The Journal of Systems & Software 169 (2020) 110697

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

A systematic literature review of model-driven security engineering for cyber–physical systems

Johannes Geismann[a,b,*], Eric Bodden[a,b,c]

[a] Software Engineering Group, Heinz Nixdorf Institute, Fürstenallee 11, 33102 Paderborn, Germany
[b] Department of Computer Science, Paderborn University, Fürstenallee 11, 33102 Paderborn, Germany
[c] Fraunhofer IEM, Zukunftsmeile 1, 33102 Paderborn, Germany

ARTICLE INFO

Article history:
Received 29 August 2019
Received in revised form 8 June 2020
Accepted 15 June 2020
Available online 16 June 2020

Keywords:
Literature survey
Systematic literature review
Model-driven security
Cyber–physical systems
Platform-specific
Security modeling

ABSTRACT

The last years have elevated the importance of cyber–physical systems like IoT applications, smart cars, or industrial control systems, and, therefore, these systems have also come into the focus of attackers. In contrast to software products running on PCs or smartphones, updating and maintaining cyber–physical systems presents a major challenge. This challenge, combined with the often decades-long lifetime of cyber–physical systems, and with their deployment in often safety-critical contexts, makes it particularly important to consider their security already at design time. When aiming to obtain a provably secure design, model-driven security approaches are key, as they allow to identify and mitigate threats in early phases of the development. As attacks may exploit both code-level as well as physical vulnerabilities, such approaches must consider not just the cyber layer but the physical layer as well. To find out which model-driven security approaches for cyber–physical systems exist considering both layers, we conducted a systematic literature review. From a set of 1160 initial papers, we extracted 69 relevant publications describing 17 candidate approaches. We found seven approaches specifically developed for cyber–physical systems. We provide a comprehensive description of these approaches, discuss them in particular detail, and determine their limitations. We found out that model-driven security is a relevant research area but most approaches focus only on specific security properties and even for CPS-specific approaches the platform is only rarely taken into account.

1. Introduction

Cyber–physical Systems (CPS) Lee (2008) are complex embedded systems that are distributed over several control units, communicate with each other, and interact with their physical environment (Fitzgerald et al., 2014). Examples are systems in the automotive domain, aerospace, medical environments, or industrial control systems. Often, such systems are used in safety-critical contexts and, therefore, have to fulfill safety-requirements, i.e., hard real-time requirements (Lee, 2008). Since modern systems interact with each other and with their environment, these systems do not operate any longer in isolation. Thus, malicious actors may gain access to the infrastructure of these systems and, therefore, security for CPS becomes a topic of high relevance (Reddy, 2015). Furthermore, in contrast to classical software systems, CPS use both the *cyber* and the *physical layer* for interaction and communication. Thus, both layers are possible attack surfaces and attack targets (Dorbala and Bhadoria, 2015) as recent incidents show like the attacks on a steel mill (Lee et al., 2014) or modern cars (Ishtiaq Roufa et al., 2010). Due to this hybrid nature of CPS and their increasing relevance in daily life, it is important to consider security for both layers as well as their combination (Rajkumar et al., 2010).

Many CPS have to operate for several years without downtime and, therefore, fixing vulnerabilities gets more complex. For example, updating a car's software has to be done manually, by a certified mechanic, or by using public update channels which may increase the attack surface. Furthermore, updating embedded systems for fixing vulnerabilities at runtime is in many cases difficult or even impossible and leads to high costs (Cigital Federal Inc., 2011), e.g., in industrial control systems where downtime of the system gets very expensive. Another problem is that some vulnerabilities might not be fixable by simply updating the software because they result from a bad design decision regarding software *and* platform, e.g., when deploying software components to executing hardware nodes of the system. Thus, considering security decisions at design time of the system can help detect and prevent vulnerabilities early in the development lifecycle (Uzunov et al., 2012).

Model-Driven Software Development (MDSD) Völter et al. (2013) has become a leading paradigm for developing CPS and to

* Corresponding author at: Software Engineering Group, Heinz Nixdorf Institute, Fürstenallee 11, 33102 Paderborn, Germany.
E-mail address: johannes.geismann@uni-paderborn.de (J. Geismann).

https://doi.org/10.1016/j.jss.2020.110697
0164-1212/© 2020 Elsevier Inc. All rights reserved.

formally verify safety requirements at their design time. In MDSD, models are used as first-class entities, which means that the models are used as primary artifacts of the development (Brambilla et al., 2012). This includes that they are formally refined and synthesized to more specific models and finally to source code guaranteeing to fulfill the safety requirements at any development step. Applying MDSD to security has to be considered in a special way, since not all security requirements can be expressed by liveness and safety properties (McLean, 1996). Additionally, security properties have to be explicitly considered for each step of the software development life cycle (in the following denoted as SDLC) because even correct refinements may not preserve proven security properties (Jacob, 1989). Similarly to approaches for safety, in the last decade, several model-driven engineering approaches for security were developed to consider security already during the design of the system (Nguyen et al., 2015). The focus of these approaches differs in the application domain, used modeling languages, kinds of analyses, and covered security properties. The same holds for the goal of these approaches because some approaches focus, for example, on the verification of security protocols and others on generating security test cases. While most approaches focus on information systems in general, there are also approaches that are tailored to the special domain of CPS.

Based on the guidelines proposed by Kitchenham (2004) and Kitchenham and Charters (2007), we conducted a systematic literature review to find out which model-driven approaches for secure CPS do exist that cover explicitly both software description ("cyber"-layer) and platform description. For CPS, this platform layer gets important because it describes not only the runtime environment (including operating system, middleware, and provided libraries) but also the physical layer which represents the connection to the physical environment. From a set of 1160 papers, we extracted 17 model-driven security approaches that consider the platform. We found out that seven of these approaches are specific to CPS and investigated these approaches in more detail. Our survey summarizes each approach, shows how the platform information is utilized in these approaches, which phases of the software development lifecycle are considered most, and which kinds of threats are covered. We state current limitations of these approaches and discuss open research questions and directions in this area. The survey showed that the selected approaches insufficiently cover the platform layer and the integration of third-party code, that threat modeling is applied only in an implicit way, and that the degree of automation of tracing and refining security requirements into implemented security solutions should be increased.

The remainder of this paper is structured as follows. In Section 2, we outline the rationale of the survey and discuss the relation to other existing surveys in this area. In Section 3, we describe the planning and conduct of the survey. We describe the selected approaches in Section 4 and summarize and discuss results of the survey in Section 5. Finally, we conclude and state future directions in Section 6.

2. Background and rationale of the survey

Following Edward Lee, "Cyber–Physical Systems [...] are integrations of computation with physical processes". Lee (2008). In contrast to embedded systems, one additional key aspect of CPS is the network communication within and between such systems (Lee, 2008; Zhang et al., 2013). From the security point of view, this becomes a highly relevant aspect because CPS and their security rely not only on cyber attacks but also on their physical environment (Rajkumar et al., 2010). It is important to consider these different kind of attacks as well es hybrid attacks (exploiting both cyber attacks and physical attacks) in early development steps (Banerjee et al., 2012). Hence, in the context of this survey, we focus on this special nature of CPS. In CPS, vulnerabilities are particularly hard to fix after deployment because maintenance is not possible or the vulnerability is based on bad design decisions. Hence, it is important to consider security in the design phase and to refine these secure designs into deployed systems. Finding and preventing vulnerabilities is up to 30 times less expensive than fixing them in a deployed system (Cigital Federal Inc., 2011). Applying model-driven engineering for security can help to address this problem and to prove that specific security properties and assumptions hold in the deployed system as well.

In the last few years, surveys in the area of model-driven security and security for distributed systems were already published. Nguyen et al. (2015) conducted an extensive systematic literature review and provide a quantitative evaluation and classification of model-driven security approaches. Yet, they do not focus on approaches for CPS nor approaches that take the platform into account. Overall, only 6% of the approaches they cover are in the area of CPS. In addition, in Nguyen et al. (2017) Nguyen et al. present a mapping study studying the research activities in "model-based security engineering for cyber–physical systems" (Nguyen et al., 2017). This mapping study provides a broad overview of publications in this research area. In contrast, we focus on model-driven approaches that explicitly take the platform into account. Uzunov et al. (2012) present an extensive survey in the area of security for distributed systems. Some parts of our survey are based on or inspired by this survey, e.g., we used the proposed taxonomy of model-driven engineering (cf. Section 3). In contrast to our survey, Uzunov et al. (2012) did not conduct a systematic literature review. Nevertheless, the survey by Uzunov et al. is the most extensive survey in this area and provides a very detailed description of a large set of methodologies for distributed systems. Yet, Uzunov et al. do not focus on the use of the platform for the approaches but classify them in more general categories. Kriaa et al. present a "survey of approaches combining safety and security for industrial control systems" (Kriaa et al., 2015). They show how important the combination of safety and security is, how requirements for both can contradict each other, and which approaches cope with this problem. In contrast to our survey, they focus more on risk analysis in the SDLC. Furthermore, they do not explicitly consider the physical layer or model-driven techniques. In Jensen and Jaatun (2011), Jensen et al. provide a survey with a focus on code generation in model-driven security approaches. In contrast to our survey, Jensen et al. do not consider the whole SDLC but focus on code-generation capabilities only. Felderer et al. present in Felderer et al. (2016) a survey on model-based security testing. In contrast to our survey, Felderer et al. do not explicitly focus on the needs of CPS. In addition, they focus not on model-driven engineering even though model-based security testing might be used by some model-driven design approaches. Lun et al. (2016) present a mapping study of security approaches for CPS and give a good overview of this area but do not consider model-driven approaches in detail.

In line with these surveys, we agree that security has to be considered in the whole SDLC and not only in one phase, e.g., design or implementation. Uzunov et al. (2012) state four distinct phases for security based on the SDLC. In particular, the authors define general phases for security along with the general phases of software development: *Security Requirements Determination* (along with Requirements Analysis), *Security Modeling* (along with design phase), *Implementation* (along with implementation phase), and *Configuration and Monitoring* (along with deployment phase). We agree to these phases in general but refined them into six steps for the purpose of better differentiation of the studied approaches. Fig. 1 shows these phases. In

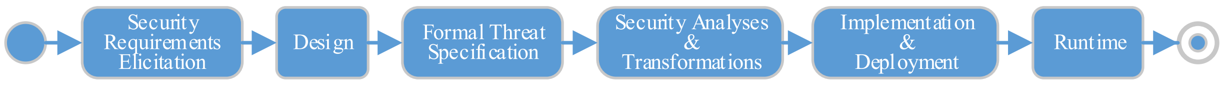
Security Requirements Elicitation
Design
Formal Threat Specification
Security Analyses & Transformations
Implementation & Deployment
Runtime

Fig. 1. Phases of the secure software development life cycle.

the first step, one specifies *security requirements*. In this step, the used artifacts are not necessarily defined formally. In the next step, one specifies the system *design*. Since we focus on model-driven approaches, formal models are used. It is important that both *cyber* and *physical layer* of the systems are considered to match the nature of CPS. Furthermore, an approach has to allow *the specification of possible threats* and/or attacks. Since this activity is crucial for model-driven security, we propose to state a dedicated phase for specifying formal threats or attacks to the system. In the best case, these threats are derived from the specified security requirements. The outcome of this step is a formal *threat model*. It is essential that the threats can be specified formally, such as to enable *Security Analyses and Transformations*. Security Analyses are used to prove or disprove certain security properties of the models, whereas transformations are used for the refinement of models or to introduce security controls into the existing models. We propose to state also a dedicated phase for this, since threat specification and security analyses might be on a different level of expressiveness, e.g., if an approach allows one to specify more kind of threats than analyses can find in the models. During the *deployment*, additional vulnerabilities might be introduced. Thus, securing the deployment (allocation of software to hardware nodes, deploying the software to concrete hardware, configuration, etc.) should also be considered. In contrast to Uzunov et al. we see the implementation coupled with the deployment, because when developing CPS, the source code often highly depends on the concrete allocation of the software. *Runtime* on the other hand should be a dedicated phase, in our opinion, since during runtime, additional security mechanisms can be applied, e.g., monitoring the system. It can be used to detect attacks or to enforce security mechanisms, by utilizing information synthesized from the system models.

In an initial literature survey, we collected requirements that a model-driven security approach should satisfy. We base our requirements on existing surveys in this area. In particular, the most influencing survey is Uzunov et al. (2012). We took an initial set of requirements from these surveys and state in the following the most important requirements. We leave out requirements regarding usability and industrial applicability since these are not in the focus of our survey but focus on requirements that are essential to platform-aware security approaches. We do not claim this list to be complete but state these requirements as the most important.

R1 *Support different layers of the system:* One key aspect of CPS is that they consist of different layers. In this survey, we distinguish two main layers: cyber layer and the platform layer. The cyber layer describes the software part of the system. The platform layer covers the whole runtime environment (as for usual IT systems) containing artifacts like operating system and middleware but also the physical part of the system, e.g. sensors and actuators. Since in this survey we focus on the use of the platform, this requirement has some significance.

R2 *Phases of the SDLC:* Security-by-Design means to consider security in all phases of the SDLC. Thus, all phases of the SDLC should be covered. First of all, the system has to be designed, e.g., by modeling the software architecture and the target platform. Furthermore, potential threats and attacks have to be specified, e.g., by a threat model. Additional phases are the application of formal analyses, (automatic) threat mitigation, deployment, or runtime analyses like monitoring or simulation. It is important that each phase is clearly defined and has specified input and output artifacts.

R3 *System of systems:* Modern CPS are complex systems that consist of several sub-systems. To handle the development of such system-of-systems, the method has to provide a structured approach, e.g., to allow compositional analyses, it has to provide the specification of a hierarchical system specification. In large systems, not all subsystems are developed by the same provider. Thus, complex systems often integrate third-party resources. Hence, in the best case, a method is able to process both fully known parts and only partially known (or even unknown) parts of the system.

R4 *Threat model:* Since the threat model defines potential threats to the system, it is important that such a model is sufficiently expressive; it should cover as many kind of threats as possible. In the best case, the threat model is extensible, allowing one to define new threats. We do not restrict the threat model to the application layer but also consider threats regarding the platform and hardware.

R5 *Formal methods and formal models:* Security-by-Design is aided by applying formal analyses and model transformations to enable consistency between all development steps. Thus, the method has to provide formal models for the system and the threats and attacks as well as formal analyses on these models.

R6 *Refinement and relation to implementation:* An approach has to provide a correct synthesis to source code, i.e., a source-code generation that preserves the analysis results, e.g., by generating secure source code for the platform. In the best case, a full code generation for the is provided but also partial code generation for the security implementations might be sufficient, e.g., if the code for secure communication is generated. Other code generations are possible, e.g., automatically generated test cases or static code analyses.

R7 *Requirements:* Typically, beside functional requirements and security requirements also non-functional requirements are essential for CPS, since CPS are often restricted by resource constraints, e.g., restricted memory or computing power. Restricted computational power and timing constraints get important properties and make the secure development of CPS more difficult, e.g., when encryption is needed. Also, such systems have to formally show that specific (hard real-time) safety-requirements are fulfilled. Hence, an approach has to provide the specification and/or verification of such non-functional requirements. In the best case, an approach has to allow the specification and verification of functional, non-functional, and security requirements.

The main question we seek to answer is to which extent existing approaches fulfill the requirements of the model-driven development of CPS. In particular, we focus on approaches for security engineering that consider in the models both software and platform. Therefore, we conducted a systematic literature review to determine (a) to which extent current approaches consider the platform of the system, (b) which requirements for the development of CPS are satisfiable by existing model-driven methods, and (c) the open research challenges.

3. Survey

To address our research questions, we conducted a systematic literature review. We next describe the organization and conduct of this review. To increase the validity of the survey and the results, we followed the guidelines proposed by Kitchenham (2004) and Kitchenham and Charters (2007). One key element to the methodology they propose is to define a survey protocol before the survey is actually conducted. This protocol defines how the survey has to be executed, "to reduce the possibility of researcher bias" (Kitchenham and Charters, 2007). During the entire survey, this protocol must be strictly followed. In Section 3.1, we describe the preparation and organization of our survey, including the survey protocol. In Section 3.2 we then describe organizational aspects during the conduct of the survey.

3.1. Preparation/organization

Following Kitchenham and Charters (2007), the survey protocol has to describe (among others) a *rationale* for the survey, *research questions*, a *search strategy*, *study selection (and exclusion) criteria*, and a *data-extraction strategy*. The rationale for our survey was already outlined in Section 2. We next describe in more detail the other parts of our survey protocol.

3.1.1. Research questions

The main questions we seek to answer are also already outlined in Section 2. For the survey protocol, we defined the questions more explicitly:

RQ1: To which extent do model-driven security approaches for CPS consider the platform? (cf. R1)

RQ2: Which of the other stated requirements (cf. R2-R7) do these approaches fulfill?

RQ3: What are current challenges and open research questions in the area of such platform-aware approaches for model-driven security?

3.1.2. Search strategy

The approaches we are looking for are covering three different areas: *Cyber–physical Systems*, *Security*, and *Model-driven Development*. To find an initial set of approaches covering all three areas we decided to apply a keyword-based title-search. For this, we defined three sets of keywords:

1. Secure, Security, Threat, Attack
2. Cyber–physical System, CPS, (Embedded AND distributed)
3. Model-Driven, MDSD, MDE, Aspect-oriented, AOM, (Method AND Model)

Using these keywords, we generated search strings to find approaches that contain at least one keyword of Set 1 and Set 2 each, or at least one keyword of Set 1 and Set 3 each. More precisely, the search string is: (1 X 2) OR (1 X 3), where X is the Cartesian product. Another possible approach would have been to simply search for publications that contain at least one keyword of each set. However, an initial search showed that too many relevant approaches would be excluded during this search. This is because the papers do not necessarily cover all three areas (depending of the research area and commonly used terms).

After that, we refined the resulting set of approaches by exclusion: first based on reading the title, second based on reading the abstract, third based on reading the full paper. Exclusion is based on clearly defined criteria explained in the next paragraph. In addition to the automatic searches in the libraries, we conducted snowballing based on the resulting set of papers. During snowballing, further papers are added manually following relevant references within the set of already selected papers. We applied forward as well as backward snowballing (Wohlin, 2014) until no additional papers were found.

We conducted our search in the digital libraries of IEEE Xplore (http://ieeexplore.ieee.org), the ACM Digital Library (http://dl.acm.org/), Springer Link (http://link.springer.com), and ScienceDirect (https://www.sciencedirect.com).

3.1.3. Exclusion criteria

We defined a set of exclusion criteria. If one of these criteria holds for an approach, this approach is excluded from the list and, to enable traceability of our decisions for later evaluation of the survey, marked with the reason for which we excluded it. We used the following exclusion criteria:

- *Unrelated Domain:* We exclude papers that describe domain-specific approaches for other, unrelated domains, e.g., cloud or web services.
- *Informal approaches:* We exclude approaches that specify threats or the system model only informally. These approaches cannot be used for MDSD because the models cannot be refined properly.
- *Not Model-driven:* In addition to the former criteria, we exclude all approaches that do not aim at the principle of MDSD. We use the definition by Uzunov et al. where we "focus on system models as first class entities" (Uzunov et al., 2012), e.g., it gets excluded when an approach considers only one model-layer and cannot be refined to more platform-specific models.
- *Only One Stage of SDLC:* We exclude approaches that cover only one stage of the SDLC, e.g., if the approach covers only threat modeling without a relation to a concrete system model.
- *Work-in-progress Papers:* Since we would like to focus on (to a certain degree) mature approaches we do not consider approaches that are considered as *work-in-progress*. This includes approaches marked explicitly as work-in-progress, but also approaches published at workshops and poster sessions only. We instead focus on approaches published in conference proceedings, journals, and book chapters. Furthermore, we exclude short papers (with fewer than five pages).
- *Language:* We exclude all papers not written in English.

3.1.4. Data extraction strategy

To give a broad overview of the selected approaches and to answer the research questions, we collected specific data for each approach. In the following, we describe in more detail which pieces of data were collected:

General Information: We extract for each approach if a method is defined, if tooling is proposed or available, how the approach is evaluated, e.g., using toy example or case studies, based on industrial scenarios, or large-scale systems. In addition, we extract for which target domain the approach is mainly developed, if possible.

Kind of Approach: We distinguish between approaches that are designed for security purposes only leaving the functional aspect of the system out, and approaches that cover both security and functional modeling. In the latter case, an approach provides

Table 1
Platform specificity of the selected approaches.

Approach	General	PS	CPS
SecureUML (Basin, 2006)	✓		
UMLsec (Jürjens, 2005)	✓		
SECTET (Hafner et al., 2006)	✓		
ModelSec (Sánchez et al., 2009)	✓		
Motii (2017)	✓		
Security4UML (Neri et al., 2013)	✓		
ISSEP (Ruiz et al., 2015)	✓		
SecureMDD (Moebius et al., 2009)		✓	
Security-enhanced SPACE (Gunawan et al., 2011)		✓	
Neureiter et al. (2016)		✓	
DREMS (Levendovszky et al., 2014)			✓
ProCom (Saadatmand and Leveque, 2012)			✓
Wasicek et al. (2014)			✓
Al Faruque et al. (2015)			✓
Eby et al. (2007)			✓
SysML-Sec (Li et al., 2018)			✓
SEED (Vasilevskaya, 2015)			✓

model elements for both functional system elements and dedicated security elements. As a special case of this and, therefore, a third case, we see approaches that integrate security features into an existing functional approach, i.e., where security solutions are integrated into the system model such that the model still conforms to the functional meta-model (based on R1 and R5).

Stages of the SDLC: We extract which phase of the SDLC (cf. Fig. 1) is covered. We distinguish between fully covered, not covered, and partially covered (based on R2 and R7).

Models for System Design: If the system model is to be refined to a security-aware system model, one requires mature functional models. Thus, we also extract how the system model is defined. In particular we distinguish in this survey between three kinds of models: architectural models describing the structure of the system, e.g., component diagrams, behavioral models describing the functional behavior of the system, e.g. state machines, and platform models describing the platform of the system (based on R1, R3, R4, R5 and R7).

Formal Threat Specification: Additionally, if model-driven methods are used for (automatic) refinement then one requires formal threat models. Therefore, we extract which threats and attacks can be formally specified in the approaches. In particular, we extract the used modeling language, concrete kind of threats and attacks (based on R4 and R5), and the focus. Corresponding to Shostack (2014a), we distinguish between three different kinds: attacker-centric, system-centric, asset-centric. Attacker-centric approaches focus on concrete attack and threat specification whereas system-centric and asset-centric approaches focus more on security properties. System-centric approaches use system models to find potential threats, while asset-centric approaches focus on concrete assets and potential threats.

Security Transformations & Analyses: Approaches for model-driven security provide transformations for refinement of the models or analyses, e.g., to find possible design flaws. We extract for each approach which kind of transformations and analyses are provided (based on R6).

3.2. Conducting the survey

We next give insights to the *conduct* of the survey. The survey was conducted in February 2019. Content-related information is discussed in the data-elicitation described in Section 4. First, we give an overview of the number of papers and approaches after each exclusion stage. Then we state threats to validity of the survey.

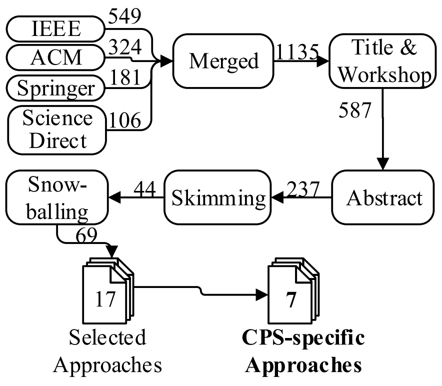

Fig. 2. Exclusion steps and number of publications after each step.

3.2.1. Selection of approaches

In the following, we shortly explain the numbers of papers after each exclusion stage. Fig. 2 shows the number of publications after each exclusion step. The title-based search resulted in a set of **1160** publications, nearly 50 percent of them in IEEE digital library. Merging the search results and eliminating duplicates resulted in a set of **1135** publications. The identification of duplicates was aided by tool support of Mendeley Mendeley Ltd. (2020). In the next step, we excluded publications based on their title. Additionally, we removed all workshop papers and, thereby, reduced the number of papers to **587**. Next, we excluded papers based on the abstract. In this step, we excluded many approaches that do not follow the model-driven paradigm, use informal models, or were not applicable for CPS. Here, we reduced the set to 50 percent of the publications that remained at this stage. Finally, we evaluated all remaining **237** publications and reduced the number of relevant publications to **44** by applying our exclusion criteria. We followed relevant references of these publications ("snowballing"), resulting in a final set of **69** publications. Those 69 publications describe a set of **17** relevant approaches. We classified these 17 approaches into three categories, based on how CPS-specific they are. In particular, we distinguish between *General Approaches* that consider the platform in an abstract way, *Platform-specific Approaches* (PS) that explicitly take the platform into account, but do not focus on CPS, and *CPS-specific Approaches* that explicitly focus on model-driven development for CPS that also take the platform into account. Table 1 shows the classification for all 17 approaches and one main reference per approach. In this survey, we focus on these *CPS-specific* approaches. Thus, we got a final set of **7** relevant approaches. A list of all surveyed papers can be found in our replication package Geismann and Bodden (2020).

3.2.2. Threats to validity

In the following we discuss known threats to validity and how we tried to mitigated them. We use the classification scheme for validity by Runeson and Höst (2009) and, therefore, distinguish between *Construct Validity*, *Internal Validity*, *External Validity*, and *Reliability*. For each classification, we discuss potential threats shortly.

Construct validity. Approaches on model-driven security for CPS come from different research domains, which is reflected in our keywords. To mitigate the threat of missing some approaches, we tried to keep the keywords not too restrictive. However, we still might have missed some relevant publications because they used other terminology, e.g., *IoT* instead of *CPS*, or were published in other libraries not included by our search. To mitigate this, we applied forward and backward snowballing to find approaches that are not covered by our keywords or were not contained in the searched libraries. All approaches found during the snowballing were studied based on the same search protocol and, therefore, may have been excluded step-by-step based on the exclusion criteria. Also, it is often not completely clear whether an approach fits into a category, e.g., if it belongs to the domain of CPS or if it is a model-driven approach. To mitigate this, each approach that got marked to be excluded, was cross-checked by at least one additional researcher. In addition, in case of doubts, we discussed these cases with members of our research group.

Internal validity. For searching, we used the online search engines of each digital library. Since the search algorithms are not publicly available, the correctness of the search results cannot be validated. To make the results more comprehensible, we split the search string into a single search string per keyword combination and checked each result set for the keywords.

External validity. Excluding all papers that cover only one stage of the SDLC might exclude approaches that cover several stages of the SDLC but distribute their contributions to several papers. Also, workshop papers and work-in-progress papers may refer to other, relevant papers. To cover this problem, we marked these papers as *Conditionally Excluded* and checked this list at the end of the survey again making sure that approaches distributed over several papers were included. We also applied snowballing ensuring that relevant references were taken into account.

Reliability. It is important to make the survey reproducible. We provide all relevant information of our goals and context in Section 2 as well as information of the search protocol in Section 3. For the purpose of replicability and transparency, we provide an archive containing the survey protocol, the search results for each library, a list documenting all papers and their inclusion and exclusion, and the data extraction template Geismann and Bodden (2020). However, as discussed for *construct validity*, exclusion of approaches might be affected by authors bias. We believe that our clear protocol and defined exclusion criteria can help to mitigate this threat.

4. Selected approaches and data elicitation

In this section, we describe the data elicitation for the seven selected approaches. This includes a description of each approach and a summary of the collected data. A discussion of the data collection and of the approaches is presented in Section 4.2.

4.1. Summary of approaches

In this section, we discuss approaches that explicitly aim at security for CPS. We discuss seven approaches, which vary in their focus and expressiveness. There are on the one hand approaches that extend existing model-driven approaches for safety-critical CPS by certain security properties like ProCom or DREMS. On the other hand, there are approaches that focus on security issues only. For each approach, we give a short overview of the approach, which stages of the SDLC are addressed and which security analyses can be applied.

4.1.1. DREMS

DREMS Balasubramanian et al. (2015) is a model-driven approach for distributed embedded systems, e.g., "cloud computing platforms built from mobile embedded devices" (Levendovszky et al., 2014). One focus is the generation of a middleware layer to ensure the correct communication and the abstraction of the subsequent deployment at design time. Here, one aspect is the secure communication between all parts of the system. Even if this might not be the main focus of DREMS, this aspect (and other security features) makes this approach relevant for this survey. DREMS aims to guarantee secure information flows within the system and secure deployment of the applications. The concepts are based on the *OMG Common Object Request Broker Architecture (CORBA)* and is implemented for C/C++ projects. The approach got evaluated on real hardware nodes by emulating the communication between three satellites.

Stages of the SDLC. DREMS covers all stages except *Requirements* which are described in the following.

System Design: The system is described component-based. When instantiating a system, components are assigned to so-called *actors*. There are two different actor types: application actors that execute the components, and system actors that provide system services. An actor represents an OS process and executes either the behavior of the components or the system services. The behavior of actors is not specified by models but has to be implemented manually (against a generated interface). Actors are deployed to hardware nodes that can also be specified as a model. Using the DREMS OS, meeting security requirements of actors is guaranteed, e.g., by addressing them in the scheduling.

Formal Threat Specification: Instead of specifying concrete threats or attacks, DREMS provides a security property specification based on the systems description. One key part of security in DREMS is a "multi-level security" policy that is based on a label-oriented modeling approach to ensure that actors cannot access data that they are not supposed to. In contrast to usual label-based approaches, DREMS uses so-called *multidomain labels*. Each label has 1 to n *domains*, which allows specifying not only different applications but also applications of different providers, e.g., companies. Each domain declares 1 to n labels with identifiers. The labels describe security levels and are ordered by a domination relation. They can be used to label ports, actors, and hardware modules. Hence, DREMS provide a system-centric threat specification.

Security Analyses and Transformations: DREMS provides analyses to check if all security requirements are fulfilled in the given design. In particular, the authors provide an automatic static analysis to verify that all information flows in the systems conform to the specified multi-level security policies.

Deployment: In addition to the already generated code-skeletons for the software components, source code for parts of the software as well as descriptive deployment plans are generated. One key advantage of DREMS is the automatic generation of a middleware. The middleware is used to ensure the correct execution of the specified system with regard to the model specification. This middleware is used at runtime to ensure security assumptions made in the model.

Runtime: In addition to the functional implementation (e.g., correct communication, scheduling, etc.) also security-related functionality like the multi-level label security policy is enforced by the middleware. Hence, assumptions that are made on the results of the label analysis also hold for the system at runtime. Additionally, DREMS allows separating actors at runtime. Each actor runs in a separate memory part, e.g., to prevent memory attacks between two distinct organizations. Additionally, DREMS uses temporal separation which ensures that predefined actors are never scheduled on the same electronic control unit (ECU) at the same time. Thus, attacks to the data of actors that share a CPU are mitigated.

Discussion. DREMS utilizes formal models, model analyses and code generation. One big advantage is the provided middleware that can handle functional tasks but also enforce security policies during runtime. Noteworthy is that DREMS is the only approach of the selected approaches providing security controls at runtime that enforces specific security properties. It also provides – in contrast to the other approaches – generated deployment plans to reduce human mistakes during deployment, which is in our opinion an important feature to secure the complete SDLC. DREMS provides a method to specify security levels in the system. Unfortunately, behavioral models are not provided. Instead, manually implemented source code is integrated. Hence, behavior of the actors cannot be analyzed for possible security threats but only the static (structural) system information is taken into account. However, DREMS utilizes model-driven techniques to increase the use of security in the SDLC, especially in the final steps Deployment and Runtime.

4.1.2. ProCom

In Saadatmand and Leveque (2012), Saadatmand et al. present an approach to integrate security features in the existing model-driven engineering approach ProCom (Bureš et al., 2008) which focuses on the development of real-time systems in automotive and telecommunication domains. The authors present an approach that enables one to explicitly specify security issues regarding *Authentication* and *Confidentiality*. It was integrated into the ProCom development tooling PRIDE (Borde et al., 2011) and evaluated on a academic case study from the automotive domain. (Saadatmand and Leveque, 2012)

Stages of the SDLC. The presented approach covers all stages of the SDLC except *Requirements* and *Runtime*.

System Design: ProCom distinguishes between two layers for the system component specification: ProSys and ProSave. ProSys is the top layer and describes concurrently running subsystems that can communicate asynchronously. ProSave is used to describe each of these systems in a component-based way. Analyses can be used to show that specific real-time requirements are fulfilled for a given system model. The developer has to specify a software component model, a data model, and a platform model that describes ECUs and their communication channels. Based on this, a mapping of software components to ECUs and software component communication to concrete hardware communication channels is specified.

Formal Threat Specification: In the second step, the data model, as well as the platform model, get annotated by *security annotations*. Thus, the threat model is asset-centric as well as system-centric. After the system design, the designer annotates data in the data model with required security properties for confidentiality and authentication. In the platform model of the system, each physical subsystem can be annotated as (physically) accessible by possible attackers or as physical secure. Similar to this, communication channels can be annotated as secure or not.

Security Analyses and Transformations: Based on the security annotations, software component communication is determined that needs to be secured, e.g., when components that share confidential data are deployed to different subsystems. Additionally, for each communication that has to be secured, an applicable security mechanism is determined. After that, for each security mechanism, two additional ProSave components are generated: One for encryption of the data at the sender component and one for the decryption at the receiver component. Since ProCom focuses on real-time systems, a worst case execution time for each algorithm is provided to determine an encryption algorithm that fits best into the real-time schedule of the system. For this step, the developer can choose from different strategies to select the most suitable algorithm, e.g., providing strongest security that is still schedulable by the system. The result is a ProCom model whose component model is enriched by security components for encrypting sensitive data.

Deployment: ProCom itself provides code generation that allows to generate parts of the source code for the target platform (Borde and Carlson, 2011). Thus, code generation for the functional part of the system exists. However, code generation for the proposed encryption components and decryption components is left for future work (Saadatmand and Leveque, 2012).

Discussion. The presented approach integrates security into the existing ProCom approach for engineering CPS. Since it utilizes existing model elements and provides runtime properties (like execution times and memory size) for security solutions, it demonstrates how basic security controls can be integrated into model-driven engineering for systems that have restricted resources and hard real-time requirements. One advantage of this approach is that the original models are annotated and then transformed automatically into a security-aware system, which still conforms to the original meta-model. Consequently, this model can be used for further analyses of the original approach. Based on simple security assumptions, only threats against authentication and confidentiality can be analyzed. Also the provided encryption techniques should be further investigated, e.g., how they can be adopted for different execution platforms. The approach assumes that asymmetric cryptographic keys for encryption and certificates are used but key management is not covered and should be further investigated.

4.1.3. Wasicek et al.

In Wasicek et al. (2014), the authors present an aspect-oriented approach for model-driven development of CPSs. A main goal of the approach is to enrich model-driven design methods by security features. Since this approach is aspect-oriented, the functional models of the system are not affected. In particular, attack models are specified and associated with the communication between software components of the functional model. The approach “aims at revealing potential vulnerabilities that manifest as malicious design and interaction faults” (Wasicek et al., 2014). It was evaluated based on a case study in the context of the automotive domain describing the scenario of adaptive cruise control. For the evaluation Wasicek et al. use Ptolemy (Eker et al., 2003) environment instead of describing a new modeling environment for their approach.

Stages of the SDLC. The authors propose four steps for the approach: 1. design, where the system is designed, 2. annotate, where attack models are specified, 3. analyze, where the system design is analyzed for effects of the specified attacks, and 4. synthesize, where an implementation for detecting or dismissing attacks is devised. According to our taxonomy, these phases can be mapped to *System Design*, *Formal Threat Specification*, *Security Analyses and Transformations*, and *Deployment* and are described in the following.

System Design: Conceptually, Wasicek et al. (2014) is not restricted to one system specification meta-model. However, the approach is evaluated using models defined with Ptolemy II (Ptolemaeus, 2014). In Ptolemy, the system is composed of several actors that are executed concurrently and can communicate via specified ports.

Formal Threat Specification: The threat specification in the presented approach is attacker-centric because general attacks to the system are modeled. Four different attacks are modeled as aspects in Ptolemy: A *FuzzAttack* to insert random data into a connection, an *InterruptionAttack* to prevent communication, a *ManInTheMiddleAttack* to eavesdrop the communication, and a *ReplayAttack* to re-transmit known data.

Security Analyses and Transformations: Following the paradigm of aspect-oriented modeling, these attacks are woven into the system specification during a simulation to monitor how each attack can affect the system by plotting the values for each incoming and outgoing data value.

Deployment: According to the approach, in the last step, an implementation for detecting or dismissing attacks has to be developed. Even if the authors state that this step is also highly applicable for automation, it is not further discussed in the approach. Since there are approaches for the automatic code generation from Ptolemy models (Zhou et al., 2007), also code for the specified attack patterns can be generated.

Discussion. This approach is one of two aspect-oriented approaches and shows how model-driven techniques can be used to harden CPS against effects of known attacks. Since aspect-oriented modeling supports the separation of concerns to a high degree, system modeling and security modeling can be easily split and be done by different modeling experts. An advantage is that the functional models also cover the system's behavior and, therefore, allow for precise analyses. In this approach, concrete attacks are modeled instead of possible threat categories. Since the attack patterns injected into the system models can be specified in the same language, the security modeling expert also needs high domain knowledge of the system models. Finding effects of attacks in the model is a big advantage. However, support for finding threats or assets in general could make this approach more powerful and would enable to find unknown attacks to the system. Also the (automatic) integration of security solutions or countermeasures would increase the expressiveness of this approach.

4.1.4. Al Faruque et al.

In Al Faruque et al. (2015), Al Faruque et al. present a framework for designing secure CPS and take the "properties of the underlying computation and communication platforms" (Al Faruque et al., 2015) into account explicitly. The main focus is to find security flaws and impact of security solutions on a given CPS represented by a functional model. For this, the approach covers steps for system design, attack specification, and security analyses. The implementation is done in the commercial design and simulation tools Amesim and Matlab/Simulink. The approach is evaluated on a real-world scenario of a ground robot steered by an attack-resilient cruise control system.

Stages of the SDLC. The approach is based on previous work on functional modeling for CPS in the automotive domain (Wan et al., 2017) and covers the phases *System Design*, *Formal Threat Specification*, *Security Analyses and Transformations*, and *Deployment*.

System Design: The presented approach is integrated into an existing approach on model-driven engineering for CPS using functional models (Wan et al., 2015b). In functional modeling, the cyber layer as well as the platform layer are represented as functions that are formal model representations of software and platform parts of the system under design (Canedo et al., 2014).

Formal Threat Specification: Correspondingly, the main idea in the approach is to specify two different types of attacks as functions, too: cyber attacks and physical attacks. The approach does not provide a dedicated modeling language or specification for attacks but provides a library of six common attacks to CPS and is, therefore, attacker-centric. In particular, functions for the following attacks are modeled: *fuzzy*, *interruption*, *man-in-the-middle*, *replay*, *overflow*, and *down-sampling*. Since the attacks are also modeled as functions, the library can be extended by using the same language as for the system specification.

Security Analyses and Transformations: After applying the attack functions to flows of the functional model, the model can be simulated using a co-simulation of Matlab/Simulink and Amesim where a physical model of the system and an environment model is used. The results can be analyzed by the system designer by "using standard time series and plotting capabilities of the simulation software" (Wan et al., 2015a). Hence, the applied analyses are not security-specific but general analyses that are used for security purposes. If security flaws in functions are detected, the functional model can be refined. Using this approach, the system can be analyzed for effects of security functions on the behavior before the system is implemented.

Deployment: In Al Faruque et al. (2015), code generation is proposed for later stages of the development. Since code generation for simulation environments is already provided, generation for a concrete target platform is possible. Further support for deployment is not discussed.

Discussion. Similar to Wasicek et al. the presented approach enables developers to analyze effects of known attacks to a specified system. The advantage is that precise effects of the attacks can be measured and analyzed on the level of the physical layer. A major drawback is that only known attacks are considered and knowledge about the attacks and about the domain is needed. A systematic threat modeling approach could help to determine where attacks should be placed instead of defining it manually to reduce the analysis space.

4.1.5. Eby et al.

Eby et al. (2007) present an approach for integrating model-driven security concepts into existing approaches for embedded systems. The general idea of this approach is to specify a security-related DSL describing security aspects of embedded systems. This DSL can then be applied to existing modeling approaches. The approach is validated on a small example case study of distributed system.

Stages of the SDLC. The approach covers *System Design*, *Formal Threat Specification*, and *Security Analyses and Transformations*. Since this approach is applied to an existing one, further stages of the SDLC might be covered by the target language.

System Design: For applying the approach in Eby et al. (2007), the system design takes place in a method or DSL of the designers' choice. Eby et al. extend the system design by introducing the *Security Analysis Language (SAL)* which enables the designer to apply two types of analyses for access control policies: *information flow analysis* and *threat model analysis*. SAL defines so-called partitions, which represent parts of the system that can communicate via ports. Each partition has an attribute for a security level. Additionally, every data object of the system is marked if it has to provide integrity and/or confidentiality. Hence, SAL describes systems in a very generic way from the security point of view.

Formal Threat Specification: For information flow analysis, the authors use a combination of two security models that are used to define access control policies: Bell-LaPadula (Bell and LaPadula, 1973) which aims at confidentiality, and the Biba model (Biba, 1977) which aims at integrity. Hence, Eby et al. do not provide a specification language for threats but base the analysis on well-established security models for access control policies. Additionally, SAL provides an attacker-centric threat modeling technique to detect possible man-in-the-middle-attacks. For this, different attacker types and encryption algorithms are specified. Each attacker defines which encryption algorithm can be cracked.

Security Analyses and Transformations: Before analyses can be applied, SAL has to be applied to an existing modeling approach. This transformation is in our opinion a transformation for security purposes and is not part of the system design since it enriches a completed system design with security-relevant information before analyses take place. The main idea is to transform an existing modeling language into a new, security-aware modeling language. For this, concepts for information flow, partitions, and annotations for security attributes are created in the source language. Since the concepts for information flow and partitions are quite generic, corresponding elements can be found in most modeling languages. After that, the SAL-based analyses can be applied. Based on this, analyses for information flow can be applied. In particular, the analysis checks if both security models are fulfilled and gives the designer feedback if a property is violated. Additionally, the designed system can be analyzed for possible man-in-the-middle-attacks. For this, every information flow in the SAL model is annotated with an attacker. Then, the security information for each attacked information flow are analyzed regarding possible violations of the security properties. The result reports possible eavesdropping of messages in the system.

Discussion. By providing the general purpose security language SAL, this approach provides basic security controls and threat analyses for model-driven approaches that do not focus on security aspects yet. Due to its quite general elements, SAL is applicable to model-driven approaches that provide architectural models. However, a drawback is that the concepts are too general and, therefore, are restricted to basic (general) threats and attacks. For more complex threats, the mapping to the SAL dialect might also become a complex task. However, this approach shows that several security threats can be handled on a more abstract level and how model-driven security can be integrated into existing approaches. During the mapping of SAL to the existing approach, model element IDs are also mapped. Thus, feedback of the analyses can refer to the original model elements (Eby et al., 2007) and, therefore, the engineer does not need any knowledge about SAL or the analyses themselves.

4.1.6. SysML-Sec

SysML-Sec (Apvrille and Roudier, 2013; Li, 2018) is a model-driven engineering approach that "aim[s] at fostering the collaboration between system designers and security experts" (Apvrille and Roudier, 2014) in all phases of the SDLC for the development of embedded systems. It is based on SysML (Weilkiens, 2011) and provides customized SysML diagrams to describe security-related parts of the system and a methodology for a systematic development, which also includes formal security analyses and validation of the model. It aims to close the gap between safety and security modeling and explicitly focuses on security issues coupled with the co-design of hardware and software. Starting with (security) requirements engineering, the approach considers refinement steps to a full system design, design validation, and generation of C code for the target system. SysML-Sec is evaluated in several quite detailed case studies from different domains, e.g., automotive domain or UAVs. The evaluation scenarios are real-world examples and since SysML-Sec was initially developed and validated within the EVITA project (EVITA, 2020), the automotive domain seems to be the main target domain.

Stages of the SDLC. SysML-Sec follows a three-phase approach, namely *system analysis*, *software design*, and the *verification phase*. In the following, we give a short overview how these steps are distributed over the phases of the SDLC defined in Fig. 1.

Requirements: SysML-Sec extends SysML (Weilkiens, 2011) requirement diagrams for specifying security requirements. In particular, affected security properties are added to the requirement description explicitly, e.g., confidentiality, access control, or integrity. In addition, attack graphs (Apvrille and Roudier, 2015) are used as threat modeling technique and describe a set of attacks and their dependencies. Like in attack trees (Schneier, 1999), dependencies like conjunction, disjunction, and sequence between attacks but also temporal operators are allowed. Attacks can also refer to specific security requirements created earlier. In Roudier et al. (2014) and Idrees (2012), the requirements stage is improved by adding an approach to formalize the requirements and by introducing the use of ontologies for security goals, system architecture, attacks, security requirements, and security mechanisms which can be used to annotate SysML-Sec model elements to refine the description, e.g., a specific attack in an attack graph.

System Design: Secondly, the software design takes place where concrete security mechanisms are defined. Related to our taxonomy, this step can be split up into at least two sub-steps: the actual design of the system and the formal specification of threats or attacks respectively which is described in the next paragraph. SysML-Sec follows the Y-chart approach (Kienhuis et al., 2002) at first and the V-Cycle for the software design (Roudier and Apvrille, 2015) in a second step. First of all, the system's structure and main functionality are described using SysML block diagrams. Additionally, the platform of the system is described. Then, following the Y-chart approach, functions of the system are mapped to specific hardware parts, which is called *mapping*. This step is crucial and requires expertise in different fields, since it can affect performance, security, and safety of the system. Analyses can be applied to find possible flaws in the mapping. Also the effect of applying security mechanisms on the functional parts of the systems can be analyzed. SysML-Sec also provides a technique to generate modeling solutions for secure communication automatically (Li et al., 2017). Once a feasible mapping is found, the *System Design* takes place. Here, the system building blocks are described by SysML models in more detail, e.g., the behavior of blocks is specified. In particular, the software functions of the system are refined, i.e., functions are specialized until automatic code generation is applicable.

Formal Threat Specification: SysML-Sec provides several extensions for general SysML models enabling formal threat specification. While in the requirements phase all possible attacks are considered, here only confidentiality and authenticity of message exchange is analyzed. SysML-Sec bases on a Dolev–Yao attacker model (Dolev and Yao, 1983) in which only messages exchanged between two entities can be eavesdropped by an attacker. Hence, the threat specification in SysML-Sec is system-centric. In case of SysML-Sec, these entities are SysML blocks. Using this attacker model, it is possible to describe direct attacks on the communication protocols. Attacks on the platform itself or attack sequences against several components are not considered. For the purpose of modeling security, three extensions are provided: 1. Communication channels can be annotated as to whether message eavesdropping is possible, 2. Blocks are extended by a set of methods to describe cryptographic algorithms, 3. The developer can define knowledge of specific blocks, e.g., shared keys.

Security Analyses and Transformations: During the different development phases, SysML-Sec provides several analyses to assist the developer and to find possible flaws regarding security and safety as early as possible. The mapping models resulting from the partitioning step can be analyzed regarding performance issues, e.g., load of CPUs and buses. Based on the specified security properties, the design models can be analyzed for specific safety and security properties using UPPAAL (Behrmann et al., 2004) and ProVerif (Roudier and Apvrille, 2015; Lugou et al., 2016). Furthermore, liveness and reachability of crypto-protocols and the impact of security mechanisms on safety-critical processes can be analyzed (Li et al., 2017). If properties cannot be proven on model level, and to harden the trust into the system, test-code for safety and security tests can be generated from the design models automatically.

Deployment: During the partitioning, the allocation of software to hardware nodes is already taken into account. During the deployment step, more concrete measurements and techniques can be applied. It is possible to compare different architectures using simulation environments, e.g., regarding bus loads and the resulting impact of key distribution at runtime (Roudier and Apvrille, 2015). Based on the fine-granular behavioral models, model-to-code transformations are proposed (Apvrille et al., 2016).

Discussion. SysML-Sec is one of the most mature approaches found in our survey. It provides a whole methodology and provides a process, languages, and tooling. Since in the last years several papers were published, it can be seen as actively researched and maintained. These papers cover model-driven security concepts and analyses in every stage of the SDLC. It is the only approach that provides a dedicated security requirements phase. One major advantage of this approach is that it allows for a very early analysis of effects on the system when integrating security controls. One use-case that is very important for CPS is a performance and resource analysis. Comparing system designs (with and without security features) enables system designers to estimate the effects of adding security to the system. Since behavioral models are provided, also behavioral aspects can be handled in the analyses. However, while all possible attacks are considered in the requirements phase, in subsequent formal phases only confidentiality and authenticity of message exchange is analyzed. If models get too complex for verification, generated tests can be used to test for specific security or safety properties on code level. One drawback is the lack of separation of concerns like in Eby et al. because currently domain knowledge, modeling expertise, and security knowledge is needed to apply this method.

4.1.7. SEED

Vasilevskaya et al. (2014) present an approach for Security-Enhanced Embedded system Design (SEED). The main goal of the approach is to “bridge the gap between [...] embedded system and security experts” (Vasilevskaya et al., 2014). SEED is a model-oriented, domain-specific approach that explicitly focuses on separation of responsibilities and concerns. It does not propose new modeling languages for the system design itself, but “increments [...] existing practices and processes” (Vasilevskaya, 2015). The approach proposes a general process which can be applied for various modeling languages (called *SEED foundations*). Vasilevskaya et al. use SPACE and MARTE for the system specification in their concrete implementation (called *SEED realization*). The approach is evaluated on several case studies including real-world scenarios.

Stages of the SDLC. SEED does not follow exactly the SDLC steps outlined by our survey, since the two essential steps for functional modeling and defining security solutions are executed in parallel. In general, these steps can be mapped to our SDLC steps. The approach does not contain an explicit threat specification but focuses on the specification of assets and corresponding protection goals. All other steps are (at least partially) covered by SEED.

System Design: SEED bases on a common systems design approach for embedded systems. Firstly, a functional model is created that describes the software of the system. Since SPACE is used for system specification, the system is described by so-called building blocks. Building blocks can either be chosen from a library or created by the system designer. Thus, a building block describes a specific part of the system, and can be decomposed into subsystems. Furthermore, the interaction of the subsystems and the behavior is described in a model-driven way by using activity diagrams and state machines. Secondly, an architectural model is created that describes the execution platform of the system. In SEED realization, the UML profile MARTE (Object Management Group, 2008) is used. Thirdly, the functional model is mapped to the architectural model. These artifacts (functional model, architectural model, and mapping) are the basis for the SEED approach (Vasilevskaya, 2015).

Formal Threat Specification: SEED does not focus on threat modeling or attack specifications but on making decisions for security solutions. For finding and defining possible threats (implicitly), two key parts are used: an asset elicitation and an application of security knowledge. The asset elicitation is based on formal rules and can be done automatically for the system model and focuses on confidentiality and integrity of the assets. Hence, the threat modeling of SEED can be seen as asset-centric. Besides that, SEED proposes to reuse existing security knowledge. For this, an approach for creating and storing security knowledge for embedded systems is provided. Security experts can use a UML class model to describe security solutions that can be transformed into an ontology representation based on a general security ontology by Herzog et al. (2007). In essence, the proposed ontology covers security goals and specific security solutions to fulfill the security goals for assets. Noteworthy is that each security solution refers to a concrete domain. Hence, SEED allows specifying domain-specific security solutions. Thus, performance analyses for security solutions can be applied before selecting them. In SEED realization, SPACE building blocks are used to describe concrete security solutions.

Security Analyses and Transformations: One key idea in SEED is to use analyses to support the engineers when selecting appropriate security solutions. Performance analyses for the security solutions can be applied before or during the selection of security solutions for the system. Based on predefined attacks, an approach for risk assessment can be applied to elicit which (secured) system design is the best with regard to costs and efforts. Since embedded systems often have restrictive resource constraints, adding additional functionality for security might lead to problems. Based on the assets, security experts can choose concrete security solutions from the security ontology. Since the solutions are domain-specific as well as provide a behavior description and resource consumption, analyses can be applied to check if the solution is applicable to the system or not.

Deployment: Since in SEED, a system model (containing software and hardware) and a specific set of security solutions are synthesized to a new security-aware system model, the deployment depends on the used modeling language. In case of SEED realization, the security-aware system model is still described by SPACE building blocks. Thus, existing code generation and deployment methods of SPACE are still applicable.

Discussion. Similar to SysML-Sec, SEED is a method and provides a full process, language, and tool support. A big advantage from the research point of view is the clear separation of SEED foundation and SEED realization. This separation makes an adaption of the concepts to other model-driven approaches possible. An additional advantage are the provided and reusable platform-specific security solutions that can be chosen from a library. It supports separation of concerns since these solutions are developed by security experts for each platform and can be easily integrated into the system models. However, a higher degree of automation would support the engineers by finding unknown threats and attacks to the system.

4.2. Classifications and results

In this section, we summarize the elicited data and classify the selected approaches based on the aspects presented in Section 3.1.4.

Table 2
General information of the selected approaches.

Approach	Main domain	Method	Tooling	Evaluation
DREMS	Distributed Embedded Systems	✓	(✓)	Industry/Realworld
SysML-Sec	Automotive & Telecommunications	✓	✓	Industry/Realworld
ProCom	Automotive & Telecommunications	✓	(✓)	Case study
Wasicek et al.	Automotive	(✓)	(✓)	Case study
Al Faruque et al.	(Automotive)	(✓)	(✓)	Industry/Realworld
Eby et al.	Embedded systems	(✓)	(✓)	Case study
SEED	Distributed Embedded Systems	✓	✓	Industry, Case study

4.2.1. General

All selected approaches are developed for the area of CPS but have a different main focus in this area. Four of the approaches stated explicitly to be targeting the automotive domain. Eby et al. in contrast, explicitly focus on extending existing model-driven approaches and, therefore, do not have an explicit target domain but adopt the target domain of the extended approach. However, all approaches are targeting in general CPS and, after studying the approaches it appears that at least the concepts in general of all of them are adaptable to most of the CPS domain to our opinion. Table 2 shows an overview of the (main) domain of all approaches as well as information regarding if a method is provided in the approach, a tool is provided and how the approaches were evaluated. The information is based on publications of the authors. We did not evaluate the tooling itself nor we reproduced the evaluation results.

All approaches provide are at least a structured procedure and aim at seamless development. However, DREMS, SysML-Sec, ProCom, and SEED provide a methodology including defined process steps and artifacts and provide therefore a more mature methodology. The fact that only SEED and SysML-Sec cover the specification of requirements explicitly (cf. Section 4.2.3), indicate that both approaches seem to be more comprehensive in this regard than the other selected approaches. Furthermore, all approaches describe beside the proposed method also a tooling. Tools for SysML-Sec, ProCom, and SEED were at least accessible to us.

The evaluation of the approaches vary. Some approaches were evaluated on toy examples or fictional *Case Studies*, e.g., ProCom. Most of the approaches claim to be evaluated on industrial or real-world scenarios. Successful industrial application (production use not case studies) is not described for any of the approaches. The most mature approaches seems to be SysML-Sec and SEED since both provide a fully defined method and mature evaluation scenarios.

4.2.2. Kind of approach

We classify the approaches into three categories as described in Section 3.1.4. Table 3 shows the classification for the selected approaches. We distinguish between approaches that consider security only, approaches that consider both security and non-security properties (Func./Sec.), and approaches that integrate security concepts into existing approaches (Sec. → Func.).

Two approaches can be classified as *security-only approaches*. Both are aspect-oriented. Due to the nature of aspect-oriented modeling, both approaches are used to model security without taking functional models into account. In Eby et al. aspects are used in the end to inject security solutions into existing functional models. Thus, this approach can also be classified into the third category to some extent. Al Faruque et al. SysML-Sec, and DREMS consider both functional and security features during the specification. Al Faruque et al. use functional models to specify attacks and use them to simulate the harm to specific functions of the system. In the other two approaches, specific modeling elements or annotations are used to enrich the system models for security purposes. We classify ProCom and SEED as approaches that

Table 3
Security category of the selected approaches.

Approach	Sec. only	Func./Sec.	Sec.→Func.
DREMS		✓	
SysML-Sec		✓	
ProCom			✓
Wasicek et al.	✓		
Al Faruque et al.		✓	
Eby et al.	✓		(✓)
SEED			✓

Table 4
Covered phases of the software development lifecycle.

Approach	RQ	DE	TM	AN	DP	RT
DREMS	–	✓	✓	✓	✓	✓
SysML-Sec	✓	✓	✓	✓	✓	–
ProCom	–	✓	✓	✓	✓	–
Wasicek et al.	–	✓	✓	✓	✓	–
Al Faruque et al.	–	✓	✓	✓	✓	–
Eby et al.	–	✓	✓	✓	–	–
SEED	(✓)	✓	✓	✓	✓	–

Table 5
Covered modeling views for system design.

Approach	Architecture	Behavior	Platform
DREMS	✓	–	✓
SysML-Sec	✓	✓	✓
ProCom	✓	–	✓
Wasicek et al.	✓	(✓)	✓
Al Faruque et al.	✓	(✓)	✓
Eby et al.	✓	–	✓
SEED	✓	✓	✓

integrate security into the functional system model. Here, also security annotations are used to express security assumptions and requirements for the model elements. However, in these approaches, transformations are used to transform the functional model into a security-aware model that still corresponds to the original meta-model.

4.2.3. Stages of the SDLC

In the following, we detail which phases of the SDLC are covered by the selected approaches. Table 4 shows the covered phases per approach. Only SysML-Sec and SEED cover requirements elicitation (RQ). In this phase, SEED only supports the developer by the automated search for assets and corresponding security properties.

SysML-Sec provides methods and tools for specifying both functional and non-functional requirements. In addition, SysML-Sec allows specifying security requirements and attack graphs that can be analyzed formally. All selected approaches provide models for specifying the system under development at design (DE). A special case is the approach by Eby et al. because it does not address the system specification but the composition of an arbitrary DSL and the proposed security extension SAL. All approaches provide models and techniques to formally specify threats or attacks to the system (TM). Additionally, all approaches

Table 6
Classification of formal threat modeling.

Approach	Language	Focus	Kind
DREMS	Multi-level labels	System-centric	Confidentiality, integrity
SysML-Sec	Model annotations	System-centric	Authenticity, confidentiality
ProCom	Model annotations	Asset-centric, system-centric	Authenticity, confidentiality
Wasicek et al.	Aspects	Attacker-centric	Confidentiality, integrity, availability
Al Faruque et al.	Functional models	Attacker-centric	Confidentiality, integrity, availability
Eby et al.	DSL	Attacker-centric asset-centric	Confidentiality integrity
SEED	Ontology (DSL)	System-centric asset-centric	Confidentiality integrity

provide security analyses or transformations (AN) based on the models for system design and threats/attacks. Both aspects are discussed in more detail in a dedicated paragraph below. All approaches except Eby et al. cover the deployment step (DP). Here, the approaches differ in the focus and methods. In most cases, the developer has to define an allocation of software parts of the system to hardware nodes, e.g., in DREMS, SEED, or ProCom. In SysML-Sec, the partitioning step, which is part of the requirements phase, is used for an initial allocation. Automatic analyses on this level support the developer when making her decisions during system design. Only DREMS explicitly focuses on runtime (RT) concepts. Even if some approaches provide code generation for parts of the system, no other approach focuses on this phase of the SDLC sufficiently. DREMS provides a mature middleware that is used to enforce – besides functional purposes – security policies at runtime. Adapter code of the middleware can be used to ensure secure communication. Hence, the final steps in the SDLC are in general not considered sufficiently.

4.2.4. Models for system design

The selected approaches use different types of models to describe the system under development. Table 5 summarizes which model types are used in the approaches. Due to the selection criteria of our survey, all approaches provide model elements that represent the platform of the system. Also, all approaches provide models that represent the architecture of the system. Wasicek et al. and Al Faruque et al. use models that describe the embedded systems on a very low level of detail. These functional models do not only cover the architecture of the system but also behavioral aspects. Since discrete (software) functions are not covered, the behavioral aspect is only fulfilled partially (cf. Table 5). DREMS, ProCom, SEED, and (to a certain degree) SysML-Sec focus on component-based specification. DREMS and ProCom provide their own meta model for the specification where as SEED and SysML-Sec base on existing modeling languages: SEED is based on SPACE and MARTE, SysML-Sec is (obviously) based on SysML. Only SysML-Sec and SEED provide dedicated models for the description of the behavior. ProCom was initially developed for the development of safe CPS and provides therefore behavior models in general but these models do not directly affect the security approach discussed in this survey. Behavioral descriptions of the generated ProSave components for encryption and decryption are not provided.

4.2.5. Formal threat specification

All approaches allow for a formal threat specification. Table 6 summarizes our elicited data. In most cases, it is an implicit threat specification, since specific security properties like confidentiality are annotated instead of specifying a threat. The main focus of the approaches is to specify confidentiality and integrity of information flow. Exceptions are the approaches by Wasicek et al. Al Faruque et al. and Eby et al. because these approaches allow specifying specific attacks targeting the stated security properties of a system. Hence, the focus of the specification of these approaches is *attacker-centric*. In general, we distinguish between three different kinds: *system-centric*, *asset-centric*, and *attacker-centric* (cf. Section 3.1.4). DREMS, SysML-Sec, ProCom and SEED focus on system-centric security specification. In addition, the ProCom approach, Eby et al. and SEED allow specifying asset-centric security properties, i.e., properties on concrete assets of the system.

Table 7
Applied security analyses and transformations.

Approach	Testing attacks	Searching threats	Securing design	Securing code
DREMS	–	✓	–	✓
SysML-Sec	–	✓	✓	(✓)
ProCom	–	–	✓	✓
Wasicek et al.	✓	–	–	–
Al Faruque et al.	✓	–	✓	–
Eby et al.	–	–	✓	–
SEED	–	✓	✓	✓

4.2.6. Security Transformations & Analyses

All approaches provide security analyses and transformations based on the models. Table 7 summarizes the kinds of analyses and transformations of the selected approaches. Only Wasicek et al. and Al Faruque et al. provide analyses to find specific attacks. Both approaches use control engineering models that allow specifying attacks on sensor values and hardware connections but do not focus on software vulnerabilities. DREMS, SysML-Sec, and SEED focus on software threats. Hence, they provide analyses to find threats in general, e.g., a potential violation of confidentiality or integrity of system messages. Five approaches also provide transformations that can be applied to make the given design more secure. Three of them (DREMS, ProCom, SEED) generate or integrate dedicated security code into the application code. In DREMS, security adapter code for communication is generated (provided by the middleware). ProCom follows a similar approach but allows to specify custom implementations for encryption and decryption components. These encryption and decryption components are added automatically by a (horizontal) model transformation keeping the model compliant to the ProCom meta model. Eby et al. on the other hand, using a vertical model transformation allowing to extend existing modeling languages with security-relevant information. SEED aims at finding, integrating, analyzing the effect of concrete security solutions specified by experts into a security knowledge base. In contrast to ProCom which focuses on secure communication, in SEED the database of available security solutions can be extended continuously by the experts and is therefore quite more expressive. SysML-Sec does not utilize security solutions on code level but allows to declare security-related functions in the design that have to be implemented correctly by the developer. Hence, the design uses the advantage of referring to secured source code and, therefore, fulfills this property partially.

5. Data analysis and discussion

In this section, we discuss the elicited data with regard to our research questions stated in Section 3.1. We discuss the selected data and answer research questions *RQ1* and *RQ2* in Section 5.1. After that, we discuss potential open research areas in Section 5.2 to answer research question *RQ3*.

5.1. Data analysis

To answer the question to which extent the platform is considered by the approaches (RQ1), we discuss the classification of all approaches presented in Table 1. Our systematic literature review eventually resulted in 17 approaches. We classified these approaches into three classes: seven approaches consider the platform but are nevertheless general approaches, e.g., UMLsec uses UML deployment diagrams to describe the deployment of the software. Typically, CPS have strict requirements regarding real-time constraints and the underlying platform, which are not covered by these approaches. Three approaches consider the platform more explicitly but are not specific to CPS. Seven approaches consider the platform in detail and are CPS-specific; those are investigated in more detail in this paper. Noteworthy is that the most papers in the final set represent the general approaches, whereas the CPS-specific approaches are represented by fewer papers. Furthermore, as also stated by Nguyen et al. (2017), the CPS-specific approaches have mostly been published during the last few years. However, considering all initial papers, we recognized a lot of *work-in-progress* approaches within the last two years which were excluded for our data analysis. Most of these papers were in the context of model-driven security in general but also in the area of CPS-specific approaches. These facts indicate the current relevance of this research topic and we expect more mature approaches in the future. In summary, we state that considering the platform of CPS explicitly is a current research topic for model-driven security. Only two of the seven approaches (SysML-Sec and SEED) provide an entire model-driven methodology that allows for design decisions based on results of security analyses. However, none of the selected approaches provide automatic design decisions with regard to software and platform, e.g., considering security policies during the allocation. Only DREMS allows enforcing such policies at runtime. Hence, the automatic design decisions for securing a system within the whole SDLC and the refinement into source code are immature areas in this field, and could be improved, e.g., by generating code or utilizing code analyses to verify assumptions made in the models. Overall, we state that **R1** (*support different layers*) is satisfied by the rationale of this survey since we focus on platform-specific approaches. Thus, all selected approaches allow specifying the hardware or platform of the system. However, Al Faruque et al. and Wasicek et al. use functional models that describe both hardware and software in one model. Thus, we state that these approaches satisfy **R1** only partially.

Table 8
Requirements for CPS covered by the selected approaches.

Approach	R1	R2	R3	R4	R5	R6	R7
DREMS	✓	✓	✓	p	p	p	p
SysML-Sec	✓	✓	✓	p	p	p	✓
ProCom	✓	p	✓	p	p	p	✓
Wasicek et al.	p	p	p	p	p	p	p
Al Faruque et al.	p	p	p	p	p	p	p
Eby et al.	✓	p	✓	p	p	p	p
SEED	✓	✓	✓	p	p	p	✓

To answer the question which requirements are met (RQ2), we state which requirement is fulfilled for each approach. Table 8 summarizes the requirements for all of the six approaches. We define **R2** (*phases of the SDLC*) as fulfilled (✓) if at least five of six phases of the SDLC are supported, and as partially fulfilled (*p*) if at least three of five phases are supported. All approaches satisfy **R2** at least partially. Three approaches support five SDLC phases, where either the requirements phase (DREMS) or the runtime phase (SysML-Sec, SEED) is not supported sufficiently. The use of functional models also affects the fulfillment of **R3** (*system of systems*), since Al Faruque et al. and Wasicek et al. do not allow for hierarchical composition of the system. The other approaches are component-based and ease the decomposition of the system into smaller parts. However, none of the approaches considers third-party software parts explicitly during the design. Approaches that utilize manually written source code give in fact the possibility to integrate not (fully) known system parts, e.g., DREMS or ProCom. Here, the manual code has to be compliant to generated interfaces but is not analyzed to ensure that it still conforms to assumptions made during the design.

R4 (*threat model*) is satisfied partially by all selected approaches. Each approach provides a formal threat model as summarized in Table 6. In the approaches that provide system-centric and asset-centric specifications, only threats for confidentiality and integrity are considered. The attacker-centric specifications are modeled by dedicated attack models. However, in general, all threat models base on more general concepts and aim to be extensible for further security properties and threats respectively. In addition, some approaches provide additional threat elicitation techniques. SysML-Sec provides the most mature one since attack graphs can be used to specify complex attacks that can be related to system models, which allows effective threat modeling. The approach does not provide a refinement for all possible attacks into a formal threat specification, that can be used during design. Nevertheless, it is the only surveyed approach that provides a mature threat model that relates to a model-driven systems engineering approach. Only the ProCom approach uses threat models to derive secure design decisions automatically. SEED guides the developer during decisions-making, but does not automate this step. All approaches utilize model-driven techniques and also provide threat models or attacker models. However, in all approaches, threat models for the platform parts are – where existent – too simple. In general, threat modeling could be more focused to find more (unknown) threats which then can be tackled by automatic transformations. Hence, we state that this requirement is only partially fulfilled.

Due to the model-driven focus of the survey, all selected approaches provide threat/attack models and corresponding analyses on a formal basis and, thus, fulfill **R5** (*formal methods*) at least partially. Four approaches use annotations to express security related information within the system models. SysML-Sec, for example, uses text annotations directly within the block diagrams whereas in SEED the integration of security ontologies is applied which allows a more flexible specification. Three approaches provide dedicated attack models that can be used to analyze concrete attacks and threats to the system models. However, these concepts are only applicable to very basic attacks, e.g., message replay or denial-of-service attacks. More general threats and solutions are not manageable using such approaches. Nevertheless, results of analyses or simulations of these attacks can be used to refine the system models into more secure ones. Most system-centric approaches focus on confidentiality and integrity of communication in the system. Further security approaches are not explicitly covered in the formal models. However, the more mature approaches, like SEED or SysML-Sec, aim to be able to integrate additional security properties. Apart from SysML-Sec, no other approach considers attack sequences.

To fulfill **R6** (*refinement*, an approach should provide a correct refinement from the requirements stage to at least platform-specific code. Correct refinement always depends on the source models of each refinement step. For example, in DREMS or ProCom, it is assumed that the implementation of the actors is provided by a C function which is implemented or loaded in the deployment step. Thus, these functions cannot be considered in the early analyses and, therefore, the behavioral aspect is also not considered during the refinement of the models. In approaches like SysML-Sec or SEED, where dedicated behavioral models are used and analyzed during the security analyses, a refinement is more complex. Considering behavioral models has the advantage of more precise analyses but can run quickly into a state-explosion problem. All of the approaches aim for fulfilling the verified security properties for the executed code. However, we base our observations on the results reported by the authors without executing a full correctness check of the provided refinement steps.

Furthermore, all surveyed approaches base on concepts or existing approaches for model-driven development for CPS. Thus, functional requirements are covered by all approaches at least implicitly, which is why **R7** (*requirements*) is fulfilled at least partially. Since the approach of Eby et al. refines an existing modeling language into a security-aware dialect, the fulfillment of this requirement depends on the chosen host language. SEED, ProCom, and SysML-Sec provide dedicated possibilities for specifying functional requirements and satisfy **R7**. SysML-Sec provides dedicated analyses to measure the impact of security features to properties of the system during the partitioning phase, e.g., performance. Also SEED, ProCom, and Eby et al. allow analyzing the impact of adding security features to the system model. Hence, these approaches also consider non-functional requirements. SysML-Sec is the only approach that allows to explicitly specify all kinds of requirements, i.e., functional and non-functional. Integrating security functionality into CPS affects the behavioral parts of the system. Here, we observed that it is possible to abstract from the functional models during threat modeling but that it is not possible to abstract from these models during the mitigation stage where security solutions are introduced.

5.2. Discussion and open research topics

Considering all surveyed approaches, it appears that the following areas should be researched in more detail.

Threat Model: Most of the approaches do not provide an explicit threat model but use formal specifications of concrete threats, and focus in general on confidentiality and integrity of communication within the system. However, when a threat model is created, e.g., by stating security requirements, a connection between a threat model (created by domain experts) and the formal models of the system is missing. Only SysML-Sec provides methods for both threat and attack modeling and the integration into model-driven development. However, a combination of both in later development steps is also not given. Creating a separate threat model and integrating it into the CPS development would be an interesting research field. One interesting direction for future research could be, how techniques of non CPS-specific approaches (cf. Table 1) could be adapted for CPS, e.g., by adding information flow specifications and analyses as done in UMLsec (Jürjens, 2005). One major question would be how the platform model and its different parts would affect such analyses. Another interesting part for a threat model would be access control. SecureUML (Basin, 2006) provides useful concepts for access control on models. However, also here further research on the impact of the platform part and in particular the physical part is necessary. Since physical attacks may also affect the cyber layer, it might not be sufficient to simply add a platform layer to an approach. Changes in the original techniques might be needed considering all aspects of CPS. Furthermore, even if in the selected approaches system and platform models are developed separately, the threat specification is not. A clear concept for separation of the different layers is also needed when specifying threats to support the interdisciplinary development of such systems. Many methodologies for CPS support interdisciplinary development for the functional part of the system. An interesting research direction would be to investigate how a common threat model that is usable by all involved disciplines and stakeholders could look like. On the one hand, we like the idea of creating a dedicated threat or security model like in ModelSec (Sánchez et al., 2009). However, if many different disciplines and roles are involved, a clear separation of security experts and other experts is needed on the one hand. On the other hand, a concept for close collaboration is needed and, therefore, also a strong connection between security model and system model is favorable. Thus, a concept for touchpoints in the development methodology is needed, where different experts are forced to communicate about problems between these areas. Beside considering the different layers and their inter-dependencies, the different view – also from not-technical disciplines – would be an interesting area. Finally, current approach focus on mostly one kind of threat specification and analysis (cf. Section 4.2.5.)

Third-party Code: Furthermore, most approaches focus on closed systems. Even if all selected approaches support modeling of hierarchical systems (requirement **R3**) at least partially, none of the approaches does consider system parts developed by third-party providers or full applications from different providers that run concurrently in the same runtime environment. The only exception is DREMS, where such kind of systems can be modeled by using the multi-domain labels. However, they have to be modeled explicitly and are expressed and handled in the same way as all other system actors and components respectively. Third-party code is getting more important for large-scaled systems. On the one hand, large CPS have to integrate code and hardware modules of other suppliers and, therefore, have different levels of trust (depending on the suppliers). Additionally, the amount of open source software is rising which has also to be considered during the design and implementation because often OSS solutions are used for critical task, e.g., OpenSSL. The advantage is that known security issues and knowledge can be added to the own threat model. The (secure) integration of third-party code into the system but also into the threat modeling approach seems to be an important research direction. In general, introducing trust boundaries in the modeling approaches could improve this integration in an easy and efficient way, e.g. as specified in the STRIDE methodology (Shostack, 2014b). This includes a classification of all model elements into security classes that describe the criticality from a security point of view as done in DREMS or Eby et al. Approaches that use source code as behavior specification allow applying analyses without knowing the source code. However, the drawback is that the source code itself is regarded as trusted and is not covered within the threat models explicitly. Hence, the integration of third-party system parts into the models, and the relation to source code should be improved, e.g., by deriving inputs for code analyses and using their results as input for security risk analysis. This would allow security expert to reason about the overall risk based on analysis results instead of assumptions made during the threat modeling phase. Thus, we see the integration of third-party code and new concepts for integrating it into the threat modeling approach in a trustworthy way as a interesting research field.

Platform Integration: In general, the platform and especially the *physical* part of the system, is only rarely taken into account when modeling threats and attack surfaces in the selected approaches, e.g., by marking communication channels as publicly accessible or as secure like done in the ProCom approach. More complex threat models for the platform might be useful to find attacks on the platform level in early steps, and to adapt software and platform models for mitigation. For example, the fact if a device is physically accessible should be considered when talking about security concepts for the software running on this device. It makes a difference if a device is operating at a construction side or in a secured area where only a few person have physical access to. Furthermore, the deployment and runtime are only barely covered. DREMS is the only approach that generates deployment plans for the system. SysML-Sec and SEED provide designers the option to analyze and compare different versions of the deployed system. However, providing concrete plans for the secure operation of a system and considering active countermeasures like intrusion detection systems during development would improve the overall security of the system and is an interesting research field. Only DREMS is able to enforce security properties at runtime. Furthermore, most approaches provide (partial) code generation for the system. However, a holistic deployment step involves other tasks despite code generation. For example, in analyses, it is assumed that keys for encryption are already shared. Such a key management could be part of the deployment, as well as configuration plans for each involved ECU and operating system. Hence, since deploying the system might add additional side effects, it would be interesting to think about the validation of countermeasures during the design and threat modeling phase. Instead of applying security only during the development, additional work on using a threat model as a reactive artifact describing the current security situation of the deployed system would be interesting. In general, a higher degree of automation in such methods would increase the degree of security since each manual step may introduce security flaws. This does not only affect model transformations (including those to source code) but also the deployment and runtime controlling of the system. In the context of security, this refinement becomes a major challenge since a correct refinement might not preserve security properties in all cases (Roscoe, 1995). Since in CPS the execution highly depends on the executing platform, secure code generation and execution (based on the design models) as well as their validation should be further investigated.

Common Evaluation Scenario: A common evaluation scenario would be of great benefit for approaches both in existence and under development. Since CPS cover still a quite wide range of domains, sticking to one domain would be restricting but is necessary when performing comparative evaluations. Another problem we see here is that there are too many different facets that could be compared, e.g., expressiveness of the threat model, kind of the threat model, traceability of threats and countermeasures, performance, usability, etc. which might need different scenarios. Also, the kind of approach is a factor that has to be considered: Approaches that can be added to an existing system modeling method could be compared based on an existing pre-defined system model. However, if an approach uses its own modeling language for defining the system, the scenario needs to be remodeled within this language. There are large-scaled projects that could be used for comparing such approaches, e.g., the EVITA project (EVITA, 2020) or the Common Component Modeling Example (CoCoMe) (Herold et al., 2008). In general, when designing an evaluation scenario, one has to choose if this scenario strengthen the internal validity or external validity of the evaluation (Siegmund et al., 2015). A concrete evaluation scenario as used in the presented approaches could strengthen the internal validity of the evaluation but then necessarily has limited external validity at the same time. To strengthen the external validity, on the other hand, one would need to compare multiple approaches to one another, which thus would demand a rather less specific scenario. Thus, as consequence a *fixed* modeling example has limited utility for comparing multiple approaches. Such a comparative evaluation could better be supported by a list of weaknesses, e.g., a subset of the CWE database and a set of properties. CWEs are well suited because they are describing general weaknesses instead of concrete vulnerabilities like CVEs. Focusing on one CWE allows one to evaluate properties such as traceability of threats and countermeasures as well as the properties of the used modeling syntax such as expressiveness or usability. Choosing a database describing scenarios from real-world projects would also be suitable to evaluate the practicality of the approaches. However, such a data set cannot replace scenarios used to strengthen internal validity (Siegmund et al., 2015) and thus should be created in addition.

6. Conclusion

Model-driven security approaches seem to be a key methodology to fully handle the security and safety needs for modern CPS. Since CPS are distributed systems and additionally interact with their environment, one requires dedicated methods that cover all requirements of CPS. In particular, the software (cyber) as well as the platform layer have to be considered during development and threat modeling. We conducted a systematic literature review to answer the question to which degree model-driven security approaches for CPS cover the platform.

We found out that during the last years several approaches were developed in this area. However, the platform integration is still immature in most approaches. Also, the integration of threat modeling and the trace to applied security solutions can be improved. The refinement of the models during the whole SDLC can be improved by increasing the degree of automation. Furthermore the tracing of security properties through all phases of the SDLC should be considered more when providing refinements of the models. Although the deployment is a crucial step, it is barely covered in most approaches. Furthermore, integration of third-party code and dependencies to other, not fully known system parts is missing. Finally, a common evaluation scenario would help to compare such methodologies in detail.

We hope that the results of our survey will encourage developers of model-driven methodologies to focus more on the open research questions. The comprehensive overview of the approaches might help to find similarities and differences, and help to provide a base for a better understanding of this area. We plan to extend our survey to other libraries and journals in order to cover more approaches in this area. Finally, we are going to investigate how classical threat modeling can be integrated into a holistic model-driven approach for developing CPS to increase the number of covered threats and the degree of automation in both the cyber layer and the platform layer.

CRediT authorship contribution statement

Johannes Geismann: Conceptualization, Data curation, Investigation, Writing - original draft, Writing - review & editing. **Eric Bodden:** Conceptualization, Supervision, Writing - original draft, Writing - review & editing.

Acknowledgments

The authors would like to thank Andreas Dann, Christopher Gerking, and Marie Christin Platenius-Mohr for the helpful discussions and their feedback on drafts of this paper. Johannes Geismann is member of the Ph.D. program "Design of Flexible Work Environments: Human-Centric Use of Cyber-Physical Systems in Industry 4.0", supported by the German federal state of North Rhine-Westphalia.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Al Faruque, M., Regazzoni, F., Pajic, M., 2015. Design methodologies for securing cyber-physical systems. In: International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2015. CODES '15, IEEE Press, Piscataway, NJ, USA, pp. 30–36.

Apvrille, L., Li, L., Roudier, Y., 2016. Model-driven engineering for designing safe and secure embedded systems. In: 2016 Architecture-Centric Virtual Integration. ACVI. pp. 4–7.

Apvrille, L., Roudier, Y., 2013. SysML-Sec: A SysML environment for the design and development of secure embedded systems. In: Asia-Pacific Council on Systems Engineering. APCOSEC.

Apvrille, L., Roudier, Y., 2014. Towards the model-driven engineering of secure yet safe embedded systems. Electron. Proc. Theor. Comput. Sci. 148, 15–30.

Apvrille, L., Roudier, Y., 2015. SysML-Sec attack graphs: Compact representations for complex attacks. In: GraMSec@CSF, Vol. 9390, pp. 35–49.

Balasubramanian, D., Dubey, A., Otte, W., Levendovszky, T., Gokhale, A., Kumar, P., Emfinger, W., Karsai, G., 2015. DREMS ML: A wide spectrum architecture design language for distributed computing platforms. Sci. Comput. Program. 106, 3–29.

Banerjee, A., Venkatasubramanian, K.K., Mukherjee, T., Gupta, S.K.S., 2012. Ensuring safety, security, and sustainability of mission-critical cyber–physical systems. Proc. IEEE 100 (1), 283–299.

Basin, D., 2006. Model driven security. In: First International Conference on Availability, Reliability and Security. ARES.

Behrmann, G., David, A., Larsen, K., 2004. A tutorial on uppaal. In: Formal Methods for the Design of Real-Time Systems, Vol. 3185. pp. 200–236.

Bell, D.E., LaPadula, L.J., 1973. Secure Computer Systems: Mathematical Foundations. Technical Report, DTIC Document.

Biba, K.J., 1977. Integrity Considerations for Secure Computer Systems. Technical Report, DTIC Document.

Borde, E., Carlson, J., 2011. Towards verified synthesis of ProCom, a component model for real-time embedded systems. In: Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering. CBSE '11, ACM, New York, NY, USA, pp. 129–138.

Borde, E., Carlson, J., Feljan, J., Lednicki, L., Lévêque, T., Maras, J., Petricic, A., Sentilles, S., 2011. Pride-an environment for component-based development of distributed real-time embedded systems. In: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture. IEEE, pp. 351–354.

Brambilla, M., Cabot, J., Wimmer, M., 2012. Model-driven software engineering in practice. Synth. Lect. Softw. Eng. 1 (1), 1–182.

Bureš, T., Carlson, J., Crnkovic, I., Sentilles, S., Vulgarakis, A., 2008. ProCom – The Progress Component Model Reference Manual. Mälardalen University, Västerås, Sweden.

Canedo, A., Wan, J., Al Faruque, M.A., 2014. Functional modeling compiler for system-level design of automotive cyber-physical systems. In: Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on. IEEE, pp. 39–46.

Cigital Federal Inc., 2011. Addressing software security in the federal acquisition process.

Dolev, D., Yao, A., 1983. On the security of public key protocols. IEEE Trans. Inform. Theory 29 (2), 198–208.

Dorbala, S.Y., Bhadoria, R.S., 2015. Analysis for security attacks in cyber-physical systems. In: Cyber-Physical Systems: A Computational Perspective. Chapman and Hall/CRC, pp. 395–414.

Eby, M., Werner, J., Karsai, G., Ledeczi, A., 2007. Integrating security modeling into embedded system design. In: 14th Annual IEEE International Conference on Engineering of Computer-Based Systems.

Eker, J., Janneck, J.W., Lee, E.A., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y., 2003. Taming heterogeneity - the Ptolemy approach. Proc. IEEE 91 (1), 127–144.

EVITA, 2020. The EVITA project webpage. https://www.evita-project.org.

Felderer, M., Zech, P., Breu, R., Büchler, M., Pretschner, A., 2016. Model-based security testing: a taxonomy and systematic classification. Softw. Test. Verif. Reliab. 26 (2), 119–148.

Fitzgerald, J.S., Larsen, P.G., Verhoef, M., 2014. From embedded to cyber-physical systems: Challenges and future directions. In: Collaborative Design for Embedded Systems. Springer, pp. 293–303.

Geismann, J., Bodden, E., 2020. Replication package for the literature review. http://dx.doi.org/10.5281/zenodo.3843478.

Gunawan, L.A., Kraemer, F.A., Herrmann, P., 2011. A tool-supported method for the design and implementation of secure distributed applications. In: International Symposium on Engineering Secure Software and Systems. Springer, pp. 142–155.

Hafner, M., Breu, R., Agreiter, B., Nowak, A., 2006. SECTET: an extensible framework for the realization of secure inter-organizational workflows. Internet Res. 16 (5), 491–506.

Herold, S., Klus, H., Welsch, Y., Deiters, C., Rausch, A., Reussner, R., Krogmann, K., Koziolek, H., Mirandola, R., Hummel, B., Meisinger, M., Pfaller, C., 2008. CoCoME - the common component modeling example. In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (Eds.), The Common Component Modeling Example: Comparing Software Component Models. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 16–53.

Herzog, A., Shahmehri, N., Duma, C., 2007. An ontology of information security. Int. J. Inf. Secur. Priv. 1 (4), 1–23.

Idrees, M.S., 2012. A Requirement Engineering Driven Approach to Security Architecture Design for Distributed Embedded Systems (Ph.D. thesis). Télécom ParisTech.

Ishtiaq Roufa, R.M., Mustafaa, H., Travis Taylora, S.O., Xua, W., Gruteserb, M., Trappeb, W., Seskarb, I., 2010. Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study. In: 19th USENIX Security Symposium, Washington DC, pp. 11–13.

Jacob, J., 1989. On the derivation of secure components. In: Proceedings. 1989 IEEE Symposium on Security and Privacy, pp. 242–247.

Jensen, J., Jaatun, M.G., 2011. Security in model driven development: A survey. In: Sixth International Conference on Availability, Reliability and Security. ARES, 2011, pp. 704–709.

Jürjens, J., 2005. Secure Systems Development with UML. Springer Science & Business Media.

Kienhuis, B., Deprettere, E.F., Wolf, P.V.D., Vissers, K., 2002. A methodology to design programmable embedded systems — The Y-chart approach. In: Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS, pp. 18–37.

Kitchenham, B., 2004. Procedures for Performing Systematic Reviews, Vol. 33, No. TR/SE-0401. Keele University, Keele, UK, p. 28.

Kitchenham, B., Charters, S., 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering, Vol. 2. Technical Report, Keele University.

Kriaa, S., Pietre-Cambacedes, L., Bouissou, M., Halgand, Y., 2015. A survey of approaches combining safety and security for industrial control systems. Reliab. Eng. Syst. Saf. 139, 156–178.

Lee, E.A., 2008. Cyber physical systems: Design challenges. In: 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing. ISORC, IEEE, pp. 363–369.

Lee, R.M., Assante, M.J., Conway, T., 2014. German steel mill cyber attack. Ind. Control Syst. 30.

Levendovszky, T., Balasubramanian, D., Coglio, A., Dubey, A., Otte, W., Karsai, G., Gokhale, A., Nyako, S., Kumar, P., Emfinger, W., 2014. DREMS: A model-driven distributed secure information architecture platform for managed embedded systems. IEEE Softw. 31 (2), 62–69.

Li, L., 2018. Safe and Secure Model-Driven Design for Embedded Systems (Ph.D. thesis). Université Paris-Saclay.

Li, L.W., Lugou, F., Apvrille, L., 2017. Security-aware modeling and analysis for HW/SW partitioning. In: MODELSWARD, pp. 302–311.

Li, L.W., Lugou, F., Apvrille, L., 2018. Security modeling for embedded system design. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 10744 LNCS. pp. 99–106.

Lugou, F., Li, L.W., Apvrille, L., Ameur-Boulifa, R., 2016. SysML models and model transformation for security. In: 2016 4th International Conference on Model-Driven Engineering and Software Development. MODELSWARD, SCITEPRESS, pp. 331–338.

Lun, Y.Z., D'Innocenzo, A., Malavolta, I., Di Benedetto, M.D., 2016. Cyber-physical systems security: a systematic mapping study. arXiv preprint arXiv:1605.09641.
McLean, J., 1996. A general theory of composition for a class of "possibilistic" properties. IEEE Trans. Softw. Eng. 22 (1), 53–67.
Mendeley Ltd., 2020. Mendeley desktop. [Online]; https://www.mendeley.com/. (Accessed May 2020).
Moebius, N., Stenzel, K., Grandy, H., Reif, W., 2009. SecureMDD: A model-driven development method for secure smart card applications. In: International Conference on Availability, Reliability and Security, 2009. ARES '09, pp. 841–846.
Motii, A., 2017. Engineering Secure Software Architectures: Patterns, Models and Analysis (Ph.D. thesis). Université de Toulouse, Université Toulouse III-Paul Sabatier.
Neri, M.A., Guarnieri, M., Magri, E., Mutti, S., Paraboschi, S., 2013. A model-driven approach for securing software architectures. In: International Conference on Security and Cryptography. SECRYPT.
Neureiter, C., Engel, D., Uslar, M., 2016. Domain specific and model based systems engineering in the smart grid as prerequesite for security by design. Electronics 5 (4), 24.
Nguyen, P.H., Ali, S., Yue, T., 2017. Model-based security engineering for cyber-physical systems: A systematic mapping study. Inf. Softw. Technol. 83, 116–135.
Nguyen, P.H., Kramer, M., Klein, J., Le Traon, Y., 2015. An extensive systematic review on the Model-Driven Development of secure systems. Inf. Softw. Technol. 68, 62–81.
Object Management Group, 2008. A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems (2008-06-08), Vol. 2. Technical Report ptc/2008-06-09, Object Management Group.
Ptolemaeus, C., 2014. System Design, Modeling, and Simulation. Using Ptolemy II. Ptolemy. org Berkeley, p. 690.
Rajkumar, R., Lee, I., Sha, L., Stankovic, J., 2010. Cyber-physical systems: the next computing revolution. In: Design Automation Conference. IEEE, pp. 731–736.
Reddy, Y.B., 2015. Security and design challenges in cyber-physical systems. In: 12th International Conference on Information Technology - New Generations. ITNG, 2015, pp. 200–205.
Roscoe, A.W., 1995. CSP and determinism in security modelling. In: Proceedings 1995 IEEE Symposium on Security and Privacy. IEEE, pp. 114–127.
Roudier, Y., Apvrille, L., 2015. SysML-Sec: A model driven approach for designing safe and secure systems. In: 3rd International Conference on Model-Driven Engineering and Software Development. MODELSWARD, 2015, pp. 655–664.
Roudier, Y., Idrees, M.S., Apvrille, L., 2014. Improved security requirements engineering using knowledge representation. In: Conf'rence sur la Scritè des Architectures Rèseaux et des Systemes d'Information.
Ruiz, J.F., Maña, A., Rudolph, C., 2015. An integrated security and systems engineering process and modelling framework. Comput. J. 58 (10), 2328–2350.
Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131.
Saadatmand, M., Leveque, T., 2012. Modeling security aspects in distributed real-time component-based embedded systems. In: Ninth International Conference on Information Technology: New Generations. ITNG, 2012, pp. 437–444.
Sánchez, Ó., Molina, F., García-Molina, J., Toval, A., 2009. ModelSec: a generative architecture for model-driven security. J. UCS 15 (15), 2957–2980.
Schneier, B., 1999. Attack trees. Dr Dobbs J. 24 (12), 21–29.
Shostack, A., 2014a. Threat Modeling: Designing for Security. Wiley, Indianapolis, Ind.
Shostack, A., 2014b. Threat Modeling: Designing for Security. John Wiley Sons, Indianapolis, USA.
Siegmund, J., Siegmund, N., Apel, S., 2015. Views on internal and external validity in empirical software engineering. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, pp. 9–19.
Uzunov, A.V., Fernandez, E.B., Falkner, K., 2012. Engineering security into distributed systems: A survey of methodologies. J. UCS 18 (20), 2920–3006.
Vasilevskaya, M., 2015. Security in Embedded Systems: A Model-Based Approach with Risk Metrics (Ph.D. thesis). Linköping University Electronic Press.
Vasilevskaya, M., Gunawan, L.A., Nadjm-Tehrani, S., Herrmann, P., 2014. Integrating security mechanisms into embedded systems by domain-specific modelling. Secur. Commun. Netw. 7 (12), 2815–2832.
Völter, M., Stahl, T., Bettin, J., Haase, A., Helsen, S., 2013. Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons.
Wan, J., Canedo, A., Al Faruque, M.A., 2015a. Security-aware functional modeling of cyber-physical systems. In: 20th Conference on Emerging Technologies & Factory Automation. ETFA, IEEE, pp. 1–4.
Wan, J., Canedo, A., Al Faruque, M.A., 2017. Functional model-based design methodology for automotive cyber-physical systems. IEEE Syst. J. 11 (4), 2028–2039.
Wan, J., Canedo, A., Faruque, M.A.A., 2015b. Functional model-based design methodology for automotive cyber-physical systems. IEEE Syst. J. 11 (99), 1–12.
Wasicek, A., Derler, P., Lee, E.a., 2014. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In: Proceedings of the the 51st Annual Design Automation Conferenc. DAC, pp. 1–6.
Weilkiens, T., 2011. Systems Engineering with SysML/UML: Modeling, Analysis, Design. Elsevier.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. EASE '14, ACM, New York, NY, USA, pp. 38:1–38:10.
Zhang, L., Fallah, Y.P., Jihene, R., 2013. Cyber-Physical Systems: Computation, Communication, and Control. SAGE Publications Sage UK, London, England.
Zhou, G., Leung, M.-K., Lee, E.A., 2007. A code generation framework for actor-oriented models with partial evaluation. In: Embedded Software and Systems: Third International Conference, ICESS 2007, Daegu, Korea, May 14–16, 2007. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 193–206.

Johannes Geismann is a research assistant at the chair for software engineering at the Heinz Nixdorf Intitute at Paderborn University. He received the degree Master of Science in computer science in 2016. He is member of the Ph.D. program "Design of FlexibleWork Environments: Human-Centric Use of cyber–physical Systems in Industry 4". His main research interests are model-driven security and threat modeling for cyber–physical systems.

Eric Bodden is heading the chair for software engineering at the Heinz Nixdorf Intitute at Paderborn University. He is also director for software engineering at the Fraunhofer Institute for Mechatronic Systems Design. Prof. Bodden's research focuses on a secure engineering lifecycle for software intensive systems. In particular, his research group designs, builds and empirically evaluates tool support for such a lifecycle.