Software and Systems Modeling
https://doi.org/10.1007/s10270-025-01300-6

SPECIAL SECTION PAPER

Automatic security-flaw detection - towards a fair evaluation and comparison

Bernhard J. Berger[1,2] · **Christina Plump**[2,3]

Received: 5 May 2024 / Revised: 6 May 2025 / Accepted: 20 May 2025

Abstract

Threat Modeling is an essential step in secure software system development. It is a (so far) manual, attacker-centric approach for identifying architecture-level security flaws during the planning phase of software systems. In recent years, academia has presented ideas to automate threat detection that do not focus on a particular class of security flaws but offer means of pattern-based security flaw descriptions. However, comparing presented ideas (tools) for automated threat detection contains the potential for unwilling bias or restricted information content. In this work, we investigate the process of comparing automatic security flaw detection tools, clarify common pitfalls during this process, and propose a fair, reproducible, and informative comparison approach to be used as a community standard. We additionally discuss the necessary steps for the community to effectively implement this approach and support improved comparisons and evaluations in the future. We use a previously published case study to determine problems with current comparison techniques and classify different levels of comparison to be used for future reference as our main contribution. As a consequence, we propose using a model-based approach for specifying security flaws and apply an existing natural language-based catalogue to this model-based approach. Furthermore, we introduce an inspection process model (for providing a standard to specify findings of a threat detection process) to streamline the evaluation and comparisons of automatic security flaw detection tools. We provide an exemplary evaluation of this detection guideline and inspection process model along the lines of both automatic approaches from the original case study. All artefacts of the work are publicly available to support the research community and to create a common baseline for future tool comparisons.

Keywords Threat modeling · Dataflow diagrams · Security flaw detection · Automation · Interoperability · Comparison

1 Introduction

It is undisputed that software security and privacy are important issues nowadays. One essential way of improving application security is architectural risk analysis since 50% of security issues relate to the software's architecture [1]. Among others, Microsoft's Threat Modeling [2] is one possibility for conducting risk analysis for software systems. During Threat Modeling, experts first create an architectural model of the software system using dataflow diagrams. This hierarchical model comprises the software *components*, *external dependencies*, *assets*, *trust boundaries*, and *interactions* with(out) *dataflows*. As Threat Modeling is an attacker-centric approach, the experts then try to identify potential attack vectors or security flaws—scenarios of how to attack the software system—that attackers may use to gain access to the software assets. Afterwards, the experts perform a risk estimation, e.g., using the STRIDE approach [3], to identify the most critical threats. Since Threat Modeling is a manual process, it is time-consuming, and the results depend on the security expertise of the experts involved in the assessment [4]. Therefore, to guarantee good results, it

Communicated by G. Taentzer, A. Cicchetti, A. Pierantonio, and T. Kühne.

✉ Bernhard J. Berger
bernhard.berger@tuhh.de

Christina Plump
christina.plump@dfki.de

[1] School of Electrical Engineering, Computer Science and Mathematics, Hamburg University of Technology, Am Schwarzenberg-Campus 3 (E), Hamburg 21073, Germany

[2] Faculty 3 – Mathematics and Computer Science, University of Bremen, Bibliotheksstraße 1, Bremen 28359, Germany

[3] Cyber-Physical Systems, DFKI GmbH, Bibliotheksstraße 5, Bremen 28359, Bremen, Germany

Published online: 21 July 2025

is necessary to involve application, domain, framework, and security experts. This, again, makes the approach resource-consuming.

Different publications identified several improvement points in Microsoft's Threat Modeling [5, 6]. Shostack concludes many of these points: *"There are two categories of features that people often ask for that are worth a brief discussion: automated model creation and automated threat identification."* [7]. The first category aims to automatically create dataflow diagrams based on an existing software implementation, a common use case when assessing legacy systems. The challenge here is that it is not trivial to automatically detect components and sensitive information of a system without additional information. The second category aims at automatically detecting security flaws in a manually created or extracted architectural view. Therefore, it is necessary to capture and formalise knowledge on architectural security flaws and make them automatically checkable.

In addition, Yskout et al. identified the need for better interoperability between different threat modelling tools that allow the exchange of threat modelling artefacts and suggested **developing a knowledge base** of threat modelling knowledge [8]. Currently, researchers **cannot exchange dataflow diagrams or security rules** (following Tuma et al., we will also call these security rules *guidelines* [9]), which slows down research in automated threat modelling. This lack of exchange is mainly due to focusing on different security aspects, using different models, and formalising different security rules. Hence, even if researchers or practitioners from the industry are willing to change their architectural views or descriptions of security flaws, they cannot necessarily be used as such. However, they must undergo some transformation to make them useful for the secondary tool or use case.

Additionally, the lack of interoperability is accompanied by a lack of standardisation. While classically, tools are evaluated using standardised measures such as *precision*, *recall* or the *f1-measure*, their results can vary depending on the methodology choices the researchers make, which we will showcase in Section 3. These issues (among others) hinder a fair and unbiased evaluation of security-flaw detection tools and a sensible comparison of one security-flaw detection tool to another. Both are, however, important to nurture research to develop ever-improving security flaw detection approaches that can be used with decreased required resources.

Our overall goal is to derive a **fair, reproducible and informative approach** (we will explain these terms in Section 2.3 in greater detail) to evaluate and compare automatic security-flaw detection tools that the community can agree on and continue to use to further research quality as a whole.

With this paper, we aim to take the first steps towards this overall goal by contributing the following aspects[1]:

C1 Analysis of pitfalls for unwilling bias in evaluation (this contribution is mainly based on the results of [10], but restructures and conceptualises the original findings in more detail).

C2 Derivation of guideline and inspection process model for security flaws and the detection process (new contribution).

C3 Standardised statistical evaluation to evaluate and compare results (partially new contribution).

C4 Detailed discussion on further steps necessary towards the overall goal of a **fair, reproducible and information evaluation and comparison approach** based on work presented in this publication (new contribution).

C5 All artefacts of the paper are published online, including the inspection guideline adhering to our introduced model for detecting securiy flaws, and the corresponding models.[a]

[a] The artefacts are available at https://doi.org/10.5281/zenodo.15477264.

The remainder of the paper is structured as follows: First, we will introduce our view on evaluating automatic security flaw detection tools and the problems that arise within Section 2. We will also state our research questions here. In Section 3, we will use our findings from MODELS'23 to showcase these problems with a case study of comparing two automated security flaw detection tools. Section 4 will discuss aspects of the process that can be standardised. Section 5 introduces our solutions for the referenced problems in Section 2. We discuss our results in Section 6 and conclude the paper in Section 7.

2 Background & problem statement

This section briefly explains how threat modeling works and how automated approaches implement it. Additionally, we represent a sketched process of the current form of evaluating automated security-flaw detection tools. With this at hand, we will formulate our overall goal and the related research questions for this paper.

[1] As this publication is based on a previously published case study [10], we state in parentheses which contributions stem from the original publication and which are new contributions.

2.1 Threat modeling

Threat modeling is an attacker-centric manual security-flaw detection process [2]. It requires a broad range of expertise to be appropriately applied. Experts, such as developers, software architects, domain experts, and security experts, provide this expertise in real-world assessments. Threat modeling consists of four subsequent steps that the experts conduct jointly.

Step 1 – View Creation Experts create data flow-centric architectural views of the system under investigation. Therefore, they use typical use cases, sketch the involved software components, and how they exchange data [2]. Additionally, they add security-relevant information [11]. Depending on the focus, the detail level of the created views differs. While it is possible to use any modelling language, informal dataflow diagrams are frequently used as they are easy to understand and sketch without tool support [2].

Step 2 – Threat Identification While the first step relies on system and domain experts, the second requires security knowledge. In this step, they systematically identify potential attack vectors. To this end, they focus on every architectural element and try to find ways attackers can compromise the element or its processed data. Typically, they use a structured way, such as *STRIDE* [2], which focuses on attacks on the information security goals *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service*, and *Privilege Escalation*. This step results in a list of potential security flaws in the system.

Step 3 – Mitigation Search Now, developers, software architects, and security experts identify mitigations to the identified potential security flaws, which the security experts then validate. This check follows the idea that a system is—by default—insecure, as long as no valid security measures are in place.

Step 4 – Risk Analysis Lastly, the experts perform a risk analysis for the remaining untackled problems, e.g., by estimating the exploitability and severity of the security flaws and using a risk matrix [12]. The estimation depends on technical, domain-specific, and organisational aspects. The risk analysis results in a ranked list of the remaining risks for decision-makers who decide which flaws should be worked on.

Due to its informal and manual nature, the threat modeling approach does not claim to identify all existing security flaws, but it provides a structured approach for improving a system's security. Its result depends on the experts' knowledge. Nevertheless, the approach helps secure systems and reduces the number of security incidents for many companies [11].

2.2 Automating threat detection and analysis

Shostack formulated the need for tool-assisted automation in threat modeling, cf. Section 1. Various tools for *Automated Threat Modeling* started investigating this topic [13]. Essentially, there are two areas of automation: *Automated View Creation* (automation of the *View Creation*) and *Automated Security-Flaw Detection* (automation of *Threat Identification* and *Mitigation Search*). Both automations use reverse engineering and static program analysis techniques to identify architectural components, data flows, and implemented security concepts. Automation concepts for the *Threat Identification* formalise knowledge of security concepts. Since dataflow diagrams are loosely defined, every tool uses a different dataflow-diagram formalisation with varying advantages and disadvantages. Consequently, since the security formalisation is applied to the chosen dataflow-diagram formalisation, the security formalisations differ between the approaches. Such a formalised security rule is a security antipattern, as it describes patterns that indicate vulnerabilities at the architecture level, thus detecting the absence of security concepts.

Tools for *Automated Threat Detection* can follow two approaches when reporting their findings. The first approach lists all potential security flaws—regardless of the fact whether they are mitigated or not—and adds information on the chosen mitigation. The second approach lists only not mitigated security flaws, reducing the list of findings. The area of conflict automated tools face here is that reporting all possible findings (even the mitigated ones) may overwhelm users (which may interpret mitigated threats as false positives). At the same time—from a security and risk analysis point of view—it is important to keep these findings. First, every decision to consider a threat as mitigated is based on various assumptions that might become incorrect at some point. Second, security experts may suggest additional security measures to implement defence-in-depth concepts[14].

Tools usually do not tackle the *Risk Analysis* aspect as the impact and severity heavily depend on the application domain, country-specific legal constraints, and may even be contract-specific.

2.3 Evaluation and comparison process

After discussing the usual form of automated approaches for security flaw detection as well as the standard threat modeling approach, we turn to focus on a standard evaluation setup. Usually, there are two main forms of evaluation: First, evaluating a single tool (mostly, the one that has just been developed, see, e.g., [15–19]), and second, comparing two (or more) different tools to one another (this includes comparing the self-developed tool to other already published tools, e.g., [10, 20]). We will refer to these settings with *evalu-*

ation and *comparison*, respectively. Of course, both forms can morph into one another, as evaluation results are mostly used for a succeeding comparison as well.

Evaluation: Let us consider Figure 1 and assume a researcher plans to evaluate his automated security flaw detection tool. As a first step, the researcher will need a set of systems (see *Systems* in Figure 1) for evaluation, for which the researcher has to create formalised architectural views as required for the tool. The formalisation approach can differ depending on the tool to evaluate. For the tool, the researcher has already decided on a set of security concepts (see *Detection Rules* in Figure 1) that the tool can detect. These are a subset[2] of all possible security concepts (see *Security Concepts)* in Figure 1). The researcher can then run the tool on the formalised views of the system to obtain a list of findings (right-hand side *Security Flaws (Findings)* in Figure 1). Depending on the tool, these might be detected or potential security flaws, with a note of whether they have been mitigated or not, cf. Section 2.2. The findings are in a format that is defined by the tool. As ground truth to compare to, the researcher will mimic an oracle, i.e., manually analyse the given systems according to some set of security concepts. This set of concepts is usually similar to the security concepts the tool is built for (see the *oracle* and the *knowledge* of the oracle in Figure 1). Again, the resulting findings for the given systems are listed as findings and potentially their mitigations (see left-hand side of *Security Flaws (Findings)* in Figure 1). Sometimes, this oracle will be mimicked repeatedly by different oracles and then merged in a predefined way to address bias issues. Both sets of findings will (hopefully) intersect with the *Gold Standard (Security Flaws)*, which represents the list of all missing security concepts (given the entire set of security concepts applied to the set of systems). As this *Gold Standard* can not be achieved, the ground truth (left-hand side of *Security Flaws (Findings)*) is used as a substitute. The set of findings based on the tool is then evaluated with respect to the oracle-based findings, and some result metrics (usually *precision*, *recall* and *f1-measure*) are returned. These metrics are then used as a means to describe the quality of the automated tool.

The process can be compared to the experiment process described in [21] in Chapter 6, where the potential relationship between cause and effect, in theory, is aimed to be shown via an experiment with treatment and outcome. Although the comparison has deficiencies, the treatment can be related to the knowledge base (or detection rules), while the cause is the security concept (or the absence thereof). A standard issue of *construct validity* would be the number of security concepts covered by the knowledge base or detection rules.

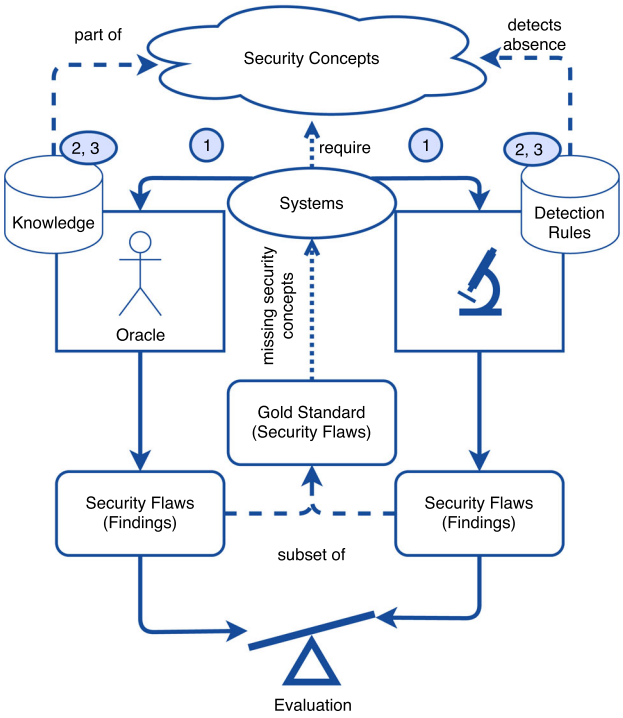

Fig. 1 Depiction of a typical evaluation process and its context. The numbers shown in the circles correspond to the process steps explained in Subsection 2.1. As mentioned in Section 2.2, the risk assessment is not tackled by tool automation and thus is not mentioned here

Comparison:

Usually, it is not sufficient to simply evaluate one's tool and report metrics, but additionally, a comparison to other tools already present in the literature is required. Several comparison levels can be seen in the literature [10, 20], although explicit (and quantitative) comparisons are not frequent[3] in the domain of automated threat modeling. Nevertheless, [21] provides a comprehensive guideline on how to perform experiments in software engineering, and therein also covers comparisons. However, these do not fit the domain and challenges of automated security flaw detection in its entirety. The lack of comparison in the literature is, however, an indicator for our introductory statement that a community standard for comparing tools is not present. The following overview and explanation are based on a mixture of experiences from other sub-domains in software engineering, thought processes prior to our comparison study, and analyses of *threats-to-validity* or *related work* sections from publications, which rather focused on evaluations instead of comparisons. Figure 2 shows an overview of comparing two tools and influencing factors that may or may not be kept consistent. Both sides show the evaluation parts of one tool,

[2] *Subset* is here used as $\subseteq$, i.e., in theory, all possible security concepts could be included in a tool. This is, however, highly unlikely, as of now, we can never be sure to have knowledge of all security concepts.

[3] This is also the reason, why we include our own comparison work that is included as case study.

respectively, hence, the numbers 1 and 2. Each tool has a set of *systems* and *rules* on which it is evaluated. An *oracle* produces *findings* for these systems and rules. The tool also produces findings for the set of systems and rules, which we refer to with *findings-analysed*. In the end, both tools produce *metrics*.

In the following, we will introduce levels of comparison by restricting these elements part by part to equality, or what may be thought of as equality (The terminology of *level 0 to level 3* is our own).

Level 0: First, the simplest of comparing tools is the *comparison of published metrics*. That is, the researcher evaluates the developed tool as described in Figure 1 and compares the metrics to the metrics other researchers have published about their own tools. This does not necessarily imply identical systems or security concepts for the evaluation. Actually, it can—in general—be assumed that there are differences in at least one of both sets. Oftentimes, it is also impossible to validate congruences between these sets, as they need not necessarily be available open-source or as published artefacts. Regarding Figure 2, this keeps all numberings different and only compares metrics.

Level 1a: Second, the next comparison level can be described as restricting the comparison to a comparable set of systems. This is usually done to keep the complexity of these investigated systems comparable or show the tool's applicability to already used systems. This necessitates a publicly available description of these systems in some form of formal manner (potentially an architectural view) that can either be directly employed for the new tool or transformed into a description the new tool can work with. In Figure 2, this relates to setting both *systems* to *System 1*.

Level 1b: Third, following the incentive to keep the testing environment comparable, the next level of comparison can be described as restricting the comparison to a comparable set of security concepts. In this case, the researcher tries to restrict the evaluation to a set of similar security concepts that have been used with the other published tools. This necessitates a publicly available list and potential interpretation of ssecurity concepts for the tools to which the researcher wants his tool to be compared to. In Figure 2, this relates to setting both rules to Rule 1.

Level 2: Fourth, to improve the comparison of *false positives* and *false negatives*, the findings from the oracle can be kept identical, i.e., the new tool's findings would be compared to those from the original tool to produce the metrics. In this case, *systems*, *rules*, *oracle* and *findings* would have number 1. To that end, it is necessary to have the findings from the original tool publicly available. This can also be seen as conceptual replication following [22].

Level 3: Up until now, only metrics have been compared. If, however, the analysed findings of the original tool are available, it is possible also to perform a comparison of the analysed findings of the tool, allowing a much more informative dive into the differences between both tools. This, however, necessitates a publicly available list of the analysed findings from the tool to compare to.

Please note that we have only described potential comparison methods so far and do not comment on their potential biases or the possibility of being carried out.

2.4 Problem statement

After presenting the current usual ways of evaluating automated security flaw detection tools in the literature, it is easy to see that it is highly dependent on the availability of information and compatibility of tools, which evaluation or comparison approach a researcher chooses. However, as we will point out in the next section, these choices can influence the results and, therefore, the assessment of the presented tool by the readers and later, users. Additionally, having no further insights into why different tools perform differently (or perform comparably but for different reasons) decreases the potential for further improvement in research.

For the reliability of research results and, additionally, the development of improved tools (with the overall goal of secure systems at one point), it is necessary to develop and provide **a fair, reproducible, and informative evaluation and comparison scheme of automated approaches**. Only then can the evaluations and comparisons be trusted and interpreted fairly, be cross-checked by different teams of researchers, and yield insightful details for advancing research for improved automated security-flaw detection tools.

The goal of our research, provided in this paper, is, therefore, to develop such a scheme. To that end, we use the following interpretation of the properties mentioned above: A comparison is **fair** if its design does not favour one of the approaches willingly or unwillingly and thus has no approach bias. It is **reproducible** if it allows others to achieve the same results given the same tool, the same security concepts for its detection rules, and the same set of systems. If the comparison provides detailled insights into performance differences, it is **informative**.

To achieve this goal, we focus on the following research questions, which will be addressed in the three following sections:

RQ1: What problems may arise when evaluating and comparing automated approaches?

RQ2: Which aspects (w.r.t. these problems) have the potential to be standardised?

RQ3: How can these aspects be standardised?

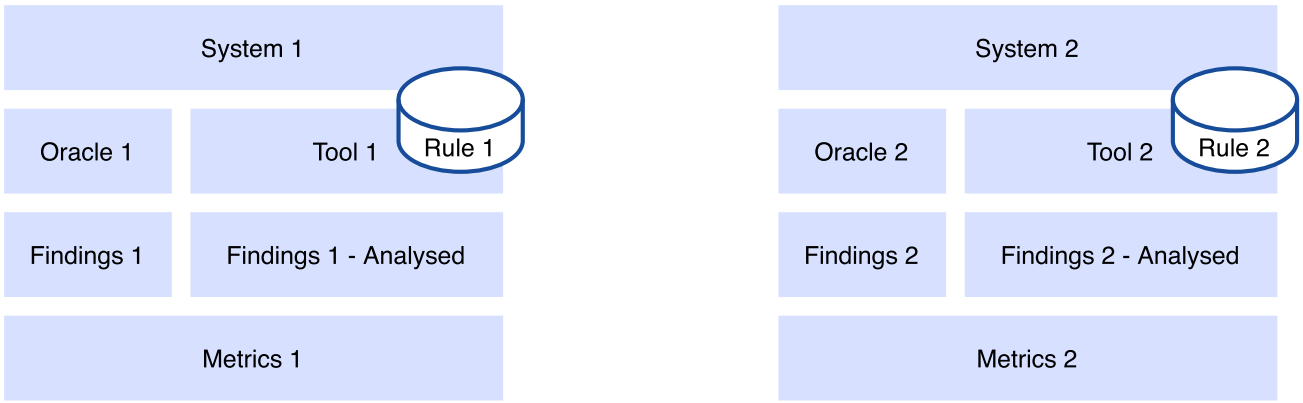

Fig. 2 Artifacts existing during a tool comparison

The three following sections will discuss these research questions one by one. In Section 3, we will use a Case Study from [10] where we approached a Level 2/3 comparison for two automated security flaw detection tools to showcase the issue that may arise when performing these types of comparisons.

3 Potential biases and case study

In this section, we plan to investigate problems that may arise when evaluating and comparing automated security flaw detection tools concerning our defined goals of **fairness**, **reproducibility**, and **informativeness**. We base this investigation on the sketched evaluation process from Section 2 and the comparison levels introduced in Section 2.3. After a more abstract discussion, we use a case study from [10] to illustrate the problems that may arise when doing so. Please note that we discuss the problems in a subset manner, e.g., all problems present in Level 1 are also present in Level 0, although not explicitly stated.

Level 0: It is quite self-explanatory that tools evaluated on different systems, with different rules and varying ground-truths, can not be compared in an informative manner, i.e., yielding insight into the actual reason for different or equal resulting metrics. Regarding fairness, we argue the following: If the sample sizes are medium or small, only a fair statement can be made about the respective setup, i.e., tool A performs better in its given setup than tool B in its given setup. If the sample sizes are high, a statement like: Tool A performs better in classes of its given setup than tool B in classes of its given setup, is possible. This is due to the high sample size, which allows the metrics to be interpreted as predictions of future performance. However, in this case, an estimation error must be considered and set in context with the potential improvement a new tool claims to make. Reproducibility is possible if all artefacts are provided by the respective authors of the investigated tools but can (a) not be ensured by the authors of just one tool and (b) can not be assumed, given a simple Level 0 comparison.

Level 1(a/b) For comparisons of the first level, i.e., identical systems, identical security concepts, or both, one central issue of Level 0 comparisons is solved: Statements need not restrict themselves to the given setup for the tool anymore. A statement like: Tool A performs better than Tool B in the given setup is—in general—possible. However, several problems still might result in non-informative comparison results. First, while the systems and security concepts might be identical, the ground truth is not. When the ground truths (remember that they are manually produced) are only produced by one assessor each, differences in the result outcomes may stem from differences in the tool and the different ground truths. This issue can be softened by using several assessors for the oracle, but it will still be present. Informativeness is hindered by that aspect, as well as the fact that only metrics are compared. Improving reproducibility is implied now, as the systems and the set of security concepts are publicly available. Nevertheless, this need not be sufficient for actually reproducing published results. Another issue regards the assumed equality of systems and rules. While one can restrict the evaluation to the same set of systems and the same set of security concepts, it is a likely assumption that both tools need a varying type of formalisation for both systems and detection rules for the mentioned security concepts. This process may contain interpretation differences, which lead to different results.

Level 2 For level 2, the automated tools are evaluated on the same set of systems and rules, and evaluated with respect to the same ground truth. Informativeness is increased, as the metrics are actually comparable in this instance. Fairness is also increased, as different ground truths can no longer bias the evaluation. However, the evaluation process itself can be biased depending on the handling of similar entries. We will illustrate this in detail in Section 3.3.3. Furthermore,

as the ground truth is provided by humans, and therefore, potentially erroneous, ambiguous, or based on a different understanding of the system modelling or detection principle of the security concepts, it may contain a bias towards the original supplier of the ground truth.

Level 3 For a level 3 comparison, we also require the list of analysed findings to be present. At this point, informativeness is greatly increased. The results of the investigated tools can now be compared in detail, yielding clearer information on where and (potentially) why *false positives* and *false negatives* were made. Due to the presence of the ground truth (see Level 2 requirements), it is also possible to identify different understandings/interpretations of system modellings or detection rules for security concepts (see issue for Level 1). However, non-compatible formats for representing findings may render this effect mute. We will illustrate this in our Case Study (see Section 3.3.5). Fairness is (potentially) increased, as the presence of a list of findings allows the reconstruction of the computation process for the standard metrics and thus, the process can be adapted to not favour one of the approaches. Reproducibility is also highly increased, as almost all necessary information needs to be present to be able to perform a Level 3 comparison. However, some artefacts may be missing, i.e., it can not be safely assumed.

Table 1 summarises all defined levels and their respective conditions and subsumption of the likelihood that a comparison at this level will fulfil the informativeness, fairness and reproducibility properties.

While some above-presented problems might seem obvious (equal setup or different ground truths, for example), others present themselves when actually performing such a comparison (they can then often be found in the *threats-to-validity* Section). Therefore, to illustrate some of the more *hidden* issues when performing a comparison of level 1+, we use our case study from Berger et al. [10][4]. In this case study, the goal was to perform a Level 2 and Level 3 comparison between Tuma et al.'s automated security flaw detection tool (as evaluated in [17]), and Berger et al.'s automated security flaw detection tool [23] with Tuma et al.'s approach as *baseline*, i.e., using their systems (as described in [17]), security concepts [9] and ground truth [17].

The remainder of this section will first introduce both automatic security flaw detection tools, then introduce the base study by Tuma et al. [17] and the basis for their study [9]. Then, we will explain our approach to compare both tools. We will focus on the issues referred to as level 1+ and explain how choices by the researchers can influence the results here. Finally, we will answer our first research question: *What problems may arise when evaluating and comparing automated approaches?*

3.1 Automated security flaw detection tools

Tarandach and Coles report on tools for automated threat modeling [13], such as the open-source software *Threatspec*[5] or *ThreatPlaybook*[6], or the commercial threat-modeling tools *IriusRisk*, *SD Elements*, or *ThreatModeler*. The tools focus on different use cases and, consequently, provide different levels of automation. This subsection introduces two approaches to automated security flaw detection [17, 23], which we compared in our case study published at MODELS'23 [10]. We decided to compare these approaches as they are actively researched, they are available and focus on automatically detecting security flaws. In contrast, other open tools, such as *Threatspec* and *ThreatPlaybook* allow documenting a threat model within or alongside the code but do not implement any automated security flaw detection.

The approaches we focus on are based on dataflow diagrams, which DeMarco first introduced in 1979 as an informal way of capturing dataflows without strictly defined semantics [24]. A starting point for the security-related use of dataflow diagrams is the publication on Threat Modeling by Swiderski and Snyder [2]. In the following, we will describe both approaches in detail and compare them to highlight their differences.

3.1.1 Extended dataflow diagrams

Berger et al. introduced extended dataflow diagrams (EDFD) to overcome the shortcomings of dataflow diagrams when used for automatic security-flaw detection [16]. EDFDs consist of two parts: A schema part and a diagram part. The diagram consists of four concepts: *Element*s, *Data Flow*s, *Trust Area*s, and *Data*. *Elements* are connected by *data flows* and can be contained in *trust areas*. *Data* is assigned to *elements* or *data flows* if the element processes or transports it. These diagram concepts are linked to the schema part by requiring every concept of the diagram to have a type, which are parts of the schema. The schema is comparable to a metamodel that allows to specify hierarchical types and attributes at runtime[7] that are, in turn, used in the diagram part to create an architectural view of a system. Attributes describe security requirements, e.g., some data sensitive, and security concepts, such as encryption mechanisms.

[4] This paper is an extension of the original MODELS'23 publication that we reference here. The replication and comparison performed in this paper are the basis for providing techniques for a fair and systematic approach to comparing security flaw detection rules. For the self-containedness of the publication, we will restate the relevant parts of the respective publication.

[5] Available at https://threatspec.org

[6] Available at https://github.com/we45/ThreatPlaybook

[7] EDFDs come along with a pre-defined schema that can be used and extended.

Table 1 Overview of Evaluation/Comparison Levels, their required elements (please view as additive entries), the respective element of comparison and the resulting properties. Properties marked with (−) cannot be assumed at all, with (+) can be assumed, if all conditions are met. The markings (∼) and (=) show increasing likelihood of the presence of this property

level	*equal* elements required	Elements of comparison	Properties
0	none	metrics	informative (−) fair (∼) reproducible (∼)
1a	systems	metrics	informative (−) fair (∼) reproducible (∼)
1b	rules	metrics	informative (−) fair (∼) reproducible (=)
2	oracle	metrics	informative (∼) fair (=) reproducible (=)
3		findings	informative (=) fair (=) reproducible (=)

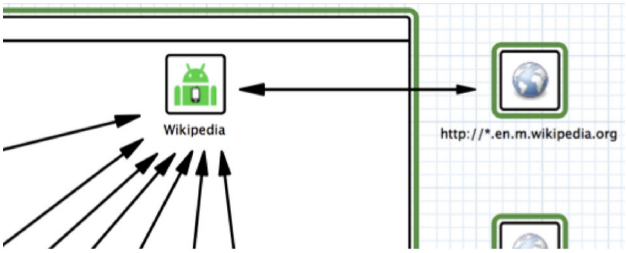

Fig. 3 Excerpt of a EDFD taken from Berger et al. [16]

To illustrate the idea behind attributes, types and the diagram, we will discuss the type *HTTPS connection* from the default schema. A user wants to create a data flow between two elements of an EDFD. Therefore, she has to decide on a dataflow type. As she is modelling a web-based system that communicates to the browser through an HTTPS connection, she decides to use the *HTTPS connection* type, a specialised inter-process communication. The default schema assigns an attribute to this dataflow type, the *TLS Encryption* attribute. A *TLS Encryption* is a specialised *Transport Encryption* attribute that states that the connection is encrypted.

Berger et al. made their threat identification approach, ArchSec, publicly available via their project website[8]. They formalised their extended dataflow diagram using Ecore. Ecore is a metamodel that allows processing models, such as extended dataflow diagrams, using all modelling features based on the Eclipse Modeling Framework [25]. Berger et al. use a model-to-model transformation to map their diagrams to property graphs. A property graph is a typed, attributed and directed graph. At this level, the approach allows the specification of security flaws and mitigation patterns using the graph query language Cypher [26], thus mapping the threat identification process to the subgraph isomorphism problem. Furthermore, ArchSec is freely extensible by creating custom *knowledge bases*—hosting custom security-flaw detection rules and their descriptions.

Figure 3 shows an automatically extracted extended dataflow diagram published by Berger et al. [27], which models an Android App in their interactive EDFD editor. Icons distinguish the different types of *Elements* visualised. The green border visible in the figure represents a trust area. Each arrow represents a *Data Flow* of the system. Instead of using icons, different arrow colours visualise different dataflow types. The data and attributes assigned to the shown nodes and edges are not depicted in their graphical representation but are accessible by an additional view [23].

3.1.2 Security-Enriched DFDs

Tuma et al. introduced security-enriched DFDs (*SecurityDFDs*) [17] to solve the aforementioned identification problem described by Shostack [7]. These dataflow diagrams stay with the original idea of dataflow diagrams and explicitly support the different diagram elements, such as *data stores*, *processes*, *external entities*, and *data flows*. Furthermore, Tuma et al. explicitly model the data processed by the system instead of simple labelling. Additionally, *SecurityDFDs* are enriched with modeling elements to explicitly model security patterns, such as *Secure Pipe*, *Secure Pipe with Client Authentication*, and *Encrypted Data Store*, and attach them to diagram elements.

Tuma et al. specify *SecurityDFDs* using Ecore and use the domain-specific language of the VIATRA model transformation framework [28] to specify security-flaw patterns. These patterns help to identify insecure parts of a *SecurityDFD*. Tuma et al. made a set of *SecurityDFDs* available but not the detection framework itself.

Figure 4 contains an excerpt of a SecurityDFD taken from the previously mentioned publication by Tuma et al. [17]. The figure shows an external entity named *User*, which sends sensitive information (*User ID* and *coordinates*) to the *Process Hazardous Conditions* process. The security pattern *Secure Pipe with Authentication Solution* is applied to the above-mentioned data flow. The security pattern references different parts of the dataflow diagram, stating their roles. Additionally, the figure shows more data flows. Some are protected, and others are not, such as the data flow from the *Process Hazardous Conditions* process to the *Updated Repudiation* process.

[8] https://www.archsec.de

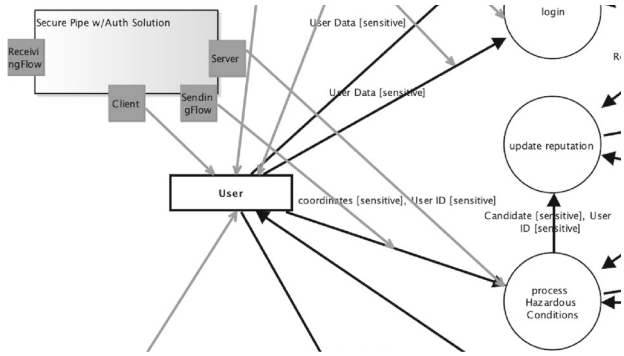

Fig. 4 Excerpt of a SecurityDFD taken from the published artifacts belonging to Tuma et al. [17]

3.1.3 Differences and similarities

Both presented approaches use Ecore to specify their dataflow diagrams. The main difference between both approaches is the flexibility. The approach by Tuma et al. supports a fixed number of diagram concepts, such as *data stores*, *processes*, and *external entities*, whereas the approach by Berger et al. provides an extensible schema that allows the customisation of dataflow diagrams and the introduction of user-defined concepts. All concepts in Tuma et al.'s approach are part of the default schema of Berger et al.'s extended dataflow diagrams. This statement, however, is not possible the other way around, as there is no way of expressing custom types in Tuma et al.'s approach without changing the Ecore model. Similarly, the security concepts supported by Tuma et al.'s approach are limited to those explicitly modelled. In contrast, Berger et al.'s approach allows adding new information on security concepts.

The chosen query languages of both approaches differ but are similar. Cypher and VIATRA allow the description of subgraphs, which are then matched by an engine. The result is a part of the diagram indicating insecure behaviour in both cases.

3.2 Base study

This section describes Tuma et al.'s study [17] (the tool description is part of the last section). We give details about the used ground truth, its generation, and the supplied material. All our information exclusively stems from their publication or openly available supplementary material.

Data Foundation Twenty-six dataflow diagrams build the foundation of Tuma et al.'s study [17], which 13 participants created manually for four software systems. These systems are *DriveSafe*, *BeSocial*, *PhotoFriends*, and *SmartTex*. Then, Tuma et al. randomly assigned these participants to the different systems, so every participant modelled two different systems. This modelling process resulted in a set of 26 DFDs (almost) uniformly distributed over the four systems[9] [17].

Table 2 List of security flaws that are part of the inspection guideline published by Tuma et al. [9]. The flaws printed in bold font are used in their automatic detection study [17]

ID	Name
1	Missing authentication
2	**Authentication bypass**
3	Relying on single factor authentication
4	Insufficient session management
5	Downgrade authentication
6	**Insufficient crypto key management**
7	Missing authorisation
8	Missing access control
9	No re-authentication
10	Unmonitored execution
11	No context when authorising
12	Not revoking authorisation
13	**Insecure data storage**
14	Insufficient credentials management
15	**Insecure data exposure**
16	Use of custom/weak encryption
17	Not validating input/data
18	**Insufficient auditing**
19	Uncontrolled resource consumption

Inspection Guideline In 2019, Tuma et al. published an inspection guideline for manually identifying security design flaws [9]. This guideline contains 19 security design flaws—shown in Table 2—and gives a description and a detection guideline for each. The target audience of the detection guideline is human beings who conduct a security flaw assessment manually. It describes which elements the flaw relates to and which circumstances hint at the presence of the security flaw. Furthermore, since human beings use the guideline, it is written in natural language and contains questions and examples for better applicability.

Listing 1 shows the inspection guideline rule 2, *Authentication Bypass, using an alternate path*. The first part of the guideline rule describes the problematic behaviour. In contrast, the second part describes how to detect instances of that rule. As one can see, the detection guideline consists of questions the assessor has to answer to review the system's security

From the 19 security flaws introduced in the inspection guideline, Tuma et al. focused on the security flaws with IDs 2, 6, 13, 15, and 18 during their automatic detection

[9] For more details on the degree of experience of the participants or differences in their modelling approaches, please see the original publication [17].

This refers to the case where although there is an authentication mechanism in place, it does not cover all possible entry points to the system. This can be due to the fact that there is a remote access point to the system aiming towards support or maintenance, a possible backdoor. Additionally, a system may make a call to invoke functionality of an external application. In such case, the external application can have access to resources and data of the system if not contained properly.

Detection:
Determine the entry points to the system.
For each entry point examine:
Does it go through an authentication point?
What kind of assets are accessible through this path? Are their security objectives still achieved?
Is the system protected against MITM and session hijacking attacks?
Are the communication channels used encrypted?
Does the system invoke functionality from third party applications?
Are these applications subject to proper access control (regarding the resources and data they have access to)?

Listing 1: Inspection Guideline Rule 2: Authentication Bypass using an alternate path

study [17]. These rules tackle different security aspects. Rule 2 relates to authentication issues. Rules 6, 13, and 15 deal with insecure transmission or storage of sensitive information. Lastly, Rule 18 focuses on secure auditing of sensitive information.

Reference – Ground Truth Two security experts independently inspected these 26 dataflow diagrams concerning the abovementioned security flaws. They reported their findings with the following information: *Participant*, *System*, *Rule ID*, *Element Type*, *Element Name*, *Data Flow*, and *Comment*. This report is referred to as *ground truth* and provides the mimicked oracle. It contains 367 entries.

Automatic Detection Tuma et al. defined the five employed rules in VIATRA to automatically identify flaws in the given dataflow diagrams [17]. Their results are of the same form as the ground truth—however, they do not contain any comments stating any details on the concrete finding.

Evaluation They evaluated their approach based on the reported findings using the standard measures of *precision*, *recall*, and *f1-measure* using the following standard formulas:

- *true positive (tp):* Results that are present in the ground truth and the findings, i.e., correctly identified flaws.
- *false positive (fp):* Results that are present in the findings but not in the ground truth, i.e., wrongly identified flaws.
- *false negative (fn):* Results that are present in the ground truth but not in the findings, i.e., overlooked flaws.
- *precision (p):* $p = \frac{tp}{tp+fp}$, the percentage of identified flaws that are correct.
- *recall (r):* $r = \frac{tp}{tp+fn}$, the percentage of present flaws that are identified.
- *f1-score (f1):* $f1 = 2 \cdot \frac{p \cdot r}{p+r}$, a harmonic mean of precision and recall.

Supplied Material They published their study material online, i.e., the dataflow diagrams, the constructed ground truth, and their final findings table, which already includes ground truth and findings. This table of final findings contains 739 entries. However, the supplied materials do not contain the VIATRA patterns used for detection.

3.3 Setup and methodology of comparison case study

Figure 5 shows a more detailed version of Figure 1 in Section 2.3 regarding the depicted steps. However, it uses the *Inspection Guideline* as an instance for the Security Concepts in Figure 1, as they are the main reference line for the Tuma study. The grey area on the left-hand side depicts the evaluation of Tuma et al.'s tool (except the Ecore model marked with a dark blue *star*), as presented in [17]. DFDs, Rule Patterns (following the Inspection Guideline) and Findings are explained in Section 3.2. The resulting metrics are *tp, fp, np*, as well as *precision, recall* and *f1-measure*. The detection process happens via their proposed tool, while the evaluation process is not explicitly explained.

To compare Berger et al.'s tool's performance to that of Tuma et al.'s tool, we (tried to) perform(ed) a Level 2/3 comparison in [10], consisting of five main steps: First, to base the comparison on the same set of systems (see Level 1a), we built an Ecore model for the Tuma et al.'s SecurityDFDs, and used a QVTo script to generate EDFDs that are compliant with the Ecore model of Berger et al.'s approach (all steps marked with a star numbered with the number *1*). Second, to ensure comparable security flaws (see Level 1b), we construct Cypher queries for rules 2, 5, 13, 15, and 18 from Tuma et al.'s Inspection Guideline (this step is marked with a star with the number *2*). Third, we (tried to) replicate(d) Tuma et al.'s evaluation process, i.e., the process of producing metrics from the list of findings and the ground truth. This yield(ed)s the evaluation process marked with a star with the number *3* and the replicated metrics of Tuma et al.'s evaluation (also marked with a star with the number *3*). Fourth, we evaluated Berger et al.'s tool with the constructed EDFDs and rule patterns, resulting in a new list of findings and corresponding metrics (see elements marked with a star with number *4*). Additionally, we compared the replicated and newly produced metrics (see Level 2 comparison). Fifth, we compare the respective lists of findings (from Tuma et al. and the ones produced in Berger et al.) (see Level 3 comparison). These steps are marked with a star with the number *5*.

All steps marked a non-framed star were developed for the publication in [10]. Steps with a framed star were explicitly created for this publication. Elements without stars stem from

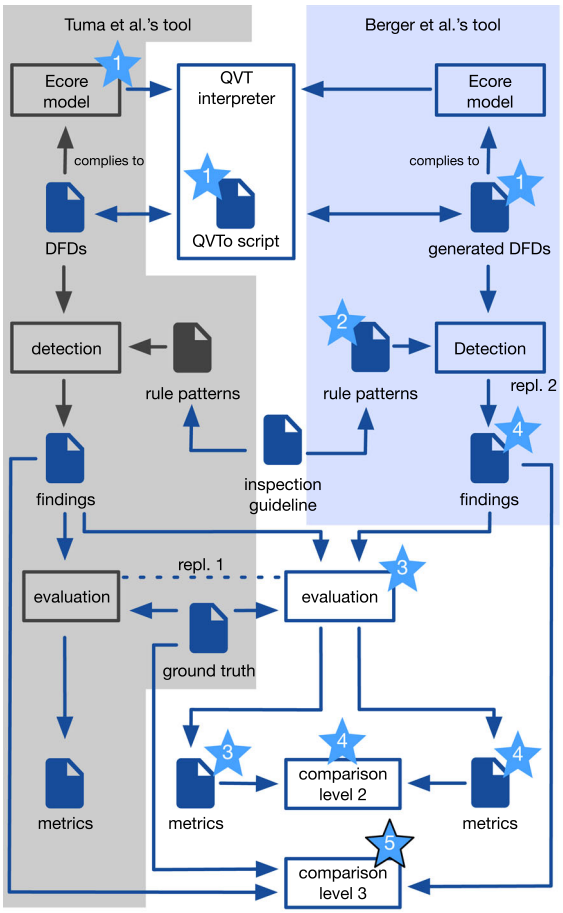

Fig. 5 Evaluation setup

Tuma et al.'s publications [9, 17] or Berger et al.'s tool [16]. We will more closely describe these five steps in the following five subsections.

3.3.1 Transformation of securityDFDs to EDFDs

Description

To transform the dataflow diagrams, we first re-created the Ecore model of Tuma et al.'s diagrams since it is not publicly available. We then implemented two model-to-model transformations with less than 1000 lines, including comments. The first script maps security-enriched DFD elements into extended dataflow diagram elements. It processes the information in the following order: First, it maps all data information, including attributes that indicate the sensitivity of the data. Second, it converts all nodes to the target. Third, the mapping creates all dataflows, connecting the generated nodes in the resulting extended dataflow diagram. Fourth, the script transfers the trust areas from the input to the target model. Lastly, the transformation generates attributes reflecting the security measures found in the security-enriched DFD that serves as the transformation's input. The second script implements the reversed mapping and transforms an extended dataflow diagram to a security-enriched DFD.

Potential Problems There are two issues worth mentioning: First, it needs to be validated that the transformation script results in equivalent models. This, however, is only possible if both models have an equivalent expressive power with respect to the considered system. Second, this explicit step has to be adapted for every newly investigated tool or added systems.

3.3.2 Rule patterns

Description Listing 2 shows an exemplary rule pattern for the approach of Berger et al. for the already shown inspection guideline rule 2 (c.f., Listing 1). First, it searches for a path (named `path1`) from an external entity to a data store. Second, it searches for a second path (called `path2`) between these elements that traverses an authentication point. The query's **`WHERE`** clause ensures that the first matched path does not traverse the authentication point and that the identified paths do not traverse multiple data stores and external entities. Besides this query, the pattern lists (which is not shown here) a reporting template that uses concrete values of the match to allow the user to understand the match description more easily. We then use these rule patterns with Berger et al.'s approach ArchSec on the generated EDFDs to obtain the findings for Berger's approach. **Potential Prob-**

```
MATCH
  path1 = (ext : "Element" {type :
  ↪  subtypeof("External Entity")})
    -[:"Channel" * .. {"IsActive" : true}]->
  (ds : "Element" {type : subtypeof("Data
  ↪  Store")}),
  path2 = (ext)
    -[flow :"Channel" * 1
    ↪  {"Mitigation.IsAuthenticating" :
    ↪  true}]->
  (ap : "Element" {type :
  ↪  subtypeof("Process"),
  ↪  "Mitigation.IsAuthenticationPoint" :
  ↪  true})
    -[:"Channel" * ..]->
  (ds)
WHERE NOT(ap IN nodes(path1))
  AND SINGLE (n IN nodes(path1) WHERE
  ↪  n.subtypeof("Data Store"))
  AND SINGLE (n IN nodes(path1) WHERE
  ↪  n.subtypeof("External Entity"))
  AND SINGLE (n IN nodes(path2) WHERE
  ↪  n.subtypeof("Data Store"))
  AND SINGLE (n IN nodes(path2) WHERE
  ↪  n.subtypeof("External Entity"))
RETURN ext AS ext, ap AS ap, path1 AS path1,
↪  ds AS store, path2 AS path2, flow AS host
```

Listing 2: Cypher pattern for Inspection Rule 2

lems The description of the inspection guideline is in natural language and thus inherently subject to potential misunderstanding. The created rule pattern does not necessarily need to be equivalent to the one created by Tuma et al. using VIATRA. Additionally, it does not need to represent the same understanding of the problem and thus might not reflect the understanding of the experts providing the oracle for the ground truth.

3.3.3 Replication

Description — Reengineering the Evaluation Process As mentioned in Section 3.2, Tuma et al. provided their ground truth and a combined list of findings with the ground truth that served as the basis for the computation of their metrics. However, it is neither straightforward to extract the list of findings (without ground truth) nor the process of combining findings and ground truth to their final results. Nevertheless, it was important to re-engineer this exact process to ensure a truthful comparison. To that end, we first compared entries in the ground truth and the reported combined list. The comparison led to several insights into the actual process: First, identified flaws—that did not belong to a specified element node or dataflow—had been deleted to compute the results. Second, they report and supply data for thirteen DFDs (Participants *A*–*M*). However, the findings include a participant *N*. As no DFD is present in the supplied data, we decided to delete this participant from the findings to allow a fair comparison. Third, we employed minor changes in capitalisation and spelling to allow an automatic evaluation process instead of a manual one (see Table 3 for an overview of the alterations)[10][11]. Finally, we condensed a findings file that only contains findings by deleting all entries stemming from the ground truth (recognisable by existing comments). With both files at hand, we re-engineered a series of *join*-operations to obtain a result file with true positive, false positive and false negative markings, which could then be used for computing the metrics. This replication can be considered an exact replication (in terms of Dennis et al.'s replication manifesto [22]). Using this re-engineered evaluation process, we reconstructed their result metrics and those of Berger et al., which finally allowed a comparison of both approaches.

Description — Validating the Basic Numbers After rebuilding their findings set (see Section 3.3.3), we applied the re-engineered process to the findings set and their ground truth and compared the results to their reported results to ensure the validity of our re-engineered process. Tuma et al.'s reported results, as well as the results from our re-engineered process, are given in Table 4. Even if the numbers are close, there are still some discrepancies. However, they can mainly be explained through the inconsistency with participant *N* (accounting for 25 *true positives*, 30 *false positives*, and six *false negatives*). Factoring this in, the results are valid for the *false positive* and *false negative* categories. The missing 22 *true positives* stem from a copy-paste mistake in the original findings file. This complete validation process has been documented in detail in the supplementary material.

Table 3 Minor alterations for automatically reproducing Tuma et al.'s results

Change	Frequency
Deletion of uncategorised flaws	62
Captialisation	25
Spelling	30
Deletion of DFD 14	25 + 30 + 6

Description — Ensuring Fairness with Different Formats We obtain the results in Table 4, when replicating Tuma et al.'s results and applying the same procedure to Berger et al. At first glance, both approaches perform similarly, with a slightly better performance of Berger et al.'s approach. However, when inspecting the results, we expect the sum of *tp* and *fn* to equal the entries in the ground truth. Generally, we expect the following equations to hold: $tp + fn = \#groundtruth, tp + fp = \#findings$. A quick look at the table shows that not only is $331 + 99 \neq 443 + 91$, but it also does not match the number of entries in the ground truth when filtered as described (and following Tuma et al.'s reported numbers), namely $288 = 367 - 62 - 17$, where 368 is the original number of entries in the ground truth, 62 is the number of uncategorised flaws in the ground truth, and 17 entries in the ground truth refer to participant *N*. Some analyses of the supplied findings explain this phenomenon: a *true positive* is counted when a finding matches the ground truth but also when a ground truth entry matches a finding, i.e., counted more than once.

However, when aiming to address this issue and producing an evaluation process that only counts *true positives* once, we realised that the nature of the comments in the ground truth, as in Berger et al.'s findings, makes this more challenging than initially anticipated.

We will first describe the different types of comments that need to be dealt with. First, there is the ground truth: Some entries have comments, and some do not. Several entries can only be distinguished through their comments (for example, the data element *User Data* has one entry with the comment "*no access logs*" and one entry with the comment "*no backup*" while everything else is identical). Second, there

[10] We produced a corresponding ground truth file as part of the supplementary material.

[11] As the computations for this chapter are rather extensive and full of peculiarities, we wrote an R-Markdown File, which contains all computations with intermediate results that can either be viewed as plain HTML or interpreted with R to make our results replicable. It is also part of the supplementary material.

Table 4 Tuma et al.'s evaluation variant applied to both approaches and the repeated values reported by Tuma et al.

Method	TP	FP	FN	Precision	Recall	F_1 Score
Berger et al.	443	313	91	0.59	0.83	0.69
Tuma et al.	331	270	99	0.55	0.77	0.64
Tuma et al. [rept]	334	300	105	0.53	0.76	0.62

are the findings by Tuma et al. [17] where no comments are present. However, sometimes there are identical entries (whether they were produced by their automated detection process or contained initial comments that were only omitted, we could not reconstruct). Third, we will discuss the findings of our security flaw patterns using Berger et al.'s approach. Here, every entry is supplied with a very extensive comment. They yield additional information, e.g. the attribute the security flaw belongs to, or the subtype the security flaw belongs to. Concludingly, one correctly identified security flaw might occur more than once in our findings due to different comments.

The evaluation process of *tp* is the most intriguing one: The basis of the evaluation process is highly influential on the resulting number of *true positives*. When taking the ground truth as the basis and counting every element of the ground truth that also occurs in the findings (ignoring the comments), one ignores all the—in principle—correct elements in the findings that are more detailed. This will lead to an incorrect second equation as the number of *tp*s is too small. However, when taking the findings as the basis and counting every element of the findings that also occurs in the ground truth, the first equation will not hold, as the *true positives* are underestimated when there are several identical entries in the ground truth with different comments.

The central idea of the above paragraph also translates to counting *false negatives* and *false positives*. When one security flaw has been incorrectly identified (*false positive*) but occurs more than once through different comments, it overestimates the number of *false positives*. This holds similarly for *false negatives*. One solution, in this case, may be to run a `distinct` on all properties of the findings but the comment.

We derived five different combinations of the above-mentioned situations: *Findings* (take findings as the basis for the computation of *true positives*), *Findings Distinct* (findings as a basis, run distinct on *false positive* and *false negative*, *Benchmark* (ground truth as a basis), *Benchmark Distinct* (ground truth as the basis, run distinct) and *Distinct* (run distinct in the beginning to avoid the above-mentioned problem altogether). The results are presented in Table 5. It shows the results of the defined evaluation process variants for the findings of Tuma et al. and Berger et al., as well as Berger et al., when the security flaws that are mitigated are counted as flaws nonetheless.

The table verifies our assumptions. The *Findings* variant fulfils the second equation, as the findings list is the basis of the computation works, but it does not fulfil the first equation. The *Benchmark* variant fulfils the first equation but violates the second one. The *Benchmark* variant is more favourable for the scheme of Tuma et al. as its dependency on comments is not as high, whereas the *Findings* variant is more favourable for Berger et al.'s detection scheme. One can see the massive effect the *distinct* operation has on Berger et al.'s *false positives* (from 313 to 188 distinct entries), which strengthens our assumption that the very detailed presentation of security flaws and comments produces an unfairly high number of *false positives*.

However, we use the clean *Distinct* variant to allow a fair evaluation. This eliminates any advantages or disadvantages regarding the base data set, and yields results that fulfil both equations.

Potential Problems There are several problems worth mentioning here: First, note that at this point, we have a level 2+ comparison (set of systems and security concepts is assumed equal, and the computation of metrics is based on the same ground truth). However, the ground truth contains detected flaws that are not categorised into one of the five investigated security concepts. It is unlikely that an automated tool is capable of detecting these flaws if it is only provided the detection rules for these investigated security concepts. Second, the computation process of *tp, fp, fn*, and thus the metrics *precision, recall, and f1-measure* is not as trivial as might be expected. Thus, it requires re-engineering if no reproducible process is supplied with the artifacts. Third, the difference in the findings format together significantly influences the outcome of the aforementioned metrics. As the comments can not be efficiently compared (at first glance), their deletion leads to many non-distinct results. Deciding on a *basis* may greatly influence the results. It is, therefore, important to be aware of this issue when comparing metrics of automated tools (even in a level 2+ comparison) that have different formats of findings (or different levels of detail).

3.3.4 Evaluation and level 2 comparison

In this subsection, we describe the results of our comparison based on metrics. We assume, however, that a differentiation of systems and detection rules is present (but not the entire findings list).

Table 5 Results of different evaluation variants with different result sets

Variant	Method	TP	FP	FN	Precision	Recall
Findings	Berger et al.	246	313	91	0.44	0.73
	Berger et al. without mitigation	248	405	90	0.38	0.74
	Tuma et al.	142	270	99	0.35	0.59
Findings Distinct	Berger et al.	246	188	68	0.57	0.78
	Berger et al. without mitigation	248	254	67	0.49	0.79
	Tuma et al.	142	266	74	0.35	0.66
Benchmark	Berger et al.	197	313	91	0.39	0.68
	Berger et al. without mitigation	198	405	90	0.33	0.69
	Tuma et al.	189	270	99	0.41	0.66
Benchmark Distinct	Berger et al.	197	188	68	0.51	0.74
	Berger et al. without mitigation	198	254	67	0.44	0.75
	Tuma et al.	189	266	74	0.42	0.72
Distinct	Berger et al.	135	188	68	0.42	0.67
	Berger et al. without mitigation	136	254	67	0.35	0.67
	Tuma et al.	129	266	74	0.33	0.64

Description — Analysing the Influence of Setup Before diving into the results, we first analyse the variance of different influencing factors. We do this to ensure the later grouping of data is appropriate. In this paragraph, we analyse the influence of the factors *Participant* (remember that in the base study, the system views were created by 13 participants), *System*, and *Security Flaw* on the metrics. We use `min` and `max` values on the precision and recall measures to describe the variance due to two reasons: First, when computing a (non-weighted) variance, we assume that every group is identical in some sense. However, due to the varying system sizes and different types of Security Flaws, it is at least questionable to use the variance. Second, we use precision and recall as they are relative values. Some systems may have a small ground truth, while others have many flaws. An analysis of the systems has already been done by Tuma et al. [17]. We aim to analyse the influence of the factors on the resulting measures.

Table 6 shows the computed results. First, the precision and recall values have a very large span for the influence factor *Participant* for both systems, although it is higher for Tuma et al.'s system. Second, for *systems*, it is very narrow: In all cases, the span is about ten percentage points. Third, when considering the influence factor *security flaws*, the span is large for all three variations but not as high as for the participants.

Usually, a small span leads to the assumption that grouping with regard to that influence factor is statistically safe. Following that, it is safe to group over *Systems*, but not over *Participants* or *Security Flaws*. However, in this particular case, the factor *System* should be highly dependent on the factor *Participant*, as the participants designed the DFDs for the different systems, and each of them only designed two, not all four DFDs. However, the grouping of six to seven participants on one system seems to have had a balancing influence such that there is little variation concerning the *System*, although the underlying factors vary greatly.

The strong variation for the factor *Security Flaw* (also Rule ID) already hints that the type of flaw influences the quality of automatic detection. This deviation should, therefore, definitely be subject to investigation.

Concludingly, we can (but need not) safely group over systems or supply data without differentiation for systems, but not over security flaws.

Description — Results of Comparing Metrics Table 7 shows the results of Tuma et al. and Berger et al. (with our developed rule patterns and mitigations) differentiated for both Security Flaw and System, following the *Distinct* variant introduced in Section 3.3.3.

We note several aspects: (1) Flaw 15 is detected the most, and this holds for both detection schemes (2) The results of both detection schemes are comparable with a slight advantage for Berger et al.'s detection scheme (see also Table 5). (3) Security Flaw 2 achieves bad results, which is also true for both systems. (4) the worse precision of Tuma et al. seems to stem from Security Flaws 2, 6, and 15. Figure 6 shows the results of Berger et al. [16] compared to those of Tuma et al. [17]. Every data point visualises precision (position on the x-axis) and recall (position on the y-axis) for a given system (no visual differentiation), a security flaw (shape of a point) and the detection scheme (dark blue for Berger, light blue for Tuma). As we have four different systems, five analysed security flaws, and two detection schemes, there are 40 data points. Some, however, overlap, e.g. the results for Security Flaw 2 are shown at the same spot and therefore look like one. It is important to note that although the data set contains 26 dataflow diagrams, for this depiction, this does not result

Table 6 Variance analysis for participant, system, and security flaw

Influence	Method	Precision		Recall	
		min	max	min	max
Participant	Berger et al.	0.18	0.56	0.15	1.00
	Berger et al. without mitigation	0.18	0.48	0.15	1.00
	Tuma et al.	0.08	1.00	0.08	1.00
System	Berger et al.	0.37	0.48	0.60	0.74
	Berger et al. without mitigation	0.31	0.40	0.60	0.74
	Tuma et al.	0.28	0.39	0.60	0.70
Security Flaw	Berger et al.	0.26	0.62	0.29	0.82
	Berger et al. without mitigation	0.26	0.62	0.29	0.82
	Tuma et al.	0.15	0.67	0.22	0.82

Table 7 Evaluation results for Tuma et al. (T) and Berger et al. (B) following the *Distinct* variant. The table shows the *true positives*, *false positives*, *false negatives*, *precision* (P) and *recall* (R) for the four different systems.

Flaw	M	BeSocial					DriveSafe					PhotoFriends					SmartTex				
		TP	FP	FN	P	R	TP	FP	FN	P	R	TP	FP	FN	P	R	TP	FP	FN	P	R
2	B	2	8	6	0.20	0.25	1	7	6	0.13	0.14	2	7	4	0.22	0.33	5	6	8	0.45	0.38
	T	2	15	6	0.12	0.25	1	14	6	0.07	0.14	3	10	3	0.23	0.50	6	14	7	0.30	0.46
6	B	3	4	0	0.43	1.00	2	3	0	0.40	1.00	4	6	2	0.40	0.67	5	4	1	0.56	0.83
	T	3	6	0	0.33	1.00	2	6	0	0.25	1.00	4	10	2	0.29	0.67	5	13	1	0.28	0.83
13	B	8	6	2	0.57	0.80	5	4	3	0.56	0.63	5	5	6	0.50	0.45	8	1	3	0.89	0.72
	T	8	4	2	0.67	0.80	5	4	3	0.56	0.63	5	4	6	0.56	0.45	8	1	3	0.89	0.72
15	B	19	31	2	0.38	0.90	20	14	3	0.59	0.87	22	22	12	0.50	0.65	11	31	5	0.26	0.69
	T	19	40	2	0.32	0.90	21	27	2	0.44	0.91	22	30	12	0.42	0.65	11	46	5	0.19	0.69
18	B	3	11	2	0.21	0.60	3	6	2	0.33	0.60	3	7	0	0.30	1.00	4	5	1	0.44	0.80
	T	1	6	4	0.14	0.20	0	8	5	0.00	0.00	2	3	1	0.40	0.67	1	5	4	0.17	0.20
Σ	B	35	60	12	0.37	0.74	31	34	14	0.48	0.67	36	47	24	0.43	0.60	33	47	18	0.41	0.65
	T	33	71	14	0.32	0.20	29	59	16	0.33	0.64	36	57	24	0.39	0.60	31	79	20	0.28	0.61

in 26 data points for every security flaw and detection scheme (then, we should have 104 data points). However, we group the dataflow diagrams by the system they represent and show the overall result for this system. For example, have a look at the data point at (0.89|0.73): Its shape is a diamond, therefore, it represents the results for Security Flaw 13. This data point represents one of the four analysed systems It is dark blue, which means it represents the results of the detection scheme of Berger et al. We abstracted the information about the system, due to its non-significant influence, especially when compared to the influence of the considered Security Flaw.

To give the reader an impression in terms of f-measure, we also show f-measure curves, i.e. the solid, dotted, and dashed lines in the figure represent isolines for an f-measure of 0.25, 0.5 and 0.75, respectively. A data point that lies to the upper right of the dashed line and to the lower left of the dotted line, for example, has an f-measure between 0.5 and 0.75. Results below the dashed line are quite bad, whereas results above the dotted line are quite good.

We see that Security Flaw 2 (authentication bypass) produces bad results for both detection schemes, whereas Security Flaw 6 (insufficient crypto key management) produces acceptable results with a good recall. Security Flaw 18 (insufficient auditing) has rather unconvincing results for Tuma et al.'s scheme. Both systems' recall is better than the precision. There is only one entry with a good f1-score of above 0.75.

Description — Influence of Berger et al.'s Mitigation Patterns Berger et al.'s approach supports mitigation patterns, which allows for retracting identified security flaws when security measures are in place. For example, a secure channel can mitigate Flaw 15, insecure data exposure. For the results shown above, we only considered flaws that we did not identify as mitigated, i.e. actual flaws (in terms of this detection). Berger et al.'s mitigation feature identified 67 security flaws to be mitigated. The results in Table 5 show that the mitigation approach accounts for $66 = 254 - 188$ fewer *fp*s and one additional *fn* (together with one missing *tp*).

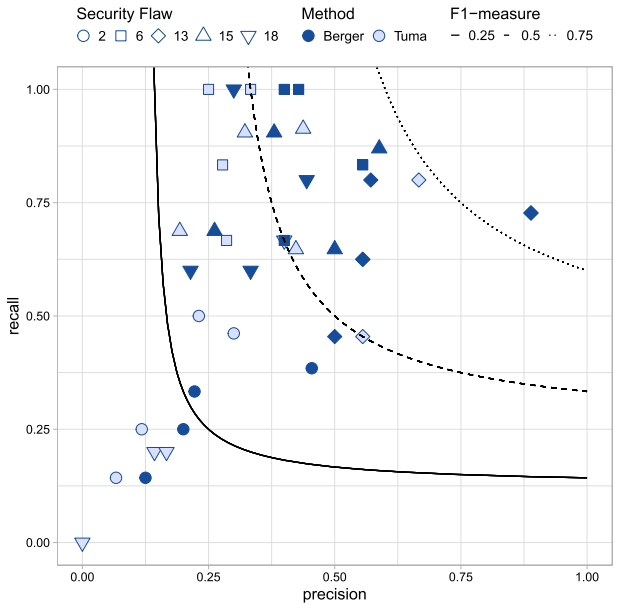

Fig. 6 Comparison of Precision and Recall for both detection schemes grouped by System and Security Flaw

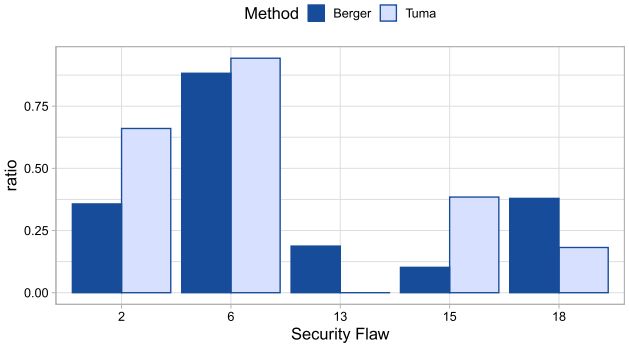

Fig. 7 Ratio of method-specific false positives to all false positives found by that method.

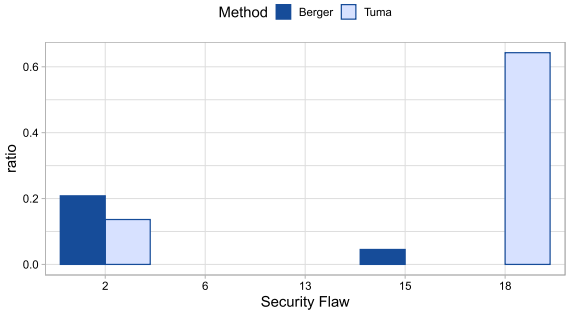

Fig. 8 Ratio of method-specific false negatives to all false negatives found by that method

Cross-referencing these 66 flaws reveals that these are also part of Tuma et al.'s *fp*s.

The numbers indicate that 66 mitigations were correct, i.e. significantly reduced the number of *fp*s of Berger et al.'s approach. On the contrary, one security flaw was incorrectly identified as mitigated—this can be seen in the variation between *tp* and *fn*. Additionally, the fact that Tuma et al.'s *fp*s cross with Berger et al.'s mitigated security flaws implies that the mitigation feature is actually an additional improvement and not already part of Tuma et al.'s model.

Description — Additional Comparisons based on Metrics We additionally investigated the nature of the received *false positives* to potentially identify common overdetections of both approaches. To that end, we compared Tuma's *false positives* to Berger et al.'s. It turns out that the majority of both *fp*s is detected by the other one as well. Figure 7 shows the ratio of detection-procedure-specific *false positives*, i.e., the ones found by our rules but not by Tuma and vice versa, to all detected *fp*s by that method.

Most values are below 50%, i.e. less than half of the *false positives* are specific to the method. However, Security Flaw 6 is an exception, with values above 75%. We choose this form of analysis because the higher the number of common false positives instead of detection-procedure-specific ones, the more likely it is that the issues lie with the automation as such, e.g., with a hard-to-automate or hard-to-model security flaw. The high values for Security Flaw 6 could—for example—point to a very different understanding of the modelling of Security Flaw 6. A comparably smaller value compared to the other approach, however, points to the superiority of this approach for that given flaw. Flaws 15 and 18 are good examples here—as for one, Tuma et al.'s approach is superior, whereas, for the other one, Berger et al.'s approach performs better.

There is an interesting difference in the results for *false positives* when analysing *false negatives*. For *false positives*, while there was substantial overlap between the different detection schemes, each detection scheme still made *its own mistakes*. The picture changes almost completely for *false negatives*: We only found detection-scheme specific, i.e. present for one detection scheme but not the other, *false negatives* for the Security Flaws 2, 15 (only for Berger et al.), and 18 (only for Tuma et al.). For Security Flaws 6 and 13, both approaches agreed on all *false negatives*. For Security Flaw 2 and 15, the ratio is around 20% or smaller. Only the value for Security Flaw 18 is over 60%.

If all *false negatives* are identical, there are different possible interpretations. First, the reason might be automation. Automated approaches may be less likely to find the missing cases. This does not mean that Flaws 6 and 13 are hard to identify, but that this type of flaw seems to either be found by both approaches or none. Second, the used dataflow diagrams are not expressive enough and thus do not contain sufficient information to answer the necessary checks correctly.

Potential Problems: First, a standard problem at this level is the correctness of the statistical evaluation. This especially holds for summarising result over heterogeneous groups, as potentially happens when summarising over all *Security Flaws* or *Systems*. Systems may have different size or difficulties, Security Flaws might be harder or easier to detect and occur more or less often in specific systems. It is therefore important to first cross-check whether a cumulation over different setups is sensible before doing so. Second, while a small form of analysis is possible when looking only at the metrics (see the analysis of false positives and false negatives), one lacks the clear information and more specific detail of comparing findings instead of only metrics. Especially for security flaws with numerous different aspects to consider, the lost detail may decrease the understanding of advantages or disadvantages an automated tool might have.

3.3.5 Level 3 Comparison

Please note that these results do not belong to the original case study anymore and are part of the addition for this specific publication.

Description — Unifying Comments In this paragraph, we describe possible results from a level 3 - comparison, i.e., a comparison based on findings. Besides a frequency analysis of different elements of the findings, the most intriguing one is actually the evaluation of the comments (please remember that the different handling of comments in the findings was the origin of some potential fairness issues in the process of computing metrics). Unfortunately, Tuma et al.'s tool does not provide findings. Nevertheless, their supplied ground truth does and Berger et al.'s tool does as well. Therefore, we will focus on the findings resulting from Berger et al.'s tool and compare them to the comments in the ground truth, as the process would be similar if Tuma et al.'s tool had comments as part of their findings.

As the comments are natural language comments, we need to map them to comparable categories to use them in an automated check. To that end, we use the inspection guideline from [9] and try to derive a mapping from the comments to the listed questions. We then add a column to both data sets, referring to the comment category, thus allowing a comparison.

Producing results in terms of the already introduced metrics—*precision*, *recall*, and *f-measure* (see Section 3.3.3)—is mostly straightforward. However, we again face the decision of which evaluation structure to follow: benchmark or findings first, and whether to ensure uniqueness or not. Since our focus is on the deeper analysis of the structure of the identification errors, we use the *findings* and *findings-distinct* variants, although in this case, *findings-distinct* is identical to the most defensive *distinct* variant. Additionally, we need to make a decision regarding identified flaws without comments (marked by *N/A*). These only occur in the ground truth since the results from Berger et al.'s approach always have a comment. Given a potential match between ground truth and Berger et al.'s results, an *N/A* comment on the ground truth side can either be interpreted as a mismatch (wrong category) or as a match (correct category). There are arguments for both variants, so we compute both. Respective metrics are computed as defined in Section 3.2.

This section describes the results of our analysis of the category-refined results. We first focus on a description and frequency-based analysis of our choice of comment categories. Second, we describe the effects of the category-refined analysis on the standard metrics of *precision* and *recall*. Third, we analyse the nature of these results by focusing on the categories and their specific results. At first glance, the comments in the given ground truth seem arbitrary in their structure. For example, for Security Flaw 2 (*Authentication Bypass*), there are entries such as *NoAuth*, *Does not go through Auth Point*, or *Wrong Auth*. However, it is straightforward to map them to a common category: **No Authentication**. For Security Flaw 6 and others, however, comments are not quite as easily mapped, and additionally, we could not directly map them to the comments in Berger et al.'s approach. Therefore, we referred to the inspection guideline. Here, every described security flaw has questions or checks attached that are supposed to guide the reader in identifying the respective security flaw. As the ground truth was created based on this inspection guideline (as well as the Cypher queries used for Berger et al.'s approach), we derived categories based on the ground truth comments and the given checks in the inspection guideline. The description of Security Flaw 6 is shown in Figure 3. Exemplary comments from the ground truth are as follows: *no backup of storage*, *insecure channel*, *store not protected*, *Keys not distributed securely*, *no logs of key access* and others. Together with the description, we derived the following categories: **No Logging**, **No Key Handling**, **No Backup**, **Insecure Distribution**, and **Insecure Storage**. Comparing these categories to Berger et al.'s comments, we see that his comments on the key handling issues (creation plan, destruction plan, replacement plan) are more detailed. Therefore, we substitute the category **No Key Handling** with **No Creation**, **No Destruction**, **No Replacement**.

As some comments in the ground truth and many comments in the reports of Berger et al. contain additional information in the comment, e.g., the special data that is sent over an insecure channel, we keep this in an extra column *Attribute* when mapping the comments to the respective categories. Table 8 shows the frequency of occurrences of the defined categories. There were more categories defined; however, none of the comments matched them, so they do not occur in the table. An empty category name refers to empty comments (the aforementioned *N/A*). The left column (dis-

Table 8 Frequencies of occurrences of identified security flaws assigned to the given categories in the ground truth

Flaw	Category	Distinct	
		no	yes
2	*None*	22	22
2	No Authentication	19	19
6	Insecure Distribution	11	11
6	Insecure Storage	6	4
6	No Backup	7	5
6	No Destruction	2	2
6	No Logging	8	6
6	No Replacement	2	2
13	Insecure Storage	1	1
13	No Access Logging	21	20
13	No Backup	43	39
13	No Encryption	5	5
15	*None*	60	60
15	Insecure Channel	50	50
18	Insecure Storage	1	1
18	Insufficient Information	1	1
18	No Access Logging	6	5
18	No Attempt Logging	2	2
18	No Backup	10	10

Table 9 Frequencies of occurrences of identified security flaws assigned to the given categories in the results of Berger et al.'s approach-label

Flaw	Category	Distinct	
		no	yes
2	No Authentication	38	38
6	Insecure Distribution	9	9
6	Insecure Storage	6	4
6	No Creation	16	16
6	No Destruction	16	16
6	No Logging	9	7
6	No Replacement	16	16
13	No Access Logging	84	42
13	No Backup	84	42
15	Insecure Channel	183	170
18	No Access Logging	85	42

tinct - no) shows the frequencies when identical flaws with different attributes are counted multiple times, and the right column (distinct - yes) shows the frequencies when ignoring the attributes. Table 9 shows the frequency of occurrences for the results of Berger et al.'s approach (left and right columns have the same interpretation as in Table 8).

We notice several aspects with the frequency distribution: (1) Different Security Flaws have a highly varying number of categories, e.g., Flaw 18 and Flaw 6 have five or more categories, while Flaw 2 and 15 have only one category (disregarding the empty comments). (2) The *distinct*-effect is marginal in the ground truth (12 flaws), while it is noticeable in Flaws 13–18 for results from Berger et al.'s approach. It even halves the results for flows 13 and 18. (3) The frequency distribution varies for both result sets, e.g., for Flaw 6, Berger et al.'s approach identifies 16 flaws each for the categories **No Creation**, **No Destruction**, and **No Replacement**, while the ground truth only lists four in total for all three categories. (4) Several categories are only listed for the ground truth and do not occur for Berger et al.'s approach (cf. Flaw 13 and 18).

Description — results for comment-based analysis After defining and mapping categories, we could perform the analysis for *precision*, *recall*, and *f1-measure*. We had four possibilities for counting data occurrences (*with N/A - distinct*, *with N/A - non distinct*, *without N/A - distinct*, *without N/A - non distinct*). Figures 9a and 9b show the results of these analyses compared to the *uncategorised* results from Berger et al.'s approach. The interpretation of both figures is analogous to the one in Section 3.3.4. The left figure shows the results for the *distinct* variants (black stands for *N/A* matches, dark blue for *N/A* does not match), while the right one shows the same for the *non-distinct* variants.

The original (uncategorised) results of Berger et al.'s approach are comparably better than the categorised variants (irrelevant of the *N/A*-match choice) in both figures. Additionally, we can see that although there are differences between the *N/A-matches* and the *N/A - does not match* variants, showing better results for matching *N/A*s, these are only for a few flaws. The results for the *non-distinct* variants are slightly better than the ones for the *distinct*. The observation from the last subsection is repeated in both figures: While producing bad results absolutely speaking, Flaws 2 is hardly influenced by the categorisation, while Flaws 6–15 show high differences to the original computation.

As these results are surprising (the approach identifies the correct flaw, element, and dataflow but then misses the correct category), we also turned to a more thorough investigation of these results. Table 10, therefore, shows the results from the comparison. However, they are differentiated by the introduced categories. The column with header *C* contains the *true positives*, *false positives*, and *false negatives* for the given categories. When categories are not given (empty comments), they are listed as *None*. The left side shows the results for the *distinct* variant, and the right side shows the results for the attribute-aware, i.e. *non-distinct* variant.

Flaw 6 shows a distinctly varying behaviour for its categories. Almost perfect identifications, e.g., for **Insecure Distribution** or **Insecure Storage**, are countered by *true positive* counts of 0, e.g., **No Creation**, **No Destruction**, and **No Replacement**. Flaw 13 also shows different behaviour

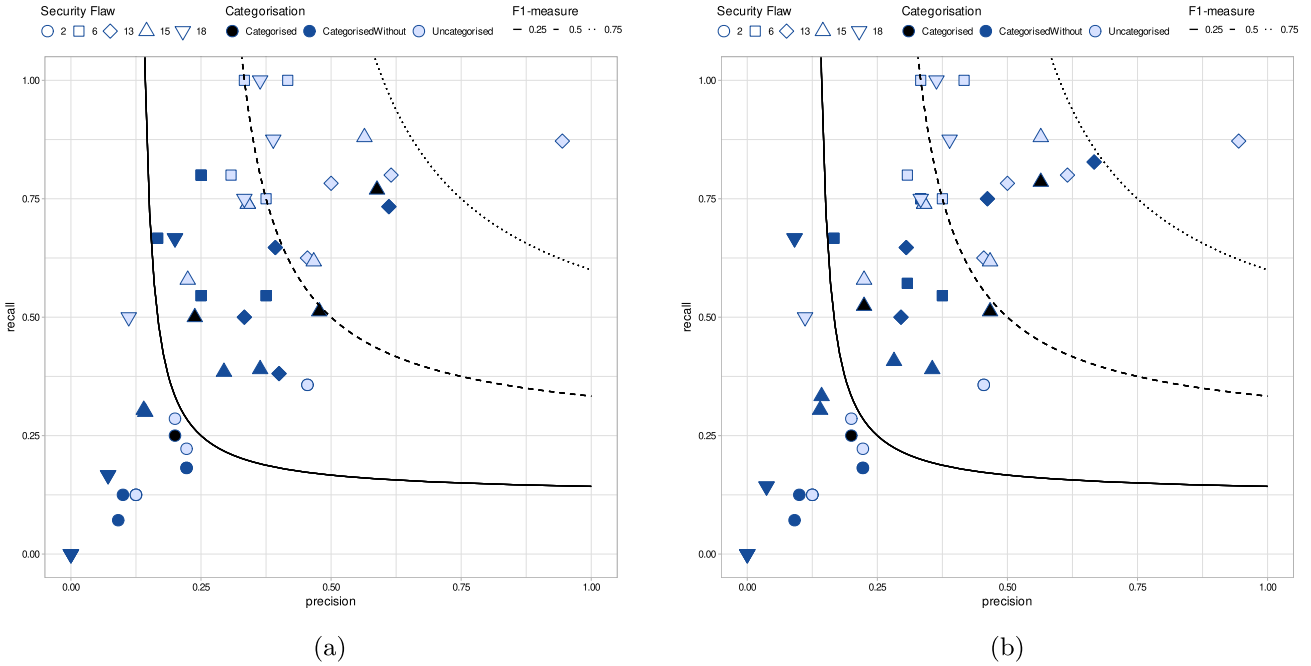

Fig. 9 **(a)** Precision and Recall for the categorised comparison, the categorised comparision without considering *N/A* as matches, and the uncategorised results (compare variant *finding*) with consideration of attribute distinction (that is, the distinct variant). **(b)** Precision and Recall for the categorised comparison, the categorised comparision without considering *N/A* as matches, and the uncategorised results (compare variant *finding*) with no consideration of attribute distinction (that is, the non-distinct variant)

concerning the different categories. Again, some differences between the *distinct* and the *non-distinct* variants can be seen. A prominent example is Flaw 13 with its categories **No Access Logging** and **No Backup**. These almost see a doubling between the *distinct* and the *non-distinct* variant, which is in congruence with our findings from Table 9. Furthermore, **No Access Logging** from Flaw 18 shows almost a doubled value for false positives.

To identify which of the identification errors (*false positive*, *false negative*) could be traced back to the inclusion of categories, we also denote the number of identification errors that could also be found with the non-categorised method (i.e. the original one from Section 3.3.4). These columns are marked with *P*. It shows that many identification errors actually stem from the original evaluation and are not added through the categorisation. For example, Flaws 2 and 15 almost show the same number of identification errors, while Flaws 6 and 18 show higher differences. Here, the categorisation-based analysis is responsible for a greater portion of the identification errors.

Potential Problems The obvious first problem that prohibits an informative Level 3 - comparison is actually present in our Case Study: Unequal finding formats for the investigated tools. In this case, one tool did not provide comments, which might have provided a better analysis of the present errors both tools make. The second problem is the variable nature of the comments. The ground truth provides them with natural language. Berger et al.'s tool (due to its automated nature) provides them in a structured manner but does not necessarily address the same issues as the ground truth comments. This leads to the third problem: A potentially different understanding or differently structured understanding of the inspection guideline that is the basis for the ground truth as well as the detection rules for the automated tools. The inspection guideline is written in natural language, which may lead to different usages. High values of *false negatives* can, for example, stem from a bad performance of the automated tool but also from a different understanding of the security concept/security flaw descriptions in the inspection guideline.

3.4 Emerging issues of comparison study

The conducted Case Study (Sections 3.3.1– 3.3.4) and its extension (see Section 3.3.5) have illustrated some of the mentioned problems in Section 3. We have listed the occurring problems after each subsection describing the conducted Case Study. However, as they sometimes occur more than once, we will quickly restate them here in a sorted manner for a better overview and to answer our first research question.

Problem Class 1 — Ensuring Equal Systems Unfortunately, Level 1a is not as easily achieved as might be thought.

Table 10 True Positives, False Positives, and False Negatives for the categorised analysis differentiated by categories of security flaw

Flaw	Category	Distinct						Non Distinct					
		TP		FP		FN		TP		FP		FN	
		C	P	C	P	C	P	C	P	C	P	C	P
2	*None*	0	0	0	0	17	14	0	0	0	0	17	14
2	No Authentication	10	10	28	28	14	14	10	10	28	28	14	14
2	Σ	10	10	28	28	31	28	10	10	28	28	31	28
6	Insecure Distribution	9	9	0	0	2	2	9	9	0	0	1	2
6	Insecure Storage	4	4	0	0	0	0	6	6	0	0	0	0
6	No Backup	0	0	0	0	5	1	0	0	0	0	7	1
6	No Creation	0	0	16	15	0	0	0	0	16	15	0	0
6	No Destruction	0	0	16	15	2	0	0	0	16	15	2	0
6	No Logging	5	5	2	2	1	1	7	7	2	2	1	1
6	No Replacement	0	0	16	15	2	0	0	0	16	15	2	0
6	Σ	18	18	50	47	12	4	22	22	50	47	13	4
13	Insecure Storage	0	0	0	0	1	0	0	0	0	0	1	0
13	No Access Logging	12	12	30	17	8	8	21	21	63	32	9	9
13	No Backup	24	24	18	17	15	15	51	51	33	32	17	17
13	No Encryption	0	0	0	0	5	4	0	0	0	0	5	4
13	Σ	36	36	48	34	29	27	72	72	99	64	32	30
15	*None*	0	0	0	0	31	19	0	0	0	0	31	19
15	Insecure Channel	68	68	102	102	11	11	71	71	112	112	11	11
1	Σ	68	68	102	102	42	30	71	71	112	112	42	30
18	Insecure Storage	0	0	0	0	1	0	0	0	0	0	1	0
18	Insufficient Information	0	0	0	0	1	0	0	0	0	0	1	0
18	No Access Logging	3	3	39	29	2	2	3	3	82	61	3	3
18	No Attempt Logging	0	0	0	0	2	0	0	0	0	0	2	0
18	No Backup	0	0	0	0	10	3	0	0	0	0	10	3
18	Σ	3	3	39	29	16	5	3	3	82	61	17	6

While the goal may be to investigate equal systems, most automated security flaw detection tools are based on their formal view (description) of a system. Even if the system is provided in one of these formal views, it is not guaranteed that the respective representation of the system in another formal view is equivalent or (at least) equivalent w.r.t. the investigated security concepts. It may not hold that these different views can express all the given information. Subproblems in this class are the development of a valid transformation, the generation of a generally accepted validation approach, and present semantics for the systems provided (to ensure correct understanding). These problems mostly tackle the informativeness (and fairness) aspect (how can two tools be fairly compared if it is not clear which aspect is responsible for differences in the results). Additionally, reproducibility is an issue if the transformation (and validation) are done manually and not via an accessible script.

Problem Class 2 — Ensuring Equivalent Detection Rules Level 1b is also not without its challenges. The inspection guideline published by Tuma et al. [9] is an enormous first step for consolidating security concepts (or the absence of those) and means of detecting them. Nevertheless, they face two issues: The first is identical to Level 1a and ensures equivalent formal views. Most automated security detection tools will use their specific means of describing the absence of security concepts. It cannot be assumed a priori that two formal detection rules of different types actually present the same detection rule. Secondly, as this description is in natural language, it is subject to misunderstanding and may result in different results. Furthermore, the formalisation of rules may influence the degree of details of the findings. Subproblems in this class are developing a valid transformation, generating a generally accepted validation approach, and presenting semantics for the systems provided (to ensure correct understanding). These problems mostly tackle informativeness and fairness. Reproducibility is an issue, especially for the creation of ground truth (why do the experts consider this an instance of a security flaw?), but also for generating detection rules for a specific tool from the inspection guideline.

Problem Class 3 — Ensuring Equivalent Findings Format A problem that occurs at several steps during the evaluation process is the equivalence of the format of the

resulting findings. This holds for the ground truth and the investigated tools' findings. Differences in this format can lead to fairness biases, as has been shown in Section 3.3.3, if not handled accordingly. Additionally, it prohibits deeper analysis of the functionality and performance of the respective tools if the findings can not be easily compared (or only if some information is ignored). Different forms of comments, findings that are not attributed to a specific security flaw or non-present findings at all are clear drawbacks to an informative and fair evaluation. Reproducibility is usually given, if the respective tool is open source, and the evaluation process (given findings, and ground truth) is published as well.

Problem Class 4 — Ensuring Correct Statistical Evaluation An evaluation may fulfil all requirements for Level 2 or even Level 3 comparisons if the statistical evaluation is not done carefully. The results may be unintentionally biased or invalid at worst. Over time, the only *remembered* aspects of an evaluation are provided metrics or short and informative (quantified) results. It is, therefore, very essential to ensure their unbiasedness and correct computation. Subproblems of this class are correct computation of metrics when given findings with different formats (see the *distinct*-issue in the Case Study), careful accumulation of data stemming from different evaluation setups (see analysis of variances in the Case Study), and precise analysis towards the reasons of the resulting metrics (can they be attributed towards the automation as such, or are they rather a problem of the approach). A careful, defensive, and precise statistic analysis can provide a fair comparison and improve the informativeness of the results. Depending on the choice of comparison level, the choices of setup and the type of statistical evaluation, it could even provide replicability[12]. To that end, however, first, reproducibility for all steps must be ensured.

Answer to RQ1: In total, we identified four different problem classes that arise when evaluating and comparing automated approaches. These are 1) ensuring equivalent systems, 2) ensuring equivalent security concepts, 3) ensuring an equivalent findings format, and 4) ensuring a sound statistical analysis.

4 Potential standardisation

This section addresses our second research question: Which aspects of the identified problem classes have the potential to be standardised to allow a more fair, informative, and reproducible comparison of automated security flaw detection tools in the future? We will address this problem class by problem class:

4.1 Problem class 1: equivalent systems

At first glance, ensuring equivalent systems seems like a problem that can easily be solved via standardisation. Simply, provide a set of systems as benchmark for the community and from now on, all automated tools need to compare themselves to another based on these systems. Unfortunately, several issues complicate this matter. First of all, even when an agreement is made on systems, deciding on a specification form for these systems is mandatory. The first option (as it was in our case study) would be for the formalisation to be introduced by researchers as a service to the community. This would, however, almost always require a transformation process for the comparison with other tools, as they have not necessarily be based on the same modelling approach, which has two drawbacks for the evaluation: The burden of showing validity would always lie with the *different* tools and aspects that their modelling approaches might be capable of expressing but the original approach could not, could not be brought to light with these set of systems. Concludingly, it is unlikely that the community would accept such a process. Another option would be to provide a textual description of these systems. This variant would not prefer any modelling approach. Nevertheless, it would result in typical problems inherent to natural-language specifications. In the end, a *same set of systems* would not be guaranteed as it would be unclear whether differences are due to different understanding of the specification or different capabilities of the modelling and the associated tool. The other extreme would be providing actual implementations of systems that could act as *benchmark sets*. In this case, generally speaking, all modelling approaches and tools would be equally disadvantaged (provided a sufficient amount of variety in the set of systems). Nevertheless, it would (most likely) require an automated approach of extracting the required information for the given modelling approach and automated tool. In a perfect world, it would be possible to develop a modelling approach without any detection tool in mind that could model an entire system as a whole. This could abstract from implementation details, allow model-to-model transformation for translating to the respective model of the detection tool, and be void of natural-language impreciseness.

Given this set of systems (in whatever form), the process of finding, agreeing, and specifying these systems had to be repeated in a specified period. Otherwise, automated detection tools could overfit on this set of systems and the results would lose their generalisability.

Nevertheless, an agreed-upon set of systems (with a repeating fresh-up) as well as an agreed-upon form of specification is undoubtedly a long way down the road.

[12] As defined by the ACM in https://www.acm.org/publications/policies/artifact-review-and-badging-current

Transformations between system representations in modelling approaches associated with given detection tools are most likely going to be standard for the foreseeable future, if one aims for a level 1a+ comparison. To that end, one aspect that can now be standardised is the process of validating this model-to-model transformation, thus indicating that the transformation does not affect the system for the given systems and investigated concepts.

4.2 Problem class 2: equivalent security concepts

Equivalent security concepts are an easier aspect to standardise. This is mainly due to the fact that usually evaluations only focus on specified set of security concepts and these can much more easily be agreed upon than a set of systems with the claim to be a good sample for possible systems. Nevertheless, since there is not yet a community consensus on what a security flaw precisely is and what aspects belong to a security concept, it is still necessary to create a collection of security concepts and specify their relation. Furthermore, even if they are easier to agree upon (see, for example, Tuma et al.'s inspection guideline catalogue), they still have the problem of a precise specification and the possibility for automated security flaw detection rules to state which elements they cover and which they do not. So, while the entirety of security concepts can most certainly not be standardised, their representation and included aspects can. Additionally, this would tackle the issue of granularity, as a security concept could come with predefined checks and ground truth, or an automated security detection tool could specify which of these checks cover, improving a fair and informative evaluation.

4.3 Problem class 3: equivalent finding formats

Three aspects influence findings produced by an automated security-flaw detection tool: First, the modelling approach the tool is based on (as findings can only report their reason if it is modelled). Second, the defined aspects of the security concept that triggered its report, and third, the decision of the developer to report this information. While it is certainly not desirable to standardise or restrict modelling approaches as this would restrict the freedom of researchers to develop new ideas and tools, it should be possible to standardise the reporting of the aspects that triggered the security flaw reporting. With standardised reporting, a level 3 comparison can unfold its complete strength. This would significantly benefit the research community and practitioners by fostering clearer benchmarks, improving tool interoperability, and encouraging the adoption of best practices in automated security-flaw detection.

4.4 Problem class 4: sound statistic evaluation

There are four main aspects—in our opinion—that can be standardised. First, the computation of the respective metrics given a setup with non-standardised findings to prohibit a bias towards a specific tool (see Section 3.3.3). Second, a variance analysis is used to determine which factors allow grouping without distorting information and which do not. This is (as explained within the case study) important for the fairness of the comparison. Third, analysing *false positives* and *false negatives* to attribute errors to either the automation as such or the detection approach, which will help to understand the strengths and weaknesses of security-flaw detection. Fourth, if both tools have an equivalent findings format, standardised requirements for the analysis of the actual findings would improve the informativeness of the comparison.

4.5 Summary

This section focused on answering the second research question:

> **Answer to RQ2:** We find that some problems can be mitigated through standardisation, especially equivalent security concepts and finding formats w.r.t. the aspects that triggered their report. Additionally, the validation of system transformation and the performance of necessary statistical evaluations have standardisation potential.

5 Standardisation methods and evaluations

In this section, we will answer our third research question: How could the aspects mentioned in the last section be standardised? We will discuss a potential approach of standardising the validation of equivalent modelling in Section 5.1, while Sections 5.2 and 5.3 will discuss the development of a model for the security concepts and the resulting findings, presenting solutions for problem classes 2 and 3. Section 5.4 will discuss our standardisation for the statistically sound evaluation and comparison, tackling problem class 4. Finally, we summarise and relate our findings to our joint goal of supporting fair, informative and reproducible evaluations of automated security flaw detection tools in Section 5.5. Please note that some approaches (especially in problem classes 1 and 4) are also challenges in other domains. Nevertheless, we mention them here, as they are still important for a fair, reproducible and informative evaluation and comparison process for automated security flaw detection tools. We will point out during the following subsections which approaches are specific to the domain and which are of general importance.

5.1 Transformation of systems

Aiming for an equivalent set of systems is necessary for a level 1+ comparison. We have discussed at length in the last section why ensuring a *perfect* equivalence is no simple task and argued that—for now—providing means to justify an equivalent behaviour of systems increases (if followed) informativeness and fairness of the comparison process.

5.1.1 Idea

We consider the situation as presented in our case study: Systems are given as instances of the model of one (original) approach, not in a general manner. A model-to-model transformation then transforms these instances into instances of the other (new) approach. The relevant question is: Are these instances equivalent representations of the system?

In general, if we have a representation of a system following model A and a representation of (presumably) the same system following model B (S_A, S_B), together with a model-to-model transformation t, yielding $S_B = t(S_A)$, i.e., the representation w.r.t to model B from the representation w.r.t. to model A, we could provide evidence for equivalence, if we define an inverse transformation t^{-1} and validate its inverseness by verifying $S_A = t^{-1}(t(S_A))$ and $S_B = t(t^{-1}(S_B))$. If both hold, the transformation is bijective and thus, both representations are isomorphic. Please note that this would not yet state whether these representations are any good at all w.r.t. actually representing the system, just that the representations of the system are isomorphic. It would also not yet state semantic equivalence, just isomorphism. There are, however, two main (albeit very different) problems that occur when tackling this issue for actual case studies (as, for example, for the one presented in Section 3.3).

P1 Both models do not have the same expressive power. One model can capture different aspects of the system than the other. We expect this to be the case because these models are a core part of proposing a new automated security flaw detection tool.

P2 The base model is not provided. Here, the base model refers to the model the system is provided in. It may be that only graphical depictions of the systems are supplied, which allow reengineering a fitting base model, but not necessarily the base model. As this may lead to open questions regarding semantic equality, validation of the equation $S_A = t^{-1}(t(S_A))$ is not possible. Please note that $S_B = t(t^{-1}(S_B))$ can still be validated, however, as the model B is given.

The idea is to provide steps for researchers to validate isomorphism and provide indicators for equivalence induced by their defined model-to-model transformation. If researchers follow these steps and the results are positive, a sufficient form of isomorphism can be assumed. For our description of the method, we assume that a model-to-model transformation exists as well as a candidate for its inversion, capable of transforming the systems from the given model to the model necessary for the automatic security flaw detection tool.

5.1.2 Method & Justification

Our main approach is to argue that it is sufficient to validate $S_B = t(t^{-1}(S_B))$ to verify the isomorphy of the transformation (please note, not the equivalence). If both models have the same expressive power for the given set of systems, this is straightforward, as S_B is itself the image of S_A under t, and as such, the equality of cardinality yields the equivalence of both equations $S_B = t(t^{-1}(S_B))$ and $S_A = t^{-1}(t(S_A))$. However, it is more intricate if the models do not have the same expressive power w.r.t. the given set of systems: If, again, the base model has lesser expressive power, the above method is valid as well. As the model-to-model transformation is deterministic, the transformation only maps information of its expressive power to the new model. The inverse function can - in turn - not artificially add arbitrary information because it would break $S_B = t(t^{-1}(S_B))$. While this situation may be unfortunate for the new approach, as it can not apply all its model's capabilities to the systems, it still implies isomorphism restricted to the base model's capabilities. It may, however, hinder the new approach in unfolding its strengths. This is another major reason for tackling the issue of finding a common representation format for systems that is not specific to a tool-specific modelling approach. If the base model has greater expressive power (w.r.t. the systems), the transformation will delete information that the inverse function can not recreate. Nevertheless, validating $S_B = t(t^{-1}(S_B))$ will still validate the transformation, but on a smaller subset. The new approach will not be capable of expressing as much information as the base approach. This is, however, due to the approach, not due to non-equivalent systems.

Additionally, a researcher can perform a series of visual checks and automatic comparisons of metrics on the represented systems to ensure their equivalence, especially with respect to semantics. While these won't substitute a formal proof, they still support the reviewer in accepting the correctness of the model-to-model transformation w.r.t. the systems. In its entire generality, this problem and its proposed method for indicating equivalence is not domain-specific but a research topic and problem in its own right. If we consider, however, the specific application of our use case, where we can, for example, assume variants of data flow diagrams as graphic representations of models and consider comparable elements of the respective systems (that might have different names but provide a comparable semantic), our arguments are rather application-specific.

5.2 Guideline model

The inspection guideline proposed by Tuma et al. [9]—consisting of natural-language security flaw descriptions—has opportunities for improvement—shown in the last sections—, which we will discuss in the following. First, we describe the base idea behind the model, followed by the model description and an evaluation.

5.2.1 Idea

The rule descriptions—as they are intended for human beings—leave room for interpretation and thus lead to different problems that hinder a good comparison of the findings. In essence, we identified the following problems:

P1 A single rule consists of several parts that can be automated. It consists of different rules for detecting elements, and then several checks are listed that must be conducted for the identified elements. This makes the tool comparison less precise, as the missing reporting granularity makes it impossible to determine which parts of a rule are supported. This leads to the problem of being unable to conclude if the detection qualities are comparable.

P2 A goal of the comparison was to compare the expressiveness of the automation and to investigate if all rules can be detected with the same quality. The combination of several aspects in a single rule distorts the results as there might be rule parts that can be easily automated, and others that are not automatable at all. Combining them into a single rule results in limited meaningfulness of the results as it is a mixture of several rule parts.

P3 The ground truth contains entries that are neither applied to elements, such as processes, external entities, or data stores, nor applied to dataflows. These findings relate to the entire system, e.g., if a key revocation mechanism is not present. During the ground truth generation, such findings were not assigned to a particular element, as there is no corresponding process.

To propose an improved guideline, we first have a look at how the guideline is used, which Figure 10 shows. In a manual detection process, a *security expert* performs an *inspection* of a *system under investigation*. The *investigation* is steered by a *guideline* and produces a set of *findings*. The *findings* are used by a *system architect* to improve the design of the *system under investigation*. *Detection tools*, such as ArchSec and SecuriDFD automate the *investigation* by realising parts of the *guideline* by automatic checks. The goal is not to replace the *security expert* but to allow her to focus on more difficult threats that a tool cannot easily rule out. Therefore, *detection tools* must always support the possibility for *security experts* to extend the manually detected security flaws.

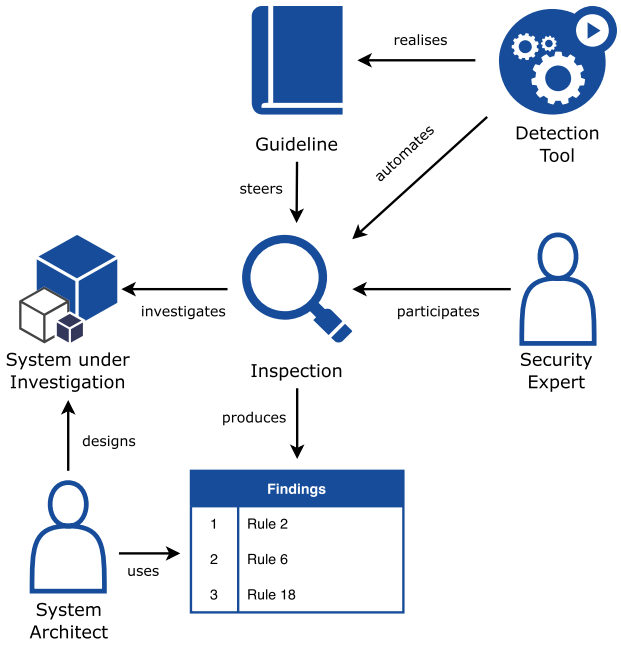

Fig. 10 Overview of the Security Flaw Detection Process

We propose a model-based guideline that helps define rules more precisely and how to match them to improve the current guideline. The new guideline does not aim to be automatically checkable but to provide a more precise way of checking. Furthermore, the goal is to provide a more fine-grained catalogue to allow detection tool vendors (and researchers) to provide a more precise list of supported and unsupported checks. This fosters an improved comparability of automatic detection approaches.

5.2.2 Method

Figure 11 shows the proposed guideline model in dark blue. The light blue classes are part of the inspection process model, which we will explain in the following subsection. A *Guideline* consists of a list of *Rule*s, which correspond to a rule by Tuma et al. (c.f. Listings 1). A rule can have multiple *Identification* instances that describe a pattern to identify elements of a dataflow diagram that might be vulnerable. To remove ambiguity, an *Identification* specifies the dataflow diagram element it targets and which might be vulnerable. Finally, an *Identification* has several associated *Check*s. A check describes a check that is required to determine if an identified element is vulnerable or not. We added the attribute *isAutomated* to the classes *Identification* and *Check* to allow tool vendors to indicate which parts of a guideline are automated and which a user has to check manually.

Problem Solution The proposed model explicitly separates *Identification*s and *Check*s to mitigate problem **P2**. Every *Identification* description is accompanied by a set of

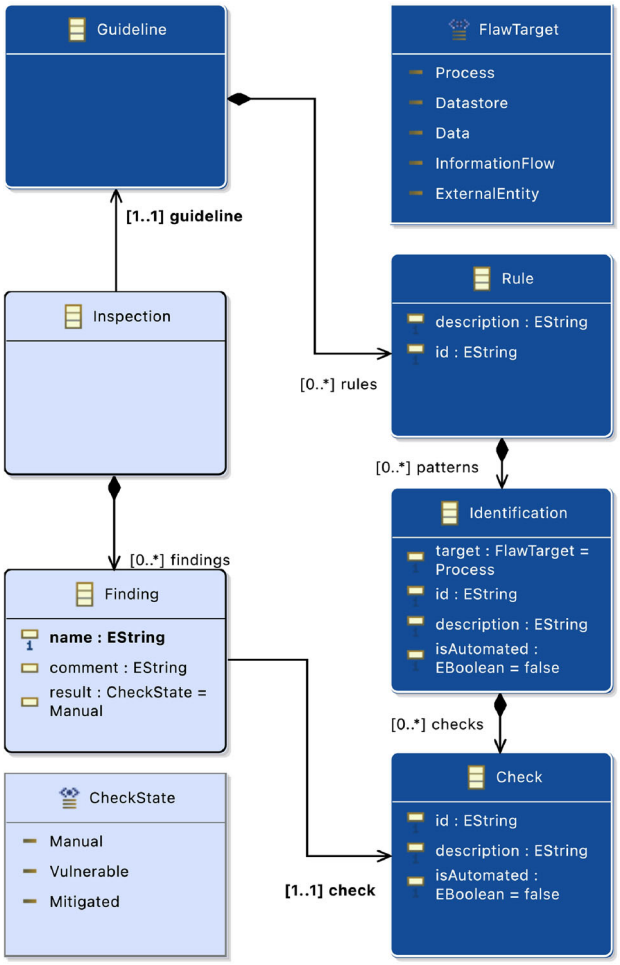

Fig. 11 Class diagram of the guideline model

associated *Check*s. To map the existing guideline to this modelling, it is necessary to map the existing checks to the identifications in order to separate them correctly.

The explicit separation described in the last paragraph, the *id*s and *isAutomated* flags allow a better comparison of findings and thus mitigate problem **P1**.

An investigation of the ground truth shows that the findings that are not associated with an element are not associated since the current scheme only allows the association to a data store or a data flow. Some identification descriptions are related to other elements, e.g. data. To make the targeted element explicit and mitigate problem **P3**, we added the *target* attribute.

5.2.3 Evaluation

Listing 3 shows *Inspection Guideline Rule 6*. When having a closer look at the description, one can see that the detection part starts with various aspects to identify, while the subsequent checks only apply to a subset of the identified elements.

This refers to the generation, distribution and storage of the cryptographic keys in the system. A compromise of a key means that every piece of information encrypted with this key is compromised. If there is no mechanism to replace a key or any auditing to aid in recovering, this can lead to serious compromise of the system.

Detection:

Determine where cryptographic keys are created, stored, used and how they are distributed.

Identify the mechanisms in place to replace a key.

Track where the keys are used, for which purpose and who has access to them.

For each key examine:

Is it generated within a cryptographic module isolated from the rest of the system?

Does the Random Number Generator (RNG) comply with latest standards?

Is the time the key is in plaintext format minimized?

Is the access to it during that time restricted only to authorized parties?

Are the keys distributed through secure channels?

Is the key stored securely?

If the key is stored locally in devices/clients, is it encrypted with Key Encryption Key (KEKs)?

Is the key integrity ensured?

the key used only by processes within the cryptographic module?

there a secure backup of the datastore that stores the keys?

Is all access to the key in plaintext format logged?

Is a key only used for a single purpose? (For example only for encrypting data but not other keys)

Is the key destroyed after it is no longer needed?

Is there a mechanism in place to renew/replace the keys in case they are compromised?

Listing 3: Inspection Guideline Rule 6: Insufficient Cryptographic Keys Management

The check *Are the keys distributed through secure channels?*, for instance, can only applied to data flows, such as the identified elements from *Key Distibution* and *Key Replacement*. Other checks, such as *Is the key stored securely?*, make only sense for the identified storage locations of the cryptographic keys. Furthermore, some aspects of the guideline cannot be checked automatically, such as if a key is destroyed when it is no longer needed. Using an EMF forms editor, we converted the guideline proposed by Tuma et al. [9] to our proposed model. Figure 12 depicts a screenshot that shows an excerpt of rule 6. The rule contains six identifications in total. Each of these identifications contains one to four checks. It is important to validate that our model can express the entire guideline to show that it is an appropriate model. To that end, we converted every listed security flaw of the inspection guideline (19 in total) to the defined model to show its expression capability. The entire model and the converted guideline can be found in the supplementary material[13].

[13] The supplemental material can be found at https://doi.org/10.5281/zenodo.15477264.

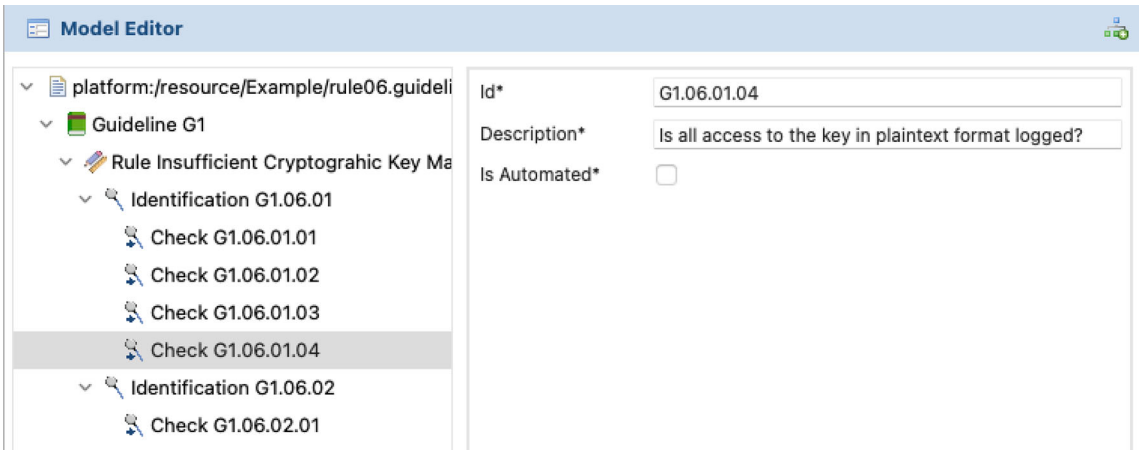

Fig. 12 Screenshot of an excerpt of the model of the Inspection Rule 6

While our evaluation can not show that our model is capable of expressing all possible security flaws, it shows that a community-accepted catalogue of 19 well-known security flaws [9] could be expressed with our model. This hints at a (so far) sufficient expressiveness of our model. Additionally, we would argue that our model-based guideline yields a more formalised approach and is thus less prone to yield ambiguous results during the detection process. Nevertheless, this has not been evaluated for this work and is left for future work.

5.3 Inspection process model

We have seen that a unified form of findings and their reporting is necessary to prevent potential bias problems, allowing a more profound and informative analysis and improved reproducibility (through the ensured reporting). Corresponding to the model guideline, we propose a model for findings or entire inspections. We will first discuss the idea of our approach, then introduce our model, and finally describe a justification for our method.

5.3.1 Idea

A model for inspections could guarantee a unified way of presenting results when applying the inspection guideline to systems. We already discussed that it is not advisable to restrict the form of findings in a way that necessitates a certain modelling approach for the detection tool, i.e., demanding elements or flows to be named, as it is not given what types of elements are present in a given modelling approach. It also does not necessarily increase the informativeness of the comparison. Relating findings to *identifications* and *checks*, however, significantly improves the informativeness, because the comparison can now focus on strengths and weaknesses with respect to flaw-categories. A tool can afterwards be provided with a statement like: *Our tool detects the following flaws based on these checks with a confidence of x%. It has weaknesses in the following categories, therefore, we suggest a manual redo for the following checks in addition to the automated process.* This statement delivers precise information for later applicants of the tool, and is more helpful than a mere: *Our tool has an overall precision and recall of x%.*

5.3.2 Method

Figure 11 shows the proposed inspection process model in light blue. A dataflow diagram inspection on a *System under Investigation* is comparable to a templating mechanism. First, all *Identification*s and the connected checks are evaluated. Given a match on a check for an *Identification*, a corresponding *Finding* is generated. Accordingly, an *Inspection* contains an unordered set of *Finding*s. A finding has a *name* representing the vulnerable element (which is the result of the *Identification* application). Additionally, it contains a *comment* holding additional information on the finding. Furthermore, it references the corresponding *Check* that corresponds to this finding. It is important to note that the combination of finding name and referenced check must be unique, meaning that a check can only produce a single result for a specific element. A tool can set the *result* to three different values: a) *Vulnerable*, b) *Mitigated*, or c) *Manual* if the rule was not evaluated automatically.

5.3.3 Justification

In the following, we discuss the validity of our approach. The main question is whether the above-described model can express findings and whether it is helpful for our original goal. The last question is easily answered: The referenced *checks* allow the linkage to the detection rule that triggered

this finding. This will also help distinguish subsets of systems that trigger the same security flaw more than once but for different reasons (checks). This is highly important for practice since a developer who tries to fix the detected security issues might otherwise only fix one of the reasons and erroneously think the task to be fulfilled. We, therefore, consider our model helpful for the original goal of *informativeness*. Additionally, it is helpful for the original goal of *fairness*, as multiple entries are now distinguished by checks, so it is less likely to run into the *distinct*-issue (see Section 3.3.3). It is still possible, however, so our approaches for statistical evaluation should not be disregarded, but it is less likely. Despite the previous discussion, we have not yet carried out an actual user study to support our arguments with empirical evidence. The expression power of our model is very high through its very open form: It has no requirements on the detection tool and its modelling approach. A finding only requires a name and the corresponding check. Suppose the finding has been obtained by following the inspection guideline. In that case, it is possible to provide the check (one would expect, in an automated tool, that respective queries are written *check-by-check*). We effortlessly transformed the findings of Berger et al.'s tool from the case study into this model.

Setting a *result* status might be an issue for tools that do not consider mitigated flaws. However, as they—by their design—only *find vulnerable* flaws, they can set this attribute to *vulnerable*. At the same time, adding this attribute allows the usage of this model also for semi-automatic tools, and—of course—the ground truth.

These arguments yield a variability of the chosen approach that allows an extension from the original dedication (automated security flaw detection tools) to semi-automatic or purely manual detections, allowing the usage for several evaluations and comparisons.

5.4 Standardising the statistical evaluation

This subsection addresses the issue of standardising the statistical evaluation of comparisons of automated security flaw detection tools. It can be seen as a list of statistical procedures or cross-checks that should be performed to guarantee the fairness of the evaluation and a suggestion of analysis to increase informativeness when only metrics are available for comparison. It is motivated and based mainly on our developments from the case study but summarised and presented in a structured manner in this subsection.

5.4.1 Idea

As we have mentioned in Section 3.4 and 4, there are three main issues that might hinder fairness, or decrease informativeness:

P1 A discrepancy in reporting findings leads to non-unique entries in the findings results, therefore making the choice of comparing findings to ground truth or ground truth to findings significant for the results.

P2 The factors that influence the evaluation setup (systems, investigated security concepts) have a significant influence on the results, but the results are presented in a cumulated manner.

P3 No findings were reported from the original study, thus only allowing a Level 2 comparison.

It should be noted that especially statistical evaluations should be rigorously documented and supplied as a script in the supplementary material for every published comparison study. Statistical methods depend on many factors and are influenced by many decisions, and a mere textual description might easily be imprecise in terms of relevant details to improve reproducibility and, thus, allow other researchers to retrace the evaluation steps. It is necessary to supply the actual routines.

5.4.2 Method

We will shortly present the solutions employed in the case study, leading to more fair and informative comparisons.

First of all, it is necessary to validate the results on computed *tp*, *fp*, and *fn*. Summing the number of true positives and the number of false positives should result in the number of all findings, as well as the sum of all true positives and all false negatives should result in the number of entries in the ground truth. A *distinct*-analysis is necessary if one of both statements does not hold for any of the investigated approaches (while applying identical metric computations for each). To perform a *distinct*-analysis, all non-distinct entries (possibly resulting from non-identical findings information) have to be unified to create an actual set of findings, i.e., no non-unique entries. Afterwards, the above-mentioned statements should hold, and one can proceed with the comparison. However, this process should be mentioned in the report, and we suggest an analysis of the reasons for the non-unique entries and whether they could be avoided. Although this issue can certainly come up in comparable analyses from other domains, it is particular to the situation of unspecified findings formats and, therefore, in its essence, is specific to the domain of security flaw detection.

Second, before *tp, fp, fn* or resulting metrics like *precision*, *recall*, or *f1-measure* are computed for the entirety of the respective study setup, they need to be computed w.r.t. the different influencing factors, usually varied systems and security concepts (when given a level 1+ comparison). Given these values, a variance analysis over these factors should be performed. Suppose there are less than five factor-levels (e.g., less than five systems or less than five investigated security concepts). In this case it is sensible to use extremal

values for this analysis (i.e., analysis the difference between the maximal and the minimal value) instead of computing the variance. The variance tends to be unstable for smaller values. High differences between extremal values or high variance values indicate that the factor has a decisive influence on the result, and thus, results should not be cumulated over this factor but should be reported independently. Additionally, high variances (or distances between extremal values) indicate that there are structural issues with some of the factor levels (i.e., some of the security concepts or some of the systems), which could motivate a more precise investigation. If the differences are small, it is valid to cumulate the results and only report aggregated values. It should, however, be stated that this analysis has been performed and what the results have been. This aspect is a general aspect that should be addressed in all statistical analyses that have different independent variables. Nevertheless, as it is essential for a fair and informative evaluation and comparison of automated security flaw detection rules, we state it here and apply its description to the domain at hand.

Third, if there are no findings reported from the original study besides comparing *precision*, *recall* and *f1-measure*, there is not much more in terms of informative comparison that can be done. However, when analysing automated approaches, an important question is whether the errors made are due to the automation or the approach. Errors due to the automation, e.g., checks for security flaws that are hard to detect automatically, will more often than not occur for all compared approaches. Errors due to the approach, e.g., the incapability of the modelling approach to express a decisive aspect, will more often than not occur only for the respective approach. This can be identified by comparing the ratio of individual *fp*s to common *fp*s and *fn*s, respectively. The higher the proportion of individual *fp*s or *fn*s, the more likely it is that the error is due to the approach. The higher the proportion of common *fp*s or *fn*s, the more likely it is due to the automation as such. The general idea to analyse which independent variable is responsible for the change in data is a general one. Nevertheless, the domain has struggled between the advantages and disadvantages of using manual vs. automatic approaches, and developing this concept for application in this domain seems a valid point.

5.5 Summary

In this section, we suggested different aspects of standardisation. A very specific contribution specifically made for this publication is the introduction of a model-based guideline for documenting security flaws that supports a model-based detection process and provides a clear findings format. This standardisation again gives rise to more informative statistical analyses that — in turn — require their form standardisation. Accordingly, we can answer the third research question:

> **Answer to RQ3:** We suggest a standardised way of validating system transformation, an approach-indepdent guideline and inspection process model, as well as, a standardised statistical evaluation that focuses on fairness.

6 Discussion

This section contextualises the results of the paper. To this end, we will first discuss the threats to validity in Subsection 6.1. Then we discuss research related to the presented paper in Subsection 6.2 and provide an outlook on the future work of this article's topic in Subsection 6.3.

6.1 Threats to validity

Throughout the paper, two parts of the evaluation need a discussion of possible threats to validity. We will discuss these parts separately.

6.1.1 Case Study

First of all, the main results of our paper are based on the case study that compares two automated security-flaw detection approaches we conducted in [10], summarised and extended in Section 3.3. To keep this paper self-contained, we will first discuss the threats to the validity of this comparison. Besides self-containedness, these threats to validity influence our identified problem and thus affect the new contributions of this paper.

Construction: Reverse engineering an existing model, as we have done with the SecurityDFD model from Tuma et al. [17], may lead to an incorrect model, invalidating all subsequent data. The given model instances are present in the XMI format that the EMF model reader reads. While parsing the data, it checks the syntactical correctness and whether the stored model strictly adheres to the model specification. This means that it loads the serialised instance graph and checks if all attributes and references are part of the specified model. If one of the members is missing, the model loader will raise an exception and refuse to load the model. We assume our reverse-engineered model is correct, as we can load all files successfully.

Replicating an already formalised rule set poses the threat of mirroring the rules of the original work. To counteract this, we based our patterns on the inspection guidelines. Furthermore, the author responsible for the evaluation process did not create security flaw patterns.

A manually created ground truth may contain mistakes, rendering the evaluation and comparison useless. Additionally, it might contain information not explicitly stated in the formal model, especially flaws that can not be mapped to specific elements of the given model.

Internal: To ensure the validity of the transformation, we tried to determine the equality of the transformed dataflow diagrams. While we checked the visual, structural, and semantical equivalence, the checked EDFDs may only be a semantic subset of the original security-enriched DFDs. Using a transformation model, such as *QVTr*, that allows bidirectional transformations would be a better choice. We decided against using this approach as the project website of Eclipse QVTr still states that it is still an experimental implementation.

The manual creation of the rule patterns based on a natural language description poses the threat of incompleteness compared to the original guideline. The measured *fn* values indicate a reasonably good execution. The formalisation in the form of a textual representation always holds the potential for impreciseness. For some rules, the identified result sets were nearly distinct. This was one inspiration for the contributions in this work.

External: In the evaluation, we focused on the same set of security flaws that Tuma et al. [17] have already analysed. Since this set only contains five different security flaws and the detection mechanism was only applied to a limited set of systems, we cannot rule out that the results are not generalisable.

6.1.2 Standardisation methods

After the comparison, we will focus on the threats to the validity of the standardisation methods presented in Section 5.

Internal: By modelling all rules of the guideline of Tuma et al. [9], it is not shown that we can model all possibly existing security flaws with the help of our guideline model. This indicates that the model can capture rules for recognising security flaws. If new rules for recognising security flaws are found in the future that do not fit into the current model, it must be ensured that the old model is compatible with the new one. The best way to do this is to specify a model-to-model transformation that transforms existing catalogues from the old model to the new one.

Similarly, an extension of the guideline model may lead to a change in the inspection model. Based on the changes, a model-to-model transformation would allow the conversion from the old inspection model to the new one.

6.2 Related work

There are several categories of related work to be discussed. Of course, the first category is other automated security flaw detection tools to justify the choice of tools for the original case study. Furthermore, related work needs to be analysed with respect to works that analyse comparisons or provide systematic studies to that end, as well as introduce standards for comparison of automated security flaw detection tools. Finally, our proposed models for the inspection guideline, as well as the inspection process, need to be compared to the respective work in the literature.

6.2.1 Automated security flaw detection tools

In the following, we will list existing approaches focusing on security flaw detection.

Tuma et al. list 26 existing threat analysis techniques in their systematic literature review [29] and compare their features. Only 15 of the presented approaches are tool support, and seven of them have an extensible knowledge base. According to Tuma et al., only two of these seven tools can be applied automatically or semi-automatically. One of them is the approach by Berger et al., and the other one is used in the later publication [17]. Tarandach and Coles report on additional tools [13]. As already mentioned the open-source tools *Threatspec* and *ThreatPlaybook* have different focuses and aim for documenting the results of manual threat modeling within or alongside the code.

Jürjens et al. presented *UMLsec* between 2001 and 2012 [30–35]. *UMLsec* is a UML profile thats aims to specify and check security requirements. The authors focus on different aspects of cryptography, cryptographic protocols, and authentication. The security-relevant aspects of a software system are added to the UML models using UML stereotypes, tagged values, and constraints to activity diagrams, state charts, sequence diagrams, static structure diagrams, and deployment diagrams. In contrast to the presented approaches, *UMLsec* does not support an extensible knowledge base.

Abi-Antoun et al. published several papers facilitating dataflow diagrams for automatic threat identification [36–38]. The authors describe DFDs as a runtime view following the Component-and-Connector view type, according to Clement et al.'s definition [39]. The proposed approach first extracts an as-implemented DFD from a software's binary and compares it to a given as-designed DFD using an extended version of Murphy et al.'s reflexion model [40]. The presented approach deals with automatic threat identification, but according to their publications, it is not extensible.

Almorsy, Grundy, and Ibrahim introduced an approach to detect implementation- and architectural-level security flaws based on formalised signatures. A vulnerability signature is expressed using OCL and consists of a set of invariants. The invariants are applied, and the vulnerability exists if the invariants hold. The authors list signatures for SQL injection, cross-site scripting, improper authorisation,

cross-site request forgery, information exposure, URL redirection and improper authentication [41]. In 2013, Almorsy, Grundy, and Ibrahim extended the approach to supporting security-relevant metrics and architecture-level security flaws [42]. The approach presented by Almorsy et al. is extensible regarding their knowledge base, but does not apply to dataflow diagrams. Furthermore, there is no prototype implementation.

6.2.2 Comparison of tools

To our knowledge, no research compares or replicates the detection capabilities of different automatic threat identification techniques nor focuses on standardising these processes, making it impossible to present related work in this direction.

However, there are qualitative comparison studies that compare used representation types, employed detection techniques, and security flaw categories, e.g., [20]. They, however, rather have the form of a systematic literature review or be mainly qualitative. These are interesting sources for obtaining an overview of different techniques for automated detection approaches, but they do not meet the scope of our quantitative evaluation and comparison approach. Harzevili et al., for instance, conduct a systematic mapping on automated software vulnerability detection approaches that use ML techniques. They compare existing tools by stating their methodology, list used benchmark sets for evaluation, and show what kind of vulnerabilities are detected [43].

Furthermore, comparing tools from a broader perspective is a well-known topic and receives wide recognition in the software engineering community. In [21], experiments in software engineering are described as a whole, also covering empirical tool comparisons. They mainly focus on correct experiment planning, setup, and evaluation. Several aspects, e.g., the variance analysis, we mentioned, can be found here as well (and can be found in any good book on experimental design, see, e.g. [44]). Nevertheless, some aspects are particular to security flaw detection and comparing manual and automatic approaches. One example is the fact that in security flaw detection, it is always necessary to base an evaluation on a ground truth or be restricted to a true positive/false positive analysis, as there never is a *definitely true* list of security flaws for a system. The ground truth can be interpreted as such, but even decade-long experts are not free of errors. Regarding the development of benchmarks and common ground for evaluations and comparison of tools, the software verification community has undertaken significant steps to build up benchmarks and standards comparably to what we hope to establish for the security flaw detection community. Examples are the workshops and competitions at ETAPS[14]. In this year's publication for the SV-COMP workshop (see [45]), they actually perform similar steps to our proposed approach, like including a property language and common formats for comparing results of the competition and providing a repository of software-verification tasks (which is similar to our set of benchmark systems). The fact that different domains address the same issues and find comparing solutions (in an abstract form) can be seen as a sign of the domain-overarching need for fair, reproducible and informative evaluation and comparison standards. Our approach to introducing a model for the format of the finding goes along these lines but considers domain-specific aspects like different detection types and security flaw categories.

One aspect for ensuring a fair comparison, we mentioned in our analysis, was the correct transformation of benchmark software systems to the respective model required for the approach (to be able to state that they have been evaluated on equivalent systems). In [46], the authors present an overview of different verification techniques for model-to-model transformations. Our approach is an attempt to *not outrule* the equivalence of systems given the minimal set of information present in our case study, for example. This, of course, can not prove equivalence, but, however, hint at it. To actually prove equivalence, one of the techniques presented in [46] should be used if possible given the respective set of information. This, furthermore, leads to the requirement of an open model for defining these benchmark systems to allow the usage of fitting verification techniques for the model-to-model transformations necessary for a fair evaluation and comparison.

6.2.3 Models for security flaws

We could not find comparable approaches for our model for security flaws. However, from a certain point of view, the natural-language catalogue of Tuma et al. [9] already is some form of model of these flaws. We more or less extended this approach, restructured it and tried to improve the precision of the presentation. Also, in [47], they introduced a taxonomy for security problems in software systems. This is a comparable step in the same direction we undertook. However, the taxonomy focuses on security problems in general, and thus summarise all architectural security problems to a single group. The publication does not investigate further into the details and structure of architectural security flaws.

Santos et al. suggested a catalogue of architectural security weaknesses. The proposed catalogue is based on the common weakness enumeration (CWE) and groups these into security concept groups, such as *encrypt data*. The catalogue does not focus on detection and automation aspects [48].

[14] https://verifythis.github.io/, https://sv-comp.sosy-lab.org

6.3 Future work

There are several aspects to extend this work. They can (in our opinion) be mainly categorised into two areas: The first area can be described as continuing down the path of establishing a community standard for evaluating and comparing automated security flaw detection tools. The second area covers aspects of improving and extending automated security flaw detection tools. We will discuss both subsequently.

6.3.1 Towards a fair evaluation and comparison standard

While our main (additional) contribution in this paper was the analysis of challenges in evaluating and comparing automated security flaw detection tools, as well as the derivation of potential for standardisation, we additionally provided first works for standardisation and the introduction of a community standard. We plan to continue down this road by focusing on the following three steps:

First, provide a deeper evaluation of our work presented in Sectiona 5.2 and 5.3. While we provided a first evaluation of the expressiveness of both models in this work, we also need to conduct a more thorough evaluation of their usefulness and applicability to other catalogues of security flaws. To achieve our overall goal of a fair, informative and reproducible evaluation and comparison process in security flaw detection research, it is important that our proposed models are widely accepted and used in the community. This can only be ensured by providing the necessary expressiveness and usability for other researchers or practitioners.

Second, the development of an agreed-upon benchmark set in terms of security flaws and investigated systems. Hand in hand with the development of a model for the inspection guideline, we need to extend the current catalogue (originally provided by Tuma et al.) to be as all-encompassing as possible. Evaluations and comparisons can still be performed on a subset of the collected security flaws, but it needs to be ensured that nomenclature and definition are common ground for all researchers. Additionally, a constant update of the catalogue is required to make sure that newly discovered security flaws are included in this catalogue. To this end, it is necessary to dive into the known architectural security flaws—those collected in the published guideline by Tuma et al.[9], as well as from other sources—and to categorise them according to the information, such as structural, dataflow, or sequence information, that is necessary to model and analyse them fully. This categorisation will help to understand which flaws can currently be detected well and which cannot. Furthermore, the community needs (in our opinion) a well-thought set of systems to be used for evaluations of security flaw detection tools. These benchmarks have to provide a good distribution of real systems and occurring security flaws. Last, but not least, they need to be updated as well, to prohibit an overfitting of the security flaw detection tools to these systems. For all of these systems, ground truths need to be provided that are widely agreed upon by the community and follow the model for findings we proposed in Section 5.3.

It might be worthwhile to investigate those threat modeling tools that focus on threat modeling in the context of continuous integration to collect more real-world systems for the ground truth. These tools use configuration files, such as YAML files, to manually document assets, use cases, and potential security flaws. Open-source tools that use such approaches would be interesting candidates for use as systems that could be incorporated.

Third, working on developing a general formalised description for systems or, if this is not successful, coming up with a common form of description for benchmark systems that allows other researchers to evaluate their tools on a comparable set of systems. We have discussed these issues at length in Section 4, and there probably will not be an easy solution to that problem. We have thought about organising respective workshops within the community to come up with a common ground for the description form of benchmark systems.

Finally, as soon as the findings model, as well as the security flaw description catalogue, are agreed upon, it is possible to provide researchers with a pre-built statistical evaluation and comparison script which will perform analyses based on these formats and flaws, as well as our proposed points from Section 5, and followingly ensure comparable results.

6.3.2 Improving and extending automated security flaw detection tools

Currently, the automatic detection rules used for the ground truth are limited to those used by Tuma et al. in their case study [17] to keep the comparison fair. Accordingly, it would be a good idea to a) investigate the remaining not-yet-automated guideline rules and b) apply the rule base provided by Berger et al. to find similarities and differences to the guideline.

Another aspect of future work is the extension of the research on security flaws and automatic security flaw detection into the context of hardware design. As part of the *DI-ExViPaS*[15] project, we research how the threat-modeling idea can be transferred to hardware designs, what kind of architectural views are of interest here and how they can be extracted automatically.

[15] Parts of the work were funded by the German Federal Ministry of Education and Research under grant number 16ME0970.

7 Conclusion

Threat Modeling is a manual attacker-centric security-analysis technique that gets more and more attention from research and practitioners as it can reveal architectural security flaws. Automating the security flaw detection on existing dataflow diagrams is one of the current research directions, thus leading to the necessity of comparing tools with one another. It is important to make these comparisons fair, reproducible, and informative to identify those aspects of the approach that require improvements.

We were motivated to this work through a case study we conducted for a publication at MODELS'23 [10], where we compared two dataflow based automated security flaw detection tools to one another. During this comparison, we came across several justified ways of comparing architectural security flaw detection tools. Every comparison method led to a different result, which shows the importance of standardising the comparison to make the comparison results meaningful.

Within this work, we take up the thread from this case study and thoroughly discuss pitfalls and unwilling biases for evaluations and comparison of automated security flaw detection processes. To that end, we first analyse and introduce different levels of comparison and their information quality. We then discuss challenges in standardisation along the lines of these levels of comparison. Mainly we identify issues of standardisation for benchmark systems, security flaws, and the reporting form of findings. Additionally, we identify some statistical effects (some of a more general nature, some specific to the domain) that researchers need to be aware of when conducting comparisons of tools for automated security flaw detection. We analyse which of these issues have potential of being standardised, and propose a guideline and inspection model for formalising security flaw guidelines and reporting respective findings. We provide a small evaluation of expressiveness with respect to the guideline model by expressing an entire catalogue of security flaws through this model.

We are aware that this work is only the beginning on the path to our overall goal, a fair, reproducible and informative evaluation and comparison approach for the tools in the area of security flaw detection. In our discussion section, we discuss approaches from other domains that could be applied to the domain of security flaw detection and suggest further steps of research and action to provide common evaluation ground for the community. In order to meet our own standards of informativeness and reproducibility, all artefacts developed and used in the paper are publicly available and thus can be used by the community to better compare tools for architectural security flaw detection. We hope that with future work and the support of the community in deriving well-constructed benchmarks, we can improve research in this important area. Being able to detect more security flaws automatically in a safe manner, would greatly improve systems across all application areas.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. McGraw, G.: Software security: building security in (Addison-Wesley Professional, 2006)
2. Swiderski, F., Snyder, W.: Threat Modeling. Microsoft Press, USA (2004)
3. Jiang, L., Chen, H., Deng, F.: A security evaluation method based on stride model for web service. In: 2nd international workshop on intelligent systems and applications, 1–5 (IEEE, 2010). (2010). https://doi.org/10.1109/IWISA.2010.5473445
4. Dhillon, D., Mishra, V.: Applied threat driven security verification. In: 2018 IEEE cybersecurity development (SecDev), 135–135 (2018)
5. Rehman, S., Mustafa, K.: Research on software design level security vulnerabilities. SIGSOFT Softw. Eng. Notes **34**, 1–5 (2009). https://doi.org/10.1145/1640162.1640171
6. Tarandach, I., Coles, M.J.: Threat modeling – a practical guide for development teams, 1st edn. O'Reilly Media Inc, USA (2020)
7. Shostack, A.: Threat Modeling: Designing for Security 1st edn (Wiley Publishing, 2014)
8. Yskout, K., Scandariato, R., Joosen, W.: Do security patterns really help designers?. In: Proceedings of the 37th international conference on software engineering - Volume 1, ICSE '15, 292–302 (IEEE Press, 2015)
9. Tuma, K., Hosseini, D., Malamas, K., Scandariato, R.: Inspection guidelines to identify security design flaws. Proceedings of the 13th european conference on software architecture - Volume 2, ECSA '19, 116–122 (Association for Computing Machinery, New York, NY, USA, 2019). https://doi.org/10.1145/3344948.3344995
10. Berger, B. J., Plump, C.: Automatic security-flaw detection replication and comparison. In: ACM/IEEE 26th international conference on model driven engineering languages and systems (MODELS), 84–94 (IEEE Press, 2023). (2023). https://doi.org/10.1109/MODELS58315.2023.00027
11. Dhillon, D.: Developer-driven threat modeling: Lessons learned in the trenches. IEEE Secur. Privacy **9**, 41–47 (2011)
12. MIL-STD-882C – System safety program requirements. In: Tech. Rep., Department of Defense (1993)
13. Tarandach, I., Coles, M. J.: Threat Modeling (O'Reilly Media, Inc, 2020)
14. Schneier, B.: We have root: even more advice from Schneier on security, 1st edn. John Wiley & Sons, Incorporated (2019)

15. Frydman, M., Ruiz, G., Heymann, E., César, E., Miller, B.P.: Automating risk analysis of software design models. Sci. World J. **2014**, 1–12 (2014). https://doi.org/10.1155/2014/805856
16. Berger, B. J., Sohr, K., Koschke, R., Caballero, J., Bodden, E., Athanasopoulos, E.: (eds) Automatically extracting threats from extended data flow diagrams. (eds Caballero, J., Bodden, E. & Athanasopoulos, E.) Engineering secure software and systems - 8th international symposium, ESSoS 2016, London, UK, April 6-8. Proceedings, Vol. 9639 of Lecture Notes in Computer Science, 56–71 (Springer, 2016). (2016). https://doi.org/10.1007/978-3-319-30806-7_4
17. Tuma, K., Sion, L., Scandariato, R., Yskout, K.: Automating the early detection of security design flaws. Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems, MODELS '20, 332–342 (Association for Computing Machinery, New York, NY, USA, 2020). https://doi.org/10.1145/3365438.3410954
18. De Rosa, F., Maunero, N., Prinetto, P., Talentino, F., Trussoni, M.: Threma: Ontology-based automated threat modeling for ict infrastructures. IEEE Access **10**, 116514–116526 (2022)
19. Schneider, S., Scandariato, R.: Automatic extraction of security-rich dataflow diagrams for microservice applications written in java. J. Syst. Softw. **202**, 111722 (2023). https://doi.org/10.1016/j.jss.2023.111722
20. Granata, D., Rak, M., Salzillo, G.: Automated threat modeling approaches: comparison of open source tools, 250–265 (Springer International Publishing, 2022). https://doi.org/10.1007/978-3-031-14179-9_17
21. Wohlin, C., et al.: Experimentation in software engineering (Springer. Berlin Heidelberg (2024). https://doi.org/10.1007/978-3-662-69306-3
22. Dennis, A. R., Valacich, J. S.: A replication manifesto. AIS Transactions on Replication Research, 1 (2015)
23. Berger, B. J., Sohr, K., Koschke, R.: The architectural security tool suite - ARCHSEC. In: 19th international working conference on source code analysis and manipulation, SCAM 2019, Cleveland, OH, USA, September 30 - October 1, 250–255 (IEEE, 2019). (2019). https://doi.org/10.1109/SCAM.2019.00035
24. DeMarco, T.: Structured analysis and system specification Yourdon computing series. Yourdon, Upper Saddle River, NJ (1979)
25. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse modeling framework 2nd edn (Pearson International, 2008)
26. Francis, N., et al.: Cypher: An evolving query language for property graphs. Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, 1433–1445 (Association for Computing Machinery, New York, NY, USA, 2018). https://doi.org/10.1145/3183713.3190657
27. Berger, B. J., Sohr, K., Kalinna, U. H.: in *Architekturelle Sicherheitsanalyse für Android* (ed.Horster, P.) D ● A●CH Security 2014: Bestandsaufnahme - Konzepte - Anwendungen - Perspektiven 287–298 (Peter Schartner and Peter Lipp, 2014)
28. Varró, D., Balogh, A.: The model transformation language of the viatra2 framework. Sci. Comput. Program. **68**, 214–234 (2007)
29. Tuma, K., Calikli, G., Scandariato, R.: Threat analysis of software systems: a systematic literature review. J. Syst. Softw. **144**, 275–294 (2018)
30. Jürjens, J., Hußmann, H.: Towards development of secure systems using umlsec. (ed.Hußmann, H.) Fundamental Approaches to Software Engineering, 4th International Conference, FASE 2001 Held as Part of the Joint European conferences on theory and practice of software, ETAPS 2001 Genova, Italy, April 2-6, Proceedings, Vol. 2029 of Lecture Notes in Computer Science, 187–200 (Springer, 2001). (2001). https://doi.org/10.1007/3-540-45314-8_14
31. Jürjens, J., Jézéquel, J.-M., Hussmann, H., Cook, S.: (eds) Umlsec: Extending uml for secure systems development. (eds Jézéquel, J.-M., Hussmann, H. & Cook, S.) <<UML>> 2002 — The Unified Modeling Language, 412–425 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2002)
32. Jürjens, J., Houmb, S. H., Reis, R.: (ed.) Risk-driven development of security-critical systems using umlsec. (ed.Reis, R.) Information Technology, Selected Tutorials, IFIP 18th world computer congress, tutorials, 22-27 August, Toulouse, France, Vol. 157 of IFIP, 21–53 (Kluwer/Springer, 2004). (2004). https://doi.org/10.1007/1-4020-8159-6_2
33. Best, B., Jürjens, J., Nuseibeh, B.: Model-based security engineering of distributed information systems using umlsec. In: 29th international conference on software engineering (ICSE'07), 581–590 (2007)
34. Jürjens, J.: Model-based security testing using UMLSEC: a case study. Electron. Notes Theoretical Comput. Sci. **220**, 93–104 (2008)
35. Ruhroth, T., Jürjens, J.: Supporting security assurance in the context of evolution: Modular modeling and analysis with umlsec. 14th International IEEE symposium on high-assurance systems engineering, HASE, Omaha, NE, USA, October 25-27, 2012, 177–184 (IEEE Computer Society, 2012). (2012). https://doi.org/10.1109/HASE.2012.35
36. Abi-Antoun, M., Wang, D., Torr, P.: Checking threat modeling data flow diagrams for implementation conformance and security. Proceedings of the Twenty-Second IEEE/ACM international conference on automated software engineering, ASE '07, 393–396 (Association for Computing Machinery, New York, NY, USA, 2007). https://doi.org/10.1145/1321631.1321692
37. Abi-Antoun, M., Barnes, J. M.: Analyzing security architectures. Proceedings of the IEEE/ACM international conference on automated software engineering, ASE '10, 3–12 (Association for Computing Machinery, New York, NY, USA, 2010). https://doi.org/10.1145/1858996.1859001
38. Vanciu, R., Abi-Antoun, M.: Ownership object graphs with dataflow edges. 19th Working conference on reverse engineering, 267–276 (2012)
39. Garlan, D. et al.: Documenting software architectures: views and beyond 2nd edn (Addison-Wesley Professional, 2010)
40. Murphy, G.C., Notkin, D., Sullivan, K.J.: Software reflexion models: Bridging the gap between design and implementation. IEEE Trans. Softw. Eng. **27**, 364–380 (2001). https://doi.org/10.1109/32.917525
41. Almorsy, M., Grundy, J., Ibrahim, A. S.: Supporting automated vulnerability analysis using formalized vulnerability signatures. 2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering, 100–109 (2012)
42. Almorsy, M., Grundy, J., Ibrahim, A. S.: Automated software architecture security risk analysis using formalized signatures. Proceedings of the 2013 international conference on software engineering, ICSE '13, 662–671 (IEEE Press, 2013)
43. Shiri Harzevili, N., et al.: A systematic literature review on automated software vulnerability detection using machine learning. ACM Comput. Surv. (2024). https://doi.org/10.1145/3699711
44. William G., Cochran, G. M. C.: Experimental Designs 2. edn (Wiley, 1994)
45. Beyer, D., Strejček, J., Gurfinkel, A., Heule, M.: (eds) Improvements in software verification and witness validation: Sv-comp. (eds Gurfinkel, A. & Heule, M.) Tools and Algorithms for the construction and analysis of systems, 151–186 (Springer Nature Switzerland, Cham, 2025) (2025)
46. Ab. Rahim, L., Whittle, J.: A survey of approaches for verifying model transformations. Software & Systems Modeling **14**, 1003–s1028 (2013). https://doi.org/10.1007/s10270-013-0358-0
47. Landwehr, C.E., Bull, A.R., McDermott, J.P., Choi, W.S.: A taxonomy of computer program security flaws. ACM Comput. Surveys **26**, 211–254 (1994). https://doi.org/10.1145/185403.185412

48. Santos, J. C. S., Tarrit, K., Mirakhorli, M., Malavolta, I., Capilla, R.: (eds) A catalog of security architecture weaknesses. (eds Malavolta, I. & Capilla, R.) 2017 IEEE international conference on software architecture workshops (ICSAW), 220–223 (IEEE, 2017). https://doi.org/10.1109/ICSAW.2017.25

Bernhard J. Berger received his diploma in computer science from the University of Bremen in 2008. He worked as a Technical Sales for Axivion GmbH, Stuttgart, between 2008 and 2010. For his doctorate, which he completed in 2022, he returned to the University of Bremen, where he worked with the Chair of Software Engineering. He has been working as a tenured lecturer at the Hamburg University of Technology since 2021. He interrupted this position in 2024 for a visiting professorship at the Software Engineering Chair at the University of Rostock and a visiting professorship at the Secure Systems Chair at the University of Bremen. His main research is model-based software engineering and its applications to many domains, with a strong focus on optimisation.

Christina Plump received her diplomas in computer science and mathematics from the University of Bremen in 2012 and 2014, respectively. She works at the research group of computer architecture with Prof. Drechsler, who is also her PhD thesis supervisor. Her main research interests are domain-oriented optimisation, the correct interleaving of ML techniques and nature-inspired optimisation. During her work in this area, she has developed a keen interest in fair and unbiased evaluations, which are particularly challenging when combining stochastic algorithms. Together with her study focus on cryptography, she is happy to apply her statistical knowledge to evaluation processes in security flaw detection.