Securing the Testing Process for Industrial Automation Software

Matthias Eckhart[a,b,*], Kristof Meixner[a], Dietmar Winkler[a], Andreas Ekelhart[a,b]

[a] *Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle (CDL-SQI), Institute of Information Systems Engineering, TU Wien, Favoritenstraße 9–11, Vienna, Austria*
[b] *SBA Research, Favoritenstraße 16, Vienna, Austria*

Abstract

The testing of automation applications has become a crucial pillar of every production systems engineering (PSE) project with the proliferation of cyber-physical systems (CPSs). In light of new attack vectors against CPSs, caused, inter alia, by increased connectivity, security aspects must be considered throughout the PSE process. In this context, software testing represents a critical activity, as a lack of adequate security mechanisms puts a variety of valuable assets (e.g., system configurations and production details) at risk of information theft and sabotage. Thus, organizations must analyze the security of their software testing process on a regular basis in order to counter these threats. Yet, due to the required security knowledge or budget constraints for security-related expenses, these undertakings may be destined to fail. In this work, we present a framework that supports the semi-automated security analysis of an organization's software testing process for industrial automation software. This framework is based on the VDI/VDE 2182 guideline and integrates an ontological approach to model the necessary background knowledge, including, e.g., data flows, assets, entities, threats, and countermeasures. The framework comprises a default model of the testing process, which users can adapt so that the target of inspection accurately

*Corresponding author
Email addresses: matthias.eckhart@tuwien.ac.at (Matthias Eckhart), kristof.meixner@tuwien.ac.at (Kristof Meixner), dietmar.winkler@tuwien.ac.at (Dietmar Winkler), andreas.ekelhart@sba-research.org (Andreas Ekelhart)

Postprint. Computers & Security 85 (2019) 156–180.

reflects their software testing environment. In particular, the testing process considered for creating the default model is based on best practices observed at a major system integrator, aligned with the ISO/IEC/IEEE 29119 series of software testing standards. Moreover, we developed a tool that enables the automatic generation of attack–defense trees from such formal models of the organization's software testing process. We demonstrate how the proposed framework can be applied to a generic software testing process to answer essential questions in conducting a security risk analysis. The results of the exemplary security analysis provide guidance, should raise awareness in the industrial domain, and support effective, yet cost- and time-efficient security analyses. Finally, we evaluate the presented framework by performing a comprehensive comparison of suitable security analysis tools.

Keywords: Security analysis, Threat modeling, Risk assessment, Security ontology, Software testing, Industrial automation software, Cyber-Physical Systems, Industrial control systems, VDI/VDE 2182, ISO/IEC/IEEE 29119

1. Introduction

In the past decades, the adoption of software in the industrial automation domain increased significantly. According to a report presented by the Mechanical Engineering Industry Association (VDMA)[1], the costs of software development activities in engineering projects for automation systems increased from approx. 20% in 2000 to more than 40% in 2010, and it is expected that this share continues to rise (Gausemeier, 2010; Vyatkin, 2013). These findings indicate that software engineering already started to dominate PSE projects, leaving other engineering disciplines (i.e., mechanics and electronics) behind in terms of spending. Strategic initiatives, such as Industry 4.0 (Kagermann et al., 2013), underpin this trend, as CPSs are considered as a stepping stone toward the realization of the "smart factory". CPSs tightly couple "cyber" (e.g., software) and physical components (e.g., sensors, actuators) and operate on both dimensions (i.e., communicating with other *cyber* systems and act in the *physical* environment) (Baheti and Gill, 2011). Due to the fact that the behavior of these systems is governed by the software that they execute, software testing is a vital activity to ensure that the CPSs

[1]VDMA: `https://www.vdma.org`.

perform as intended. Since CPSs interact with the real world, e.g., by controlling manufacturing processes in case of industrial control systems (ICSs), the functional safety but also security of these systems must be guaranteed.

As a matter of fact, security and safety are interdependent properties (Knowles et al., 2015), meaning that successful cyber attacks against CPSs may damage plant equipment, put human health at risk or harm the environment. For example, past CPSs or, more specifically, ICSs security incidents[2] caused sewage to flow into waterways in Maroochy Shire (Slay and Miller, 2008), the destruction of centrifuges at Iran's Natanz nuclear facility (Langner, 2013; Falliere et al., 2011) and severe physical damages to a German steel mill's blast furnaces (Lee et al., 2014). To counter cyber threats for CPSs, security must be integrated into each phase of the PSE process, following the principle "security by design". PSE processes are embedded in a multi-disciplinary environment, where engineers of different domains work together using various specialized tools that produce heterogeneous planning artifacts (Biffl et al., 2017). Unprotected PSE data (i.e., engineering artifacts) in general, pose a severe security threat, as adversaries may be able to steal know-how or even introduce vulnerabilities into artifacts (e.g., blueprints or code of the CPS), for exploitation later on in the system's lifecycle (Kieseberg and Weippl, 2018; Weippl and Kieseberg, 2017).

Software testing of automation applications, in particular, represents a critical phase in every engineering project, as a compromised testing process may allow adversaries to steal or manipulate engineering artifacts. Besides software piracy or the theft of intellectual property (IP), test artifacts may be leveraged to launch highly effective and covert attacks against CPSs during plant operation. For instance, if these artifacts enhance the attacker's knowledge of the physical process under control, he or she may be able to introduce subtle changes in a way that covertly degrades the operation of the plant (de Sá et al., 2017). Stuxnet is one of the most prominent examples of such a covert attack, which required in-depth knowledge of the target systems and the controlled industrial processes (Langner, 2013; Falliere et al., 2011).

On the other hand, the manipulation of test results may allow adversaries

[2]Interested readers may refer to (Miller and Rowe, 2012; McLaughlin et al., 2016; Stouffer et al., 2015) for a list of documented ICSs security incidents that occurred in the past.

to conceal malicious code that has been injected during the testing process or previous PSE phases. The placed malware could then become active during test execution or lie dormant until triggered during plant operation. The criticality of the involved assets and the unique characteristics of the testing process for industrial automation software motivate the need for a thorough threat and risk assessment of software testing activities. Furthermore, as the software testing approaches typically differ between organizations, it is crucial to assess the individual situation of each organization.

In this article, we present a comprehensive framework that facilitates the analysis of the security aspects of the software testing process for industrial automation software. The overall objective of this article is to support organizations in securing their software testing approach and to increase the awareness of cyber threats that target the PSE process.

First, we develop a generic process model for software testing that considers the special characteristics of the industrial domain to define our assessment scope. To define this model, we conducted interviews with a major Austrian systems integrator, reviewed it together with a software quality consulting company, and finally, aligned it with internationally recognized standards for software testing.

Second, we introduce the developed security analysis framework, which is based on the procedural model for risk analysis specified in the VDI/VDE 2182-1 (2011) guideline to ensure conformance with the recommended state of the art. This framework also integrates a STRIDE-based threat modeling approach (Shostack, 2014) for identifying relevant threats to the assets involved. To ensure that the threat models are applicable to variants of the herein described testing process, we developed a tool that enables users to automatically generate attack–defense trees (ADTrees) (Kordy et al., 2011), specifically tailored to their environment. For the quantitative assessment of risks, we take advantage of the open-source software *ADTool* (Kordy et al., 2013a). In this way, users are able to answer questions, such as, *"Which roles are authorized to access which assets?"* or *"Which threats may exist for the software testing process and how can they be mitigated?"*.

Finally, we conduct a comprehensive evaluation of our framework by comparing it to other security analysis tools.

The contributions of this paper can be summarized as follows:

- We investigate the state of practice in testing industrial automation software by (i) analyzing the testing approach of a major systems inte-

grator, and (ii) aligning it with international standards for software testing (viz., the ISO/IEC/IEEE 29119 series). The outcome is a generic and profound version of the software testing process that is well applicable for industrial automation software.

- We present a novel framework for conducting semi-automated security analyses of software testing processes in PSE projects, based on the VDI/VDE 2182-1 (2011) guideline. Furthermore, we demonstrate how this framework can be applied to understand threats and answer security-relevant questions pertaining to the software testing process.

- Finally, we introduce a publicly-available prototype implementation of the framework, and data models of the underlying security and process knowledge. It includes *ADTGenerator*, a tool that allows users to automatically generate ADTrees (Kordy et al., 2011) for specific testing setups, in order to facilitate threat modeling and a quantitative risk assessment.

The remainder of this paper is structured as follows: Section 2 discusses the methodology of our work and briefly reviews existing security concepts that have been leveraged in our research. In Section 3, we introduce a generic software testing process for automation applications, which also defines the assessment scope for the proposed framework. Section 4 details the security analysis framework and how it can be applied in the context of software testing. In particular, this section first describes the ontologies that are used to model relevant knowledge and then demonstrates how the proposed framework can support each step of the security analysis. After presenting the main contribution of this work, in Section 5, we evaluate the developed framework by comparing it to other tools, some of which support (semi-)automated security analyses. Next, in Section 6, we discuss related work in the areas of threat modeling for CPSs, automated threat modeling, and information security ontologies. Finally, Section 7 concludes the article and provides suggestions for future research directions.

2. Methodology

This work is based on the *Design Science* approach by Hevner et al. (2004). The *Design Science* process, outlined in Figure 1, is on the one hand influenced by the *Environment*, and on the other hand, by a *Knowledge Base*

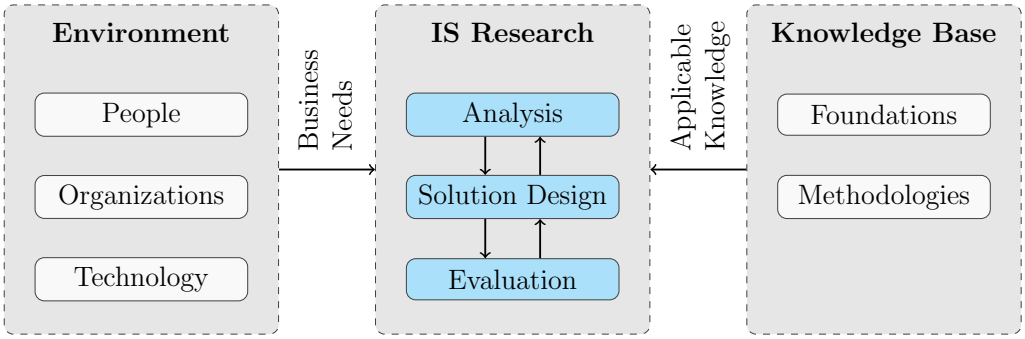

Figure 1: A simplified adaptation of the *Design Science in Information Systems Research* methodology based on (Hevner et al., 2004).

that provides foundations and methodologies. The research process is divided into three tasks, viz., *Analysis*, *Solution Design*, and *Evaluation*, which build on but also contribute to each other. In the *Analysis* phase, the research problem is investigated by means of the state of the art and the applicable knowledge. In the *Solution Design* phase, a solution approach is drafted, which may consist of theories, a particular design or other artifacts. The proposed approach is then examined and justified in the *Evaluation* phase.

To build a generic software testing process for automation applications, we first conducted qualitative, unstructured interviews with different roles from a major Austrian-based systems integrator, aligned the resulting model with existing standards for software testing and then discussed them with a company experienced in testing industrial automation software. More specifically, we interviewed four individuals to analyze their experiences and viewpoints on software testing in depth. All male interviewees are working in different departments and are filling various roles, viz., (i) Project Manager in the department *Automation Engineering*, (ii) Head of the department *PLC Programming*, (iii) Head of the department *SCADA*, and (iv) Head of Software Quality in the department *MES*.

Prior to conducting the interviews, we prepared our view of the organization's testing process based on background information that we had obtained from an independent research collaboration. We graphically mapped the testing process with the Business Process Model and Notation (BPMN) (OMG, 2011), as it is a standardized notation for the modeling of business processes. However, we used the prepared BPMN model only as a tool to guide discussions. Owing to the unstructured interviews, we were able to

focus on unanswered questions and blank spots. Furthermore, this setting encouraged the interviewees to expand on software testing issues and security concerns.

Subsequently, we reviewed our results together with an Austrian consultant company that specializes in software quality in order to ensure that the modeled software testing process is sound. In addition, we aligned the state of practice of testing automation applications with the internationally recognized standards for software testing, viz., the ISO/IEC/IEEE 29119 series, and relevant literature (Spillner et al., 2011; Lewis, 2008). In this way, we further refined our process model to establish a common ground for the security analysis.

In developing our security analysis framework, we analyzed and leveraged security models and standards, which are briefly described in the following. The analysis steps are based on the VDI/VDE 2182-1 (2011) guideline, which specifies a uniform approach to increase the security of automation systems. The VDI/VDE 2182-1 (2011) guideline starts with a *Structure Analysis* analysis to define the assessment scope. Subsequently, the following eight steps are performed: (i) identify assets, (ii) analyze threats, (iii) determine relevant security objectives, (iv) analyze and assess risks, (v) identify measures and assess effectiveness, (vi) select countermeasures, (vii) implement countermeasures, and (viii) perform process audit.

Moreover, we adopt STRIDE (Shostack, 2014), as it provides a structured model of threats and thereby supports the threat analysis step. STRIDE is an acronym that denotes a set of security threats, viz., (i) Spoofing, (ii) Tampering, (iii) Repudiation, (iv) Information Disclosure, (v) Denial of Service, and (vi) Elevation of Privilege (Shostack, 2014). Using STRIDE may facilitate discovering threats, especially when applying one of its variants, namely, STRIDE-per-element or STRIDE-per-interaction (Shostack, 2014). However, before applying STRIDE, we plot the data flows within the target of inspection by using a data flow diagram (DFD), consisting of the following elements: (i) processes (i.e., code that is executed), (ii) data flows (i.e., exchanged data between elements), (iii) data stores (i.e., elements that store data), (iv) and external entities (i.e., people) (Shostack, 2014).

Furthermore, we use ADTrees (Kordy et al., 2011) for graphically representing the enumerated threats in a tree structure. ADTrees extend attack trees (Schneier, 1999; Mauw and Oostdijk, 2006) by defense nodes, allowing users to analyze the security of a system from both an attacker's and defender's point of view (Kordy et al., 2011). Adequate tool support is

available, viz., the ADTool (Kordy et al., 2013a), which may foster wider adoption.

Taking into account that the testing approach differs between organizations, the modeling of data flows must be dynamic so that the threats can be accurately identified. As a result, we provide an ontology that allows users to model a DFD, representing the data flows within their testing process. Additionally, we created an ontology to model generic ADTrees that are based on the threat trees defined in (Shostack, 2014). To ease the process of modeling threats, we developed a tool, named *ADTGenerator*, that automatically extracts the DFD and generates ADTrees for chosen threat scenarios. The ADTree generator was developed in Scala and the source code, including both ontologies, is publicly available on GitHub[3].

Furthermore, we demonstrate how competence questions (cf. Table 2) can be answered by means of the developed framework. Due to space constraints, we can neither show the SPARQL queries that attempt to answer these questions, nor the corresponding query results. However, we provide the SPARQL queries as well as the knowledge base, which can be used to execute the queries, via the aforementioned GitHub repository[3].

Finally, we evaluate the framework by performing a comprehensive comparison with other state-of-the-art tools for threat modeling and security analyses.

3. Generic Software Testing Process for Automation Applications

In general, the process of testing industrial automation software is quite similar to that of testing traditional IT software. One of the main differences lies in the fact that industrial automation software runs on CPSs that integrate physical components in order to interact with the real-world (Baheti and Gill, 2011) (e.g., a robot arm as part of an assembly line). As a consequence, the system under test (SuT) consists of software that may run within a simulation or a testbed of the production system. Another consequence is that testing in the automation domain is still manually done on a regular basis due to the difficulty to automate specific tests and the lack of effective test measures (Dubey, 2011).

As, to the best of our knowledge, there is no industry standard specifically defining the testing process in PSE, we based our testing process for industrial

[3]ADTGenerator: `https://github.com/sbaresearch/adtgenerator`.

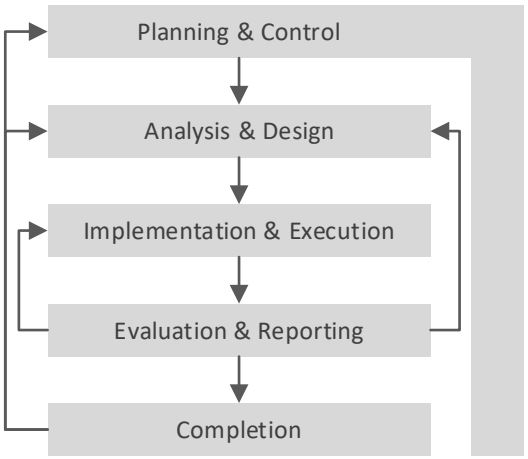

Figure 2: The high-level testing process (Graham et al., 2008; Spillner et al., 2011).

automation systems, which is shown in Figure 2, on the high-level software testing processes of (Graham et al., 2008) and (Spillner et al., 2011). In both works, the testing process consists of the following five tasks: (i) *Planning & Control*, (ii) *Analysis & Design*, (iii) *Implementation & Execution*, (iv) *Evaluation & Reporting*, and (v) *Completion*, which are executed in the testing phase of a software system. The *Planning & Control* activity spans over the other activities to enable the adaptation of the testing process if necessary. The results and artifacts of tasks are handed over to the next task of the process, for example as logical test cases and testing source code. Furthermore, to allow the control mechanism to work properly, particular results of the tasks are fed back to previous tasks as a basis for further decisions and to improve the specific testing process and quality.

Utilizing the process descriptions from (Spillner et al., 2011; Graham et al., 2008), we developed a detailed, yet comprehensible software testing process that especially considers requirements for industrial automation software. Very similar to non-automation software, testing an engineering solution are tested on different levels (Dubey, 2011) — from unit and module testing to on-site testing. However, the site acceptance testing level (Dubey, 2011) during the commissioning phase of a production system is out of the scope of this article. Figure 3 illustrates this software testing process. For further explanation, we will provide a description of the process steps and describe the involved stakeholder roles.

The first of the five BPMN swimlanes in Figure 3 represents the activities

of *Test Management*, which corresponds to the *Planning & Control* task in the high-level process illustrated in Figure 2. We chose this different name of the *Test Management* swimlane after interviewing the test expert of the consultancy company and aligning the process with the ISO/IEC/IEEE 29119 series to better conform to the standard. All other names follow the convention in the depicted high-level process. The management activities in this process define the test strategy (e.g., automated behavior-driven testing), create a test plan that considers the quality expectations, define the requirements for testing, and monitor and control the testing process. The second swimlane contains the *Test Analysis & Design* activities, where engineers analyze the test basis, derive test conditions, define the logical test cases and the test environment, and describe the test procedure. In contrast to the business software testing process, the testing process for industrial automation software has to describe the environment with its simulation or physical systems in the *Environment Description*. In the *Test Implementation & Execution* swimlane, the activities for implementing the test cases with the correct test data, preparing the environment and executing the tests, as well as collecting the test results are embedded. Our interviews and investigation showed that for automation software engineering it is crucial that the SuT is appropriately prepared before, and torn down after the test execution. This necessity is, e.g., for safety reasons due to a possible interaction between physical systems and employees or the physical damage a component may cause. Furthermore, physical systems and their components cannot be reset to a clean state as easily as software systems and often require engineers to perform manual activities. Therefore, we introduced the activities *Prepare SuT* and *Tear down SuT* which are separated from the preparation of the testing environment, covering the testing system, its tools, and the automation software itself. The fourth swimlane is the *Test Reporting* lane, which consists of creating the test report and evaluating the test results, or, in case of a test incident, the documentation of this incident. The last swimlane of the BPMN process contains the tasks for *Test Completion*, which include cleaning up the environment, archiving the tests and creating a completion report that includes the lessons learned from the test run.

After interviewing our partners, we compared the testing process with the ISO/IEC/IEEE 29119 standard and amended missing parts. For clarity reasons we combined the *Test Management Process* and the *Dynamic Test Process* and their relevant sub-processes to an integrated software testing process, but also left out the *Organizational Testing Process* that describes

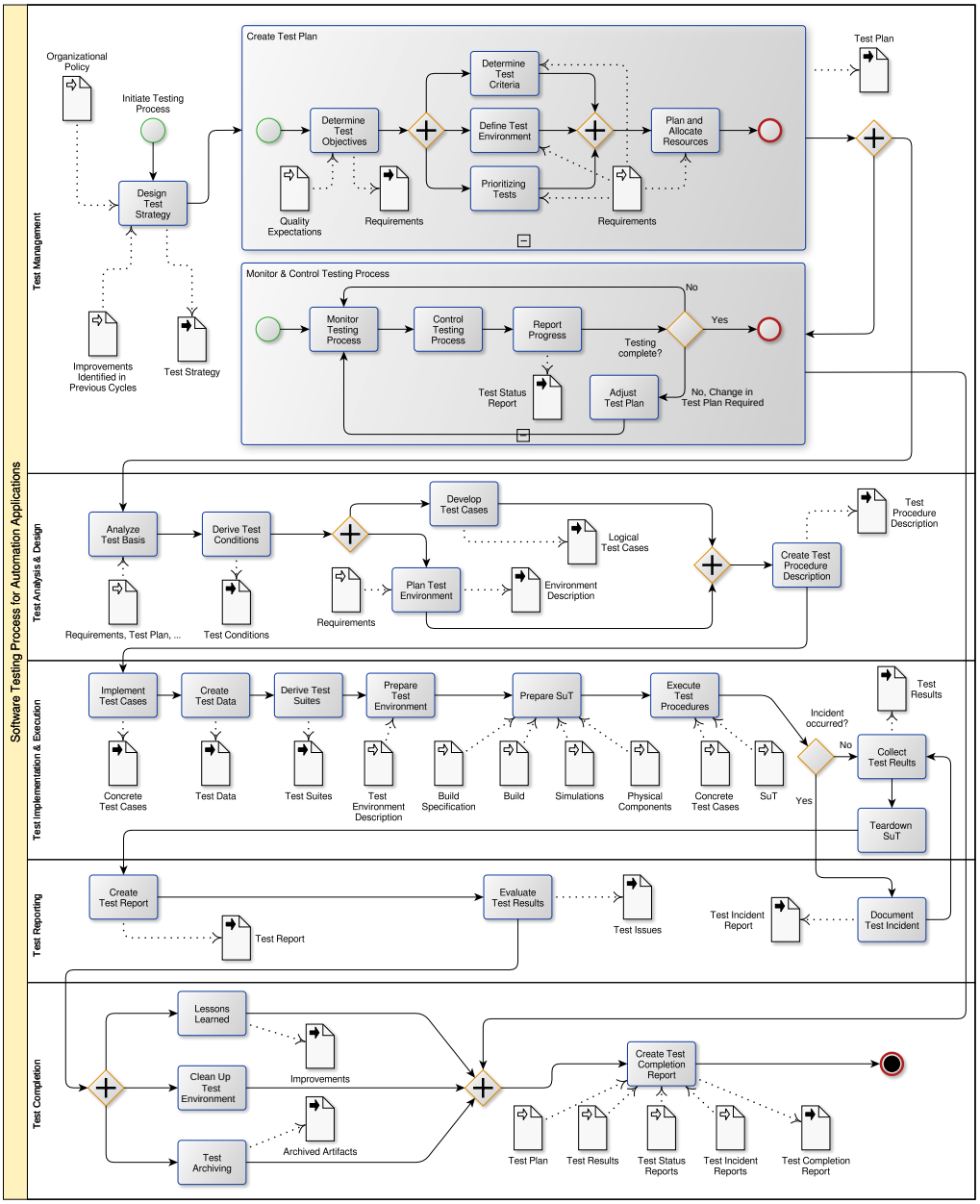

Figure 3: The testing process for industrial automation software based on (Spillner et al., 2011; Lewis, 2008; ISO/IEC/IEEE 29119-2, 2013).

processes on a higher level than the project level. Furthermore, we omitted a few tasks of the standard that seemed to make it less understandable in a combined form, such as consenting on the test plan. Nevertheless, our software testing process is a generic blueprint that still needs to be implemented according to the specific requirements of the respective automation engineering company that uses it. This also means that the testing process as depicted might not be rigorously followed, e.g., on the supervisory or control level.

Each of the tasks in our testing process is associated with a particular role. Table 1 defines these roles and their responsibilities. In practice, a single person often fills several roles, even though a clear separation of roles would be preferable (Winkler et al., 2018).

Abbrv.	Role	Responsibilities
PM	Project Manager	Project Controlling, Test Controlling
TM	Test Manager	Test Planning, Test Design, Test Management, Test Controlling
RM	Release Manager	Software Packaging, Software Releasing
D	Developer	SuT Code Implementation
T	Tester	Test Data Creation, Testcase Implementation
TAE	Test Automation Engineer	SuT Preparation & Tear Down, Environment Preparation & Tear Down, Test Execution
DE	Domain Expert	Test Data Provisioning, Business Testcase Derivation, Domain Knowledge Specification, Requirements Specification

Table 1: The roles and responsibilities in a software testing team (Winkler et al., 2018; ISO/IEC/IEEE 29119-2, 2013).

According to the developed testing process, the security analysis will focus on an organizational and high-level technical view of the testing process. Consequently, specific tools used as part of the testing process are out of scope but could be extended by users or future research. However, we do consider types of tools (e.g., test management tool) that are used for testing activities in the threat modeling process.

4. Security Analysis Framework

This section discusses a framework for the semi-automatic security analysis of software testing processes. First, an overview of the framework is given by describing its structure and how it can support an organization's efforts

to secure its software testing process. Second, we outline in Section 4.1 how we represent knowledge about (i) the software testing process, and (ii) potential threats including their respective countermeasures by using ontologies. Third, Section 4.2 demonstrates the steps of the security analysis and highlights the merits of the framework.

As stated in Section 2, the presented framework is based on the VDI/VDE 2182-1 (2011) guideline and aims to semi-automate multiple steps of the procedural method. As can be seen in Figure 4, the procedural method is cyclic in nature and the framework focuses on the first six steps, including the *Structure Analysis* (the remaining two steps marked with ⓧ are out of scope, as they cannot be automated).

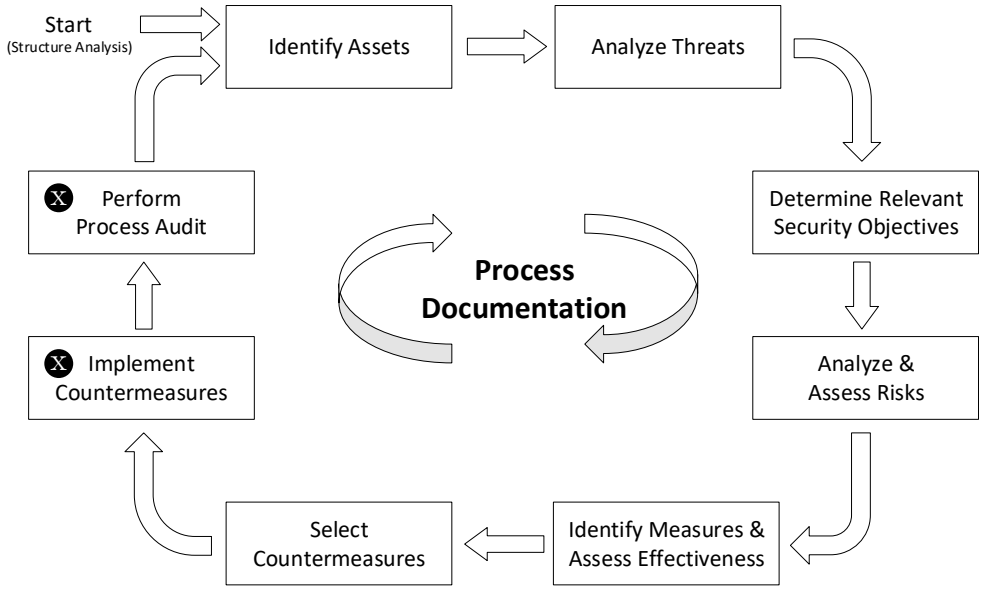

Figure 4: The procedural method, as described in and adapted from (VDI/VDE 2182-1, 2011); steps marked with ⓧ are out of scope of the proposed framework.

Before the procedure can be performed, a structure analysis must be conducted in order to define the assessment scope. The result of this analysis is a specification of the *target of inspection*, i.e., a detailed description of the object (e.g., device) to be assessed. Furthermore, the specific moments in time when the security analysis ought to be conducted is dictated by certain intervals (e.g., regular audits) or events (e.g., discovered vulnerabilities, a

major change in the target of inspection). To ensure a traceable process, the VDI/VDE 2182-1 (2011) guideline suggests creating a *process documentation* that is developed step by step throughout the procedural method. (VDI/VDE 2182-1, 2011)

Table 2 summarizes the contribution of this work, or, more precisely, how the herein presented framework supports the phases of the procedure method described in the VDI/VDE 2182-1 (2011) guideline (cf. Figure 4). Since we adopt an ontological approach, the knowledge models described in the next section lay the foundation for the framework. For this very reason, the framework also facilitates creating the process documentation.

4.1. Knowledge Representation

Due to the fact that the software testing process steps, used tools, and defense mechanisms in place vary among organizations, the target of inspection is dynamic to a certain extent. To ensure that the scope of the security analysis is clearly defined, yet flexible enough to analyze various software testing processes, we developed a tool-supported semi-automated threat modeling approach.

The foundation of this approach consists of two ontologies to model, on the one hand, data flows within the software testing process, and ADTrees on the other. Both ontologies were developed in the Web Ontology Language (OWL) by using the open-source ontology editor Protégé (Noy et al., 2001). Although we provide a set of individuals for both ontologies in advance as a starting point, we expect users to adapt the modeled DFD to their testing process, and to extend the predefined ADTrees if necessary.

The DFD ontology is formed by the typical elements of a DFD, viz., `Process`, `DataFlow`, `DataStore`, and `ExternalEntity` classes. Furthermore, an `Asset` class, with the subclasses `Hardware`, `Software`, and `Document` are used to associate individuals of these types with elements of the modeled DFD. Assets can be linked to DFD elements, via the object property `usedBy`. Furthermore, bidirectional data flows are denoted by adding the `flows` object property twice (one for each endpoint), while unidirectional data flows can take on the object properties `flowsFrom` and `flowsTo`.

On the other hand, the ADT (ADTrees) ontology consists of the following classes: `AttackNode`, `DefenseNode`, `Goal` (i.e., the root node of the ADTree), `TargetElement` (i.e., a DFD element type), `Threat` (according to STRIDE), and `Connector` with the two subclasses `AndConnector` and `OrConnector`

Phase	Answered Question	Semi-Auto.	Provided Support
Structure Analysis	*What is the target of inspection?*	◐	The generic DFD for software testing depicted in Figure 5 is modeled with the DFD ontology and can be adapted if needed. The knowledge model can be queried to gain insights into the organization's testing process.
Identify Assets	*What are the assets to be protected?*	◐	Assets listed in Table 3 are modeled with the DFD ontology and can be adapted if needed. The knowledge model can be queried to gain insights into the testing process and involved assets.
Analyze Threats	*What are the threats to assets?*	●	ADTrees can be automatically generated based on the modeled DFD and generic ADTrees. The threat model can be easily shared and extended to reflect new and organization-specific threats.
Determine Security Objectives	*Which security objectives are at risk?*	●	The security objectives can be automatically identified by querying the knowledge models.
Analyze & Assess Risks	*Which risks need to be treated?*	○	The generated ADTrees can be imported into ADTool (Kordy et al., 2013a) in order to conduct a (manual) quantitative risk assessment (Kordy et al., 2013b).
Identify Measures & Assess Effectiveness	*What are the countermeasures to mitigate the risks?*	◐	The generic countermeasures are modeled with the ADT ontology and can be adapted if needed. Users can query the knowledge base to identify countermeasures to protect assets.
Select Countermeasures	*What are the most cost-effective countermeasures?*	○	The automatically generated ADTrees can be analyzed with the quantitative risk assessment features of the ADTool (Kordy et al., 2013a), allowing to identify cost-effective countermeasures.

Legend: ● applicable, ◐ partially applicable, ○ not applicable.

Table 2: An overview of how the proposed framework supports the security analysis according to VDI/VDE 2182-1 (2011).

for representing conjunctive and disjunctive refinements of nodes, respectively. Goals, attack, and defense nodes are connected to connectors, and vice versa, by using the object property `connectedTo`. Furthermore, goals point to DFD target elements and STRIDE threats with the object properties `hasTarget` and `hasType`, respectively. These two object properties can also be used by subclasses of `Connector` in order to reference assets that are put at risk by certain threats. For the purpose of identifying the most cost-effective countermeasures, attack and defense nodes can have the data properties `successProbability` and `cost`.

To provide users with a generic set of threats, including corresponding countermeasures, we modeled the threat trees defined in (Shostack, 2014) with the ADT ontology. Although Shostack (2014) provides mitigation strategies that are geared towards developers and IT operations staff, we focus only on the latter, as we assume that users of the proposed threat modeling approach cannot modify the software used as part of the software testing process (e.g., test management tool). Owing to the modeled DFD and threat trees, some threat scenarios (e.g., obtain test data) can be automatically derived by executing SPARQL queries, due to the fact that the DFD elements that are associated with assets (e.g., test data) and the modeled threats can be queried jointly. However, to cover multi-stage attacks (e.g., covert sabotage of the test code), we modeled additional ADTrees that represent specific threat scenarios pertaining to the software testing process for automation applications. These predefined ADTrees can reference other threat trees as subtrees, but the acyclic nature of the trees must be preserved.

4.2. Security Analysis Steps

This subsection first outlines the assessment scope and then shows how the framework allows to carry out a security analysis of the software testing process in a semi-automated manner to answer the questions outlined in Table 2.

4.2.1. Structure Analysis

In this work, the target of inspection is the software testing process of automation applications. Thus, other activities of CPS testing, such as hardware testing, are out of scope. More specifically, we focus on analyzing the security of the testing process described in Section 3. Furthermore, the security analysis covers two levels of testing, namely unit testing, and integration testing. Other levels of testing, e.g., factory acceptance testing (FAT), are

not covered as they are often too diverse to include in a general model but could be considered in future work.

To further break down how information *may* be transferred, we modeled the data flows of the software testing process explained in the previous section by using a DFD (cf. Figure 5). We assigned each element in the DFD an identifier so that we can reference them throughout the security analysis; a way of dealing with references in a STRIDE-based threat analysis, which we adopted from Khan et al. (2017). The DFD depicted in Figure 5 was also modeled with the DFD ontology in order to provide users a starting point for the structure analysis. In this way, they can adapt the modeled DFD so that the target of inspection accurately reflects their setting. For instance, if no test data generator is used, or instead of one source code repository, two are deployed, the modeled DFD can be easily refined by adding or removing the respective process elements. Moreover, the target of inspection can be expanded (e.g., to consider IT operations components for the purpose of DevOps) by extending the DFD ontology.

As can be seen in Figure 5, the tools used during the software testing activities are modeled as processes, while the data repositories are denoted as data stores. Furthermore, we used external entities for modeling the roles in a software testing team and data flows to indicate communication among tools, repositories, and roles. Owing to the modeled DFD, we can execute SPARQL queries that can provide answers to a variety of questions related to the target of inspection, such as: *Which processes, external entities or data stores transfer data to the test management tool?*

4.2.2. Asset Identification

The first step of the procedural method is to identify assets within the assessment scope. The VDI/VDE 2182-1 (2011) guideline states that the components of automation device(s), the communication infrastructure (e.g., data flows) and the legal positions (i.e., claims that may arise from security incidents) have to be considered. (VDI/VDE 2182-1, 2011)

For the identification of assets involved in the software testing process, we utilized the BPMN diagram (cf. Figure 3) and the DFD (cf. Figure 5). In particular, the data inputs and outputs depicted in the BPMN diagram, except simulations, physical components, and the build, were classified as documents. Furthermore, the DFD process and data store elements were categorized as hardware and software assets.

Since all this information is modeled in the DFD ontology, we can derive

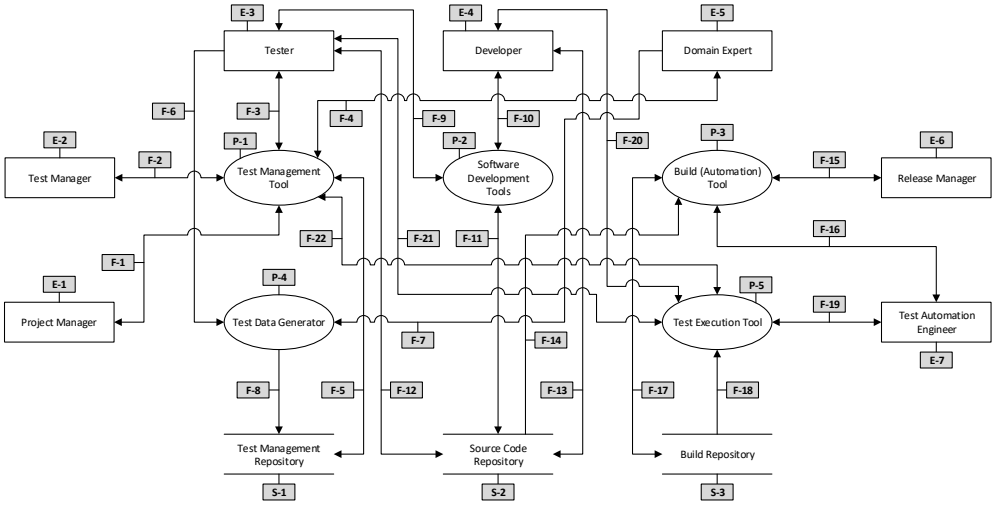

Figure 5: A data flow diagram of the software testing process.

valuable know-how about the assets at risk, simply by executing SPARQL queries. On the one hand, these modeled links allow deducing the attack targets for a given asset. On the other hand, we can determine which role (i.e., external entity) has access to which asset. Furthermore, additional knowledge about possible attack strategies (e.g., the target's attractiveness, defined by the number of involved assets) may be gained by using a semantic reasoner.

Besides assigning the DFD-IDs and authorized roles to each asset, we also analyzed the potential legal implications of a compromise thereof. Hence, we checked the potential posture that an organization might take in the event of litigation that arises from compromised assets. In particular, the following legal positions are examined: (i) protection of know-how, (ii) product liability, and (iii) safety. For instance, a breach of the confidentiality of documents may represent an infringement of the organization's intellectual property rights, while a violation of the integrity may lead to claims to be filed against the organization due to safety issues or product defects, causing, for example, machinery breakdowns.

The identified assets, together with the DFD-IDs, legal aspects to be considered, and the roles authorized to use them, are listed in Table 3.

	#	Asset	DFD-IDs	Legal Position			Authorized Roles						
				K	L	S	PM	TM	RM	D	T	TAE	DE
Hardware	1	Test management server	P-1										
	2	Test management repository server	S-1		●								
	3	Source code repository server	S-2		●								
	4	Build (automation) server	P-3										
	5	Build repository server	S-3		●								
	6	Physical components	-	●		●					●		●
Software	7	Test management tool	P-1				●	●			●		●
	8	Test management repository	S-1		●								
	9	Test data generator	P-4	●		●					●		●
	10	Software development tools	P-2							●	●		
	11	Source code	P-2, S-2, F-11, F-12, F-13, F-14	●	●	●				●	●		
	12	Source code repository	S-2		●					●	●		
	13	Build (automation) tool	P-3						●			●	
	14	Build	P-3, F-17, S-3, F-18	●	●	●							
	15	Build repository	S-3		●								
	16	Simulations	F-4, F-21, P-5	●	●	●					●		●
	17	Test execution tool	P-5		●					●	●	●	
Document	18	Organizational policy	F-1, F-2, F-5, P-1, S-1				●	●					
	19	Test strategy	F-1, F-2, F-5, P-1, S-1	●			●	●					
	20	Test plan	F-1, F-2, F-5, P-1, S-1	●			●	●					
	21	Test status report	F-2, F-3, F-4, F-5, P-1, S-1					●			●		●
	22	Test conditions	F-2, F-3, F-5, P-1, S-1	●				●			●		
	23	Logical test cases	F-3, F-4, F-5, P-1, S-1	●	●	●					●		●
	24	Test environment description	F-3, F-4, F-5, P-1, S-1	●							●		●
	25	Test procedure description	F-3, F-5, P-1, S-1	●							●		
	26	Concrete test cases	F-3, F-4, F-5, P-1, S-1	●	●	●					●		●
	27	Test data	F-6, F-7, F-8, P-4, S-1	●	●	●					●		●
	28	Test suites	F-3, F-5, P-1, S-1								●		
	29	Build specification	F-15, F-17, P-3, S-3	●	●	●			●			●	
	30	Test incident report	F-2, F-3, F-5, P-1, S-1	●	●			●			●		
	31	Test results	F-2, F-5, F-19, F-20, F-21, F-22, P-5, P-1, S-1	●	●			●		●	●	●	
	32	Test report	F-2, F-3, F-5, P-1, S-1	●				●			●		
	33	Test issues	F-1, F-2, F-3, F-5, P-1, S-1	●	●		●	●			●		
	34	Improvements report	F-1, F-2, F-3, F-5, P-1, S-1	●		●		●			●		
	35	Archived artifacts	F-2, F-5, F-17, P-1, P-3, S-1, S-3	●	●			●					
	36	Test completion report	F-1, F-2, F-3, F-5, P-1, S-1	●			●	●			●		

Table 3: The identified assets and the corresponding IDs of the involved elements in the DFD (cf. Figure 5), including a consideration of legal positions (K = protection of know-how, L = product liability, S = safety) and the roles of the software testing team (cf. Table 1 for role abbreviations) that are authorized to access the assets.

4.2.3. Threat Analysis

According to the VDI/VDE 2182-1 (2011) guideline, the threat analysis step is performed in two phases, viz., the identification of threats and the creation of a threat matrix. Prior to conducting the threat modeling activities, we will first discuss potential threat actors, their goals and attack targets.

Attacker Profiles. As past incidents in the industrial sector have shown, common threat actors are organizations with nation-state resources and criminal groups or individuals (Miller and Rowe, 2012; Langner, 2013; McLaughlin et al., 2016). Since these attacks are often launched in a targeted fashion, a special focus should be placed on Advanced Persistent Threats (APTs), i.e., threats that emanate from attackers who pursue a multi-staged attack strategy, targeting specific organizations with the aim to remain undetected for an indefinite period.

In general, the adversary's goals to attack the testing process for industrial automation software can be categorized into two groups, information theft and (covert) sabotage. The motives behind attacks that aim to steal testing artifacts may be (i) monetary profit (e.g., imitating a competitor's SuT), (ii) reputation damage (e.g., a data breach involving the loss of artifacts), and (iii) reconnaissance (e.g., obtaining knowledge about the SuT for subsequent steps in the kill chain). On the other hand, motives behind sabotage attacks targeting the testing process may be as follows: (i) equipment damage (e.g., increased deterioration of systems), (ii) environment and human safety risks (e.g., sabotaged test execution on real hardware to put human health at risk), (iii) manipulation of manufactured products (e.g., decreased product quality), (iv) denial of service (e.g., impeding the PSE process). Finally, gaining credit can be a further motive behind a cyber attack.

Table 4 summarizes the considered threat actors in the context of testing automation applications, including their estimated capabilities and potential motives behind cyber attacks.

Note that an adversary may also exploit a vulnerable testing process for industrial automation software to attack the integrator's customer who is operating the plant. Consequently, a lack of proper security measures to protect the testing process may lead to severe consequences, not only for the integrator who is in charge of the testing process but also for operators who rely on automation software to control physical processes.

Threat Actor	Capabilities	Credit	Monetary Profit	Reputation Damage	Reconnaissance	Equipment Damage	Safety Risks	Product Manipulation	PSE DoS
Basic User	Low	●	●						
Insider	Medium			●	●	●	●		●
Competitor	Medium		●	●	●	●		●	●
Hacktivist	Medium	●		●	●	●			●
Terrorist	Medium	●			●		●		
Cybercriminal	Medium		●	●					
Organized cybercrime group	High		●	●	●	●	●	●	●
Nation-state threat group	High				●	●	●	●	●

Table 4: The attacker profiles relevant to the software testing process.

Threat Modeling. As already mentioned in Section 2, we adopt a STRIDE-based threat modeling approach. In particular, we use STRIDE-per-element (Shostack, 2014) to facilitate the automated identification of threats. In essence, this variant of STRIDE predetermines which types of threats are relevant to which DFD elements (cf. Table 5). Based on this, threat trees (Shostack, 2014) can be defined, which provide generic attack vectors that ease the threat modeling process.

DFD Element	S	T	R	I	D	E
External Entity	●		●			
Process	●	●	●	●	●	●
Data Flow		●		●	●	
Data Store		●	◐	●	●	

Legend: ● applicable, ◐ partially applicable.

Table 5: The applicability of threats to DFD elements according to STRIDE-per-element (Shostack, 2014).

Due to the fact that we modeled the threat trees with the ADT ontology, we can derive specific threat scenarios that show, by means of ADTrees, how an attacker's goal may be achieved. Furthermore, we grouped these threat scenarios by attacker goals, i.e., information theft or (covert) sabotage, and also analyzed potential consequences. Table 6 shows the results of this threat

modeling activity. Since these threat scenarios describe the actions that are executed with malicious intent in order to achieve a certain goal, the attacker constitutes the proponent.

To formally represent ADTrees in Table 6, we adopt attack–defense terms (ADTerms) (Kordy et al., 2011). In a nutshell, ADTerms are elements of T_Σ, where $T_\Sigma = T_\Sigma^{\mathrm{p}} \cup T_\Sigma^{\mathrm{o}}$ (the union of the set of proponent's and the set of opponent's ADTerms), and $\Sigma = (S, F)$, i.e., an AD–signature representing a pair consisting of a set of types, $S = \{p, o\}$, and a set of function symbols, F, which is composed of disjunctive ($\vee$) and conjunctive ($\wedge$) refinement operators, and counteractions (c) for both the proponent (p) and the opponent (o) (Kordy et al., 2011). Furthermore, the ADTerm used to represent the ADTree T is denoted by $\iota(T)$ (Kordy et al., 2011).

Thus, we denote the ADTerm used to represent a generic ADTree, which is based on a threat tree defined in (Shostack, 2014) and concerns one or multiple DFD elements depicted in Figure 5, by $\iota(T^{\dagger}_{i,j})$, where i represents the DFD-ID and j the threat according to STRIDE. For the sake of brevity, we group generic ADTrees of a certain threat type by assets, $\iota(T^{*}_{a,j}) = \vee^{\mathrm{p}}\big(\iota(T^{\dagger}_{\mathrm{i_1,j}}), \ldots, \iota(T^{\dagger}_{\mathrm{i_k,j}})\big)$, where a represents the assigned number of the asset (cf. Table 3), i the DFD-ID, j the threat type (if applicable to the DFD element), and $k \geq 1$. For instance, $\iota(T^{*}_{27,T}) = \vee^{\mathrm{p}}\big(\iota(T^{\dagger}_{\mathrm{F\text{-}6,T}}), \iota(T^{\dagger}_{\mathrm{F\text{-}7,T}}), \iota(T^{\dagger}_{\mathrm{F\text{-}8,T}}), \iota(T^{\dagger}_{\mathrm{P\text{-}4,T}}), \iota(T^{\dagger}_{\mathrm{S\text{-}1,T}})\big)$ is the ADTree consisting of the threat trees for tampering test data (asset no. 27). Moreover, the ADTerm used to represent an ADTree for a threat scenario (cf. Table 6) is denoted by $\iota(T^{\ddagger}_{n})$, where n is the number of the respective threat scenario.

	#	Goal	ADTerm	Potential consequences
Information Theft	1	Obtain source code	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-11,I}}), \iota(T^{\dagger}_{\text{F-12,I}}), \iota(T^{\dagger}_{\text{F-13,I}}), \iota(T^{\dagger}_{\text{F-14,I}}), \iota(T^{\dagger}_{\text{P-2,I}}), \iota(T^{\dagger}_{\text{S-2,I}})\right]$	The disclosure of algorithms, design concepts, and configurations would allow adversaries to gain profound knowledge about the developed automation application. This knowledge can be used to replicate the application, develop exploits that take advantage of newly discovered weaknesses, and may also provide insights into the industrial processes under control, especially if the application is custom-tailored. Furthermore, an understanding of the code coverage may incite attackers to manipulate only those parts of the code that are not executed during test runs, so that the sabotage remains undetected.
	2	Obtain build	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-17,I}}), \iota(T^{\dagger}_{\text{F-18,I}}), \iota(T^{\dagger}_{\text{P-3,I}}), \iota(T^{\dagger}_{\text{S-3,I}})\right]$	Capturing the binary would allow adversaries to perform reverse engineering activities, such as static analysis, which may result in similar consequences as the theft of source code has.
	3	Obtain build specification	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-15,I}}), \iota(T^{\dagger}_{\text{F-17,I}}), \iota(T^{\dagger}_{\text{F-19,I}}), \iota(T^{\dagger}_{\text{P-3,I}}), \iota(T^{\dagger}_{\text{S-3,I}})\right]$	Various malicious actions can be undertaken based on knowledge gained about the build process, as a result of obtaining the build settings. In particular, the configured build steps, parameters, build schedule, versioning information, build failure conditions, clean-up rules, dependencies, and reporting information are valuable targets to be used for reconnaissance purposes. Furthermore, captured version control settings and SSH keys may allow adversaries to pivot to other systems. Settings related to code signing may be abused by attackers to disguise malware as benign, legitimate applications, as was the case with Stuxnet (Falliere et al., 2011; Langner, 2013).
	4	Obtain simulations	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-4,I}}), \iota(T^{\dagger}_{\text{F-21,I}}), \iota(T^{\dagger}_{\text{P-5,I}})\right]$	Depending on the type of the obtained simulation, adversaries may acquire comprehensive knowledge about systems, industrial processes, and even the overall plant. This knowledge can either be used to recreate components and manufacturing processes or as an attack preparation step. To give two examples of the former, a 3D simulation of a robot provides details about its mechanical structure, whereas a plant simulation could reveal the material flow and the resources used as part of the manufacturing process. On the other hand, system simulations may be exploited to learn the physical dynamics, allowing opponents to launch covert attacks (cf., for instance, (de Sá et al., 2017)). Furthermore, simulations may also reveal the features of physical components (e.g., payload or rotation capabilities of a robotic arm), which represent valuable information for performing attacks that aim to cause physical damages or injure individuals. In this context, adversaries can also benefit from simulating failures caused by attacks, ensuring that physical damage is induced in a controlled manner.
	5	Obtain test plan	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-1,I}}), \iota(T^{\dagger}_{\text{F-2,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\right]$	According to the ISO/IEC/IEEE 29119 standard (ISO/IEC/IEEE 29119-3, 2013), the test plan includes, inter alia, the features to be tested, a risk register, roles and responsibilities, and the overall testing approach. If this information were to fall into unfriendly hands, adversaries might be able to increase the sophistication of planned attacks significantly. For example, depending on the attacker's objective, the test scope can be used to sabotage either specific test items for the purpose of causing harm during the test execution (e.g., destroying the SuT) or manipulating only the ones that are excluded from testing in order to go unnoticed during PSE and wreak havoc during plant operation. Furthermore, the risk register may suggest additional targets that may be worth attacking, whereas staffing information could be exploited to attack particular users in order to gain a foothold in specific testing activities. Details about the overall testing approach (e.g., schedules, third-party tools used for testing, paths of stored documents, versioning information) could be a target for further reconnaissance activities.
	6	Obtain test cases	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-3,I}}), \iota(T^{\dagger}_{\text{F-4,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\right]$	Attackers being equipped with knowledge about test cases may be able to understand the happy paths, but also how the SuT can be run into error conditions. In particular, this know-how is useful for exploit development.
	7	Obtain test environment description	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-3,I}}), \iota(T^{\dagger}_{\text{F-4,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\right]$	Since the test environment description comprises details related to the used testbed, including information about the hardware, software, and configurations, the disclosure thereof may allow adversaries to gain knowledge about the structure and inner workings of physical components. This knowledge, again, may be useful for creating product clones or reconnaissance.
	8	Obtain test data	$\vee^{\mathrm{P}}\left[\iota(T^{\dagger}_{\text{F-6,I}}), \iota(T^{\dagger}_{\text{F-7,I}}), \iota(T^{\dagger}_{\text{F-8,I}}), \iota(T^{\dagger}_{\text{P-4,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\right]$	Test data can be composed of highly sensitive, business-critical information. While data privacy may be an issue when it comes to using realistic, unmasked test data for typical IT software, inputs for test cases related to automation applications may reveal valuable know-how about the manufacturing processes and systems. To give a concrete example, the interviewed company partner uses real-world data collected from customers during plant operation for the purpose of testing customized MES applications. Since these data dumps contain, inter alia, details about past orders, production logs, as well as scheduling and sequencing information, the leakage thereof as a result of industrial espionage activities would obviously damage the company's customers.

	#	Goal	ADTerm	Potential consequences
Information Theft	9	Obtain test incident report	$\vee^{\mathrm{P}}\big[\iota(T^{\dagger}_{\text{F-3,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\big]$	Incidents that occur during testing (e.g., unexpected failures) are recorded in a test incident report (ISO/IEC/IEEE 29119-3, 2013). Furthermore, these reports typically document if, and how to reproduce the incidents, while also providing supplementary material (e.g., logs, screenshots) (ISO/IEC/IEEE 29119-3, 2013). The disclosure of this information may allow adversaries to force the SuT into a critical state, provided that the exploited issues have not been fixed before releasing the software.
	10	Obtain test results	$\vee^{\mathrm{P}}\big[\iota(T^{\dagger}_{\text{F-2,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{F-19,I}}), \iota(T^{\dagger}_{\text{F-20,I}}), \iota(T^{\dagger}_{\text{F-21,I}}), \iota(T^{\dagger}_{\text{F-22,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{P-5,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\big]$	Stealing test results may constitute preparatory work an adversary undertakes as part of an attack that is launched during plant operation. In particular, failed tests would indicate that the SuT exhibits undesired behavior, providing attackers clues on potential flaws that might still be exploitable after the software release if the corresponding bugs have not been fixed in a timely manner.
	11	Obtain test report	$\vee^{\mathrm{P}}\big[\iota(T^{\dagger}_{\text{F-2,I}}), \iota(T^{\dagger}_{\text{F-3,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{S-1,I}})\big]$	The test report summarizes test activities, outlines the progress made based on the achieved test results, and discusses incidents as well as new or changed risks that the team faces in future iterations (Spillner et al., 2011; ISO/IEC/IEEE 29119-3, 2013). While stealing the test report is certainly not an attacker's top-priority goal, the test metrics, defect summary, open issues with their estimated severity, and the distribution of discovered bugs in software modules, still provides valuable hints where to search for vulnerabilities.
	12	Obtain archived artifacts	$\vee^{\mathrm{P}}\big[\iota(T^{\dagger}_{\text{F-2,I}}), \iota(T^{\dagger}_{\text{F-5,I}}), \iota(T^{\dagger}_{\text{F-17,I}}), \iota(T^{\dagger}_{\text{P-1,I}}), \iota(T^{\dagger}_{\text{P-3,I}}), \iota(T^{\dagger}_{\text{S-1,I}}), \iota(T_{\text{S-3,I}})\big]$	Upon test completion, test assets, such as test plans, test procedures, test data, and artifacts related to the test environment, are archived for reuse (ISO/IEC/IEEE 29119-2, 2013). The disclosure of archived artifacts has similar consequences as the theft of individual assets that are included in the archived bundle. However, the aggregated nature of archived test artifacts makes them generally more attractive targets for information theft.
(Covert) Sabotage	13	Inject malicious code into source code	$\wedge^{\mathrm{P}}\big[\iota(T^{\ddagger}_{1}), \vee^{\mathrm{P}}\big(\iota(T^{*}_{11,S}), \iota(T^{*}_{11,T}), \iota(T^{*}_{11,E})\big), \wedge^{\mathrm{P}}\big(\iota(T^{*}_{13,T}), \iota(T^{*}_{17,T}), \iota(T^{*}_{26,T}), \iota(T^{*}_{30,T}), \iota(T^{*}_{31,T})\big)\big]$	Injecting malicious code into the codebase is certainly a worthy goal for attackers since the resulting damage can have devastating effects for victims. For instance, attackers can tamper with the code to put either the SuT or, if the malicious code gets through QA and then shipped, the automation application in production into a critical state, potentially destructing equipment and putting human health at risk. Furthermore, attackers may be able to plant backdoors in order to gain command and control of the targeted system during plant operation. This would then also enable adversaries to exfiltrate data related to the industrial process under control.
	14	Substitute SuT	$\wedge^{\mathrm{P}}\big[\iota(T^{\ddagger}_{3}), \vee^{\mathrm{P}}\big(\iota(T^{*}_{14,S}), \iota(T^{*}_{14,T}), \iota(T^{*}_{14,E})\big), \wedge^{\mathrm{P}}\big(\iota(T^{*}_{13,T}), \iota(T^{*}_{15,T}), \iota(T^{*}_{30,T}), \iota(T^{*}_{31,T}), \iota(T^{*}_{35,T})\big)\big]$	If an attacker is able to substitute the SuT with a stale version and, in further consequence, conceals this malicious act, the organization may ship a defective software product. Besides product liability claims that may arise from failing to meet quality standards, the organization may face a project delay and cost overrun due to the caused impediments, provided that the attack was detected during a subsequent phase of the PSE process.
	15	Manipulate test environment description	$\wedge^{\mathrm{P}}\big[\iota(T^{\ddagger}_{5}), \vee^{\mathrm{P}}\big(\iota(T^{*}_{24,S}), \iota(T^{*}_{24,T}), \iota(T^{*}_{24,E})\big), \wedge^{\mathrm{P}}\big(\iota(T^{*}_{17,T}), \iota(T^{*}_{24,T}), \iota(T^{*}_{30,T}), \iota(T^{*}_{35,T})\big)\big]$	Manipulating the test environment description may trick domain experts and users into misconfiguring the testbed. While errors in the setup of hardware or software may be obvious to domain experts, an incorrect parametrization may be more difficult to spot. A misconfigured testbed could destroy physical components either during the preparation of the test environment or during text execution and certainly impedes the testing process.
	16	Manipulate test results	$\wedge^{\mathrm{P}}\big[\wedge^{\mathrm{P}}\big(\iota(T^{\ddagger}_{5}), \iota(T^{\ddagger}_{6})\big), \vee^{\mathrm{P}}\big(\iota(T^{*}_{31,S}), \iota(T^{*}_{31,T}), \iota(T^{*}_{31,E})\big), \wedge^{\mathrm{P}}\big(\iota(T^{*}_{30,T}), \iota(T^{*}_{32,T})\big)\big]$	Adversaries who aim to manipulate the test results may try to conceal malicious acts by disguising failed tests as passed ones. In this way, stakeholders of the testing process may erroneously think that the SuT fulfills the QA requirements (potentially leading to PSE process impediments), successfully buried malware, and ultimately an undesirable behavior of the automation application.
	17	Manipulate archived artifacts	$\wedge^{\mathrm{P}}\big[\wedge^{\mathrm{P}}\big(\iota(T^{*}_{18,I}), \iota(T^{*}_{19,I})\big), \vee^{\mathrm{P}}\big(\iota(T^{*}_{35,S}), \iota(T^{*}_{35,T}), \iota(T^{*}_{35,E})\big)\big]$	If manipulated artifacts are archived, subsequent iterations of the testing process may be fundamentally flawed if they use these artifacts as a basis. Furthermore, potential legal or regulatory requirements for documenting information related to QA may not be met.

Table 6: The identified threat scenarios (no. 1–12: information theft, no. 13–17: covert sabotage).

The construction of ADTrees for threat scenarios related to information theft (no. 1–12) is fully automated since we can retrieve those generic ADTrees with the STRIDE threat *Information Disclosure* for the DFD elements that use the asset at risk. However, the threat scenarios that belong to the (covert) sabotage group (no. 13–17) are more complex and, in further consequence, deriving them requires prior modeling with the ADT ontology. In particular, we assume that for sabotaging an asset in a covert manner, an adversary must launch reconnaissance attacks prior to the actual asset manipulation, and finally also execute certain attacks to cover up the sabotage. Thus, ADTrees for representing these threat scenarios are composed of multiple subtrees that are used to accomplish the attack phases (i) reconnaissance, (ii) actual sabotage of the asset, and (iii) concealment. All (covert) sabotage threat scenarios described in Table 6 comprise these phases, except threat scenario no. 17, as the target of inspection does not include the reuse of archived artifacts (cf. Figure 3) and therefore lacks the concealment phase. While the subtrees for the sabotage phase can be constructed in a fully automated fashion by retrieving the generic ADTrees with the STRIDE threats *Spoofing*, *Tampering*, and *Elevation of Privilege* that are applicable to the asset at risk, we had to model the subtrees that belong to the other two phases. More specifically, we modeled links between the ADTrees representing these two phases and the assets that must either be obtained prior to (reconnaissance) or manipulated after the sabotage (concealment) for each (covert) sabotage threat scenario. To give an example, the covert manipulation of test results (cf. threat scenario no. 16 in Table 6) can be performed by (i) obtaining the test plan and test cases to provide the attacker with an accurate understanding of the tests to be executed, (ii) spoofing, tampering with or elevating privileges of applicable DFD elements that use the test results, and (iii) tampering with the test incident report and test report to conceal the sabotage.

Since the ADTrees described in Table 6 can be derived from the ontology, users can select a threat scenario and start the ADTGenerator in order to generate an XML file, which includes the corresponding ADTree. Upon completion, they can import the generated XML file into the ADTool (Kordy et al., 2013a). In this way, users gain a comprehensive view about potential threats and can then start directly to inspect each attack path in detail, focusing on the particular circumstances involved (e.g., system's software version). For instance, Figure 6 depicts an excerpt of the ADTree for obtaining the source code (threat scenario no. 1 in Table 6).

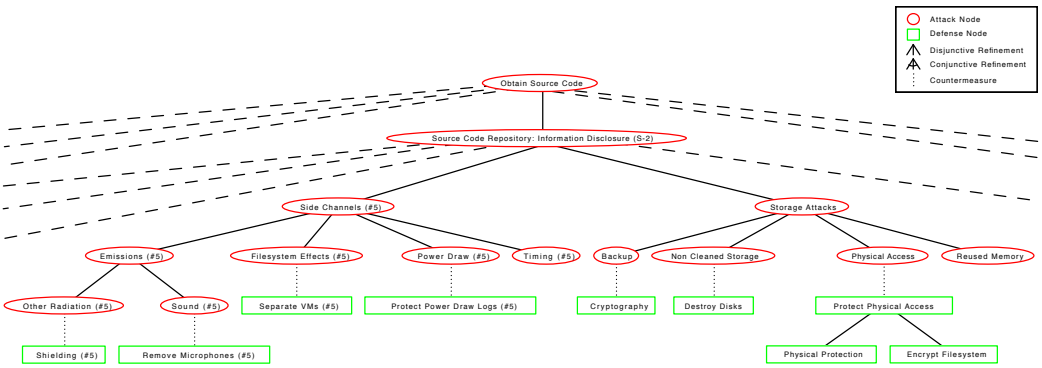

Figure 6: An excerpt of the ADTree for obtaining the source code.

The VDI/VDE 2182-1 (2011) guideline prescribes to create a threat matrix to complete the threat analysis step. This threat matrix may include the threats, which assets are affected, causes, vulnerabilities, and the resulting direct consequences. The information required to create such a threat matrix, excluding concrete vulnerabilities, can be obtained from the knowledge base or from Table 3 and Table 6. However, we omit to present a dedicated threat matrix in this work due to space constraints.

4.2.4. Security Objectives

After the threat analysis, the relevant security objectives have to be determined. In essence, the security objectives of assets, which may be compromised by one or multiple threats, are added to the threat matrix. (VDI/VDE 2182-1, 2011)

Since each of the modeled generic ADTrees represents a specific threat according to STRIDE, and in turn, each threat corresponds to a security objective, users can automatically derive the following objectives that are at risk by executing SPARQL queries: (i) Authenticity (Spoofing), (ii) Integrity (Tampering), (iii) Non-repudiation (Repudiation), (iv) Confidentiality (Information Disclosure), (v) Availability (Denial of Service), and (vi) Authorization (Elevation of Privilege). The ontology can be further extended to include other security objectives, such as auditability.

4.2.5. Risk Assessment

According to the VDI/VDE 2182-1 (2011) guideline, this step focuses on assessing the probability that threats manifest and the extent of damages. The guideline does not specify a certain risk measurement technique that

has to be applied, nor does it state whether a qualitative or quantitative risk analysis should be favored. Yet, the guideline provides a risk matrix as an example, which is used to categorize risks into three groups, namely low, medium and high. Furthermore, it is mentioned that a qualitative approach represents the usual choice when measuring risks. (VDI/VDE 2182-1, 2011)

However, since a certain degree of vagueness is inherent to a qualitative approach (Bojanc and Jerman-Blažič, 2008), a quantitative risk analysis may be indispensable. Thus, we demonstrate in this section, how quantitative risk measurement techniques can be applied in order to ensure a cost-effective treatment of risks.

ADTrees allow answering a variety of quantitative questions related to threat scenarios (Kordy et al., 2013b). In particular, Kordy et al. (2013b) define three classes of questions that can be answered by means of ADTrees, viz., (i) *"questions referring to one player"* (e.g., minimal costs for the proponent), (ii) *"questions where answers for both players can be deduced from each other"* (e.g., success probability for a threat scenario), and (iii) *"questions referring to an outside third party"* (e.g., maximal power consumption of the threat scenario) (Kordy et al., 2013b).

Recall that ADTGenerator allows the generation of XML files, which represent ADTrees, based on the knowledge base. These XML files can then be imported into ADTool for the purpose of conducting a manual quantitative risk analysis. To illustrate how the risk analysis can be performed, we reuse the exemplary ADTree depicted in Figure 6. Due to space constraints, we focus on the attack nodes *Side Channels* and *Storage Attacks*, which are connected to the goal of the generic ADTree representing the disclosure of information from a data store (source code repository). This generic ADTree is in turn connected to the root node of the entire ADTree, which constitutes threat scenario no. 1 described in Table 6.

To determine which risks need to be treated in order to achieve an acceptable level of risk, we used the ADTool to compute the probability of success for the exemplary threat scenario. Assuming that all actions are independent, the success probability of a node with a disjunctive refinement, connecting the basic actions A and B, can be calculated by the equation $P_{\cup}(A, B) = P(A) + P(B) - P(A)P(B)$, whereas the success probability of a node with a conjunctive refinement can be calculated as follows: $P_{\cap}(A, B) = P(A)P(B)$, where P represents the probability distribution (Kordy et al., 2013b). We assigned for each basic action in Figure 6 a value for the probability that either the attack or defense, depending on the respec-

tive node type, is successful. In this context, it is worth mentioning that the success probability of defense nodes represents the estimated countermeasures' effectiveness, which is also determined by the fact of whether they are in place. After the manual assignment, the probabilities for the remaining nodes, including the ADTree's goal, are automatically computed by ADTool. The assigned and calculated values for the success probabilities of nodes can be found in Table 7. As the table shows, the probability for an adversary to successfully obtain data from the source code repository is 0.798.

Note that for the success probability computation with ADTool, it is assumed that all actions are independent (Kordy et al., 2013b). However, this limitation can be overcome by combining ADTrees with Bayesian networks (Pearl, 1988), as proposed by Kordy et al. (2014b).

	Node	Probability	Computed Probability
Attack Node	Source Code Repository: Information Disclosure	-	0.798
	Side Channels	-	0.211
	Emissions	-	0.011
	Other Radiation	0.01	0.01
	Sound	0.001	0.001
	Filesystem Effects	0.1	0.1
	Power Draw	0.03	0.015
	Timing	0.1	-
	Storage Attacks	-	0.744
	Backup	0.75	0
	Non Cleaned Storage	0.65	0.65
	Physical Access	0.25	0.188
	Reused Memory	0.1	-
Defense Node	Shielding	0	-
	Remove Microphone	0	-
	Separate VMs	0	-
	Protect Power Draw Logs	0.5	-
	Cryptography	1	-
	Destory Disks	0	-
	Physical Access Countermeasure	-	0.25
	Physical Protection	0.25	-
	Encrypt Filesystem	0	-

Table 7: The success probability of ADTree's nodes depicted in Figure 6.

The beauty of the proposed approach is that the results from the risk

analysis can be transferred back to the knowledge base so that they can be reused for subsequent iterations of the procedure. However, the results of the quantitative assessment gained from ADTool must be manually transferred between the knowledge base and ADTool, as we leave the development of a prototype that automates these steps for future work.

4.2.6. Countermeasures

After analyzing the risks, countermeasures have to be identified and their effectiveness against threats evaluated. Furthermore, the VDI/VDE 2182-1 (2011) guideline states that the implementation costs of countermeasures must be taken into account to aid the decision-making process when selecting cost-effective countermeasures. (VDI/VDE 2182-1, 2011)

This step can be supported by the use of SPARQL queries, provided that the success probabilities of attack and defense nodes as well as the implementation costs of countermeasures are represented in the knowledge base. For instance, the respective SPARQL query that we provide via the GitHub repository[3] retrieves a list of potential countermeasures, including the estimated effectiveness, implementation costs, the corresponding attacks they aim to mitigate, and the estimated probability that these attacks will be carried out successfully. The query returns the results ordered by the success probability of attacks in descending order, and by the success probability of countermeasures, as well as the implementation costs thereof in ascending order. In this way, users can develop a defense strategy with the objective of reducing risks to an acceptable level, while also considering the costs that are associated with its implementation. Although we used a simplistic example for the sake of clarity, the ontology can be extended to support additional factors (e.g., business requirements) that may be of significance for investment decisions.

Now that the most cost-effective countermeasures have been identified, their implementation can be initiated. Finally, a process audit has to be carried out by someone not involved in the preceding steps of the procedure (VDI/VDE 2182-1, 2011). Although these steps are crucial for ensuring that the security measures take effect, they are out of the scope of the article at hand.

5. Evaluation

In this section, we evaluate the proposed framework in the context of selected security analysis tools. Therefore, we first identify a set of security analysis tools (step 1) by following a generic approach for tool selection according to Poston and Sexton (1992), and then evaluate selected tools in the context of the proposed approach based on identified requirements (step 2). Figure 7 illustrates the basic steps of the evaluation process, its inputs, and outputs.

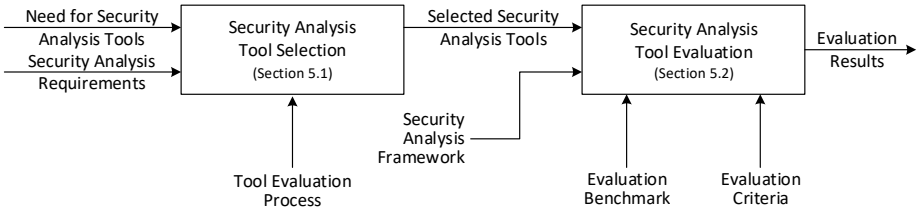

Figure 7: The overall evaluation process.

The *Security Analysis Tool Selection Process* step (cf. Section 5.1) takes as inputs (i) the need for tool support, and (ii) security analysis requirements. We use a generic tool selection process proposed by Poston and Sexton (1992) for testing tool evaluation, which also has been successfully adapted and applied in the context of business process modeling tool evaluation in (Winkler et al., 2014). The output of this process step is a set of available and promising security analysis tools.

The *Security Analysis Tool Evaluation Process* step (cf. Section 5.2) takes as inputs (i) the set of selected security analysis tools, and (ii) the novel approach, proposed in this article as a security analysis framework. Furthermore, evaluation criteria and benchmarks have been used to provide a common foundation for the evaluation. The output represents evaluation results (cf. Section 5.3) in terms of a comparison of selected security analysis tools and frameworks.

5.1. Security Analysis Tool Selection

A wide range of security analysis tools and frameworks are available in industry and academia. Due to the different focus of individual candidate

tool solutions, there is a need for selecting the most promising security analysis tools in the context of this research. To enable traceable and repeatable results, we follow a generic tool evaluation approach, proposed by Poston and Sexton (1992). Although this approach focuses on evaluating testing tools, the basic process steps are applicable for selecting security analysis tools as well. These basic tool selection process steps include: (i) *Requirements Identification* with a focus on requirements that need to be supported by candidate tools, (ii) *Selection Criteria Identification and Prioritization* to pre-select promising candidate tools (e.g., based on critical requirements) and to evaluate the remaining set of candidate tools, (iii) *Identification of a Set of Available Tools* that represent the input for pre-selection and evaluation, and (iv) *Tool Evaluation* that is based on the needs and requirements for performing security analyses to identify the most suitable approaches. However, an adaptation of this basic process approach is needed to (i) address individual aspects in the context of conducting security analyses, and (ii) to provide a step-by-step approach for traceable tool selections.

Figure 8 presents the basic process steps for systematically selecting suitable security analysis tools.

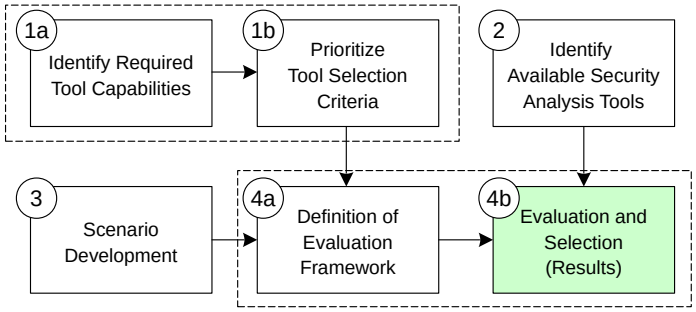

Figure 8: The tool selection process based on (Poston and Sexton, 1992) and adapted in (Winkler et al., 2014).

Step 1a: Identify Required Tool Capabilities. Based on related work and workshops as well as discussions with industry partners, we derived a set of requirements and needed tool capabilities. Note that we followed the Easy-Win-Win (EWW) Process approach (Gruenbacher, 2000) to collect and classify collected requirements. The output of this process step is a list of

agreed and categorized requirements. In particular, we obtained the following categories: (i) basic requirements (e.g., modeling of DFDs), (ii) modeling (e.g., modeling of assets, consequences), (iii) integration (e.g., report generation, API access), (iv) risk assessment (e.g., quantitative risk assessment, risk matrix), and (v) additional features (e.g., built-in threat library, attack simulation). In total, 31 requirements were collected.

Step 1b: Prioritize Requirements and Tool Selection Criteria. In this process step, experts from industry and academia prioritize the identified requirements in the EWW workshop according to expected benefits from individual viewpoints. The outcome is a list of *critical*, *important*, *less important*, and *nice-to-have* requirements. Note that critical requirements are mandatory for candidate tools. Tools that do not provide solutions for critical requirements will be excluded without further evaluation. Furthermore, for evaluation purposes, we used a four-level scale to weight the priorities accordingly (i.e., critical: 10, important: 5, less important: 2.5, and nice-to-have: 1).

Step 2: Identify Available Security Analysis Tools. This process step focuses on collecting available candidate solutions. In the context of this article, we execute an informal web search complemented by recommendations from security experts of our working group and industry experts. It is worth mentioning that we searched for tools that are being advertised as threat modeling or security risk assessment solutions, as we deem tools of both categories relevant to our evaluation context. Table 8 shows the list of candidate security analysis tools.

Step 3: Scenario Development. Application scenarios represent typical sequences of tasks that need to be executed (and supported) by tools. Therefore, we developed a set of typical application scenarios as input for the tool evaluation. Note that these scenarios were reviewed by security experts to ensure correctness and completeness of derived scenarios.

Step 4a: Definition of the Evaluation Framework. Similar to (Winkler et al., 2014), we developed an evaluation framework as a spreadsheet solution that holds categorized, classified, and prioritized requirements (y-axis) and preselected candidate tool (x-axis). Note that out of the bunch of available tools (step 2), critical requirements (derived in step 1b) have been used to exclude less promising tools from the evaluation.

Name	Version	Free	URL
CORAS	1.4	●	https://sourceforge.net/projects/coras/
IriusRisk	02/2019		https://continuumsecurity.net/threat-modeling-tool/
Isograph AttackTree	4.0		https://www.isograph.com/software/attacktree/
Microsoft Threat Modeling Tool (TMT)	2016	●	https://www.microsoft.com/en-us/download/details.aspx?id=49168
OWASP Threat Dragon	0.1.26	●	https://threatdragon.org/
SeaSponge	0.2.0	●	https://mozilla.github.io/seasponge/
SecuriCAD	1.4.7	◐	https://www.foreseeti.com/
Secur*IT*ree	4.7		https://www.amenaza.com/
ThreatModeler	02/2019		https://threatmodeler.com/
Tutamen	Beta		http://www.tutamantic.com/

Legend: ● applicable, ◐ free version with reduced feature set.

Table 8: The security analysis tools considered in the tool selection process.

Step 4b: Evaluation and Selection (Results). The last step of the tool evaluation focuses on the evaluation of selected tools with respect to the requirements (and priorities) by applying the identified application scenarios. Two researchers executed the scenarios and rated the compliance to the derived requirements on a scale ranging from 0 (requirement is not met) to 5 (requirement is met). Disagreements were discussed to come to a consensus. Based on the agreed ratings and weighted requirements, a common score for each tool was calculated, which describes the tool's capability with respect to the given requirements in the evaluation context. Finally, the one with the highest overall score represents the security analysis tool that fits best to the set of identified requirements. As can be seen from Table 9, SecuriCAD achieves the highest weighted score and will, therefore, be selected for conducting the tool evaluation. In addition to SecuriCAD, we selected the Microsoft Threat Modeling Tool (TMT) for the tool evaluation, even though it achieved only a moderate score. However, since it is free to use and given its apparent popularity, it will be considered for the evaluation.

5.2. Security Analysis Tool Evaluation

To perform the actual evaluation, we model a realistic software testing setup with all tools selected in the previous subsection in order to deter-

Tool	Basic Requirements	Modeling	Integration	Risk Assessment	Additional Features	Total		
						Score	Weighted	Weighted (%)
CORAS	77%	67%	0%	17%	4%	39	190	42%
IriusRisk	100%	68%	75%	77%	63%	105	362	80%
Isograph Attack-Tree	82%	45%	52%	83%	10%	66	238	52%
Microsoft TMT	94%	36%	43%	17%	43%	59	265	58%
OWASP Threat Dragon	91%	24%	26%	0%	29%	52	218	48%
SeaSponge	83%	6%	9%	0%	6%	28	153	34%
SecuriCAD	100%	94%	74%	100%	76%	126	405	89%
Secur*IT*ree	62%	18%	26%	77%	6%	39	161	35%
ThreatModeler	100%	94%	100%	63%	69%	130	402	88%
Tutamen	94%	2%	65%	0%	37%	47	237	52%
					Maximum:	155	455	

Table 9: The results of the tool selection.

mine their effectiveness in supporting security analyses of software testing processes. The underlying testing process of the input models for the tools is based on the generic process discussed in Section 3, albeit it misses certain steps that have been omitted on purpose. As already mentioned, we observed that the generic software testing process for automation applications, as depicted in Figure 3, may not be rigorously followed on the supervisory or control level; thus, evaluating with a subset of the activities corresponding to the generic process model seemed reasonable. In particular, the testing process considered for the evaluation (i) misses certain organizational activities related to test management (e.g., designing a test strategy), (ii) does not require the generation of test data, and (iii) lacks a dedicated test automation engineer. Moreover, note that the input models of the selected security tools have to contain technical details to varying degrees. Thus, we selected representative, off-the-shelf tools as well as necessary software and hardware components to obtain a plausible, realistic architecture of the software testing setup. For instance, we selected Jira[4] as the test management tool, which is hosted on a server that satisfies the respective system requirements for

[4]Jira: `https://www.atlassian.com/software/jira`.

running this software.

After modeling the testing setup with each tool, we compare them with each other and determine how their characteristics may facilitate security analyses of software testing processes. The evaluation criteria used for the tool comparison, which are based on (Lemaire et al., 2017)[5], can be divided into the categories (i) input (e.g., system model, attacker model), and (ii) output (e.g., threat generation, risk matrix). In essence, assessing the tools' modeling capabilities, the results they yield and how they are specialized gives some indication of the assistance that users can expect when using these tools for analyzing the security of their software testing setup.

Besides comparing the functional characteristics, we evaluate how well the tools support the steps of the procedural method defined in the VDI/VDE 2182-1 (2011) guideline (cf. Figure 4). The criteria used for this qualitative evaluation are the effectiveness in terms of accomplishing the procedural method's activities and the manual effort required to perform security analyses by using the tools. A five-point Likert scale is applied to rate the degree to which the tools meet the criteria specified. Due to the fact that the nature of outputs of the tools selected for the evaluation varies significantly, we abstain from performing a quantitative comparison (e.g., the number of generated threats).

5.3. Evaluation Results

The results of the security analysis tool evaluation are summarized in Tables 10 and 11. In the following, we present the main findings of this evaluation.

Microsoft Threat Modeling Tool (MS TMT). As can be seen in Table 10, the input model of the MS TMT comprises a system model in the form of a DFD. Furthermore, a variety of manifestations of DFD elements are built-in (e.g., OS process as a child of the process DFD element) and can be further extended or customized by creating templates. Since the focus of the MS TMT is on DFDs, the tool adopts a software-centric modeling approach (Shostack, 2014). Thus, the tool's modeling approach neither gives priority to assets nor attackers. Owing to this software-centric nature of the tool, essentially

[5]In their paper, Lemaire et al. (2017) evaluate five security analysis tools for CPSs. The therein presented tool comparison is organized by the tools' input model and the feedback that they provide.

			MS TMT	SecuriCAD	Our Approach
Input	System Model	DFD	●	●	●
		Built-in DFD Elements	●	●	◐
		Custom DFD Elements	●	●	●
		Assets		◐	●
		Testing Process Model			●
	Attacker Model	Attack Steps		●	●
		Capabilities		◐	●
		Consequence		◐	●
	Security Expertise Required				◐
Output	Threat Generation		●	●	◐
	Risk Matrix			●	
	Quantitative Analysis			●	●
	Effectiveness of Countermeasures			●	●

Legend: ● applicable, ◐ partially applicable.

Table 10: The comparison of security analysis tools based on key characteristics.

little to no security expertise is required for creating the input model. Based on the modeled target of inspection, threats can be automatically generated. The output that MS TMT provides is a list of threats, which are categorized according to the STRIDE model and include a brief explanation describing the attack vector as well as options for mitigation. Table 11 shows that the MS TMT is of minimal value to users who seek to analyze the security of their software testing setup systematically. Assets are only considered in terms of the modeled DFD elements, such as a web application or SQL database. It is evident that the MS TMT has its strengths in analyzing threats, as it has been designed for that specific purpose. However, support for subsequent steps, apart from determining relevant security objectives, is entirely lacking and therefore needs to be manually performed.

SecuriCAD. This tool relies on an accurate model of the IT infrastructure that conceptually resembles a DFD. Similar to the MS TMT, SecuriCAD provides a set of components that can be used to build the system model, albeit a more comprehensive one. SecuriCAD also allows the creation of new components, in case certain modeling elements are missing or need to be customized. The assets are implicitly modeled when sketching out the

Step	Effectiveness			Effort		
	MS TMT	SecuriCAD	Our Appr.	MS TMT	SecuriCAD	Our Appr.
Structure Analysis	+	++	−	○	−	○
Identify Assets	−	○	+	−−	○	+
Analyze Threats	+	+	+	+	++	+
Determine Relevant Security Objectives	+	○	+	+	○	++
Analyze & Assess Risks	−−	++	+	−−	++	+
Identify Measures & Assess Effectiveness	−−	+	+	−−	+	+
Select Countermeasures	−−	+	○	−−	+	+

Legend: ++ very good, + good, ○ average, − weak, −− very weak.

Table 11: The results of the tool evaluation.

IT environment (e.g., keystore). In contrast to the MS TMT, the input model in SecuriCAD also considers the steps attackers can take and the associated costs as well as consequences. However, the focus of SecuriCAD is not on the attacker but system model, meaning that a detailed understanding of the target of inspection suffices. As Table 10 shows, among the tools evaluated, SecuriCAD stands out in terms of the feedback it provides to users. In particular, SecuriCAD allows running automated attack simulations in order to obtain sequences of potential attack steps and corresponding quantitative risk measurements (e.g., time-to-compromise). In this way, the tool aims to reveal the most critical weak spots in the infrastructure and provides suggestions for mitigation. Due to its rich feature set, SecuriCAD is well suited for supporting the steps of the procedural method defined in the VDI/VDE 2182-1 (2011) guideline (cf. Table 11). However, since users need to input a detailed model of their IT environment, performing the structure analysis by means of the tool requires considerable manual effort. Thus, offering default models of reference architectures would be a valuable addition in order to enable users to generate meaningful results more quickly.

Our Security Analysis Framework. The herein presented framework is based on two ontologies that are used to represent knowledge about the target of inspection on the one hand, and the attacker on the other (cf. Section 4.1). While the system model consists of a typical DFD, the attacker model is composed of threat trees. The beauty of the adopted ontological modeling

approach is that it provides flexibility and scalability, allowing users to modify, extend and integrate the models as needed. In line with this rationale, the framework offers a model that reflects the DFD of the software testing process depicted in Figure 5 and a model that includes threat scenarios specific to testing automation software, which both can be used out of the box or adapted as needed. Furthermore, it is apparent from Table 10 that our framework fuses the system- and attacker-centric modeling approach. Thus, extending the provided attacker model or creating one from scratch requires security expertise. Output-wise, our framework achieves a moderate level of decision support. Owing to the developed prototype ADTGenerator, ADTrees can be automatically created from the knowledge base. This way of generating threats is fundamentally different from that of MS TMT and SecuriCAD. In our framework, STRIDE is used to link the roots, i.e., goals of threat trees to specific security threats (e.g., information disclosure of a data store) and the child nodes detail how these threats can manifest. Based on this, complex threat scenarios, which are represented as ADTrees and may comprise multiple threat trees, can be created. After the threats have been generated, the output of ADTGenerator can be imported into ADTool (Kordy et al., 2013a) for subsequent analysis. Since ADTool provides multiple attribute domains (Kordy et al., 2013b), the ADTrees can be utilized for quantitative risk assessments. Regarding the effectiveness in supporting the security analysis, our framework was found to be less valuable for performing the structure analysis but achieved good results for subsequent steps (cf. Table 11). The reason for the weak support of the structure analysis is that users have to draw on ontology visualization tools for sketching the target of inspection and that the DFD ontology lacks technical depth. The latter, in turn, leads to the fact that attack and defense nodes of ADTrees modeled with the ADT ontology apply to the basic types of DFD elements and therefore do not consider specific components of the infrastructure from a technological perspective. On the contrary, this also means that generally less modeling effort is required for constructing ADTrees that can be used as a basis for the security analysis. Finally, it is worth noting that the framework in its current state does not provide a seamless workflow for conducting the security analysis, as the tools used (i.e., ontology editor, ADTGenerator, ADTool) are not tightly integrated.

6. Related Work

Existing work that is related to the article at hand can be categorized into (i) threat modeling for cyber-physical systems (CPSs), (ii) automated threat modeling, and (iii) information security ontologies. The following subsections discuss selected representatives of these categories and explain how this work is connected to them.

6.1. Threat Modeling for CPSs

Numerous works have investigated how threat modeling can be applied in the CPS domain in order to identify potential security issues, which ultimately also affect the safety of these systems.

In (Khan et al., 2017), the authors propose a five-step threat modeling methodology for CPSs, which is based on STRIDE. Their work is motivated by the lack of studies that demonstrate how threats to CPSs can be systematically identified, using a real-world example. Thus, Khan et al. (2017) show step by step how their proposed methodology can be applied to discover threats pertaining to a real synchrophasor-based system. The article at hand adopts a threat modeling approach that is similar to the methodology proposed in (Khan et al., 2017) since we also use a data flow diagram (DFD) to represent how data is transmitted within the target of inspection and subsequently apply STRIDE to identify threats. However, while the work conducted by Khan et al. (2017) focuses only on the modeling of threats, we present a more comprehensive security analysis, including the analysis of threats as a single step of the adopted procedure described in VDI/VDE 2182-1 (2011). On top of that, our methodology attempts to semi-automate threat modeling activities.

Martins et al. (2015) present a tool that supports modeling threats to CPSs in a systematic manner. This tool leverages the Generic Modeling Environment (GME) (Ledeczi et al., 2001) to design a metamodel, which is used for modeling the components of CPSs, optionally also in the form of DFDs. Moreover, the tool utilizes GME interpreters to perform systematic model analyses that may yield potential security weaknesses. The authors of (Martins et al., 2015) also present a case study in which they demonstrate how their tool can be applied to discover vulnerabilities in a wireless temperature monitoring system. Although our semi-automatic threat modeling approach is conceptually similar to the one proposed by Martins et al. (2015)

(i.e., modeling the target of inspection and leveraging tool support for analyzing threats), there are two important distinctions. First, in addition to the analysis of threats, our proposed method also facilitates other steps of the conducted security analysis (e.g., identifying roles authorized to use assets, determining violated security objectives, performing a risk analysis based on generated ADTrees). Second, instead of using an UML-based modeling language, we model the target of inspection with OWL and can therefore take advantage of semantic reasoners.

The authors of (Schlegel et al., 2015) attempt to bridge the gap between existing threat modeling tools that follow either a specific or generic modeling approach, and by implication suffer either from being too restrictive or from the lack of structure. In essence, the rationale of their work is to leave the level of detail of the threat model to the user's discretion. Schlegel et al. (2015) propose a methodology that attempts to fulfill this requirement. In a nutshell, their methodology is based on (i) a data model, which consists of objects of type component, threat, impact or security control, and (ii) relationships between objects. Schlegel et al. (2015) have also implemented a web-based tool and created a threat model for a substation automation system in order to validate their methodology. While Schlegel et al. (2015) highlight the well-balanced nature between generality and specificity of their approach, we argue that by combining DFDs, STRIDE, and ADTrees, this feature can be achieved similarly. In fact, we believe that leveraging these concepts can ease adoption, as they are already well established in the community.

Several works such as (Byres et al., 2004; Ten et al., 2007; Xie et al., 2013) utilize attack trees (Schneier, 1999) to analyze the security of CPSs. While the applicability of attack trees certainly goes beyond mere threat modeling[6], we adopt a variant thereof — supplementary to STRIDE — in order to support threat modeling activities. In particular, our proposed tool generates ADTrees (Kordy et al., 2011) for selected threat scenarios based on the threat trees defined in (Shostack, 2014), which we modeled with our ADT ontology. Users can then import the generated ADTrees into ADTool (Kordy et al., 2013a) and use them as a basis for threat models or proceed

[6]See (Kordy et al., 2014a) for a comprehensive survey on attack and defense modeling techniques based on directed acyclic graphs, including a brief discussion on the use cases of attack trees.

with subsequent steps of the security analysis. While the idea of generating ADTrees or attack trees is not new (cf., for instance, (Depamelaere et al., 2018; Lemaire et al., 2018; Vigo et al., 2014; Paul, 2014; Ivanova et al., 2015)), the novelty of our work lies in using an ontological approach for deriving ADTrees based on the modeled DFD. The derived knowledge can then also be reused for ongoing steps in the security analysis.

It is also worth mentioning that several tools have been developed to analyze the security of CPSs. In (Lemaire et al., 2017), the authors provide a comprehensive comparison of five relevant tools, viz., CSET[7], CySeMoL (Sommestad et al., 2013; Holm et al., 2013), ADVISE (LeMay et al., 2011), FAST-CPS (Lemaire et al., 2014, 2015), and CyberSAGE (Vu et al., 2014). Their evaluation reveals that the considered tools vary to a great extent in terms of the input they require and the results they provide. However, since none of the tools stood out from the rest, Lemaire et al. (2017) suggest using the tools and underlying modeling methodologies in combination.

On a final note, although our work does not specifically investigate potential vulnerabilities in CPSs, we still consider previous research in the area of threat modeling for CPSs related to the article at hand, as the conducted security analysis falls within this context (i.e., discovered threats may affect the PSE process and CPSs likewise).

6.2. Automated Threat Modeling

Over the past few years, researchers have proposed various methods to automate the enumeration and analysis of threats.

For example, in 2010, Yee et al. (2010) proposed an automated approach for identifying threats based on UML diagrams. Their approach makes use of an expert system that relies on a knowledge base consisting of security-relevant information extracted from UML diagrams (facts) and threat scenarios (rules). To demonstrate the viability of the proposed method, the authors of (Yee et al., 2010) apply their implemented prototype to analyze a web service. Yee et al. (2010) also point out a few issues they have discovered related to the use of UML. In particular, the authors indicate that UML may not be an ideal candidate for automated threat identification, as it permits a certain degree of ambiguity and vagueness. The article at hand circumvents this issue, as the knowledge base is composed of explicit security-relevant

[7]CSET: `https://cset.inl.gov/`.

information, modeled with the developed ontologies. On the other hand, the benefit of the approach discussed in (Yee et al., 2010) is that the UML models that have been created during the development of systems can be directly used for automatically identifying threats.

Berger et al. (2016) introduce the concept of *extended data flow diagrams* (EDFDs), which expand DFDs by an EDFD schema that provides additional semantics to model (i) *Elements* (i.e., the entities *Data Store*, *Process*, *Interactor*), (ii) *Channels* (i.e., data flows with relationships, such as one-to-many), (iii) *Trust Areas*, and (iv) *Data* (i.e., types of data, such as credentials). In this way, additional knowledge can be represented, e.g., defensive measures that are already in place. Furthermore, Berger et al. (2016) state that this makes security-relevant know-how represented in DFDs more explicit; thus, facilitating an automated threat analysis. The automated analysis proposed by Berger et al. (2016) works as follows: First, security experts create a knowledge base that is composed of rules about security flaws. Furthermore, they provide a pattern catalog that allows to model characteristics of systems (e.g., type of a database). Users, which may not necessarily have a background in information security, can then create an EDFD with the introduces EDFD schema and the pattern catalog. For the automated threat modeling, a rule checker can be run, which transforms the EDFD into a graph and attempts to detect the rules defined in the knowledge base. The beauty of this approach is that the authors of (Berger et al., 2016) used the Common Weakness Enumeration (CWE)[8] and the Common Attack Pattern Enumeration and Classification (CAPEC)[9] as a basis for the knowledge base. Incorporating the CWE and CAPEC into the knowledge base of the herein presented framework may constitute a valuable extension of our work. However, we believe that the use of ADTrees for automating threat modeling is indispensable, as they also serve as a foundation for quantitative risk analysis.

The *Automated Architecture for Threat Modeling and Risk Assessment for Cloud Computing*, named *Nemesis*, presented in (Kamongi et al., 2014) has two similarities with our proposed framework. First, Kamongi et al. (2014) also use an ontological modeling approach. Second, STRIDE serves as one of the main pillars of Nemesis. However, the architecture of Neme-

[8]CWE: `https://cwe.mitre.org/`.
[9]CAPEC: `https://capec.mitre.org/`.

sis differs fundamentally from the one of our proposed framework. Nemesis integrates security ontology bases from the VULCAN framework (Kamongi et al., 2013) to retrieve vulnerabilities, attacks, and defenses for specific cloud configurations. Furthermore, STRIDE is applied to automatically classify discovered vulnerabilities. The identified threats serve then as an input for the threat probability estimator (Fenz, 2011) in order to obtain an aggregated risk indicator. In contrast to (Kamongi et al., 2014), our work aims to provide support for threat modeling activities by automating certain steps (e.g., constructing ADTrees), rather than computing a risk indicator. Although Nemesis can also output other information, such as severity ranks, we believe that our framework represents a precious tool to ease the process of discovering particularly unknown weaknesses.

6.3. Information Security Ontologies

As already mentioned, we chose an ontological approach to model the necessary background knowledge, which the presented security analysis framework is based on. Ontologies provide a formal specification of the concepts and relationships within a domain and facilitate data linking as well as integration. The main advantages in our context are (i) a highly flexible and extensible knowledge base that can be easily extended by interested readers and domain experts, (ii) that reasoners can infer new knowledge and thus contribute to compact queries and consistent data, (iii) that knowledge can be freely navigated and integrated into applications, and (iv) the data can be easily shared by interested parties and hence, has the potential to form a community-based reference.

Existing knowledge and standards that we included in the DFD and ADT ontologies have been outlined in Sections 2 and 4. While multiple ontologies to represent information security knowledge have been proposed in the past, including, inter alia, (Fenz and Ekelhart, 2009; Oltramari et al., 2014; Herzog et al., 2007), some approaches specifically address risk analysis, e.g., (Ahmed et al., 2007; Ekelhart et al., 2007a, 2009a, 2007b, 2009b). However, the domain of software testing according to the VDI/VDE 2182-1 (2011) guideline combined with ADTrees has not yet been covered. The defined models are less extensive than the previously mentioned works but are instead tailored to the presented framework. Still, they share concepts from existing ontologies but are not explicitly aligned yet, which is reserved for future work. Other security ontologies, such as (Zareen Syed and Joshi, 2016; Gao et al., 2013)

focused on technical vulnerabilities, which could also be integrated in the future in order to model specific security weaknesses of test environments.

7. Conclusions

In this paper, we have presented a novel framework for semi-automatically conducting a security analysis of the testing process for industrial automation software. This framework is based on the procedural method described in the VDI/VDE 2182-1 (2011) guideline and uses an ontological modeling approach to represent knowledge relevant to the security analysis. In particular, we argue that analyzing the security of a software testing process can be semi-automated. Furthermore, we introduce a prototype named ADTGenerator, which allows to automatically generate ADTrees for threat scenarios that apply to the modeled testing setup. These ADTrees can then be imported into ADTool (Kordy et al., 2013a) to perform further threat modeling activities with the graphical representation of the ADTrees or conduct a quantitative risk assessment (Kordy et al., 2013b). To demonstrate the viability of the proposed framework, we show how querying the knowledge base and leveraging the generated ADTrees can reduce manual effort and, in further consequence, facilitate the security analysis.

In future work, we want to further automate the security analysis by extending the knowledge base. On the one hand, the framework lacks simulation approaches that would allow running attack simulations for the purpose of exploring threat scenarios, e.g., as presented by (Ekelhart et al., 2015; Kiesling et al., 2014). On the other hand, we could extend the ontologies to represent knowledge about vulnerabilities and technical details regarding the specific test setup (e.g., software versions, network setup). Furthermore, `SAND` attack trees, i.e., attack trees with sequential conjunction (Jhawar et al., 2015), are not yet supported by the ADT ontology. Besides extending the knowledge base, we also plan to improve the developed prototype so that the attribute domains for ADTrees (Kordy et al., 2013b) used in the course of quantitative risk assessments can be automatically transferred between the knowledge base and ADTool.

Finally, it is worth highlighting that leveraging security-relevant knowledge, which has been modeled with ontologies, represents a powerful approach for the purpose of automating security analyses. We will further explore how we can apply this method to other phases of the production

systems engineering (PSE) process as well as to specific types of CPSs, such as cyber-physical production systems (CPPSs).

Acknowledgements

The financial support by the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development, and COMET K1, FFG - Austrian Research Promotion Agency is gratefully acknowledged.

References

Ahmed, M., Anjomshoaa, A., Nguyen, T. M., Tjoa, A. M., 2007. Towards an ontology-based risk assessment in collaborative environment using the semanticlife. In: Proceedings of the The Second International Conference on Availability, Reliability and Security. ARES '07. IEEE Computer Society, Washington, DC, USA, pp. 400–407.

Baheti, R., Gill, H., 2011. Cyber-physical systems. The impact of control technology 12, 161–166.

Berger, B. J., Sohr, K., Koschke, R., 2016. Automatically extracting threats from extended data flow diagrams. In: Caballero, J., Bodden, E., Athanasopoulos, E. (Eds.), Engineering Secure Software and Systems. Springer International Publishing, Cham, pp. 56–71.

Biffl, S., Gerhard, D., Lüder, A., 2017. Introduction to the Multi-Disciplinary Engineering for Cyber-Physical Production Systems. Springer International Publishing, Cham, pp. 1–24.

Bojanc, R., Jerman-Blažič, B., 2008. An economic modelling approach to information security risk management. International Journal of Information Management 28 (5), 413 – 422.

Byres, E. J., Franz, M., Miller, D., 2004. The use of attack trees in assessing vulnerabilities in scada systems. In: in IEEE Conf. International Infrastructure Survivability Workshop (IISW '04). Institute for Electrical and Electronics Engineers.

de Sá, A. O., d. C. Carmo, L. F. R., Machado, R. C. S., Aug 2017. Covert attacks in cyber-physical control systems. IEEE Transactions on Industrial Informatics 13 (4), 1641–1651.

Depamelaere, W., Lemaire, L., Vossaert, J., Naessens, V., 2018. Cps security assessment using automatically generated attack trees. In: Proceedings of the 5th International Symposium for ICS & SCADA Cyber Security Research 2018. British Computer Society (BCS).

Dubey, A., July 2011. Evaluating software engineering methods in the context of automation applications. In: 2011 9th IEEE International Conference on Industrial Informatics. pp. 585–590.

Ekelhart, A., Fenz, S., Klemen, M., Weippl, E., Jan 2007a. Security ontologies: Improving quantitative risk analysis. In: System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on. pp. 156a–156a.

Ekelhart, A., Fenz, S., Neubauer, T., 1 2009a. Aurum: A framework for information security risk management. In: Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS2009). pp. 1–10.

Ekelhart, A., Fenz, S., Neubauer, T., Weippl, E., 1 2007b. Formal threat descriptions for enhancing governmental risk assessment. In: 1st International Conference on Theory and Practice of Electronic Governance. ICEGOV '07. ACM, New York, NY, USA, pp. 40–43.

Ekelhart, A., Kiesling, E., Grill, B., Strauss, C., Stummer, C., 2015. Integrating attacker behavior in it security analysis: a discrete-event simulation approach. Information Technology and Management, 1–13.

Ekelhart, A., Neubauer, T., Fenz, S., April 2009b. Automated risk and utility management. In: 6th International Conference on Information Technology: New Generations (ITNG 2009). IEEE Computer Society, pp. 393–398.

Falliere, N., Murchu, L. O., Chien, E., 2011. W32. stuxnet dossier. White paper, Symantec Corp., Security Response 5 (6).

Fenz, S., 2011. An ontology- and bayesian-based approach for determining threat probabilities. In: Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security. ASIACCS '11. ACM, New York, NY, USA, pp. 344–354.

Fenz, S., Ekelhart, A., 2009. Formalizing information security knowledge. In: Proceedings of the 4th International Symposium on Information, Computer, and Communications Security. ASIACCS '09. ACM, New York, NY, USA, pp. 183–194.

Gao, J.-b., Zhang, B.-w., Chen, X.-h., Luo, Z., Oct 2013. Ontology-based model of network and computer attacks for security assessment. Journal of Shanghai Jiaotong University (Science) 18 (5), 554–562.

Gausemeier, J., 2010. Zuverlässigere mechatronik – forschungsergebnisse kompakt: Transfer von forschungsergebnissen aus 11 verbundprojekten zur steigerung der zuverlässigkeit mechatronischer systeme. Tech. rep., Heinz Nixdorf Institut.

Graham, D., Van Veenendaal, E., Evans, I., 2008. Foundations of software testing: ISTQB certification. Cengage Learning EMEA.

Gruenbacher, P., 2000. Collaborative requirements negotiation with easywinwin. In: Proceedings 11th International Workshop on Database and Expert Systems Applications. IEEE, pp. 954–958.

Herzog, A., Shahmehri, N., Duma, C., October 2007. An Ontology of Information Security. International Journal of Information Security and Privacy (IJISP) 1 (4), 1–23.

Hevner, A. R., March, S. T., Park, J., Ram, S., Mar. 2004. Design science in information systems research. MIS Q. 28 (1), 75–105.

Holm, H., Sommestad, T., Ekstedt, M., Nordström, L., June 2013. Cysemol: A tool for cyber security analysis of enterprises. In: 22nd International Conference and Exhibition on Electricity Distribution (CIRED 2013). pp. 1–4.

ISO/IEC/IEEE 29119-2, 2013. Software and systems engineering – software testing – part 2: Test processes.

ISO/IEC/IEEE 29119-3, 2013. Software and systems engineering – software testing – part 3: Test documentation.

Ivanova, M. G., Probst, C. W., Hansen, R. R., Kammüller, F., 2015. Attack tree generation by policy invalidation. In: Akram, R. N., Jajodia, S.

(Eds.), Information Security Theory and Practice. Springer International Publishing, Cham, pp. 249–259.

Jhawar, R., Kordy, B., Mauw, S., Radomirović, S., Trujillo-Rasua, R., 2015. Attack trees with sequential conjunction. In: Federrath, H., Gollmann, D. (Eds.), ICT Systems Security and Privacy Protection. Springer International Publishing, Cham, pp. 339–353.

Kagermann, H., Helbig, J., Hellinger, A., Wahlster, W., Apr. 2013. Recommendations for implementing the strategic initiative industrie 4.0 – securing the future of german manufacturing industry. Final report of the industrie 4.0 working group, acatech – National Academy of Science and Engineering, München.

Kamongi, P., Gomathisankaran, M., Kavi, K., 2014. Nemesis: Automated architecture for threat modeling and risk assessment for cloud computing. In: Proceedings of the 6th ASE International Conference on Privacy, Security, Risk and Trust (PASSAT). pp. 1–10.

Kamongi, P., Kotikela, S., Kavi, K., Gomathisankaran, M., Singhal, A., June 2013. Vulcan: Vulnerability assessment framework for cloud computing. In: 2013 IEEE 7th International Conference on Software Security and Reliability. pp. 218–226.

Khan, R., McLaughlin, K., Laverty, D., Sezer, S., Sept 2017. Stride-based threat modeling for cyber-physical systems. In: 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe). pp. 1–6.

Kieseberg, P., Weippl, E., 2018. Security challenges in cyber-physical production systems. In: Winkler, D., Biffl, S., Bergsmann, J. (Eds.), Software Quality: Methods and Tools for Better Software and Systems. Springer International Publishing, Cham, pp. 3–16.

Kiesling, E., Ekelhart, A., Grill, B., Stummer, C., Strauss, C., 1 2014. Evolving secure information systems through attack simulation. In: 47th Hawaii International Conference on System Sciences (HICSS 2014). pp. 4868–4877.

Knowles, W., Prince, D., Hutchison, D., Disso, J. F. P., Jones, K., 2015. A survey of cyber security management in industrial control systems. International Journal of Critical Infrastructure Protection 9, 52 – 80.

Kordy, B., Kordy, P., Mauw, S., Schweitzer, P., 2013a. Adtool: Security analysis with attack–defense trees. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P. R. (Eds.), Quantitative Evaluation of Systems. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 173–176.

Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P., 2011. Foundations of attack–defense trees. In: Degano, P., Etalle, S., Guttman, J. (Eds.), Formal Aspects of Security and Trust. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 80–95.

Kordy, B., Mauw, S., Schweitzer, P., 2013b. Quantitative questions on attack–defense trees. In: Kwon, T., Lee, M.-K., Kwon, D. (Eds.), Information Security and Cryptology – ICISC 2012. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 49–64.

Kordy, B., Pièvre-Cambacédès, L., Schweitzer, P., 2014a. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. Computer Science Review 13-14, 1 – 38.

Kordy, B., Pouly, M., Schweitzer, P., 2014b. A probabilistic framework for security scenarios with dependent actions. In: Albert, E., Sekerinski, E. (Eds.), Integrated Formal Methods. Springer International Publishing, Cham, pp. 256–271.

Langner, R., 2013. To kill a centrifuge: A technical analysis of what stuxnet's creators tried to achieve.

Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P., 2001. The generic modeling environment. In: Workshop on Intelligent Signal Processing.

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

Lemaire, L., Lapon, J., De Decker, B., Naessens, V., 2014. A sysml extension for security analysis of industrial control systems. In: Proceedings of the 2Nd International Symposium on ICS & SCADA Cyber Security Research 2014. ICS-CSR 2014. BCS, UK, pp. 1–9.

Lemaire, L., Vossaert, J., De Decker, B., Naessens, V., 2017. An assessment of security analysis tools for cyber-physical systems. In: Großmann, J.,

Felderer, M., Seehusen, F. (Eds.), Risk Assessment and Risk-Driven Quality Assurance. Springer International Publishing, Cham, pp. 66–81.

Lemaire, L., Vossaert, J., De Decker, B., Naessens, V., 2018. Security evaluation of cyber-physical systems using automatically generated attack trees. In: D'Agostino, G., Scala, A. (Eds.), Critical Information Infrastructures Security. Springer International Publishing, Cham, pp. 225–228.

Lemaire, L., Vossaert, J., Jansen, J., Naessens, V., 2015. Extracting vulnerabilities in industrial control systems using a knowledge-based system. In: Proceedings of the 3rd International Symposium for ICS & SCADA Cyber Security Research. ICS-CSR '15. BCS Learning & Development Ltd., Swindon, UK, pp. 1–10.

LeMay, E., Ford, M. D., Keefe, K., Sanders, W. H., Muehrcke, C., Sep. 2011. Model-based security metrics using adversary view security evaluation (ADVISE). In: 2011 Eighth International Conference on Quantitative Evaluation of SysTems. pp. 191–200.

Lewis, W. E., 2008. Software Testing and Continuous Quality Improvement, Third Edition, 2nd Edition. Auerbach Publications, Boston, MA, USA.

Martins, G., Bhatia, S., Koutsoukos, X., Stouffer, K., Tang, C., Candell, R., Aug 2015. Towards a systematic threat modeling approach for cyber-physical systems. In: 2015 Resilience Week (RWS). pp. 1–6.

Mauw, S., Oostdijk, M., 2006. Foundations of attack trees. In: Won, D. H., Kim, S. (Eds.), Information Security and Cryptology - ICISC 2005. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 186–198.

McLaughlin, S., Konstantinou, C., Wang, X., Davi, L., Sadeghi, A. R., Maniatakos, M., Karri, R., May 2016. The cybersecurity landscape in industrial control systems. Proceedings of the IEEE 104 (5), 1039–1057.

Miller, B., Rowe, D., 2012. A survey of scada and critical infrastructure incidents. In: Proceedings of the 1st Annual Conference on Research in Information Technology. RIIT '12. ACM, New York, NY, USA, pp. 51–56.

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., Musen, M. A., March 2001. Creating semantic web contents with protege-2000. IEEE Intelligent Systems 16 (2), 60–71.

Oltramari, A., Cranor, L., Walls, R., McDaniel, P., 01 2014. Building an ontology of cyber security. CEUR Workshop Proceedings 1304, 54–61.

OMG, Jan. 2011. Business Process Model and Notation (BPMN), Version 2.0. [Online; accessed 2018-05-12].
URL http://www.omg.org/spec/BPMN/2.0/

Paul, S., 2014. Towards automating the construction & maintenance of attack trees: a feasibility study. In: Kordy, B., Mauw, S., Pieters, W. (Eds.), Proceedings First International Workshop on Graphical Models for Security. Vol. 148 of Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, pp. 31–46.

Pearl, J., 1988. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Poston, R. M., Sexton, M. P., May 1992. Evaluating and selecting testing tools. IEEE Software 9 (3), 33–42.

Schlegel, R., Obermeier, S., Schneider, J., July 2015. Structured system threat modeling and mitigation analysis for industrial automation systems. In: 2015 IEEE 13th International Conference on Industrial Informatics (INDIN). pp. 197–203.

Schneier, B., 1999. Attack trees. Dr. Dobb's Journal: Software Tools for the Professional Programmer 24 (12), 21–29.

Shostack, A., 2014. Threat Modeling: Designing for Security, 1st Edition. Wiley Publishing.

Slay, J., Miller, M., 2008. Lessons learned from the maroochy water breach. In: Goetz, E., Shenoi, S. (Eds.), Critical Infrastructure Protection. Springer US, Boston, MA, pp. 73–82.

Sommestad, T., Ekstedt, M., Holm, H., Sept 2013. The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures. IEEE Systems Journal 7 (3), 363–373.

Spillner, A., Linz, T., Schaefer, H., 2011. Software Testing Foundations: A Study Guide for the Certified Tester Exam, 3rd Edition. Rocky Nook.

Stouffer, K., Pillitteri, V., Lightman, S., Abrams, M., Hahn, A., Jun 2015. Guide to industrial control systems (ics) security. NIST special publication 800 (82r2).

Ten, C., Liu, C., Govindarasu, M., June 2007. Vulnerability assessment of cybersecurity for scada systems using attack trees. In: 2007 IEEE Power Engineering Society General Meeting. pp. 1–8.

VDI/VDE 2182-1, 2011. Sheet 1: It-security for industrial automation - general model.

Vigo, R., Nielson, F., Nielson, H. R., July 2014. Automated generation of attack trees. In: 2014 IEEE 27th Computer Security Foundations Symposium (CSF). Vol. 00. pp. 337–350.

Vu, A. H., Tippenhauer, N. O., Chen, B., Nicol, D. M., Kalbarczyk, Z., 2014. Cybersage: A tool for automatic security assessment of cyber-physical systems. In: Norman, G., Sanders, W. (Eds.), Quantitative Evaluation of Systems. Springer International Publishing, Cham, pp. 384–387.

Vyatkin, V., Aug 2013. Software engineering in industrial automation: State-of-the-art review. IEEE Transactions on Industrial Informatics 9 (3), 1234–1249.

Weippl, E., Kieseberg, P., Sept 2017. Security in cyber-physical production systems: A roadmap to improving it-security in the production system lifecycle. In: 2017 AEIT International Annual Conference. pp. 1–6.

Winkler, D., Meixner, K., Biffl, S., Sept 2018. Towards flexible and automated testing in production systems engineering projects. In: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA). Vol. 1. pp. 169–176.

Winkler, D., Schönbauer, M., Biffl, S., 2014. Towards automated process and workflow management: A feasibility study on tool-supported and automated engineering process modeling approaches. In: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, pp. 102–110.

Xie, F., Lu, T., Guo, X., Liu, J., Peng, Y., Gao, Y., Oct 2013. Security analysis on cyber-physical system using attack tree. In: 2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing. pp. 429–432.

Yee, G., Xie, X., Majumdar, S., July 2010. Automated threat identification for uml. In: 2010 International Conference on Security and Cryptography (SECRYPT). pp. 1–7.

Zareen Syed, Ankur Padia, M. L. M. T. F., Joshi, A., February 2016. UCO: A Unified Cybersecurity Ontology. In: Proceedings of the AAAI Workshop on Artificial Intelligence for Cyber Security. AAAI Press.