Methods

Maximilian Walter*, Sebastian Hahner, Tomáš Bureš, Petr Hnětynka, Robert Heinrich and Ralf Reussner

Architecture-based attack propagation and variation analysis for identifying confidentiality issues in Industry 4.0

Architekturbasierte Angriffsausbreitungs- und Variationsanalyse zur Identifizierung von Vertraulichkeitsproblemen in Industrie 4.0

https://doi.org/10.1515/auto-2022-0135
Received October 19, 2022; accepted January 10, 2023

Abstract: Exchanging data between entities is an essential part of Industry 4.0. However, the data exchange should not affect the confidentiality. Therefore, data should only be shared with the intended entities. In exceptional scenarios, it is unclear whether data should be shared or not and what the impact of the access decision is. Runtime access control systems such as role-based access control often do not consider the impact on the overall confidentiality. Static design-time analyses often provide this information. We use architectural design-time analyses together with an uncertainty variation metamodel mitigating uncertainty to calculate impact properties of attack paths. Runtime access control approaches can then use this information to support the access control decision. We evaluated our approach on four case studies based on real-world examples and research cases.

Keywords: attack propagation; confidentiality; software architecture.

Zusammenfassung: Der Datenaustausch zwischen Partnern ist ein zentraler Bestandteil von Industrie 4.0. Der Austausch sollte aber nicht die Vertraulichkeit verringern. Daher sollten die Daten nur mit den vorgesehenen Partnern geteilt werden. In Sonderfällen ist unklar, ob Daten ausgetauscht werden sollen oder nicht und welche Auswirkungen die Zugriffsentscheidung haben kann. Zugriffskontrollsysteme wie die rollenbasierte Zugriffskontrolle berücksichtigen die Auswirkungen auf die Vertraulichkeit oft nicht. Entwurfszeitanalysen liefern häufig diese Informationen. Wir verwenden Architekturanalysen zusammen mit einem Ungewissheitsvariationsmetamodel, um Angriffspfade zu berechnen. Zugriffskontrollsysteme können diese Informationen während der Entscheidung nutzen. Wir haben unsere Analysen anhand von vier Fallstudien evaluiert, die auf realen Beispielen und Forschungsbeispielen basieren.

Schlagwörter: Angreiferpropagation; Vertraulichkeit; Softwarearchitektur.

1 Introduction

Industry 4.0 is the digitalization of the production process. It promises, for instance, more efficient usage of resources and more customization for products. These benefits are achieved by connecting different systems and enabling data-exchange for them. The used technologies are Cyber-Physical-Systems (CPS) and Internet of Things (IoT) devices. However, with the connected systems also the questions regarding the security and confidentiality arise. It is vital that systems only share selected data and keep the other data secure and confidential. Additionally, these systems are also often critical and cannot be offline or blocked. For instance, a stopped assembly line is costly. Still, there could

***Corresponding author: Maximilian Walter**, Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany, E-mail: maximilian.walter@kit.edu. https://orcid.org/0000-0003-0358-6644
Sebastian Hahner, **Robert Heinrich and Ralf Reussner**, Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany,
E-mail: sebastian.hahner@kit.edu (S. Hahner), robert.heinrich@kit.edu (R. Heinrich), ralf.reussner@kit.edu (R. Reussner)
Tomáš Bureš and Petr Hnětynka, Faculty of Mathematics and Physics, Charles University, Malostranské Náměstí 25, 118 00 Praha 1, Czech Republic, E-mail: bures@d3s.mff.cuni.cz (T. Bure), hnetynka@d3s.mff.cuni.cz (P. Hntynka)

be incidents happening like accidents or system breakdowns. Some of these incidents can be foreseen. Thus, they are already considered during the system's design. However, some situations are not foreseeable or just forgotten through mistakes and therefore unknown to the system. These unknown situations lead the system to an unknown state, and there is uncertainty regarding the access decision. To support the system's decision-making in this state, it can be important that the runtime system gets information about the impact of its decision.

We now introduce our Industry 4.0 running example. We could have a system like in Figure 1. It is based on a scenario from our former Industry 4.0 security research project with industrial partners [1]. We extended it in [2] for attacker information. The scenarios consist of two companies with one manufacturer (M) and one service contractor (S). S maintains the machine of M. For maintaining the machine, S needs access to the log data of the machine. However, access should only be granted when the machine has a failure since the log data might contain sensitive data. The system architecture contains three devices (Storage Server, Terminal Server, Machine Controller) that are connected by a Local Network. Additionally, it contains four components (a) the Production Data Storage which stores production data such as the log data of the machine, (b) the Product Storage which contains confidential blueprints, (c) the Terminal with the interface for the technician, (d) the Machine generates the log data and provides a state.

Analyzing this system during the design-time is beneficial, since there are often security problems already in the design of the software [3] or it has already known vulnerabilities [4]. Design-time analyses such as our attack propagation [2] can find potential attack paths based on the proposed software architecture by using these known vulnerabilities or can identify critical data elements such as our dataflow analysis [5, 6]. However, the results of these analyses are rarely used during the runtime. Additionally, because of their nature as design-time analyses, there is uncertainty regarding the actual runtime properties. For instance, we do not know the initial breaching points or the concrete attack capabilities for our attack propagation. These uncertain properties directly affect the results and reduce the usefulness of the design-time analysis results during runtime.

To increase the usefulness and bridge the gap between design-time and runtime analyses, we looked primarily at runtime access control systems and developed an approach to provide design-time properties such as attack paths or classifications based on static design-time analysis. Our contributions are:

(i) We use a newly developed uncertainty model to calculate attack paths for different initial breach locations and attacker capabilities. These attack paths are used to identify potentially affected data. Additionally, we use our dataflow analysis [5, 6] to categorize this found data based on their criticality. (ii) These properties can then support the access decision during runtime by providing the impact for access decisions.

We evaluated our modified design-time analyses, including the transformation and our uncertainty metamodel for functional correctness. We used four case studies based on real-world system breaches, research case studies, and industrial example scenarios. The results suggest that our analyses produce correct results. Additionally, we investigated the applicability of our design-time properties. For this, we gave access control experts our design-time properties, and they developed access control rules which can consider these design-time properties.

Our paper is structured as follows. We give in Section 2 an overview of our approach. In Section 3 we present our variation metamodel in more detail. We describe in Section 4 our evaluation. Afterwards, we present in Section 5 the related work and Section 6 concludes the paper.

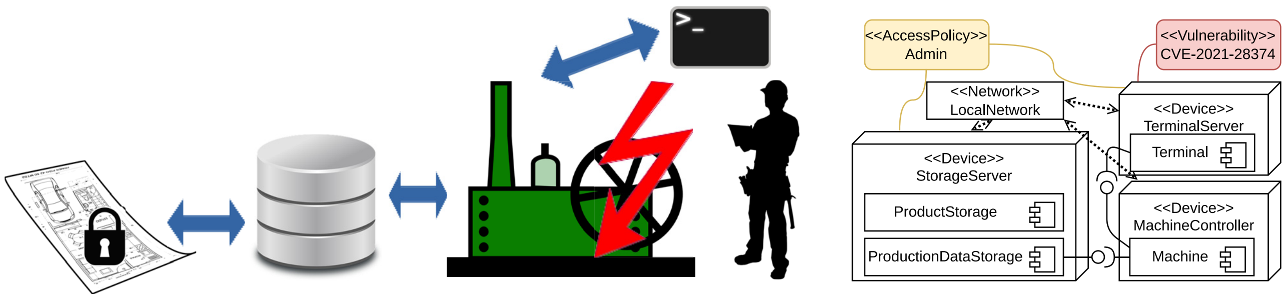

Figure 1: Graphical and architectural representation of the maintenance scenario from Walter, Heinrich, and Reussner [2].

2 Approach overview

Our main idea is to use static design-time analyses results to enhance the decision-making process for runtime access control approaches. This allows considering the impact of enabling potential attack paths to critical data during uncertain access control decisions. In this paper, we focus on the creation of the design-time properties and illustrate how these properties can be used during runtime.

Figure 2 gives an overview of our approach. It consists of the design-time part and the runtime part. The syntax is based on activity diagrams to describe the process. Rectangles with rounded corners describe activities, and regular rectangles describe input or output objects. Our design-time analyses are based on the Palladio Component Model (PCM) [7]. We use an extended version, which supports dataflow definitions [6] and vulnerability specifications [2]. The first part of our approach is the modeling of the software architecture. Here, architects can specify the criticality of the data and define vulnerable system elements such as components or hardware resources. Afterward, our dataflow analysis [6] calculates for each data element the criticality and saves it with a unique ID for later references. During the design-time, it is not clear which components can be attacked or what capabilities attackers have. Therefore, we generate multiple architectural variations for different attack start points and with different attacker capabilities. Architects can define variation points by using our newly developed uncertainty variation metamodel (see Section 3). These variation points are then automatically transformed into different architectural variations. Each architectural variation is then analyzed by our attacker propagation analysis [2], which is based on [8]. We extended the analysis, so it can output the attack complexity of an attack path. For each variation, we store the attack complexity, the starting point, and the affected data for the runtime part. Additionally, architects can define runtime access control policies.

These properties alone can already help architects to increase security. The data classification determines the critical data. Based on our data model, architects can use this information to determine components with critical data. This can then be used to harden these components against attacks. Using our variation model, we can calculate an attack path from every component. Architects and security experts can determine which component is critical and can easily be attacked with the attack paths. Based on these results, they can decide which components need special protection mechanisms or mitigation approaches. This can help already during the design-time to increase the security and confidentiality of the system.

Additionally, our design-time properties, namely data criticality and attack paths with affected data, can then be considered during runtime. However, this consideration is only possible if there is no architectural erosion between the modeled architecture and the runtime system. For this, we differ between architecture evolution and change in runtime properties. If structural information of the

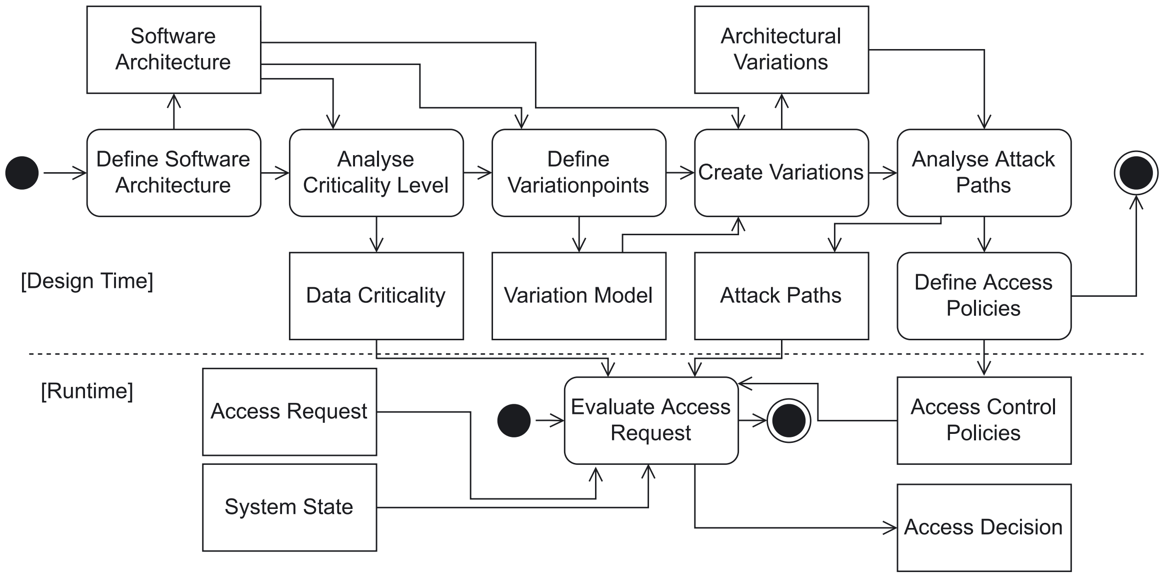

Figure 2: Process overview as an activity diagram with different actions and artifacts during design-time and runtime.

architecture is changed, such as the selection of components or hardware resources, we consider these as an evolution step. For this kind of step, the architect needs to adapt the architectural model and rerun our analysis. In other cases, for instance, that the access request is changed to another component, the architect does not need to update the architectural model. Our uncertainty variation model could also consider some structural changes in the software architecture, such as the known reconfiguration step. However, the runtime access control system would then need to know the complete system state and need to choose the appropriate attack path.

At the runtime, the evaluation of the access control policies could, for instance, consider the access request, the system state, and especially the results of the design-time analysis. This would lead then to an access decision. Notably, the stored list of the criticality and the information about the attack paths are considered to decide whether access can be granted or whether granting access would enable an attacker to access critical data.

3 Enabling security analysis at design-time

Our existing analyses do not consider uncertainty, yet. However, at design-time, much is yet unknown about the system. This includes the system usage and misuse, i.e., information about the attacker and the attacker's behavior. The lack of knowledge introduces uncertainty, e.g., about the starting point of the attack, possible attack paths, and also the system's environment. Although this uncertainty impedes deterministic statements about the overall confidentiality of a system, it is not unmanageable at all. By modeling them as variations of the software architecture, we can make more precise statements about the impact on confidentiality [9].

The concept of analyzing different versions of an architectural model and measuring the impact on quality attributes including for confidentiality has already been introduced by a PerOpteryx extension [10]. However, PerOpteryx so far does not consider attacker propagation and the data classification. Thus, we present an uncertainty variation metamodel to analyze software architecture variations due to uncertainty for attack propagation analyses.

Figure 3 shows the UML class diagram of the *Uncertainty Variation Metamodel*. It represents *Scenario Uncertainty* as Uncertainty Variations, i.e., variations in architectural models due to uncertainty as set of zero or more Variation Points. Each Variation Point has a name and provides its type by the State Handler that can be used to calculate architecture variants. The Variation Points also references all elements of the architectural model that are affected by the modeled variation, i.e., *varying subjects*. To describe all alternatives to choose from as part of the variation, the model defines the Variation Description. This description holds one or multiple alternative values or collections of values that a Variation Point can represent. Examples are different resources to deploy to or different sets of characteristics describing the sensitivity of data. All values, value collections as well as varying subjects are identifiers that reference elements of the model.

To generate the variations, we first compute the state space of all possible combinations. Given n Variation Points $v_0 \ldots v_n$ that have all a Variation Description, let $|v_i|$ the number of values of the Variation Description i. Then, the size of the state space s can be calculated as $s = \prod_{i=0}^{n} |v_i|$, i.e., by multiplying the count of all Variation Description's values. We iterate over this state space and apply the

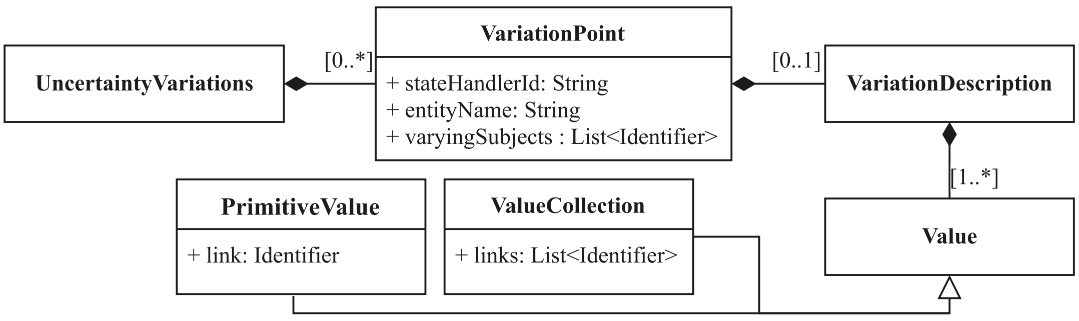

Figure 3: Uncertainty variation metamodel class diagram.

variation to the software architecture. In each iteration, we call the state handlers to modify the referenced varying subjects of the software architecture. Depending on the type of the Variation Point, this means changing references and model elements. Because of our limited scope on our attack propagation, where we only vary the starting point, the creation of all variants should not be problematic. For other scenarios, we plan to further investigate it. Each variation represents one possible runtime scenario and can be analyzed using the attacker analysis.

For our running example, we want to create for each component as a starting point an attack path. We created a new Variation Point and added as the Variation Description the attacker. Afterward, the analysis would create for each component an attacker model. Therefore, we could analyze potential attack paths from each component.

4 Evaluation

We structure the evaluation based on the Goal Question Metric approach. Our first goal **(G1)** is to evaluate the functional correctness for the properties derived from the design-time. Our evaluation questions are: **Q1.1:** *Can we correctly classify the criticality of the data?* **Q1.2.** *Can we correctly determine the affected data of the attack paths?* **Q1.3** *Can we correctly identify the complexity of the attack paths?* **Q1.4** *Can we correctly generate all variations?* In contrast to our previous publication [6], we focus in **Q1.1** not on access violations but on the classification of data elements. Determining the data criticality is essential since a wrong classification might later result in a wrong access decision during runtime. For **Q1.2**, we focus only on the data elements within an attack path since the attack path itself is already evaluated in [2]. **Q1.3** focuses on the new attack complexity element. Therefore, evaluating whether it is calculated correctly is essential. The same applies to **Q1.4**. This describes our newly developed variation model. Our metric for answering **Q1.1**–**Q1.4** is the Jaccard Coefficient (JC) [11]: $JC(A,B) = \frac{|A \cap B|}{|A \cup B|}$. It is used to compare the similarity between two sets, and its values range from 0.0 for no intersection in the sets and 1.0 if both sets are identical. We can apply this metrics, since the order of the sets are irrelevant. A detailed description of the sets can be found in the Evaluation Design. Our second goal (**G2**) concentrates on the applicability for the runtime. The evaluation question is **Q2.1**: *Can access control policies be formulated that consider our design-time properties (data criticality and attack paths with affected data)?* This question is important, since so far we only evaluated what scenario we can model and how correct the results of the analyses are. However, if our design-time properties cannot be considered during the runtime, we would only have the design-time benefits for the analyses. Therefore, it is important to investigate, whether an access control framework can theoretically consider our properties. Our metric for answering **Q2.1** is the ratio (a) between the scenario (s_1), which can consider our properties and all scenarios s_{all}: $a = \frac{s_1}{s_{\text{all}}}$.

4.1 Evaluation design

We choose four case studies for the evaluation since case studies might provide better insights, show the applicability and increase the comparability between different approaches. We reused three case studies from [2]. An in depth description of the case studies can be found due to space reason in [2]. Additionally, we investigated the running example. While the three reused case studies are not directly settled in the Industry 4.0 environment, they share important similarities with Industry 4.0 cases such as the usage of CPS, IoT devices together with business backends. Therefore, the results should be transferable to the industrial domain. Two case studies (Target, Power Grid) are based on real-world system breaches. The other ones are based on a research case study in the confidentiality community (Travel Planner) or on a scenario from a previous project with industrial partners (Maintenance Scenario). All the used models and results can be found in our dataset.[1]

For **G1**, we created for each question a reference output and compared it to the analysis results. For **Q1.1** we considered the output of our analysis as a set of tuples. Each tuple contains a data identifier and a classification. The ordering of this set is irrelevant since only the tuple describes a data element, and there is no dependency between the tuples in the output. We manually created the reference set by creating a reference list with tuples. For **Q1.2**, we looked at one concrete attacker with a concrete starting point and generated the list of affected data elements. This is a set of data identifiers for each case study. For the reference output, we manually investigated the attack path with an attacker with the same capabilities and the same starting point. Afterward, we created a set of data objects.

1 Available online: https://doi.org/10.5281/zenodo.7214894.

Again the set of the data elements can have an arbitrary ordering since there are no dependencies between the different elements in the output, and the ordering is not relevant. For **Q1.3**, the output can be seen as a triple of attacker name, attacker starting point, and attack complexity. We reuse here the attack paths from the previous question. For this attack path, we analyzed the complexity of the attacks and then generated the reference output set. Again the ordering of the set is irrelevant since there is no dependency between the triples, and the order is not considered. For **Q1.4**, we considered the selected attacker as an element in the set. We created the manual set by referencing the generated attacker variations. The ordering is not relevant since each attacker analysis is performed independent, and only the referenced attacker is necessary and not the position in a list.

We evaluated our second goal (Applicability) G2 by giving our project partners the list of design-time properties and asking them to model access control policies considering these properties. Our project partners developed during a previous project together with industrial partners an adaptive access control system [12] for dynamic access control in Industry 4.0, which they used to specify the policies. The procedure was, that we gave them for each case study a scenario to model, and they have written rules including the properties.

4.2 Correctness results and discussion

For each case study and question, we got a JC of 1.0. These are perfect results. It means that our analysis returns, for each evaluation question and case study, similar results to our manual-created reference output. This means that for our cases, the analysis correctly classifies data in our architectural model (**Q1.1**). Additionally, it indicates that our variations creations work correctly (**Q1.4**). These variations are then used to calculate the correct list of affected data for our cases and assign the correct attack complexity (based on **Q1.2** and **Q1.3**). Thus, the results indicate that our design analyses produce correct properties for the runtime access control decision.

4.3 Applicability results and discussion

Overall each scenario could be modeled. Therefore we achieved for our case study a ratio of 1.0, which is a perfect result. In the rest of the section, we will describe one example in more depth.

```
1  class Worker(/*...*/ )  extends Component { /*...*/ }
2  class Machine ( /*...*/ ) extends Component { /*...*/ }
3  class FactorySystem(factory: Factory) extends RootEnsemble
       {
4    class ShiftTeam(shift: Shift) extends Ensemble {
5      object BrokenMachines extends Ensemble {
6        val brokenMachines = factory.machines.filter(p =>
             p.hasFailure)
7        val accessBrokenMachine =
             rules(brokenMachines.map(new
8            AccessBrokenMachine(_)))
9         class AccessBrokenMachine(machine: Machine) extends
              Ensemble {
10          val technican = oneOf(factory.technicans.filter(w
                => w.canRepair(machine)))
11          allow(technican.selectedMembers,
                "read.machineData", machine)
12          notify(technican.selectedMembers,
                RepairMachineNotification(machine))
13          constraints{
14            attackDifficulity(machine, CRITICAL) > EASY
15          }
16        }
17        constraints {
18          accessBrokenMachine.map(_.technican).allDisjoint
19        }
20      }
21    }
22    /*...*/
23 }
```

Listing 1. Ensemble for the maintenance case study.

Listing **one** shows an excerpt of the access control specification of the maintenance case study. A system is specified via two main concepts – components and ensembles. Components represent actual entities of a system (e.g. the machine). Ensembles are dynamically established groups of components. In detail, an ensemble determines a dynamically emerging situation, identifies components that take part in the situation, and specifies access rules for the components. Ensembles can be hierarchically. The top-level ensemble describes an overall goal of a system, and its nested ensembles represent individual sub-goals (the top-level FactorySystem ensemble – line 3 – describes the whole factory system while its nested ShiftTeam ensemble – line 4 – describes goals for an individual shift). At runtime, ensembles are continuously evaluated via an adaptation controller, and the evaluation also considers the results of the design-time analysis. The Broken Machines ensemble is our maintenance scenario. (lines 9–20), which manages access to machines in need of repair. This ensemble selects all the broken machines in the factory (line 6) and then, for each broken machine, instantiates the AccessBrokenMachine ensemble (instantiation at line seven and the ensemble at lines 9–16). The AccessBroken Machine selects an appropriate technician, assigns him the rights to access the machine (line 11, and notifies him to start the machine fix (line 12). However, the technician is allowed only in case the access to the machine does not offer an easy way to attack the machine (or others in the factory) sensitive and critical data (the constraint at line 14). This is implemented in the function attackDifficulty, which internally takes results computed during the design-time analysis and directly tests the possible attack paths starting from the given machine and returns the difficulty of the potential attack.

4.4 Threats to validity

We categorize our threats to validity based on the guidelines from [13].

4.4.1 Internal validity

The internal validity describes that only expected factors affect the results. Regarding G1, the results are affected mainly by the reference sets. We lowered the risk by using real-world examples and existing literature [3, 14–16] to create them. While our case studies are comparatively small, they already cover together most of our design-time functionality. Thereby, adding more architectural elements or considering bigger system does not bring new insights regarding our evaluation goal. The implementation of the access control policies mainly affects the result of G2. Based on our evaluation goal, we only investigate, whether policies considering our properties can be expressed. To avoid conformation bias, the runtime time policies are formulated by a different group then the group of the design-time analysis. This allowed the runtime experts to come to their own results and reduced the bias of the design-time group.

4.4.2 External validity

This describes how generalizable the results are. Using case studies and application scenarios increase the insight but might decrease the generability. For avoiding this specialization, we choose different external case studies. The Target and Power Grid case studies are based on actual system breaches. The Travel Planner case study is also considered by other confidentiality approaches such as [17]. Additionally, we used these three case studies already in previous publications [2, 6]. Our fourth case study is based on a scenario described by industrial partners in a former research project [1]. Therefore, we assume the risk to be low. Nevertheless, we plan to extend the evaluation of our design-time approaches in the future. The choice of the access control system could affect our external validity. The chosen access control system is very specific. Other more restricted access control system might not be useable. However, our design-time properties can be reduced to attributes. Therefore, it should be possible to transfer it to other more commons approaches like ABAC.

4.4.3 Construct validity

This describes whether our metrics can be used to answer our research goal. Regarding G1, the correctness is the comparison between the result set and the expected set, and this is the intended usage of our chosen metric. Additionally, the order of the set is not relevant. Therefore, our metric can be applied. For answering G2 we chose a very simple ratio between the scenarios where we could formulate access control policies and the overall scenarios. Therefore, we assume that this is acceptable.

4.5 Assumptions & limitations

One fundamental assumption is that we have architectural models of the systems. While it is beneficial to design the software architecture and integrate our design-time analysis already in the design, legacy systems might not have architecture models. Generating these models is quite cumbersome, and it could limit our approach. However, using existing reengineering approaches might help here and reduce the effort. Nevertheless, there is still some manual effort required. Currently, our approach only considers simple mitigation strategies such as access control or network segregation. More advanced strategies such as Trusted Execution Environments are not supported. A possible solution might be to remove vulnerabilities manually when mitigation strategies prohibit them. In the future, we want to investigate this aspect more and maybe combine a mitigation model with our vulnerability model.

5 Related work

We split the discussion about related work along our different analyses. In [6], we discuss already related approaches for the dataflow analysis. Other related model-driven approaches can be found in [18]. Overall, these related approaches often do not analyze the dataflow at the architectural level. We discuss related approaches regarding our attack propagation in [2]. Kordy et al. [19] also lists different approaches. Most of the related approaches do not consider fine-grained access control and vulnerabilities. They focus only one aspect, or a very specialized for one application domain. Regarding the variation modeling, there exists the Peropteryx extension for confidentiality [10], which generates architectural variations and analyzes them. Also in the domain of product lines, they use the concepts of variants [20]. In contrast, our model is more specialized, since it only should cover uncertainty for confidentiality analyses.

6 Conclusions

In the paper, we presented an approach that analyzes what data is affected by system breaches based on the software architecture in Industry 4.0. This is achieved by assigning data a category for criticality based on our dataflow analysis and metamodel. The approach considers the uncertainty regarding the attack capabilities and initial breach point during the design-time with our uncertainty variation model. This information can be used during the runtime to enhance the decision-making of access control approaches. We evaluated our approach on four case studies, and the evaluation indicates correct results. Additionally, we illustrated that our design-time properties can be used in access control rules.

Using our approach is beneficial to identify potential problematic breach locations during the design-time. This can help architects and system designers to harden their systems and fix security/confidentiality problems in the design. Additionally, our approach can be used to document known security problems in the system, and this eases the communication between different stakeholders. During the runtime, our generated properties can provide the access control system with information regarding the impact of access decisions.

In the future, we want to investigate more case studies and evaluate the benefit of integrating the design properties in the access control system. For this, we will look at different case studies and example models. Regarding the design-time analyses, we want to integrate approaches, which can automatically update the vulnerabilities and the system architecture. This would increase the accuracy of our impact descriptions for the runtime access control system.

Acknowledgement: We like to thank Patrick Mehl, who supported us in the development of the uncertainty variation metamodel during his practical course.

Author contributions: All the authors have accepted responsibility for the entire content of this submitted manuscript and approved submission.

Research funding: This work was supported by the German Research Foundation (DFG) under project number 432576552, HE8596/1-1 (FluidTrust), as well as by funding from the topic Engineering Secure Systems (46.23.03) of the Helmholtz Association (HGF) and by KASTEL Security Research Labs. Additionally, it was supported by the Czech Science Foundation project 20-24814J, and also partially supported by Charles University institutional funding SVV 260588.

Conflict of interest statement: The authors declare no conflicts of interest regarding this article.

References

[1] R. Al-Ali, H. Robert, H. Petr, J.-V. Adrian, S. Stephan, and W. Maximilian, "Modeling of dynamic trust contracts for Industry 4.0 systems," in *ECSA-C'18*, Madrid, Spain, ACM, 2018.

[2] M. Walter, R. Heinrich, and R. Reussner, "Architectural attack propagation analysis for identifying confidentiality issues," in *ICSA'22*, Honolulu, HI, USA, IEEE, 2022.

[3] OWASP, *OWASP Top Ten Web Application Security Risks*, 2021. Available at: https://owasp.org/www-project-top-ten/ [accessed: Oct. 25, 2021].

[4] HP, *HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack*, 2014. Available at: https://www.hp.com/us-en/hp-news/press-release.html1744676 [accessed: Oct. 05, 2021].

[5] S. Seifermann, R. Heinrich, and R. Reussner, "Data-driven software architecture for analyzing confidentiality," in *ICSA'19*, Hamburg, Germany, IEEE, 2019, pp. 1–10.

[6] S. Seifermann, R. Heinrich, D. Werle, et al., *Journal of Systems and Software*, vol. 184, 2022, Art. no. 111138.

[7] R. Reussner, S. Becker, J. Happe, et al., *Modeling and Simulating Software Architectures – the Palladio Approach*, Cambridge, MA, MIT Press, 2016, p. 408.

[8] R. Heinrich, S. Koch, K. Busch, R. Reussner, and B. Vogel-Heuser, "Architecture-based change impact analysis in cross-disciplinary automated production systems," *JSS*, vol. 146, no. 146, pp. 167–185, 2018.

[9] S. Hahner, S. Seifermann, R. Heinrich, and R. Reussner, "A classification of software-architectural uncertainty regarding confidentiality," in *ICETE. To Appear*, Cham, Springer, 2023.

[10] M. Walter, S. Hahner, S. Seifermann, et al., "Architectural optimization for confidentiality under structural uncertainty," *ECSA*, vol. 2021, pp. 309–332, 2022.

[11] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.

[12] R. Al-Ali, P. Hnetynka, J. Havlik, et al., "Dynamic security rules for legacy systems," in *ECSA 19 – Volume 2*, New York, NY, USA, ACM, 2019, pp. 277–284.

[13] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” *Empir. Softw. Eng.*, vol. 14, no. 2, pp. 131–164, 2008.
[14] B. A. Hamilton, “Industrial cybersecurity threat briefing,” *Tech. rep.*, p. 82, 2016.
[15] M. Plachkinova and C. Maurer, “Security breach at target,” *J. Inf. Syst. Educ.*, vol. 29, no. 1, pp. 11–20, 2018.
[16] X. Shu, K. Tian, A. Ciambrone, and D. Yao. “Breaking the target: an analysis of target data breach and lessons learned.” In: arXiv:1701.04940 [cs], 2017.
[17] K. Katkalov, *Ein modellgetriebener Ansatz zur Entwicklung informationsflusssicherer Systeme.” doctoralthesis*, Augsburg, Germany, Universität Augsburg, 2017.
[18] P. Nguyen, M. Kramer, J. Klein, and Y. L. Traon, “An extensive systematic review on the model-driven development of secure systems,” *Inf. Softw. Technol.*, vol. 68, pp. 62–81, 2015.
[19] B. Kordy, L. Piétre-Cambacédès, and P. Schweitzer, “DAGbased attack and defense modeling: don’t miss the forest for the attack trees,” *Comput. Sci. Rev.*, vols. 13–14, pp. 1–38, 2014.
[20] S. Ananieva, S. Greiner, T. Kühn, et al., “A conceptual model for unifying variability in space and time,” in *SPLC ’20 Volume A Online*, New York, NY, USA, Association for Computing Machinery, pp. 148–158, 2020.

Bionotes

Maximilian Walter
Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany
maximilian.walter@kit.edu
https://orcid.org/0000-0003-0358-6644

Maximilian Walter has been a researcher at the chair of Dependability of Software-intensive Systems Karlsruhe Institute of Technology (KIT), since 2017. He holds an M.Sc. in Informatics from the University of Augsburg. He focuses on research regarding security modelling and software architecture.

Sebastian Hahner
Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany
sebastian.hahner@kit.edu

Sebastian Hahner has been a researcher at the KIT since 2020 where he graduated with M.Sc. in Informatics. His research is centered around the impact of uncertainty on confidentiality and access control. Besides his university activities, he has more than a decade of experience in video production and has one of Germany’s largest social media presence in tech education.

Tomáš Bureš
Faculty of Mathematics and Physics, Charles University, Malostranské Náměstí 25, 118 00 Praha 1, Czech Republic
bures@d3s.mff.cuni.cz

Tomáš Bureš is a professor at the Department of Distributed and Dependable Systems, Faculty of Mathematics and Physics of the Charles University, Prague. His research focus is on self-adaptive systems, smart cyber-physical and IoT systems, machine learning for self-adaptive systems, edge-cloud systems, model-driven development, and modern programming languages.

Petr Hnětynka
Faculty of Mathematics and Physics, Charles University, Malostranské Náměstí 25, 118 00 Praha 1, Czech Republic
hnetynka@d3s.mff.cuni.cz

Petr Hnětynka is an associate professor at the Department of Distributed and Dependable Systems, Faculty of Mathematics and Physics of the Charles University, Prague. His research focus is on self-adaptive systems, dynamic software architectures, component-based development and model-driven development.

Robert Heinrich
Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany
robert.heinrich@kit.edu

Robert Heinrich heads the Quality-driven System Evolution research group at KIT and the Mobility Lab at KASTEL. His research interests include software quality modeling and analysis, in particular, the (de)composition of model-based analysis to provide more flexibility in model-driven engineering. He is involved in the organization committees of several international conferences, initiated and organized various workshops, and is reviewer for international premium journals.

Ralf Reussner
Dependability of Software-Intensive Systems Group (DSiS), Karlsruhe Institute of Technology (KIT), Institute of Information Security and Dependability (KASTEL), Am Fasanengarten 5, 76131 Karlsruhe, Germany
ralf.reussner@kit.edu

Ralf Reussner has been a full professor for software engineering at Karlsruhe Institute of Technology (KIT) since 2006. He holds the chair for Dependability of Softwareintensive Systems, heads the Institute for Information Security and Dependability and is Director at the FZI Research Center for Information Technology. His research group works in the interplay of software architecture and predictable software quality as well as on view-based design methods for software-intensive technical systems.