Exploring Security in Software Architecture and Design

Michael Felderer
University of Innsbruck, Austria

Riccardo Scandariato
Chalmers University of Technology, Sweden & University of Gothenburg, Sweden

A volume in the Advances in Information Security, Privacy, and Ethics (AISPE) Book Series

IGI Global
DISSEMINATOR OF KNOWLEDGE

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

Copyright © 2019 by IGI Global. All rights reserved. No part of this publication may be reproduced, stored or distributed in any form or by any means, electronic or mechanical, including photocopying, without written permission from the publisher.
Product or company names used in this set are for identification purposes only. Inclusion of the names of the products or companies does not indicate a claim of ownership by IGI Global of the trademark or registered trademark.

Library of Congress Cataloging-in-Publication Data

Names: Felderer, Michael, editor. | Scandariato, Riccardo, editor.
Title: Exploring security in software architecture and design / Michael Felderer and Riccardo Scandariato, editor.
Description: Hershey, PA : Information Science Reference, [2018] | Includes bibliographical references.
Identifiers: LCCN 2018008109| ISBN 9781522563136 (h/c) | ISBN 9781522563143 (eISBN)
Subjects: LCSH: Computer security. | Software architecture--Security measures. | Software engineering--Security measures.
Classification: LCC QA76.9.A25 E996 2018 | DDC 005.8--dc23 LC record available at https://lccn.loc.gov/2018008109

This book is published in the IGI Global book series Advances in Information Security, Privacy, and Ethics (AISPE) (ISSN: 1948-9730; eISSN: 1948-9749)

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

All work contributed to this book is new, previously-unpublished material.
The views expressed in this book are those of the authors, but not necessarily of the publisher.

For electronic access to this publication, please contact: eresources@igi-global.com.

Chapter 1
Threat Modeling in Agile Software Development

Martin Gilje Jaatun
SINTEF Digital, Norway

Karin Bernsmed
SINTEF Digital, Norway

Daniela Soares Cruzes
SINTEF Digital, Norway

Inger Anne Tøndel
SINTEF Digital, Norway

ABSTRACT

Threat modeling is a way to get an overview of possible attacks against your systems. The advantages of threat modeling include tackling security problems early, improved risk assessments, and more effective security testing. There will always be limited resources available for security, and threat modeling will allow you to focus on the most important areas first. There is no one single "correct" way of doing threat modeling, and "agile" is no excuse for not doing it. This chapter describes the authors' experiences with doing threat modeling with agile development organizations, outlining challenges to be faced and pitfalls to be avoided.

DOI: 10.4018/978-1-5225-6313-6.ch001

Copyright © 2019, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

1. INTRODUCTION

Threat modeling has been identified as one of the most important activities in the Security Development Lifecycle (SDL) (Howard & Lipner, 2006). According to Jeffries (Jeffries, 2012), Microsoft SDL author Michael Howard states: "If you're only going to do one activity from the SDL, it should be threat modeling". The main idea behind threat modeling is to *think like an attacker*. A well-defined threat model helps to identify threats to the different assets of a system by utilizing well-grounded assumptions on the capabilities of any attacker interested in attacking such a system. It enables the teams to identify critical areas of design, which need to be protected. Over time, various threat modeling approaches and methodologies have been developed, and are being used in the process of designing secure applications (Cruzes, Jaatun, Bernsmed, & Tøndel, 2018). The approaches vary from conceptual frameworks to practical methodologies. To speed up software delivery, many organizations have adopted an agile software development approach, in which development teams produce code in shorter iterations with frequent feedback loops. In agile software development, however, threat modeling is not widespread, and the practitioners have few sources of recommendations on how to proceed to adopt the practice in their process. In addition, in agile software development, it is often challenging in itself to adopt security practices, either because security practices are not prioritized, or because the practitioners are not able to see the relevance and importance of the activities to the improvement of the security in the project (Cruzes et al., 2018). Studies in software security usually focus on software security activities in general, and there are few empirical studies focusing on specific practices in agile software development. The threat modeling activity is particularly important in software security, since many security vulnerabilities are caused due to architectural design flaws (McGraw, 2004). Furthermore, fixing such vulnerabilities after implementation may be very costly, requiring workarounds which sometimes increase the attack surface. A well-defined threat model helps to identify threats to different assets of a system by utilizing well-grounded assumptions on the capabilities of any attacker interested in exploiting such a system. It also enables the development teams to identify critical areas of the design which need to be protected, as well as mitigation strategies. However, threat modeling can also be challenging to perform for developers, and even more so in agile software development.

This chapter is based on results from the ongoing *SoS-Agile - Science of Security for Agile Software Development* research project (https://www.sintef.no/en/digital/sos-agile/) which investigates how to meaningfully integrate software security into agile software development activities. The project started in October 2015 and will end in October 2020, and involves many software development companies in Norway. The method of choice for the project is Action Research, which is an appropriate research methodology for this investigation because of the combination of scientific and practical objectives that aligns with the basic tenet of action research, which is to merge theory and practice in a way such that real-world problems are solved by theoretically informed actions in collaboration between researchers and practitioners (Greenwood & Levin, 1998), (Davison, Martinsons, & Kock, 2004).

The remainder of this chapter is structured as follows: Section 2 outlines our approach to threat modeling in broad strokes. In Section 3 we explore some particular challenges associated with agile software development, which influence how we think about threat modeling. Section 4 offers additional recommendations on how to successfully perform threat modeling in agile software development. We conclude in Section 5.

2. FUNDAMENTAL THREAT MODELING ACTIVITIES

Threat modeling is a wide concept that encompasses a broad range of techniques that can be utilized to make a system more secure. Threat modeling usually employs two types of models; one that represents the system that is to be built and another one that represents the actual threats to the system (Shostack, 2014b). In the context of software development, what is being built can be almost anything that will use the developed software, for example a website, a mobile application or a distributed system. The threats will then represent what can go wrong with the system, which includes all the potential reasons why the software may not function as intended. Depending on the scope of the analysis, one may choose to consider not only threats that are due to malicious intervention, i.e., all the different types of attacks that may occur, but also unintentional events, which are caused by legitimate users that make mistakes. In some cases, random failures are also included as potential threats.

Our approach to threat modeling in agile software development organizations consists of a visual representation of three main elements:

- Assets that are essential or critical for the system;
- An overview over how assets are stored, processed or otherwise interact with the system, which includes systems interfaces and potential attack surfaces;
- Threats to the system, which will affect one or more of the identified assets.

These elements can be found, to a greater or lesser extent, in many threat modeling approaches (Dhillon, 2011). In the following we will briefly describe each in turn.

2.1 Asset Identification

In the organizations where we have applied threat modeling, the first step has been asset identification. An *asset* is something that needs to be protected within the system. Usually, assets are the information or services that are vital for the business operation and success, however, the concept of "assets" can also comprise other parts of the system, such as hardware, network components, domains or even people. Although Shostak (Shostack, 2014b) presents this step as optional, and Dhillon (Dhillon, 2011) does not mention the concept explicitly at all, we always perform this activity. In our experience, it creates awareness about security in the organization, it helps the developers understand which components in their systems that need to be protected, and it makes it easier to focus the discussion on relevant threats later on during the threat modeling activities. Asset identification is also included as a compulsory step in most of the existing standards for risk assessments, including ISO/IEC 27005 (ISO, 2011). Even Dhillon (Dhillon, 2011) does this implicitly, e.g., by mentioning annotations of "processes that perform critical security functions" and "encrypted or signed data flows and data stores".

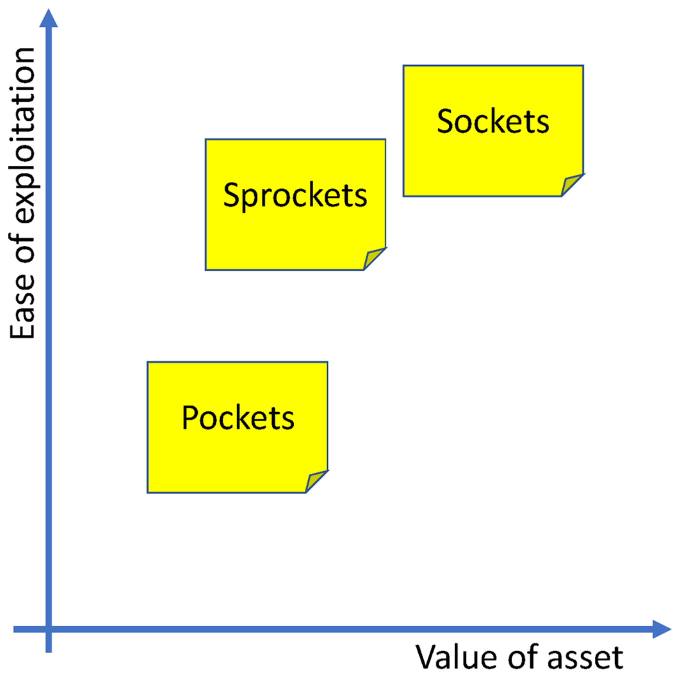
Asset identification, when performed at all, is often done from the top of one's head, with the results documented in a simple list. However, we have found it useful to employ the explicit method formulated by Jaatun & Tøndel (Jaatun & Tøndel, 2008), which we have since evolved: Briefly, the developers, together with the most important stakeholders in the project, participate in a semi-structured brainstorming session, where the first step is to get all possible assets on the table. As a second step, we then engage the participants in an interactive classification session where we try to determine the assets' relative importance or value (referring to Figure 1, we don't worry about the absolute value of "sprockets", but we want to know if they are worth more or less than "sockets", etc.). Based on the developers' knowledge of the system, we finally try to determine the assets' relative ease of exploitation, and end up with a grid as exemplified in Figure 1.

In our experience, many agile teams seem to think that asset identification is a waste of time; spending too much time on documenting something that is often perceived to be obvious. However, we still believe this is a useful exercise that should be performed at least once in each project. Even if the architecture changes, it is unlikely that the assets change - and if there are fundamental changes in the system that introduce new assets, that is a natural trigger for re-doing the asset analysis.

2.2 Data Flow Diagram

The second step in the threat modeling exercise is to get an overview over where the identified assets are stored, processed or otherwise interact with the system. As

Figure 1. Identification and classification of assets

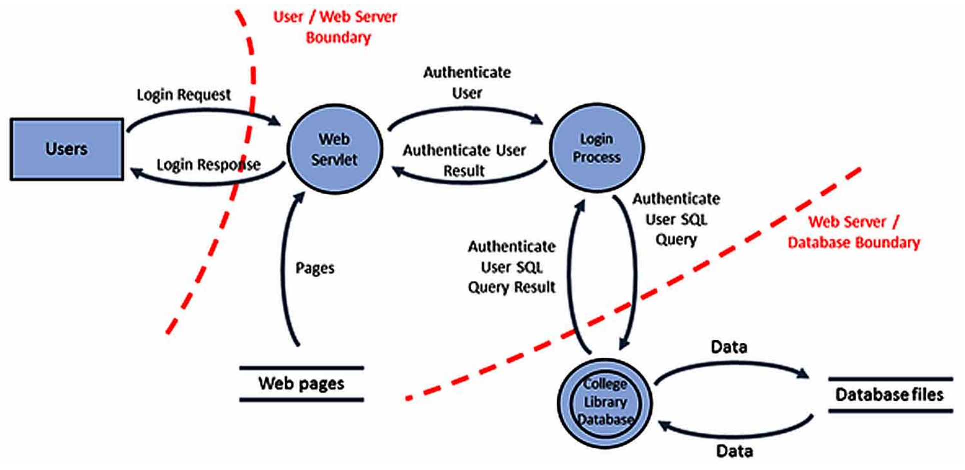


part of this step, it is also useful to define the interfaces of the system that is being analyzed and to identify potential attack surfaces.

As a means to get an overview of the system to be analyzed, we recommend making a Data Flow Diagram (DFD). A DFD is a graphical representation of the most important actors, processes, services, components and data stored in the system, and it highlights how information flows between each of them. An example of a DFD is shown in Figure 2, based on an OWASP threat modeling tutorial (Conklin, 2014). This example shows a simple university library system, with a web front end for access by students and staff, a login process to authenticate users, and a database system (which can be decomposed further). Users are represented by squares, and processes by circles (complex processes that can be decomposed into more detailed data flow diagrams are represented by double circles). Data flows are represented by curved arrows, and data stores are represented by two parallel lines. Finally, trust boundaries are indicated by dashed curved lines.

Drawing the DFD is usually perceived by the team to be worth it, but strictly speaking it takes too long to draw. Participating in a session where the DFD is created is good for newcomers; can be a good onboarding exercise giving new team members a useful overview of the system. The agile culture of no documentation is a detriment to gaining such an overview.

Figure 2. Example data flow diagram (redrawn from example at http://owasp.org)



The DFD is a very suitable backdrop for performing the next step in the threat modeling exercise: threat identification and analysis.

2.3 Identification and Analysis of Threats

The last step in the threat modeling exercise is to identify and analyze all relevant threats to the system. This can be done in several ways. Here we present the approach that we have applied, using the STRIDE framework.

Back in the day, Microsoft employees Loren Kohnfelder and Praerit Garg created the STRIDE mnemonic for typical attacks (Shostack, 2014b), based on the first letter of each of the attack types:

- **Spoofing:** An attacker can pretend to be another user, component or system.
- **Tampering:** An attacker can modify data.
- **Repudiation:** An attacker (or other actor) can deny having performed an action or having sent a message if the system has insufficient mechanisms or evidence.
- **Information Disclosure:** An attacker can get read access to protected data.
- **Denial of Service (DoS):** An attacker can prevent legitimate users from using the normal functionality of the system.
- **Elevation of Privilege:** An attacker uses illegitimate means to reach an access level with other privileges than [s]he is supposed to have.

STRIDE can be used to identify threats by analyzing each of the interfaces defined in the DFD, and assess whether any of the attack types are relevant and how they could be executed. STRIDE can also be applied directly on the identified assets, to see whether they are vulnerable to spoofing, tampering, repudiation, etc. attacks.

As stated by Adam Shostak, “STRIDE is a good framework, bad taxonomy” (Shostack, 2014a). This implies that it would be dangerous to consider STRIDE an exhaustive list of threats; there could be other threat types that are relevant for the system. Privacy could be an example; even though some privacy breaks are simply information disclosure, there are more subtle issues that may need to be addressed. An obvious example is when personal data is used for other purposes than the users have consented to. Such a threat will not be identified by the sole use of STRIDE.

To drill down on selected attacks, and explore the attacker’s goals and strategies, an attack tree (Schneier, 1999) can be constructed. In this way we can detail all the elements required in order to succeed with the attack, and also explore any alternatives that are available to the attacker. This will also be a good starting point for identifying security tests and possible countermeasures.

In the next section, we discuss the challenges we have experienced when applying these threat modeling activities in agile software development projects.

3. CHALLENGES IN AGILE SOFTWARE DEVELOPMENT PROJECTS

As shown by Camacho et al. (Camacho, Marczak, & Cruzes, 2016), adoption of security practices pose challenges in agile, either because security activities are not prioritized or because the developers do not see the importance of security activities in their projects. In a recent case study (Cruzes et al., 2018), in which we observed eight threat modeling sessions performed by agile software development teams, we identified a number of challenges to applying threat modeling in agile development practices. Here we report the most prominent.

Because of the lack of focus on documentation in agile teams, the teams had very little material to use as a starting point for the meetings. Further, the teams did not take the time to properly document the results from the asset identification activity, mainly because (at this point) they did not see why it would be useful. Also, even though the identification of assets naturally led to a discussion of whether they had the necessary mechanisms to protect those assets, such input was not documented during this phase and was hence not brought forward to the next phases.

Regarding the Data Flow Diagram, it was challenging both to motivate the teams to draw them, and to decide on the right level of abstraction. Some teams spent very long time on this activity. It also turned out to be challenging to map the interfaces of the system that they were modeling with the parts of the systems that other teams were responsible for. Some of the teams mentioned that they actually did not know how the system was really implemented, and the DFDs did not provide confidence that the system was actually implemented that way, especially where there was a lot of legacy code.

The study also revealed the need for better guidelines on how to organize the threat modeling activities, and the need to involve security experts in these activities. To agile teams, 1-2 hour meetings means a long time taken from the sprint hours and the teams sometimes expressed frustration when they did not manage to complete, for example, the DFD during the allocated meeting slot. It was also hard for the teams to say something about how long the models they created would be valid. In our experience, it is highly beneficial to let a security expert lead the threat modeling activities, however, most agile teams do not have such a person.

Threat modeling is a team sport (Jeffries, 2012), and this is also true for agile projects. When drawing a DFD, we like to be in a single room with all team members participating. However, when the software development division consists of distributed teams, which was the case for several of the organizations that we studied, this activity becomes quite challenging (Šmite, Moe, & Torkar, 2008). Even though the teams were used to video conferencing to replace or complement physical meetings, this turned out to be challenging when doing threat modeling. Most of the teams preferred to use the whiteboard when brainstorming around the different models that they created. However, with today's videoconferencing equipment it is generally not possible to see both the people at the other location(s) and their whiteboard at the same time, and often not even the whole whiteboard at a resolution that allows everyone to read everything. We have experimented with different configurations, but the major challenges remain unresolved. We have tried making identical drawings at each participating site, but maintaining consistency of such drawings is hard. Eventually, we expect that technological advances will iron out the kinks in such global collaboration sessions.

One thing we have found is that when threat modeling spans several sessions, it is important to share any diagrams created previously upfront. Since the diagrams may contain company sensitive information, they should be shared in a secure manner, e.g., not sent in regular email.

4. KEY FACTORS FOR SUCCESSFUL THREAT MODELING

The challenges identified in our study (Cruzes et al., 2018), together with challenges previously identified by Tuma et al. (Tuma, Calikli, & Scandariato, 2018), Dhillon (Dhillon, 2011), Assal and Chiasson (Assal & Chiasson, 2018), and Jaatun et al. (Jaatun, Jensen, Meland, & Tøndel, 2011), enabled us to identify a set of key factors for successful threat modeling in agile software development organizations.

4.1 Motivate the Team

Motivating the team is a prerequisite for a successful result! Make sure to explain the purpose of the threat modeling activities up-front and inform the team what they can expect as output of the activity. In particular they need to understand how the outputs can be used in the software development process. Once the purpose has been explained, hopefully the team is already motivated. Further motivation may be provided through educational activities such as short courses, preferably built around previous incidents that were caused by security bugs or flaws in the software of the system that is being developed.

4.2 Invite a Security Expert

Make sure that a security expert participates in all the threat modeling activities. This will help to focus the discussions, leverage relevant results and speed up the work. The security experts should both facilitate the discussions and document the results. He or she does not have to be a member of the development team, in fact, as already stated, most agile teams do not even have a security expert. Engaging someone external to the team has been shown to be a successful approach (Cruzes et al., 2018). However, in the long run it is unrealistic to expect that the software development organization will fund such an external resource. In order to make the security work self-sustainable, the role of the security expert should be eventually assumed by the security champion in the team. What security expertise is needed? Jeffries (Jeffries, 2012) refers to required security training in the SDL guideline (Microsoft, 2012), and states that it is "ideal that team members have an interest in security". The basic concepts in the SDL training include Secure Design, Threat Modeling, Secure Coding, Security Testing, and Privacy. Intriguingly, among the recommended resources are the Bell-LaPadula model and the Biba model – not something we would have thought the average developer would try to tackle!

We have found that teams that had a good architect from the beginning usually have many things covered, so that when threat modeling is performed on an existing system, few new unmitigated threats emerge. In general, this rhymes with results from Oyetoyan et al. (Oyetoyan, Jaatun, & Cruzes, 2017), who found that years of experience correlate with level of software security expertise.

Threat modeling may not be perceived as worthwhile for old systems, particularly for components which are slated for being phased out. Management support for security work is clearly vital (Assal & Chiasson, 2018); lack of such support is detrimental to software security morale, as exemplified by this skeptical quote from a team member at the end of a session: “Will there be hours for implementing security?”

4.3 Manage System Complexity

If threat modeling is performed for the first time with a team, then a good strategy is to limit the scope by zooming in on a selected part of the system. You should also spend some time on understanding the organization’s mission and business goals. This will ensure that the asset identification activity includes information and services that are vital to the organization’s interest and hence critical to protect. Draw a DFD upfront and decide which part of the system has priority to be analyzed first.

4.4 Short and Efficient Meetings

To agile teams, a 1-2 hour meeting means a long time taken from the sprint hours. Therefore, a thorough preparation of the meetings is crucial for success. The security expert facilitating the meeting needs to make sure that all preparations that could speed up the meeting are done on beforehand. In cases where a team has one or two very strong personalities, the facilitator may have to be active in ensuring that all voices are heard and that the meetings do not exceed the allocated time slot.

4.5 Involve the Remote Team(s)

Whenever the threat modeling activities involve two or more teams connected via videoconferencing, pick one location to be the “main” location, and provide a whiteboard with the data flow diagram here, visible on video to the other locations. Usually, the main location will be the site with the most developers participating. The current technological limitations require that some extra preparations are required for a successful remote session. All groups of remote participants should produce a large printout of

- Asset diagram (placed in value/ease of exploitation grid)

- Data flow diagrams
- STRIDE definition

If the remote participants have access to more display units in addition to those used for the videoconferencing solution, these may be used instead of the printouts; the important thing is to have these items easily available for reference during the discussion. The STRIDE definition may seem superfluous, but we have found that it is a useful reminder for the teams, and even when the definition is well known by the participants it serves as a minimal checklist for guiding the discussion.

4.6 Pick the Best Tool to Solve the Task

Most of the existing tools for threat modeling activities either lack maturity, or they are targeted for security experts (Tuma et al., 2018). These tools may therefore be too complicated for the average developer, and it will be counter-productive to try to force developers to learn a new tool for doing something they already fear is a waste of time. Tools should be chosen carefully, preferably involving security experts, security champions, and representatives from the average developers. It is normally better to err on the side of choosing too few or too simple tools.

4.7 Know the "Definition of Done"

When an organization is just starting out doing threat modeling, knowing when "enough is enough" will be hard. However, instead of letting this cramp the threat modeling initiative into indecision, it is better to simply make some broad requirements that every team can follow, e.g.: "Each team must perform at least one threat modeling session per major system component per quarter." The work will then be "done" (for now) when this session has been completed.

One problem we have experienced is that when a team really gets into the spirit of threat modeling, they sometimes "go wild". How to focus on valid threats? In this case it is particularly useful to have a facilitator (or experienced security champion) to steer the discussion in the right direction.

4.8 Document the Results

All identified assets need to be recorded in central repository, such as the team's issue tracking system (Jira, TFS, or similar). The DFDs should similarly be stored in a central repository. The issue tracking system should also be used to ensure that identified threats are handled in the code (see below).

4.9 Ensure Results Are Propagated to the Code

We have found that the best way of ensuring that threat modeling results end up in the code, is to create a “risk” ticket for each security issue that arises during the modeling sessions. This allows the issue to be handled in a manner that will be familiar to the developers, without introducing any new process elements. Broadly speaking, such tickets will fall into two categories: Either something that needs to be added by the developers, or something that needs to be discussed by other parts of the team. The second type of ticket will then either be closed after discussion (nothing needs to be done), or be transformed into the first type.

5. CONCLUSION

We have studied how threat modeling can be performed in modern software development, and have outlined here a recipe that should work in most agile organizations.

In his keynote at the XP conference in Porto, Portugal on May 23rd 2018, Kent Beck responded to a prioritization question: “If I had to choose, I’d drop security”. This reinforces the impression that agile developers are still prone to falling into the trap of “we’ll fix security later; we just need to make this work first”. It is clear that in some cases, temporarily deprecating security requirements is the right thing to do, but the problem is that most security requirements treated this way end up being simply forgotten. In a previous discussion with another agile developer (Hellesøy, 2017), the point was raised that developers need to start treating technical debt (and thus, security debt) just like financial debt – you may be able to live with some security issues for a limited time, but if you cannot keep up with the down payments, you will eventually be facing bankruptcy.

An important side-effect of involving a development team in threat modeling is a general improvement in security awareness, to the point where the developers potentially become enthusiastic about security (Jeffries, 2012).

Finally, it’s not enough to identify threats, management must also prioritize mitigating them!

ACKNOWLEDGMENT

This work was supported by the SoS-Agile project: Science of Security in Agile Software Development, funded by the Research Council of Norway (grant number 247678).

REFERENCES

Assal, H., & Chiasson, S. (2018). Security in the software development lifecycle. In *Fourteenth symposium on usable privacy and security (SOUPS 2018)* (pp. 281-296). Academic Press.

Camacho, C. R., Marczak, S., & Cruzes, D. S. (2016, Aug). Agile team members perceptions on non-functional testing: Influencing factors from an empirical study. In *2016 11th international conference on availability, reliability and security (ARES)* (p. 582-589). Academic Press. doi: 10.1109/ARES.2016.98

Conklin, L. (2014). *CRV2 AppThreatModeling*. Retrieved from https://www.owasp.org/index.php/CRV2AppThreatModeling

Cruzes, D. S., Jaatun, M. G., Bernsmed, K., & Tøndel, I. A. (2018). Challenges and Experiences with Applying Microsoft Threat Modeling in Agile Development Projects. *Proceedings of the 25th Australasian Software Engineering Conference (ASWEC).*

Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, *14*(1), 65–86. doi:10.1111/j.1365-2575.2004.00162.x

Dhillon, D. (2011, July). Developer-driven threat modeling: Lessons learned in the trenches. *IEEE Security and Privacy*, *9*(4), 41–47. doi:10.1109/MSP.2011.47

Greenwood, D., & Levin, M. (1998). *Introduction to action research: Social research for social change.* SAGE Publications. Retrieved from https://books.google.no/books?id=nipHAAAAMAAJ

Howard, M., & Lipner, S. (2006). *The security development lifecycle*. Microsoft Press.

ISO/IEC 27005:2011 Information technology - Security techniques - Information security risk management. (2011). Retrieved from https://www.iso.org/standard/56742.html

Jaatun, M. G., Jensen, J., Meland, P. H., & Tøndel, I. A. (2011). A Lightweight Approach to Secure Software Engineering. In *A Multidisciplinary Introduction to Information Security* (pp. 183–216). CRC Press.

Jaatun, M. G., & Tøndel, I. A. (2008). Covering your assets in software engineering. In *The third international conference on availability, reliability and security (ARES 2008)* (pp. 1172-1179). Barcelona, Spain: ARES. 10.1109/ARES.2008.8

Jeffries, C. (2012). *Threat modeling and agile development practices*. Retrieved from https://technet.microsoft.com/en-us/security/hh855044.aspx

McGraw, G. (2004). Software security. *Security & Privacy, IEEE*, *2*(2), 80–83. doi:10.1109/MSECP.2004.1281254

Microsoft. (2012). *Pre-SDL Requirements: Security Training*. Retrieved from https://msdn.microsoft.com/en-us/library/windows/desktop/cc307407.aspx

Oyetoyan, T. D., Jaatun, M. G., & Cruzes, D. S. (2017). A lightweight measurement of software security skills, usage and training needs in agile teams. *International Journal of Secure Software Engineering*, *8*(1), 1–27. doi:10.4018/IJSSE.2017010101

Schneier, B. (1999). Attack trees. *Dr. Dobb's Journal*, *24*(12), 21–29.

Shostack, A. (2014a). Elevation of privilege: Drawing developers into threat modeling. *2014 USENIX summit on gaming, games, and gamification in security education (3GSE 14)*.

Shostack, A. (2014b). *Threat modeling: Designing for security*. Wiley.

Smite, D., Moe, N. B., & Torkar, R. (2008). Pitfalls in remote team coordination: Lessons learned from a case study. In A. Jedlitschka & O. Salo (Eds.), *Product-focused software process improvement* (pp. 345–359). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-540-69566-0_28

Tuma, K., Calikli, G., & Scandariato, R. (2018). Threat analysis of software systems: A systematic literature review. *Journal of Systems and Software*, 144.