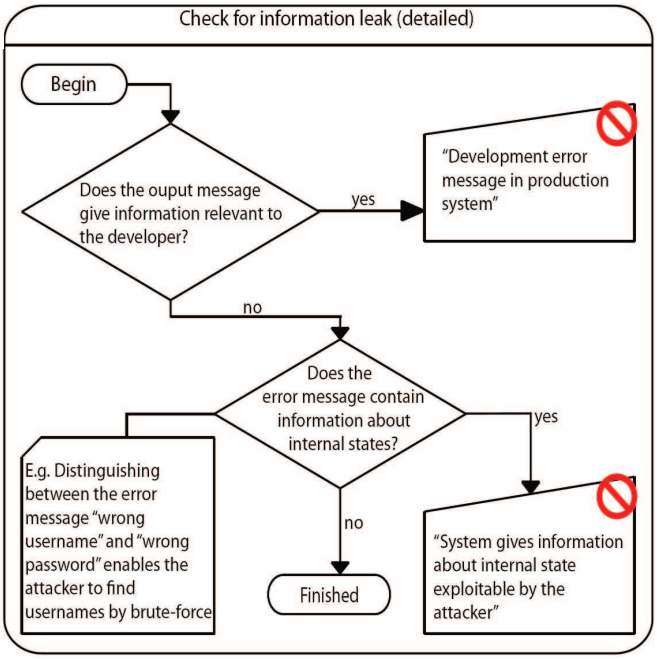

Fig. 1. VID for task 1.

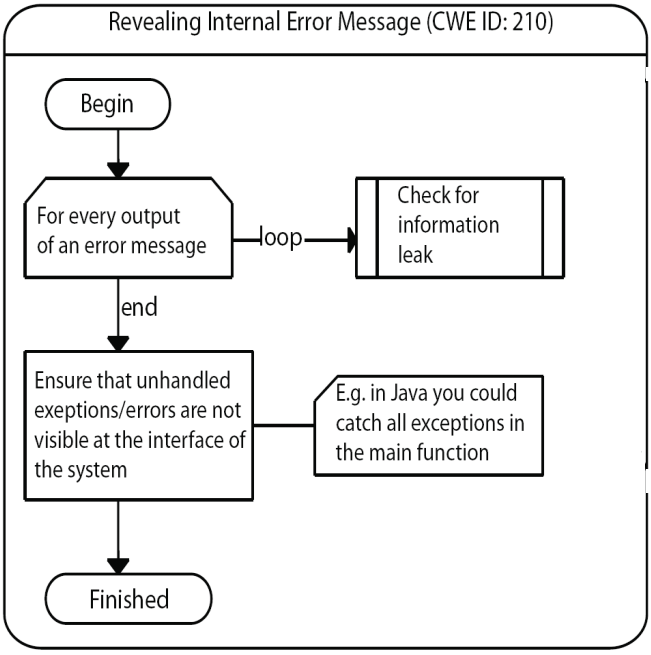

Fig. 2. VID for task 2.

functionality was slightly different than they were used to for the VID tool. In the overall usability rating (in a scale from 1 to 6, where 1 implies "easy" and 6 implies "hard") for the VID tool the average score 2,1.

From the observing the test subjects, the following points were noted:

- None of the testers complained about the icons and their corresponding description on the palette during the execution of the task. Every tester looked at the given VID (figure 1) and found the corresponding icon in the palette without any delay in the process.
- None of the testers complained about the appearance of the VID elements while creating the diagram in figure 1. Every tester typed the text in the correct VID element.

Task 2: Table II shows the time each tester used to complete task 2. From this, it is possible to see that even for a simple VID, there is a notable difference regarding the time spent in creating a VID with the VID tool versus Visio. More precisely:

- The average time spent in completing task 2, using the VID tool, is 2 minutes and 30 seconds.
- The average time spent in completing task 2, using Visio, is 5 minutes and 49 seconds.
- Taking the two points above into consideration shows that the VID tool increases the efficiency (regarding time spent in creating a VID) by 60%.

The reason for why we only have comparison results with Visio is simply that all the test subjects chose to use this tool. Although other drawing/modeling tools like Rational Rose and Poseidon were mentioned by some of the test subjects, they all chose Visio in the end. We think this indicated that Visio was familiar to the students and used for many different modeling purposes.

At the end of task 2, the test subjects were asked which application they preferred to use to create a VID. 9 of 10

TABLE II
TIME EFFICIENCY TEST: VID TOOL VERSUS VISIO

Tester \ Application	VID tool	Visio
Tester 1	2,10min	5,10min
Tester 2	2,43min	7,15min
Tester 3	2,07min	5,00min
Tester 4	3,50min	12,57min
Tester 5	1,24min	3,22min
Tester 6	1,35min	4,03min
Tester 7	1,30min	3,38min
Tester 8	3,45min	5,36min
Tester 9	2,48min	6,03min
Tester 10	3,09min	6,44min

test subjects said they preferred to use the VID tool. The one person who preferred to use Visio argued that if he had the option between installing a specialized tool and using Visio, he would choose the latter (Visio is usually pre-installed on all student PCs).

Weaknesses with the experiment: First of all a weakness with the experiment, the way it was executed, is related to the "learning effect". By repeating something, you will probably be faster, and the result is likely to be better for every time you do it. During the execution, task 1 and task 2 were executed in a chronological order. This means that the test subjects first used the VID tool, then Visio, and at last the VID tool again. One could use this as a reason to explain why the results in column "VID tool" in table II are faster. However, the fact that the test subjects chose Visio to execute task 2 indicates that they had good knowledge of it, and had used it earlier.

With only ten participants it is clear that the experiment can only give us a qualitative indication. Furthermore, all test subjects had nearly the same background, they all rated their previous modeling experience as low or medium. This means that the results reflect how the VID tool works for one type of users, but there might be other groups of users that would not give the same results.

Only very small VIDs were created during the experiment. This could potentially give a different result than if all possible VID elements were used in a larger model. This is because the time used on modeling a very large, complex VID, might not be linear to the time used on modeling a small, simple VID.

Observations from the experiment: First of all, when using Visio, the test subjects had a lot of problems using connections, getting a suitable font for the text, and also when resizing some figures. All of these seem to be related to the large amount of possibilities the user has in Visio. Many of the test subjects seemed to get frustrated by this. For instance, when linking two nodes together, a lot of different connections can be used, but the users had a hard time finding a good way of linking nodes anyway. Most of the connections had some sort of an issue, for instance text that is a part of the connection was hard to get rid of. Considering that a palette was already defined for the users, one can only imagine what would happen if the test subjects had to make all figures from scratch.

When using the VID tool, some test subjects had problems with text appearing outside of the node figures. Most of the test subjects also tried to drag-and-drop nodes and links from the palette. Since all of the users were familiar with Visio, and since drag-and-drop is a normal way of adding nodes in Visio, it is possible that Visio has affected the users, and that another group of tester would not have done the same. Another observation made was that some test subjects tried to add the comment link between two nodes in a way that was not allowed when using the VID tool. It seemed that the test subjects assumed it was the wrong model element, and tried other ones.

Comments from users: After the test subjects had completed the tasks, they were asked to comment the tools. In the VID tool, missing drag-and-drop was mentioned by the majority of the users. Still, they all said that the VID tool's way of adding nodes worked very well, so we regard this more of an observation, not a problem. The only issue that was considered a "problem" was that the text appeared outside of the boundaries when resizing some figures. Another comment was about arrows not always touching the figures, meaning that there are in some cases a small white space gap between the arrows and the figures.

VII. CONCLUSION AND FURTHER WORK

The literature shows that there is a need for better tool support when it comes to software security and security modeling. However, there has been very few experiments giving evidence to these claims. This paper has shown that by having a dedicated tool it can take 60% less time to create a security model compared to a general purpose modeling/drawing tool. In our experiment 9 out of the 10 test subjects preferred the dedicated tool, even though they were much more familiar with the general tool.

For this experiment we chose to use a modeling formalism for vulnerability inspection named VID, which is basically a UML Activity Diagram representation of security checklists. However, we think that similar results on efficiency improvements at creation time can be achieved with other security modeling languages and formalisms as well, given that proper tool support is available.

Even though the current version of the VID tool is probably more efficient than the general purpose tools used today, there are several issues that should be addressed in a future version. It would be beneficial to have empirical data on this or similar security modeling tools in actual use, and compared with other modeling formalisms. This work has focused on evaluating the tool itself, including efficiency and usability. Another interesting study would be the readability and usability of actual models created using security modeling tools.

ACKNOWLEDGMENT

This work has been performed at the Norwegian University of Science and Technology, and supervised by the SHIELDS

project (funded by the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement no 215995). We would especially like to thank everyone who participated in the experiment.

REFERENCES

[1] G. McGraw, "Software security," *Security & Privacy Magazine, IEEE*, vol. 2, no. 2, pp. 80–83, 2004.

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

[3] G. McGraw, *Software Security: Building Security in*. Addison-Wesley, 2006.

[4] *The CORAS UML Profile*, SINTEF, Telenor, http://coras.sourceforge.net/ Last date accessed 2008-11-11.

[5] P. H. Meland, D. G. Spampinato, E. Hagen, E. T. Baadshaug, K.-M. Krister, and K. S. Vell, "SeaMonster: Providing tool support for security modeling," NISK 2008.

[6] P. H. Meland and J. Jensen, "Secure software design in practice," *Availability, Reliability and Security, International Conference on*, vol. 0, pp. 1164–1171, 2008.

[7] S. Ardi, D. Byers, P. H. Meland, I. A. Tondel, and N. Shahmehri, "How can the developer benefit from security modeling?" *Availability, Reliability and Security, International Conference on*, vol. 0, pp. 1017–1025, 2007.

[8] *Attack trees*, Dr Dobbs, http://www.ddj.com/architect/184411129 Last date accessed 2008-10-22.

[9] D. Byers, S. Ardi, N. Shahmehri, and C. Duma, "Modeling Software Vulnerabilities With Vulnerability Cause Graphs," Linköpings universitet, Department of computer and information science, Tech. Rep., 2006.

[10] S. Ardi, D. Byers, and N. Shahmehri, "Towards a structured unified process for software security," in *Proceedings of the 2006 international workshop on Software engineering for secure systems*. ACM Press New York, NY, USA, 2006, pp. 3–10.

[11] S. Ardi, "A model and implementation of a security plug-in for the software life cycle," 2008.

[12] D. Byers and N. Shahmehri, "A Cause-Based Approach to Preventing Software Vulnerabilities," in *Third International Conference on Availability, Reliability and Security*, 2008, pp. 276–283.

[13] H. Peine, M. Jawurek, and S. Mandel, "Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection," *11th IEEE High Assurance Systems Engineering Symposium*, 2008.

[14] ESI, SINTEF, and LiU, "SHIELDS: Detecting known security vulnerabilities from within design and development tools. Research project within the European Community's Seventh Framework Programme. D1.2 Initial SHIELDS Approach Guide," Tech. Rep., 2009, http://www.shields-project.eu.

[15] Fraunhofer, LiU, GET/INT, MI, and SL, "SHIELDS: Detecting known security vulnerabilities from within design and development tools. Research project within the European Community's Seventh Framework Programme. D4.1 Initial SHIELDS Approach Guide," Tech. Rep., 2009, http://www.shields-project.eu.

[16] A. Klaus, F. Elberzhager, and R. Eschbach, "Preventing Vulnerabilities with Security Inspection Scenarios," in *International Symposium on Engineering Secure Software and Systems*, 2009.

[17] G. Sindre and A. Opdahl, "Eliciting security requirements by misuse cases," in *Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Pacific 2000. Proceedings. 37th International Conference on*, 2000, pp. 120–131.

[18] M. Fowler, *UML Distilled Third Edition: A Brief Guide To The Standard Object Modeling Language*. Addison-Wesley, 2004.

[19] D. Kulak and E. Guiney, *Use Cases: Requirements in Context*. Addison-Wesley Professional, 2003.

[20] J. Juerjens, *Secure Systems Development With UML*. Springer, 2005.

[21] J. Jurjens, "UMLsec: Extending UML for Secure Systems Development," *Lecture notes in Computer Science*, pp. 412–425, 2002.

[22] H. E. I. Dahl, I. Hogganvik, and K. Stlen, "Structured semantics for the coras security risk modelling language. report stf07 a970," SINTEF Information and Communication Technology, Tech. Rep., 2007.

[23] I. Hogganvik, "A graphical approach to security risk analysis," Ph.D. dissertation, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

[24] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-functional Requirements in Software Engineering*. Springer, 2000.

[25] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[26] "Secur*IT*ree," Amenza, http://www.amenaza.com/ Last date accessed 2008-11-17.

[27] "AttackTree+," Isograph, http://www.isograph-software.com/ftpover.htm Last date accessed 2008-11-17.

[28] "Non-Functional Requirements Modeling Tool," The University of Texas at Dallas, http://www.utdallas.edu/~supakkul/ Last date accessed 2008-11-25.

[29] F. Massacci, J. Mylopoulos, and N. Zannone, "Computer-aided Support for Secure Tropos," *Automated Software Engineering*, vol. 14, no. 3, pp. 341–364, 2007.

[30] J. Jürjens and P. Shabalin, "Tools for Critical Systems Development with UML (Tool Demo)," *Lecture Notes in Computer Science*, vol. 3297/2005, pp. 250–253, 2005.