The Journal of Systems and Software 104 (2015) 90–111

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Investigating security threats in architectural context: Experimental evaluations of misuse case maps

Peter Karpati [a,1], Andreas L. Opdahl [b,*], Guttorm Sindre [a]

[a] Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælands vei 7-9, NO-7491 Trondheim, Norway
[b] Department of Information Science and Media Studies, University of Bergen, P.O. Box 7802, NO-5020 Bergen, Norway

ARTICLE INFO

Article history:
Received 28 June 2014
Revised 13 February 2015
Accepted 14 February 2015
Available online 23 February 2015

Keywords:
computer security
Intrusion analysis
Use case maps

ABSTRACT

Many techniques have been proposed for eliciting software security requirements during the early requirements engineering phase. However, few techniques so far provide dedicated views of security issues in a software systems architecture context. This is a problem, because almost all requirements work today happens in a given architectural context, and understanding this architecture is vital for identifying security vulnerabilities and corresponding mitigations. Misuse case maps attempt to provide an integrated view of security and architecture by augmenting use case maps with misuse case concepts. This paper evaluates misuse case maps through two controlled experiments where 33 and 54 ICT students worked on complex real-life intrusions described in the literature. The students who used misuse case maps showed significantly better understanding of intrusions and better ability to suggest mitigations than students who used a combination of two existing techniques as an alternative treatment. Misuse case maps were also perceived more favourably overall than the alternative treatment, and participants reported using misuse case maps more when solving their tasks.

1. Introduction

Security should be addressed already at the earliest stages of software development (Jürjens, 2005), and a variety of methods and techniques have recently been proposed for secure software development (e.g., Jürjens, 2002; Dimitrakos et al., 2003; Van Lamsweerde, 2004; Giorgini et al., 2005), including security requirements (e.g., McDermott and Fox, 1999; Sindre and Opdahl, 2000; Firesmith, 2003; Liu et al., 2003; Lin et al., 2004) and secure architecture design (e.g., Jensen, 1998; Deng et al., 2003; Ali et al., 2009; Wang et al., 1999). However, few authors have focussed on the relationship between security requirements and software systems architecture, and very few methods or techniques (e.g., Karpati et al., 2010a,b; Herrmann et al., 2012) have been proposed that provide dedicated overviews of high-level security issues in a software systems architecture context. This is a problem, because almost all requirements work today happens in a wholly- or partially-predetermined architectural context (Nuseibeh, 2001; Jarke et al., 2011), which is given, for example, by an *existing system* to be modified or extended; by the *neighbouring systems* it needs to communicate with; and by preexisting architecture and infrastructure *standards*. Understanding this, often complex, existing architecture is vital when identifying security vulnerabilities and corresponding mitigations during early requirements engineering. Even in later requirements stages, it has become widely accepted that software requirements and architecture design are concurrent activities (Nuseibeh, 2001; Mayer et al., 2005), especially in complex problem domains (Jarke et al., 2010) where security often needs extra attention.

In previous work (Karpati et al., 2010a,b; Katta et al., 2010), we have therefore proposed *misuse case maps (MUCM)*, which combine ideas from our own previous work on *misuse cases* (MUC; Sindre and Opdahl, 2000, 2005) with ideas from Amyot, Buhr and Casselman's *use case maps* (UCM; Buhr, 1996; Buhr and Casselman, 1996; Amyot, 1999). The purpose of MUCM is to encourage investigation of potential security vulnerabilities and corresponding mitigations in a systems architecture context. The technique is intended to help identifying vulnerabilities of many different types and to help suggesting mitigations of many different types for each vulnerability. This paper presents an experimental evaluation of MUCM through two controlled student experiments, which extend our previous work on MUCM. In Karpati et al. (2010b), we have presented a first, formative evaluation of MUCM through a small survey of researcher colleagues—the present paper instead uses controlled experimentation that involves more participants, uses more realistic tasks, and measures

* Corresponding author. Tel.: +47 5558 4140; fax: +47 5558 9149.
E-mail addresses: Peter.Karpati@hrp.no (P. Karpati), Andreas.Opdahl@uib.no, AndreasOpdahl@gmail.com (A.L. Opdahl), Guttorm.Sindre@idi.ntnu.no (G. Sindre).
[1] Present address: Institute for Energy Technology, P.O. Box 173, NO-1751 Halden, Norway.

http://dx.doi.org/10.1016/j.jss.2015.02.040
0164-1212/

more variables. In Katta et al. (2010), we have compared MUCM experimentally with another experimental technique (MUSD)—the present paper instead compares MUCM with existing notations that are commonly used in practice. In Karpati et al. (2010a, 2011), we have presented limited analyses of an experiment—the present paper reports the first full analysis of this experiment, including all dependent variables.[2] The analysis of a second, additional experiment is presented for the first time here, along with an overall analysis of both experiments.

The purpose of our evaluation is to investigate further *whether MUCM is potentially a useful addition to the arsenal of modelling techniques* that developers and security experts have at their disposal while collaborating to develop secure software. And if so, we would like to know more about *how it is potentially useful, and why*. Because we are not aware of directly comparable techniques out there, our aim is not in any way to show that MUCMs are objectively "better than" or "superior to" existing techniques. Indeed, we advocate a multi-perspective approach to analysing requirements in general, and security requirements in particular, to which we hope MUCMs can provide one new and complementary perspective alongside existing ones. The rest of the paper is organised into related work—including our previous work on misuse cases and misuse case maps—(Section 2), research methods (Section 3), results (Section 4), discussion (Section 5), and conclusion (Section 6).

2. Background and related work

A number of techniques have been proposed for security requirements engineering, as can be seen for instance in surveys like (Tøndel et al., 2008; Fabian et al., 2010; Mellado et al., 2010; Karpati et al., 2011; Salini and Kanmani, 2012). Also, the relationship between security requirements engineering techniques and risk analysis has been reviewed by Muñante et al. (2014), and secure systems development onwards from requirements to design has been reviewed by Uzunov et al. (2012) and Lucio et al. (2014). Our purpose in this section is therefore not to make a complete review of previous research in the area, only to cover the most important techniques for security requirements (Section 2.1), related system development techniques (Section 2.2), secure architecture design (Section 2.3), and the relationship between security requirements and architecture (Section 2.4), in order to provide a context for our own work. Finally, Section 2.5 will provide necessary background material for the rest of the paper, including our previously published work on misuse case maps (Karpati et al., 2010a,b, 2011; Katta et al., 2010), in order to make it possible to understand the starting point for this paper without having to read the previous papers.

2.1. Techniques and methods for security requirements

Modelling techniques and methods that are specific to security requirements work include, abuse cases (McDermott and Fox, 1999), misuse cases (Sindre and Opdahl, 2000), and security use cases (Firesmith, 2003), which are security-oriented variants of use cases (Jacobson, 1992). Abuse and misuse cases represent behaviours that potential attackers want to perform using the system, whereas security use cases represent countermeasures intended to avoid or repel these attacks. The difference between abuse and misuse cases is that the latter show use and misuse in the same picture, whereas abuse cases are drawn in separate diagrams (for more details about misuse cases, see Section 2.5). Secure i* (Liu et al., 2003) is an extension of the i* modelling language, where malicious actors and their goals are modelled with inverted variants of the usual icons. Elahi et al. (2010) propose extensions to i* to support systematic security requirements elicitation and analysis centred around vulnerabilities, attackers, and the security impacts of attacks. Elahi (2012) extends i* with Analytic Hierarchy Process (AHP) analysis of security trade-offs. Abuse frames (Lin et al., 2004) extend problem frames with anti-requirements that might be held by potential attackers. Languages for secure business process modelling have been proposed based on both BPMN (Rodríguez et al., 2005) and UML activity diagrams (Sindre, 2007; Rodríguez et al., 2006). Katta et al. (2010) adapts UML sequence diagrams to security analysis.

Generic security techniques can also be used to analyse software security requirements, such as threat trees (Amoroso, 1994) and attack trees (Schneier, 2011). The latter represents a high-level attack as the root node of a tree so that it can be decomposed through AND/OR branches into lower-level attacks that must succeed in particular combinations for the high-level one to succeed. Attack trees are intuitive, and many extensions have appeared, such as defense trees (Bistarelli et al., 2006), protection trees (Edge et al., 2006), attack response trees (Zonouz et al., 2009), attack countermeasure trees (Roy et al., 2010), and unified parametric attack trees (Wang et al., 2011). The formal specification language Z has also been used to specify security-critical systems (Boswell, 1993; Hall and Chapman, 2002).

The Common Criteria method (CCITSE 2002) provides, among other things, a classification of various types of functional security requirements. A sophisticated taxonomy for security requirements is also provided in Firesmith (2005). Massacci et al. (2011) integrate and extend the conceptual frameworks behind i*/Tropos and Problem Frames to a security ontology, grounded in DOLCE (Gangemi et al., 2002), in order to provide clear definitions of security concepts. Mellado et al. (2010) and Karpati et al. (2011) review and compare broad selections of current security requirements engineering approaches. Mayer et al. (2007) use ontology to aid security requirements specification. Raspotnig and Opdahl (2013) review and compare risk identification techniques for safety and for security requirements, whereas Mayer et al. (2007) present ISSRM, a reference model for information system security risk management. Matulevicius et al. (2008) align misuse cases with ISSRM. Mayer et al. (2007) and Dubois et al. (2010) develop it further into a framework for security risk management, which is used to align mal-activity diagrams with risk management in Chowdhury et al. (2012). Raspotnig et al. (2012) propose a method for coordinated elicitation and analysis of safety and security requirements.

Other researchers have also investigated the effectiveness and user perception of security requirements techniques empirically. Labunets et al. (2013) compare CORAS and SREP in a controlled experiment with 28 master students, measuring effectiveness by the students' performance of given tasks and perception by the participant responses to a TAM-based questionnaire. Massacci and Paci (2012) empirically compare of four techniques—CORAS, Problem Frames, Secure I*, and Secure Tropos—through a qualitative investigation relying on observations, recordings, interviews, and focus group discussions. Other empirical evaluations of security modelling techniques are reviewed in Opdahl and Sindre (2009).

2.2. Techniques and methods for secure software development

Other security techniques and methods attempt to cover later software development phases in addition to requirements. Secure Tropos (Giorgini et al., 2005) extends the Tropos method (Bresciani et al., 2004) with security-related concepts for ownership, permission, and delegation. It is adapted in Matulevičius et al. (2008) for security risk management in the early phases of information systems development based on ISSRM (Mayer et al., 2007). Dardenne et al. (1993), extend KAOS with anti-goals (Van Lamsweerde, 2004) that can be used to express the goals of an intruder who aims to exploit

[2] Because we have reconsidered the coding and introduced some new tests, the detailed results reported here sometimes differ from these earlier reports.