Thesis for The Degree of Licentiate of Engineering

Towards Efficiency and Quality Assurance in Threat Analysis of Software Systems

Katja Tuma

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2018

Towards Efficiency and Quality Assurance in Threat Analysis of Software Systems

Katja Tuma

Technical Report No 187L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

This thesis has been prepared using LaTeX.

Printed by Chalmers Reproservice,
Gothenburg, Sweden 2018.

Abstract

Context: Security threats have been a growing concern in many organizations. Organizations developing software products strive to plan for security as soon as possible to mitigate such potential threats. In the design phase of the software development life-cycle, teams of experts routinely analyze the system architecture and design to find potential security threats.

Objective: The goal of this research is to improve on the performance of existing threat analysis techniques and support practitioners with automation and tool support. To understand the inner-workings of existing threat analysis methodologies we also conduct a systematic literature review examining 26 methodologies in detail. Our industrial partners confirm that existing techniques are labor intensive and do not provide quality guarantees about their outcomes.

Method: We conducted empirical studies for building an in-depth understanding of existing techniques (Systematic Literature Review (SLR), controlled experiments). Further we rely on empirical case studies for ongoing validation of an attempted technique performance improvement.

Findings: We have found that using a novel risk-first approach can help reduce the labor while producing the same level of outcome quality in a shorter period of time. Further, we suggest that the key for a successful application of this approach is two fold. First, widening the analysis scope to end-to-end scenarios guides the analyst to focus on important assets. Second, appropriate model abstractions are required to manage the cognitive load of the human analysts. We have also found that reasoning about security in a formal setting requires extending the existing notations with security semantics. Further, minimal model extensions for doing so include security contracts for system nodes handling sensitive information. In such a setting, the analysis can be automated and can to some extent provide completeness guarantees.

Future work: In the future, we plan to further study the analysis completeness guarantees. In particular, we plan to improve on the analysis automation and investigate complementary techniques for analysis completeness (namely informal pattern based techniques). We also plan to work on the disconnect between the planned and implemented security.

Keywords

Secure Software Design, Threat Analysis (Modeling)

Acknowledgment

First and foremost, I am thankful to my supervisor Riccardo Scandariato for his support and mentorship along my journey. His guidance helped plunge into the world of research while keeping my life in balance. I am especially grateful to my examiner Robert Feldt and co-supervisor Gul Calikli for their helpful advice. I am also grateful to Christian Berger and Francisco Gomes for supporting my passion for teaching.

Many thanks to my office mates Rebekka Wohlrab, Pierguiseppe Malozzi, and Sergio García for making me feel excited to come to work every morning. Also, thanks to all my friends here and abroad, for making my life fuller.

Finally, to my parents Tanja and Tadej and my brother Samo. My achievements would not have been possible without your unconditional support and encouragement. Looking back I am proud of the last two years, and I can not wait to see what challenges lie ahead.

This work would not have been possible without the support of the Computer Science and Engineering Department at the University of Gothenburg and Chalmers University of Technology. This research was partially supported by the Swedish VINNOVA FFI project "HoliSec: Holistic Approach to Improve Data Security".

List of Publications

Appended publications

This thesis is based on the following publications:

[A] K. Tuma, G. Calikli, and R. Scandariato. "Threat Analysis of Software Systems: A Systematic Literature Review"
Journal of Systems and Software (2018), 2018.

[B] K. Tuma and R. Scandariato. "Two Architectural Threat Analysis Techniques Compared"
Proceedings of the European Conference on Software Architecture (ECSA 2018), 2018.

[C] K. Tuma, R. Scandariato, M. Widman, C. Sandberg. "Towards security threats that matter"
Proceedings of the International Workshop on the Security of Industrial Control Systems and Cyber-Physical Systems (CyberICPS 2017), 2017.

[D] K. Tuma, M. Balliu, and R. Scandariato "Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis"
In submission to *The 9th ACM Conference on Data and Application Security and Privacy (CODASPY 2019), 2019*

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] S. Jasser, K. Tuma, R. Scandariato, M. Riebisch. "Back to the Drawing Board"
Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP 2018), 2018.

Research Contribution

I have contributed with planning and conducting the systematic literature review (Paper A). In this work I was responsible for the selection of studies, creating of the assessment criteria, data extraction and dissemination of results. In the empirical study reported in Paper B, I helped conducting the experiments in the second year. I also contributed with building the base line analysis (ground truth), assessing the reports based on the ground truth for both experiments, data analysis, and dissemination of results. For Paper C, I have contributed to forming the approach in during the workshops with our industrial partners, and writing the paper. Finally, I mainly contributed to the implementation of the Eclipse plug-in, the validation and dissemination of results in Paper D.

Contents

Chapter 1

Introduction

Today security threats to software systems are becoming a growing concern in many organizations. Thus security is considered early-on in the software development life cycle (SDLC) [1]. Practitioners that value security in their products adopt well established best practices, e.g. by applying secure design principles [2] and patterns [3]. Design models are often analyzed to assure the desired properties of the system. Model analysis can be done for different architectural perspectives (e.g., topological view, data view, access control and permissions, functional view, etc.) and on several levels of abstraction.

Threat analysis (threat modeling) is a method that strives towards validating the software architecture and discovering potential design weaknesses. It includes activities which help to identify, analyze and often prioritize potential security and privacy threats to a software system and the information it handles. The main reason for performing threat analysis is discovering potential risks early-on in the development and thereafter eliciting (or refining) security requirements.

First, reports show that only about one third of the surveyed organizations adopt threat analysis as part of their design process [4]. The manual effort that is today required for performing threat analysis may be a limiting factor for a more wide-spread adoption. In fact, empirical evidence indicates that techniques such as STRIDE can be repetitive and time consuming [5]. Meanwhile, there is a lack of security experts in organizations to handle such manual effort. To address the above issues, this work contributes with an investigation on how to find the most important threats faster. To cater to our partners from the automotive industry, we have focused on model-based threat analysis. We have proposed an extended notation (eDFD) and an accompanying analysis approach (eSTRIDE) in Paper C. The approach relies on making reductions to the problem and solution space before and during the analysis, respectively. Such reductions enable the analysts to focus the analysis and avoid discussions about threats with low priority. As such, this analysis approach is not appropriate for achieving threat coverage. The approach is initially validated with an illustration. A more extensive validation is ongoing work. We are conducting a case study in collaboration with two large automotive industries. The participants are split

into a control group using STRIDE and an experimental group using eSTRIDE. The two groups are given the task to analyze a large system using the assigned techniques. We plan to measure and compare the group performances based on the quality of analysis outcomes (i.e., the number of important threat that were discovered) and the timely discovery of important threats (when they were discovered in the process).

Second, evidence suggests a low recall of model-based threat analysis techniques such as STRIDE (about 0.5, as recorded in Paper B). This means that on average the number of overlooked threats is very high for such techniques. From our experience, threat analysis is in practice performed using design notations with little semantics. Further, for many existing techniques there is no correctness or completeness guarantees of analysis outcomes (as recorded in Paper A). This makes automated reasoning for quality assurance a very challenging task. Such guarantees (and their automation) are important for compliance to security standards (e.g., ISO26262 [6]) and software certification. Therefore, this issue is particularly relevant for the automotive domain. To address this issue we propose a formalized approach (SecDFD Analysis) for performing threat analysis on the design-level (Paper D). In information flow security, low level code is statically analyzed for a particular set of inputs to determine potential leaks of sensitive information. Initially the inputs are assigned so called *security labels*. Typically, a high label refers to a private input and a low label refers to a public input. The proposed approach contributes with a formal specification language (SecDFD) for a design-level notation and an automated analysis of information disclosure threats with label propagation and a security policy checker. The benefits of this approach are twofold: (i) the design-level specification language SecDFD is kept simple, yet equipped with security semantics to enable the automation and (ii) the automated analysis rests on the semantics of SecDFD security labels and thus provides completeness guarantees in the context of the modeled system.

The remainder of this Chapter is organized as follows. Section 1.1 narrows the research focus, whereas Section 1.2 provides background on the main topics of research. In Section 1.3 we provide a summary of the publications appended in the respective Chapters. The collective results and answers to the presented research questions are discussed in Section 1.4, followed by a conclusion and charted vision for future work in Section 1.5.

1.1 Research Focus

This section describes the research tracks and research questions of this work.

Figure 1.1 depicts a completion bar of three research tracks. Works drawn with a dotted line are either part of ongoing work or planned for future work. The systematic literature review (SLR) presented in Paper A is not shown in this Figure, as it was a result of an in-depth study of the state-of-the-art.

High effort. The first research track was oriented towards an industrial collaboration with the automotive industry. With respect to the current state

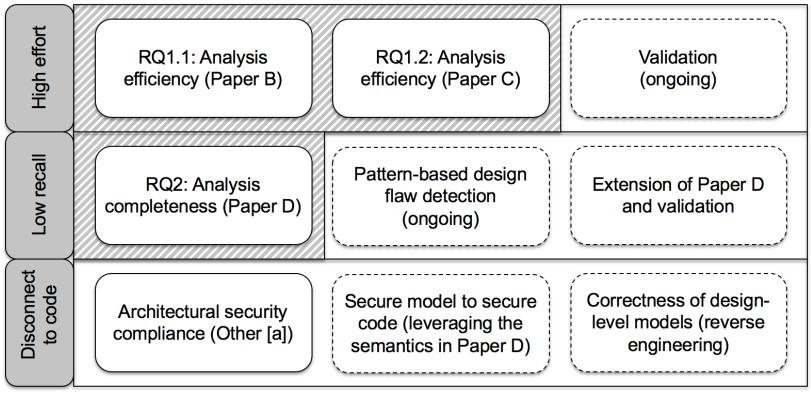


Figure 1.1: Research tracks and research questions of this work.

of practice, the most pressing issue is the lack of security experts. Increasing the efficiency of threat analysis has the potential to free valuable resources. For this reason, we were interested to look into possibilities to reduce the time spent on analyzing threats without sacrificing the quality of outcomes. Evidently, there is a trade-off between systematicity and timely discovery of important threats. In Paper C we explore this trade-off and provide a "short-term" solution with value to the industry. In parallel to this study we worked on building a deeper understanding on how the analysis procedure effects the technique performance. Specifically, we were interested to understand how the procedure of visiting the architecture facilitates designers in identifying threats. To this aim we looked into the scope of analysis, i.e. the number of elements analyzed at once. On the one hand, there exist such techniques that suggest practitioners to find threats to architectural components in isolation (e.g., STRIDE-per-element). Further down the line, some techniques consider a slightly larger analysis scope and suggest finding threats to a set of components (e.g., STRIDE-per-interaction). Finally there are end-to-end analysis techniques that suggest finding threats to a chain of components (e.g., proposed approach in Paper C). Our hypothesis was that the technique performance linearly increases with the increase of the analysis scope. We conducted an empirical comparison of two existing techniques to test this hypothesis (Paper B). The first goal of this work is to introduce an approach for finding important threats faster by enlarging the analysis scope.

RQ1. What are the effects of broadening the analysis scope on the quality of analysis outcomes? (Paper B, C)
To achieve this goal, we faced two challenges.

RQ1.1. What changes are required in the design model to facilitate a threat analysis focusing on important threats? (Paper C)
The first challenge was related to enlarging the analysis scope from a single component (or a pair) to a chain of architectural components. By doing so, the cognitive load of the human analyst increases also. A large cognitive load makes problem solving more difficult for the human brain [7]. Therefore, the first challenge is to enlarge the scope but at the same time abstract un-important

details (RQ1.1).

RQ1.2. What changes are required for a model-based threat analysis procedure to focus on important threats? (Paper C)
The second challenge was to reduce the manual effort as much as possible by introducing short-cuts during the analysis. The procedure of striving towards systematicity was not appropriate anymore. Therefore, changes to the analysis procedure were required (RQ1.2).

Low recall. The second research track was oriented towards increasing the formalism of model-based threat analysis to achieve completeness and correctness guarantees of analysis outcomes. Threat analysis is considered *complete* when all existing threats to the system and its assets have been identified and documented. For instance in Paper B we measure the recall of the techniques as the ratio of the correctly identified threats and all existing threats (including the threats that were overlooked). The analysis *correctness* (or precision) refers to the ratio of correctly identified threats and all identified threats (including falsely identified threats). The lack of formalism in threat analysis techniques makes analysis automation difficult. Therefore, the second goal of this work is to provide a practical formalized approach with certain completeness guarantees of outcomes (Paper D).

RQ2. What formalism is sufficient for an easy-to-use design-level threat analysis with completeness guarantees about analysis outcomes? (Paper D)
In contrast to the research goal mentioned above, this work aims towards finding a more "long-term" solution, which can be leveraged for assuring the quality of analysis outcomes. To achieve this goal we faced the challenge of understanding what level of formalism is required for a correct and complete threat analysis and can still preserve the simplicity of design-level notations, such as DFDs (RQ2.).

Disconnect to code. Design-level models that are used during threat analysis often become obsolete soon after the design phase. These models are seldom kept up to date due to resource constraints. Therefore, there is a disconnect between the planned security and implemented security. The third research track is focused on bridging the gap between the intended and implemented security architecture. We have started working on this track (other publications [a]), yet more effort is planned for the future. We refer the reader for more details about our future plans to Section 1.5.

1.2 Background

In this section we provide the background on the main topics related to this research. We clarify the terminology and introduce the topic of threat analysis of design models, automated analysis of design models, and security correspondence between design models and code.

1.2.1 Threat Analysis of Design Models

A necessary condition for a secure system is the correct definition and implementation of its security requirements [8]. However, the sufficient condition for a secure software is an unknown [1]. Thus we can only aspire to build software systems with acceptable levels of security. Sometimes referred to as security by design, secure software design is a term used to describe methodologies, techniques, tools, and best practices that facilitate building security into products throughout the entire software development life-cycle (SDLC). To this aim, threat analysis (modeling) techniques are used in the design phase of the SDLC.

We describe threat analysis in Paper A as such:

> "*Threat analysis* includes activities which help to identify, analyze and prioritize potential security and privacy threats to a software system and the information it handles. A threat analysis technique consists of a systematic analysis of the attacker's profile, vis-a-vis the assets of value to the organization. Such activities often take place in the design phase and are repeated later on during the product life-cycle, if necessary. The main purpose for performing threat analysis is to identify and mitigate potential risks by means of eliciting or refining security requirements."

Existing threat analysis techniques are commonly categorized into three non-exclusive categories, depending on the focus of the technique. *Risk-centric* threat analysis techniques focus on assets and their value to the organization. They aim at assessing the risk and finding the appropriate mitigations. Their main objective is to estimate the financial loss for the organization in case of threat occurrence. Therefore, when risk-centric techniques are used assets dictate the priority of elicited security requirements. For instance, CORAS [9] is an approach consisting of a modeling language, tool, and method for performing a risk analysis. CORAS uses asset diagrams for refining the target system and threat diagrams for identifying threats. The approach uses structured brainstorming in the context of workshops as the method for identifying threats. All the while, the identified threats are documented with threat diagrams. Finally, the approach enables risk estimations and their treatment with treatment diagrams.

On the other hand, *attack-centric* threat analysis techniques focus the analysis around the hostility of the environment. They put emphasis on identifying attacker profiles and the complexity of attacks exploiting any system vulnerability. Attacker profiles are commonly distinguished with respect to attacker capabilities, motivation, window of opportunity and number of attackers and their organization (i.e., singleton, groups, hacktivists, terrorists, etc.). The main objective of attack-centric techniques is to achieve high threat coverage and identify appropriate threat mitigations. Notably, attack trees [10] can be used in order to decompose attacker actions into possible events (leaf nodes) that could trigger the attack. Similarly, Fault-tree analysis [11] (FTA) and Event-tree analysis (ETA) also use trees for analyzing safety properties of a system.

Finally, the literature also mentions so-called *software-centric* threat analysis techniques. This group of techniques focus the analysis around the software. For example, in STRIDE [4] [12] the analysis is performed on Data-Flow Diagrams (DFDs), which provide a high-level architectural view of the software. STRIDE is an acronym for categories of security threats that are considered during the analysis. In essence, this method strives for achieving threat coverage by systematically visiting each element (or interaction) in the DFD and by mapping threat categories to DFD elements in to help brainstorming security threats. For a more complete list of existing threat analysis techniques we refer the reader to Paper A.

1.2.2 Automated Threat Analysis of Design Models

Many approaches propose to automate the analysis of design models to minimizing the resources needed for performing threat analysis in organizations. Often such approaches are able to semi-automate the analysis. That is, they automate parts of the analysis technique, while some parts still require manual effort. Depending on the sophistication of the analysis automation, we continue to describe knowledge-based automation of threat categories, graph-based automation, and formal approaches.

The Microsoft Threat Modeling tool (MTM) [13] is a tool developed to support the STRIDE methodology. MTM provides the ability to graphically represent the DFDs. The tool further enables the generation of threat categories for individual DFD elements with the use of the STRIDE threat-to-element mapping table. Other works approach threat analysis automation in a similar way. For instance, Sion et al. [14] present an approach which aims to automate threat mitigations (i.e., possibly matching security threats to existing security solutions). Yet both approaches automatically generate threat categories, rather than actual security threats.

Design models (e.g., software architecture) can be sometimes represented with graphs. A common method for automating the analysis of design models is by discovering patterns in such graphs. Depending on the analysis focus, such patterns represent threats, vulnerabilities, or security solutions. For example, Almorsy et al. [15] proposed an approach for automating the security analysis by capturing vulnerabilities and security metrics. To this aim, they develop an approach for modeling a system and specify formal signatures of vulnerabilities and metrics with the Object Constraint Language (OCL). Similarly, Berger et al. [16] proposed a knowledge-based approach for abstracting the architecture to graphs and querying such graphs to detect vulnerabilities.

When more effort for modeling (and analysis of) system design is justified, formal approaches can be adopted. Such approaches typically require the modelers to have a strong background in formal methods and topics alike. On the other hand, the automation of analysis reasoning in a formal setting comes sometimes for free due to the underpinned semantics. Yet the efficiency of such approaches is often still a challenge. For instance, early work of Sheyner et al. [17] automate the generation of attack graphs, based on the well understood formalism of attack trees. Later work by Ou et al. [18] builds on that to increase

the scalability of attack graph generation. Further, due to the crosscutting nature of security concerns, Xu et al. [19] approached automating threat analysis with aspect-oriented petri nets. The authors model the intended functions and security threats with Petri nets, whereas they model threat mitigations with Petri net-based aspects. Given the presented semantics, the authors are able to construct a search tree and verify whether certain threat paths are possible in the model.

1.2.3 Security Correspondence Between Model and Code

Building software from an architectural design does not guarantee a correct implementation. Further, correct implementations of planned architectures undergo maintenance which can gradually lead to architecture degradation. We provide a brief introduction into the existing work on bridging the gap between the intended and implemented security architecture.

1.2.3.1 Secure Models to Secure Code

One of the basic principles of Model-Driven Engineering (MDE) considers models as first class entities and any software artifact as a model or as a model element [20]. One aspect of MDE is model transformation from model to code (i.e., generating code). Code generation has the potential to carry over security solutions designed in the model. In the case of MDE the link between the model and code is explicit. Therefore, the implemented security is likely to correspond to the modeled security, but is not guaranteed. We provide a short description of a few Model-driven Security (MDS) approaches.

UMLSec. UMLsec is one of the most popular UML-based MDS approaches [21]. With UMLSec security requirements, threat scenarios, security concepts, security mechanisms, and security primitives can be modeled by using stereotypes (UML profiles), tags, goal trees and constraints. It is further possible to formally analyze UMLSec diagrams against security requirements. As opposed to other MDS approaches UMLSec considers multiple security objectives, namely, confidentiality and integrity. The approach has been combined with Secure Tropos [22] tackling security from the requirements phase [23]. It has been used in industrial case studies [24–26] and provides tools support [27]. UMLSec is considered as the most complete and mature MDS approach. Yet, the approach lacks support for automated model to code transformations.

SecureUML. SecureUML is an approach that aims to bridge the gap between security modeling languages and design modeling languages [28]. It is used for modeling role-based access control (RBAC) and is limited to one security objective, namely, authentication. Compared to UMLSec, SecureUML provides some automated transformations. For instance, access control infrastructures for server-based applications can be generated automatically from SecureUML models. SecureUML provides semantics [29–31] for a formal analysis of security design models.

SecureMDD. SecureMDD is a MDS approach that facilitates the development of security-critical applications based on cryptographic protocols [32]. In particular, it is a domain-specific approach tailored to smart card applications. Starting from a platform-independent model, SecureMDD generates a formal abstract state machine specification and Java code. The state machine specification is used for formally proving security properties of the therefrom generated Java code.

For a complete review of existing secure design methodologies and notations we refer the interested reader to systematic literature reviews [33], [34], and survey [35].

1.2.3.2 Security Implementation to Secure Model

Existing approaches (e.g., SECORIA [36]) have studied how to obtain security facts about the architecture just by observing the code. To this aim, a combination of static and dynamic program analysis techniques are used. *Static* code analysis techniques perform the analysis "off-line", e.g. based on a model instance or source code, without accounting for program inputs. Such techniques analyze all possible executions of the program and are forced to make certain approximations. *Dynamic* code analysis techniques take program inputs into account (typically a single input). They use runtime program information (e.g., executable files, external libraries, dynamic memory allocation, etc.) to perform the analysis. This allows greater precision for a particular input, but also causes lower correctness guarantees for other program inputs.

Empirical evidence suggests that in general the accuracy of obtaining system architectures from the implementation is low [37, 38]. We refer the reader to the related work and discussion sections of Paper D and other publications ([a]) for more background on this topic.

1.3 Paper Summaries

This section includes a summary of the appended papers. In the summaries we describe our research goals, adopted methods, main contributions, limitations, and position our research with respect to the related work.

1.3.1 Threat Analysis of Software Systems: A Systematic Literature Review (Paper A)

The number of existing threat analysis techniques makes it difficult for practitioners to make informed decisions about selecting the appropriate method for adoption in their organizations. Further, the existing literature on systematizing the knowledge about threat analysis is limited [39] and does not provide a complete set of existing techniques. The initial goal of Paper A was to catalog and characterize the existing threat analysis techniques. The second goal was to provide future research directions and to address when the techniques could be adopted by practitioners. In this study we compare 26 identified methodologies for what concerns their applicability, characteristics of the required input for analysis, characteristics of analysis procedure, characteristics of analysis outcomes, ease of adoption, and their validation. The study was conducted by strictly following the existing guidelines [40] and included an elaborate strategy, including backwards snowballing [41] for searching the literature and extracting the data. This study also discusses the obstacles to be overcome for adopting identified techniques to current trends in software engineering (i.e., Development and Operations, Agile development) and their generalization across domains. In addition, the study provides recommendations to practitioners for technique adoption depending on the amount of planned resource investment.

Contributions. The main findings of the SLR are: (i) Existing threat analysis techniques lack in quality assurance of outcomes, (ii) the use of validation by illustration is predominant, (iii) the tools presented in the primary studies lack maturity and are not always available, (iv) there is a lack of Definition of correctness and completeness guarantees for analysis outcomes.

Limitations. Substantial work was done by a single researcher, therefore we consider a risk of subjectivity as an internal threat to the validity of Paper A. Particularly, the selection of studies and the data extraction was mainly performed by the first author. We have mitigated this threat by planning and executing random quality assessments. Generally, the validity of SLR results depends on the external validity of the selected studies. To mitigate this threat we have developed and applied a rather conservative inclusion criteria, excluding gray literature papers, position papers and such. Further, Paper A restricts the search of literature to a subset of existing venues and digital libraries. Searching a non-representative set of existing literature would have harmed the validity of our conclusions. Yet we included top ranked venues and searched among the most influential works in software engineering.

Related work. To the best of our knowledge Paper A presents a first

extensive systematic literature review of existing threat analysis methodologies. Previous work has presented reviews (e.g., Mellado et al. [42]) and surveys (e.g., Salini et al. [43]) of security requirements engineering (SRE) methodologies and techniques. While a subset of techniques is also mentioned in these works (such as [44], [45], [46] and [47]), our contributions are unique in its research questions and in providing a more complete list of existing threat analysis techniques. With respect to risk analysis and assessment, Latif et al. [48] present a systematic literature review in the field of cloud computing with a focus on risk assessment. Further, Cherdantseva et al. [49] present a state-of-the-art review of the literature on cyber security risk assessment methods for SCADA systems. Both of the aforementioned reviews are narrowly scoped to one domain and do not assess the characteristics of the studied works.

Summary. In summary, this study provides an empirical analysis of existing threat analysis techniques. It was performed as part of an in-depth study of the state-of-the-art, hence it does not contribute to any of the research questions listed in Section 1.1. One of the main goals for conducting a SLR is to identify knowledge gaps, which we summarize in the main contributions above.

1.3.2 Two Architectural Threat Analysis Techniques Compared (Paper B)

Among other things, threat analysis techniques may differ in the scope of analysis. We were interested to study the effects of a different analysis scope on the technique performance. To this aim, Paper B rigorously compares two existing techniques with different scopes, namely STRIDE-per-element and STRIDE-per-interaction [4]. In particular, this study measures the respective techniques' performance in terms of their productivity, precision, and recall. The study was conducted in the context of in-vitro experiments with master students. We have adopted a standard design for a comparative study of one independent variable with two values, namely, Element and Interaction [50]. The participants were split into two treatment groups, the Element and Interaction treatment group. They were further assigned to teams. The teams were instructed to (i) create a DFD and (ii) perform a threat analysis of a familiar system using the respective technique in a fixed time frame and report the analysis results. We collected the measure of effort (in minutes) spent by each team on both sub-tasks (DFD creation and threat analysis). The final reports were compared to a ground truth analysis to collect the measure of true positives (TP), false positives (FP) and false negatives (FN). On that basis, the study collects evidence about statistical significant differences (SSD) between (i) the average productivity (number of TP per hour) of treatments, (ii) the average precision ($TP/(TP + FP)$) of treatments, and (iii) the average recall ($TP/(TP + FN)$) of treatments. Beyond that, the study controls for any possible discrepancies between the populations of the treatment groups (i.e., with an obligatory entry and exit questionnaire) and gathers subjective feedback on the usability of the techniques.

Contributions. We observed slightly better results for the STRIDE-per-

element technique (SSD between the average recall of treatments, ELEMENT : 0.62 INTERACTION : 0.49). We also observed a slightly better average productivity (no SSD, ELEMENT : 4.35 $TP/hour$ INTERACTION : 3.27 $TP/hour$). One possible explanation for the difference in treatment performance is that STRIDE-per-interaction is more difficult to perform for novice analysts [4] (such as our participants). STRIDE-per-interaction requires the consideration of pair-wise interactions of elements, thus increasing the cognitive load for the analyst [7]. Accordingly, we have observed that on average the INTERACTION teams produced larger DFDs, indicating that interactions lead to participants constructing a more complex problem space. The increased cognitive load and lack of domain expertise might have effected the performance of the INTERACTION teams. In terms of overall performance, this study concludes that there is no significant difference in the observed treatments.

Limitations. The report analysis was carried out by a single researcher, which contributes to the internal threat of subjectivity. In addition, possible mistakes could have been made while building the ground truth and assessing the reports handed in by the participants. The size of the target system was purposefully large enough to position the analysis in realistic circumstances. Therefore, participants were facing a complicated and time consuming task. This has raised the risk of over-loading the participants. Yet, we have mitigated this threat by supervising the experiment in a span of 4 hours. In these four hours we advised the participants to first elicit all the threats. Afterwards, additional time was given for writing the reports, which were assessed to include the exact threats that were identified under supervision. The use of student participants instead of professionals is considered problematic for generalizing the results. Commonly referred to as convenience sampling [51], this kind of participant sampling is considered controversial due to drawbacks [52]. Yet, studies have shown that the difference in performance of professionals and graduate students is often small [53–55].

Related work. Scandariato et al. [5] analyzed a previous version of STRIDE-per-element and evaluated the productivity, precision, and recall of the technique in an academic setting. We remark that our study has some discrepancies with respect to the observed productivity (4.35 in our study vs. 1.8 threats per hour), precision (0.6 vs. 0.81), and recall (0.62 vs. 0.36). However, a direct comparison is not entirely possible, as the two studies use different versions of STRIDE-per-element (our being the most up-to-date). A privacy oriented analysis methodology (LINDDUN [56]) has been evaluated with 3 descriptive studies [57]. LINDDUN is inspired by STRIDE and is complementary to it. Both techniques start from a representation of a system, which is described as a DFD. Similarly, the authors of the descriptive studies assess the productivity, precision (correctness) and recall (completeness) of the technique, as well as its usability. The work of Karpati, Sindre, Opdahl, and others provide experimental comparisons of several techniques, namely, misuse cases (MUC) and mal-activity diagrams [58–60]. Finally, several empirical comparisons were performed where MUC were compared to other techniques (e.g., mal-activity diagrams [60], attack trees and Common Criteria [61]). In comparison to the above mentioned studies the novelty of our work is the study of performance impact due to the difference in the analysis scope.

Summary. In summary, this study shows that there are little performance differences between the studied techniques. This finding indicates that the performance of techniques visiting architectural elements in isolation does not differ significantly to the performance of techniques considering pair-wise interactions. It further suggests that possible performance differences might occur when the analysis scope is further enlarged to end-to-end scenarios.

may be observed when comparing only by further enlarging the analysis scope and changing the analysis procedure. Therefore, Paper B contributes towards answering **RQ1**.

1.3.3 Towards Security Threats That Matter (Paper C)

This paper is motivated by the need to increase efficiency of threat analysis techniques in the automotive industry. To this aim, we enlarge the analysis scope and tailor the analysis procedure to focus on important assets. STRIDE [4] is a popular approach used in the automotive today. Yet, empirical evidence indicates that this technique can be tedious and time-consuming due to the threat explosion problem [5]. For this reason, we have proposed a novel approach inspired by the well-known STRIDE. The proposal comes as a result of numerous workshop sessions with our industrial partners that further highlighted the needs and shortcomings of existing approaches. As a collection of lessons learned, the first author synthesized the approach and validated it with an illustration.

Contributions. We propose to prioritize threats before the analysis begins based on assets and their priorities. This requires practitioners to enrich the architectural model (i.e., built an extended DFD or eDFD) with assets, their sources, targets, security concerns and priorities, domain assumptions, communication channels, and existing security solutions. The DFD extensions are made to end-to-end user scenarios around highly prioritized assets. During the analysis procedure, such scenarios become the scope of the analysis. Finally, the approach proposes initial guidelines for handling threat explosion by introducing "short-cuts" both before and during the analysis. The initial illustration shows a reduced number of low prioritized threats, yet does not provide sufficient evidence for the potential benefits of the approach.

Limitations. The proposed approach relies on the correctness of the DFD enrichments (e.g., domain assumptions). This means that the presence of a domain expert is mandatory. We see potential in countering this limitation by enriching future tool support with domain knowledge. Another draw back is that the approach assumes that there are indeed non-critical areas in the architecture, which can be disregarded for threat elicitation. If the architecture is highly critical and contains only assets with high security objectives, the abstractions will be limited. Finally, the approach was initially validated using an illustration by the first author. Ongoing work is extending this validation in the context of an empirical case study with practitioners.

Related work. Significant work has been done in the area of threat analysis and risk assessment (TARA) methods in the automotive domain. Macher et

al. [62] performed a review of TARA methods in the automotive context. In their main findings, the authors identify most applicable TARA methods for early phase analysis (i.e., EVITA [63], HEAVENS [64] and SAHARA [65]). Yet, these techniques do not tackle the threat explosion problem. Beyond the domain of automotive, there are existing risk-centric threat analysis techniques (e.g., CORAS [66], PASTA [67]). These techniques are not semi-automated and do not explore the possibilities for solution or problem space reduction. When it comes to semi-automation of threat analysis, several approaches have been proposed in the past. Notably, the approach in Paper C relates to extracting threats from DFDs by J. Berger et al. [16]. Similarly to our work, the authors extend the DFD notation with assets, security objectives and topological behavior. Furthermore, they also develop a set of guidelines, which are used to build a threat model of the architecture. However, these rules are used to discover only cataloged threats and do not aim to handle threat explosion.

Summary. In summary, this paper proposes a novel approach for performing an risk-first threat analysis with an enlarged scope of analysis. The approach leverages the enriched architectural model (eDFD) and guidelines for handling threat explosion to guide the analysis towards the most important threats. As a result, such analysis can lend itself useful for practitioners in favor of sacrificing systematicity for a timely discovery of highly prioritized threats. This paper contributes to answering **RQ1**.

1.3.4 Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis (Paper D)

Paper D is motivated by the low recall of existing techniques using informal design notations, (such as STRIDE [5]). On the one hand, literature describes formalizations of DFDs [68] which often result in a complicated language hindering their usability. On the other hand, several studies propose threat analysis automation (e.g., by means of pattern matching [15, 16]) with no correctness or completeness guarantees of analysis outcomes. Inspired by code-level information flow security [69,70], we propose a formal approach to analyze security information flow policies at the level of the design model.

Contributions. The main contributions are two-fold: (i) a light-weight extension of the modeling capabilities of DFDs, and (ii) a tool-supported, formally-based flow analysis technique. The extension of the DFD notation requires the designer to provide the intended security policy for system assets. In addition, the designer is required to specify an abstract input-output security contract for the computational nodes. The designer also specifies a global security policy for all system assets, based on which the design flaws are identified. The additional information mentioned above is leveraged in the analysis procedure. The second contribution of this work is a formally-based flow analysis technique that propagates security labels across the design model. The approach is implemented and packaged as a publicly available plug-in for Eclipse. We have validated the approach using 4 real world case studies.

Limitations. We have only considered a subset of possible node types and their semantics. For instance, we initially consider only nodes of type store,

compare, join, use, split, forward, copy, encrypt, and decrypt. The formal model can be extended to support more node types, such as authentication and authorization. Further, the provided Eclipse plug-in currently supports the specification of local temporal dependencies by explicitly enumerating data flows. A possible extension would include an algorithm which can detect such dependencies with label propagation. For instance, a stable state of security labels can be used as the satisfying condition for exiting the propagation algorithm. Finally, the validation is limited and was largely done by a single researcher. Particularly, it would be beneficial to validate the approach on larger cases with known and unknown design flaws.

Related work. Jürjens et al. [21, 71] have proposed UMLSec, an extension for UML to model security aspects in system design and prove security properties, such as secrecy. UMLSec's formal semantics scales very well as they apply to a variety of model types (such as activity diagrams, statecharts, sequence diagrams, etc.). Similarly to our approach, UMLSec defines a system as a composition of subsystems and enables modeling a security policy and attackers. In contrast, Jürjens et al. [21] focus the analysis on attacker behavior, rather than the semantics of security labels on flows. Existing works have introduced formal semantics to DFDs. The extensions made to the DFDs are mainly used for expressing functional correctness, rather than non-functional properties. For instance, Leavens et al. [72] propose a DFD semantics that allows to specify the dynamic behavior of a concurrent system, and Larsen et al. [73] leverage formal specifications in the Vienna Development Method (VDM) language to formally reason about DFDs. We refer to the work by Jilani et al. [74] for an overview.

Summary. This paper proposes a formally-based threat analysis approach with minimal security extensions required in the design notation. Therefore, this paper contributes towards answering **RQ2**.

1.4 Discussion

This section provides the answers to the main research questions of this work. To this aim, we discuss the main findings of the appended papers. First, we discuss the main findings of studies conducted for the purpose of an in-depth study of the state-of-the-art (Paper A and Paper B).

Our assessments in Paper A show that existing threat analysis techniques are mainly applicable on the level of requirements, architecture and design. This is not very surprising considering that the main purpose for performing threat analysis is to elicit security requirements. Further, most of the studied techniques use architectural design models and requirements (usually in textual form) as inputs to the analysis procedure, which is in line with the first finding. Interestingly, the precision of most threat analysis procedures is based on templates and examples, such as textual descriptions of example threats. About half of the studied techniques consider risk to prioritize the analysis outcomes. The analysis outcomes of the studied techniques in turn are mostly threats. Yet, half of the techniques also produce security mitigations or requirements. Finally, we have found that not many of the studied techniques have a way to assure the quality of analysis outcomes.

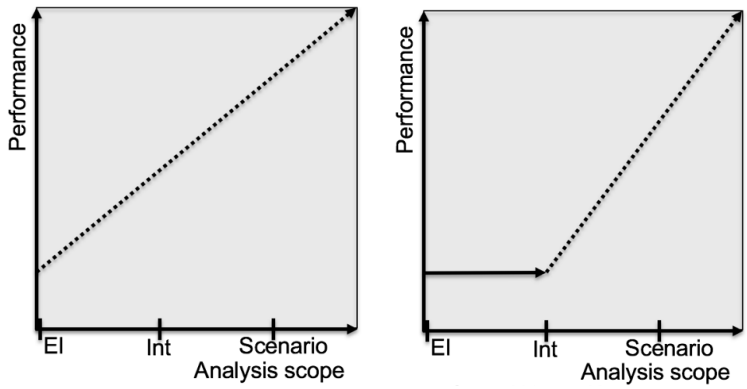
Paper A also investigates the ease of adoption for the studied techniques. About half of the studied techniques do not provide any tool support. We draw attention to the fact that most of the existing tools are used as visual aids for representing the architecture, rather than for actually analyzing it. The target audience for most of the studied techniques are security experts and security trained engineers. We hypothesize which characteristics are important for adopting the techniques in practice and provide the following guidelines for technique selection:

(a) If the organization plans to make *small* investments into adopting a threat analysis technique and security is *not prioritized* by management, we recommend selecting a technique that can be used without further modifications. Important criteria: Tool availability and maturity, sufficient documentation, low target audience and a light-weight analysis procedure.

(b) If the organization plans to make *small* investments into adopting a threat analysis technique and security is *prioritized* by management, we recommend selecting a technique that is systematic. Important criteria: Systematic analysis procedure, expert-based and preferably semi-automated.

(c) If the organization plans to make *large* investments into adopting a threat analysis technique, we recommend developing an "in-house" adaption of a promising technique. Important criteria: Systematic analysis procedure, potential for improvement (e.g., technology improvement).

Paper B investigates the effects of enlarging the analysis scope on technique performance. Figures 1.2(a) and 1.2(b) show the hypothesis and observed reality about the linear dependency between technique performance and the analysis scope. Empirical evidence shows that the productivity and precision are not significantly different for the observed treatments. Thus, in the context of the controlled experiments reported in Paper B, the analysis scope does not



(a) Hypothesis about technique performance.

(b) Reality about the performance of two techniques.

Figure 1.2: Hypothesis about technique performance in relation to the cognitive load required for brainstorming threats (left) and its reality for two specific techniques (right).

have a significant effect on the overall technique performance. On the other hand, we have collected interesting observations about the differences between treatments. Namely, the INTERACTION teams have on average produced slightly larger DFDs. Further, on average the ELEMENT teams have produced more duplicated treats, indicating that the notion of interactions is useful for correctly applying reductions suggested by STRIDE. In an ongoing case study we are measuring performance differences that might occur when the scope is further enlarged to end-to-end scenarios.

Table 1.1 presents a list of main findings ordered by the research questions defined in Section 1.1.

RQ1: What are the effects of broadening the analysis scope on the quality of analysis outcomes? The findings of Paper B indicate that techniques with a "small" difference in analysis scope do not differ in terms of overall performance. However, techniques with a "larger" difference in analysis scope (e.g., one element vs. a scenario) might differ in terms of performance. Further, Paper C suggests that enlarging the analysis scope might help manage the threat explosion problem, and in turn produce outcomes of similar quality in a shorter period of time.

RQ1.1: What changes are required in the design model to facilitate a threat analysis focusing on important threats? Reasoning about risk early-on requires a good understanding of the assets and their whereabouts in the system. During the asset analysis, the assets first need to be identified in the model (incl. asset source, target(s)). The importance of assets can only be deduced by discussing their security objectives (i.e., confidentiality, integrity, availability, accountability) and the priorities of those (high, medium, low). The annotated assets are required in the model to indicate where the model should be further extended. By focusing on highly prioritized assets, the

Table 1.1: Main research questions 1 and 2 with shortened answers.

RQ	Main findings	Paper
RQ1	• A risk-first approach is required to focus only on threats that will be mitigated.	C
	• The domain and security knowledge of the team has a large impact on the quality of outcomes and needs to be present in the architectural model *before* the analysis begins.	C
	• The design notation needs to be extended with more security information.	C,D
	• The complexity of the architectural model needs to be managed by making model abstractions wherever possible while enrichments are made only around assets with highest priorities.	C
	• During the analysis threats are only elicited for scenarios containing high priority assets.	C
	• During the analysis threats are only elicited for scenarios containing high priority assets.	C
	• During the analysis only threats that directly threaten a highly prioritized security objective are considered.	C
	• During the analysis only threats that are possible despite annotated domain assumptions and existing security solutions are considered.	C
	• Automation is required to further reduce manual effort.	C, D
RQ2	• The design notation needs to be extended with processing node types to determine what operations they performs over the assets.	D
	• A security specification language is required for design-like models (e.g., SecDFD).	D
	• The security condition and the attacker model need to be defined to reason about analysis completeness.	D
	• The semantics of the analysis procedure needs to be developed (e.g., security contracts of node types).	D

analysis is performed on parts of the architecture (reduction *before* the analysis starts). Domain assumptions, communication channels, and existing security solutions are notation extensions that are used to make reductions during the analysis. In the context of ongoing validation of eSTRIDE, we have found that analysts feel like a large amount of time is spent for establishing a consensus regarding domain assumptions (at the beginning of each session). Initial results of this validation indicate that in the beginning of sessions participants frequently discussed the domain without noting assumptions down explicitly. When domain assumptions were not explicitly written down more time was spent on re-visiting the same issues later-on.

RQ1.2: What changes are required for a model-based threat analysis procedure to focus on important threats? In Paper C we provide guidelines for handling threat explosion before (visited in RQ1.1) and during the analysis. To reduce the effort *during* threat analysis, we propose a slight departure from the analysis procedure suggested by STRIDE. First, eSTRIDE analysis is performed using eDFDs. Second, threats are only elicited for scenarios containing high priority assets. Third, the scope of the analysis is an end-to-end scenario (per asset). This means that the entire scenario is considered during threat elicitation, rather than single elements (or their interactions). Further, only threats that directly threaten a highly prioritized security objective are considered. For instance, tampering threats are compromising the integrity objective. Finally, only threats that are possible despite annotated domain assumptions and existing security solutions are considered. Initial results from the ongoing validation indicate that the guidelines for problem space minimization and effort reduction during analysis (i.e., eSTRIDE) indeed help focusing on important threats. Yet, we also observed that the participants showed hesitation when marking assets as un-important (i.e., assets with low priorities were rare). This limited the amount of possible effort reduction. Finally, we found that automation is required for further effort reduction. Ongoing validation confirms this claim. In particular, participants were often idle due to threat documentation.

RQ2. What formalism is sufficient for an easy-to-use design-level threat analysis with completeness guarantees about analysis outcomes? In Paper D we present the semantics of a security DFD (dubbed SecDFD), an extended design notation and a security specification language which is used in our analysis approach inspired by code-level information flow security. The extensions to the regular DFD are intensionally kept simple. First, it is important to specify what assets are under analysis. Thus modelers are required to extend their model with assets their sources, targets, and security objectives (i.e., confidentiality). Assets and their properties are required in order to be able to reason about information flows. Second, it is important to specify what happens to assets (and their properties) when they traverse nodes. Therefore, modelers are also required to specify the type of nodes (e.g., forward, store, encrypt, etc.). Further, the security condition and the attacker model need to be defined to reason about analysis completeness. The security condition defines what it means for a SecDFD to be secure, whereas the attacker model defines the attacker abilities. Finally, the semantics of the label propagation is required. At this point the type of nodes become

of paramount importance as they dictate which security contract should be applied upon label propagation. We present the semantics of SecDFD together with the complementary analysis approach in the appended paper.

1.5 Conclusion and Future Work

To conclude, this research contributes to four main knowledge gaps regarding threat analysis methods. First, empirical evidence and experience from practice suggest that most popular techniques (i.e., STRIDE) are manually-intensive and require a lot of resources in organizations. In this work we have presented an approach for leveraging enriched models to reduce this effort (Paper C). Second, manual analysis techniques (such as STRIDE) have been recorded to have a low recall (i.e., number of overlooked threats is high). Further, we provide evidence (Paper A) about a lack of correctness and completeness guarantees for the analysis outcomes. With respect to this issue, we contribute with a new formalism of SecDFD, coupled with an analysis approach (Paper D).

We envision three research directions in the future. First, we plan to continue our industrial collaboration. In this direction, we have ongoing work which empirically validates the approach proposed in Paper C. In this validation, we are comparing the end-to-end approach proposed in Paper C to the STRIDE-per-element technique. Therefore, we will complete our investigation about the possible performance differences caused by enlarging the analysis scope (previously studied in Paper B). We also envision that the approach will be tailored and optimized after the empirical results are analyzed. Following this, we plan to implement tool support for semi-automating the analysis. Second, we plan to continue working towards improving the recall of threat analysis techniques. In this respect, we are conducting ongoing work related to strengthening analysis automation with pattern-based design flaw detection. We also plan to extend the node types and their semantics in SecDFD (Paper D). As a follow-up study we plan to validate the approach on larger open source projects with known and unknown design flaws. As securing design-level models does not guarantee a correct implementation of the intended security, we plan to extend the work in Paper D to relate the security contracts to the implementation. The proposed semantics have the property of compositionality, which opens up possibilities for applying the semantics to annotated classes in object oriented programming languages (e.g., by leveraging code annotations in Java). Finally, in practice informal design notations are often preferred due to their flexibility and ease of use. The correctness of the models that can be created with such notations is sometimes under dispute. Without hindering such design notations, we are planning to look into other means to assure correctness. In particular, we plan to look into possibilities for design refactoring after implementation. To this aim, we plan to leverage existing reverse engineering tools to obtain call graphs, program interfaces, and dependency hierarchies. Those in turn can be analyzed for correspondence to the refactored model (e.g., by means of approximation and heuristic approaches). By doing so, the modelers would have

the opportunity to check that the refactored model is correct (i.e., corresponds to the current implementation) for the parts which should be unchanged, and that the refactorings only effect the desired parts of the model.