Architectural Attack Propagation Analysis for Identifying Confidentiality Issues

Maximilian Walter
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
maximilian.walter@kit.edu

Robert Heinrich
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
robert.heinrich@kit.edu

Ralf Reussner
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
ralf.reussner@kit.edu

***Abstract*—Exchanging data between different systems enables us to build new smart services and digitise areas in our world. This digitalisation leads to more efficient usage of resources, and an increased monetary value. However, the connection of different systems also increases the number of potential vulnerabilities. The vulnerabilities on their own might be harmless, but attackers could build attack paths based on the combination of different vulnerabilities. Additionally, attackers might exploit existing access control policies to further propagate through the system. For analysing this dependency between vulnerabilities and access control policies, we extended an architecture description language (ADL) to model access control policies and specify vulnerabilities. We developed an attack propagation analysis operating on the extended ADL, which can help to determine confidentiality violations in a system. We evaluated our approach by analysing the accuracy and the effort compared to a manual analysis using three case studies with different scenarios. The results indicate that our analysis is capable of identifying attack paths and reducing the effort compared to manual detection.**

I. Introduction

We are in the progress of digitising various areas in our life. This digitalisation includes, for instance, the energy sector with smart meters, the health sector with various eHealth services, or the industrial sector with Industry 4.0. These systems have in common that they often use or require dynamic access control systems together with a multitude of different connected components or devices. These connected components are often from different vendors and organisations. However, studies such as [1], [2] suggest that these connected elements are often vulnerable, even with known vulnerabilities. Additionally, a study of the UK government [3] revealed that about 20% of the Windows installation in companies are outdated. Therefore, they may contain known security vulnerabilities. Keeping these systems up to date is a costly process and for some components may not be possible, especially in embedded systems. On their own, the vulnerabilities also might not be problematic since they often require certain privileges to exploit or are hidden within internal networks with no access from the outside. Using techniques, such as access control or network segregation, vulnerabilities can be mitigated.

However, with the rise of advanced persistent threat (APT) attacks [4], uncritical vulnerabilities can be combined into more complex attacks. This leads to problems for the security of the system. Attackers may exploit these uncritical vulnerabilities to gain more and more credentials of the system till they find critical elements. Especially since the attackers are often using phishing attacks to get credentials to the system [3]. These attacks are then combined with other attacks or gained privileges to breach the system even more [3]. All the combined attacks build an attack path that can be used to retrace the attack. Additionally, the specification of dynamic access control policies, such as attribute-based access control (ABAC) [5] is complicated [6] because of the range of possible values, and it is hard to estimate the impact of access control policies on the confidentiality of the system [7]. This holds especially for more subtle consequences like enabling a malicious user to propagate easier through the system. Considering these consequences is especially relevant since the increase in connected elements also add new vulnerabilities and allow additional attack vectors.

Therefore, analysing this dependency between access control policies and vulnerabilities is useful. While there exist already approaches for the automatic generation of attack paths based on vulnerabilities and access control such as Bloodhound [8] for the Active Directory or approaches from the Darpa Cyber Grand Challenge (CGC) [9], these approaches require at least an already deployed system and cannot be used during the design time or downtime of the system. Therefore, modelling approaches are useful. Approaches based solely on the network topology such as in [10], [11] do not consider deployment or behaviour. Architecture description languages (ADLs) can provide this information and, therefore, architectural analysis can provide better results.

Additionally, an ADL still provides useful abstractions to the implementation. This helps architects and security experts to identify the relevant parts and develop solutions for the confidentiality issues or avoid them. On the other hand, there exist design-time approaches to analyse the access control on an architectural level, e.g. [12], [13]. However, these approaches do not consider the interplay between system vulnerabilities and dynamic access control policies. Our contributions are:
C1) We developed an extension for an architecture description language (ADL) [14], that allows modelling vulnerabilities and access control policies. In contrast to existing approaches (see section IV), we use a fine-grained access control model together with an vulnerability modelling based on commonly used attack classifications.
C2) This extension can be used in our attack propagation. It analysis how a malicious user propagates through the system by

using stolen credentials and exploiting existing vulnerabilities. Our analysis can consider the additional information of the architecture such as deployment, behaviour and considers fine-grained access control policies together with vulnerabilities. These contributions give an understanding of the dependencies between access control policies and system vulnerabilities and how attackers might exploit both for attack propagations.

We evaluated the accuracy, and effort reduction for our approach in contrast to a manual analysis. We used two case studies based on real-world descriptions of system breaches (Target [15], [16], Power Grid [17]) and a confidentiality research case study (TravelPlanner [18]).

The paper is structured as follows. We first introduce the Foundations in section II. We describe our running example in section III. With the example, we motivate the gap in the state of the art in section IV. We give an overview of our envisioned process in section V. In section VI and section VII, we introduce our metamodel and our analysis. Section VIII describes the evaluation and the limitations of our approach. Finally, section IX concludes the paper.

II. Foundations

We base our approach on an ADL, access control model and an existing propagation framework.

Architectural Modelling Language: Our ADL is the Palladio Component Model (PCM) [14] since there exists already confidentiality analyses, such as [12], [19]. Additionally, the modelled system can be reused in other analyses such as performance or reliability [14], which splits the overall effort to create the model. PCM supports the component-based development process with components and provided and required interfaces. Components and their required interfaces are specified in a repository. Components can provide different services, called ServiceEffectSpecification (SEFF). Their signatures are declared in the provided interfaces. A service can call another service, which is called external call. In the system model (or assembly), the different components are instantiated and wired to each other. It also describes the public services a system provides and that an actor (user) can call. The user behaviour is modelled in the usage model. The hardware of the system is modelled in the resource environment. It contains hardware resource containers, which models servers or other processing units and linking resources, which are network nodes. The allocation model stores the deployment information. We consider the following PCM elements as architectural elements: 1) `LinkingResource` representing network switches, 2) `ResourceContainer` representing hardware devices such as servers, 3) `AssemblyContext` representing executions environments of components or instantiated components. Additionally, we consider the provided service of an instantiated component as an architectural element. This is not yet considered in PCM.

Access Control Model: Attribute-Based Access Control (ABAC) [5] is an access control model that considers the system context for granting access to resources. Different attributes model the context. The attributes are grouped into four groups. 1) Attributes of the subject, which requests the resource 2) Attributes of the environment, such as the time 3) Attributes of the requested operation, such as read or write 4) Attributes of the requested resource, such as the filename. The policies are then specified by a conjunction of boolean expressions, which must be fulfilled to grant access. An ABAC system commonly consists out of multiple different components. For us, only the Policy Decision Point (PDP) is interesting. The PDP does the actual evaluation of the policy and sends the decision to the Policy Enforcement Point (PEP), which then guarantees that the decision is respected. ABAC is often used in dynamic environments, where access should be granted dynamically based on the context. An industrial standard for ABAC is the eXtensible Access Control Markup Language (XACML) [20]. It provides a framework of how to structure access control policies in general and especially for ABAC.

Propagation Concept: Our attacker propagation concept is based on the Karlsruhe Architectural Maintainability Prediction (KAMP) approach [21], [22], which calculates the propagation of maintenance tasks. The basic idea is to provide a change metamodel and propagation rules, which then propagate changes. For our approach, we created an adapted change metamodel for our extension metamodel and developed new propagation rules for the attack propagation.

III. Running Example

Our running example is based on the scenario from Al-Ali et al. [23]. While it is settled in an Industry 4.0 environment, the approach itself is not restricted to such environments and can be used for other component-based systems. The example consists of two companies. The first one is a manufacturer (M), and the second one is a service contractor (S) to maintain the machines of M. During normal operations, S should not be able to read the log data of M's machines. The access is restricted because they may contain sensitive information, such as the names of the employees of M or detailed operation information. However, in case of a machine failure, a technician from S can check the log data. Figure 1 illustrates our running example with its components and their deployment. It contains four components: a) the `Terminal` for the technician interface, b) the `Machine` for the log data and its current state, c) the `ProductionDataStorage` for storing machine data, d) the `ProductStorage` for storing the blueprints. The system is deployed on three hardware resources and are connected via a local network.

IV. State of the Art

We categorised the state of the art in approaches for Policy Analysis, Model-based Confidentiality Analysis, and Attacker Modelling. While there exist many approaches in these areas, we concentrated on the most relevant ones for our approach.

Policy Analysis: Margrave [24], introduced by Fisler et al., is a tool to analyse XACML [20] policies. Margrave transforms the policy into binary decision trees. Using the trees, Margrave can verify whether a specific user can access certain operations or calculate a change impact on the policies. Alberti et al. [25]

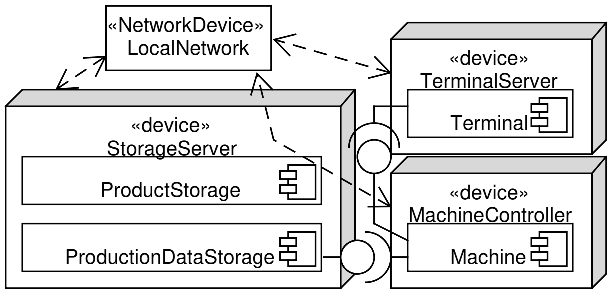

Fig. 1. Simplified component and deployment view of the running example

developed a policy analysis for a modified Role-based access control (RBAC) [26] approach, which additionally supports some attributes that might be context properties. They analyse, for instance, whether through the delegation of RBAC policies, an untrusted role can be reached. Turkmen et al. [27] developed an XACML analysis based on SMT. They support different types of analyses, such as a change impact analysis and attribute hiding [27]. In contrast to the mentioned approaches, our approach cannot provide these formal properties. However, none of the mentioned approaches supports attack propagations or an analysis based on the software architecture. Nevertheless, the approaches based on XACML could be used in addition to ours to verify the access policies.

Model-driven Confidentiality Analysis: UMLsec [28] is an extension to UML for modelling security properties. They provide different security analyses, such as a secure communication analysis and a dedicated attacker model. SecureUML [29] is another UML extension for security. Here also the access control policies are investigated. They use an RBAC approach which can be extended with OCL statements for dynamic access control. They also provide an automatic analysis [13]. SecDFD [30] is a dataflow analysis for information security. They extended classical data flow diagrams with a security DSL. Gerking and Schubert [31] presents another information flow approach. They present rules for decomposing security policies and keeping the real-time properties. The iFlow approach [18], [32] also provides an information flow analysis by using a UML profile. Data-centric Palladio [12] extends the Palladio approach [14] with dataflow definitions to identify confidentiality issues. There also exists an extension to support context-based access control [33]. Kramer et al. [19] provide an attacker intrusion analysis. However, they do not consider the propagation of attacks through vulnerabilities. In conclusion, all mentioned approaches are missing the attack propagation based on vulnerabilities in combination with credentials.

Attacker Modelling: Many approaches model system attacks based on directed acyclic graphs [34]. One popular way is the attack tree introduced by Schneier [35]. It is similar to a fault tree in failure analysis. These attack graphs or trees can be used to describe attack paths through the system. Different approaches automatically calculate these paths. For instance, the Cyber Security Modeling Language (CySeMoL) [36] analyses the vulnerability of enterprise architectures by calculating an attack graph. CySeMoL uses probabilistic values to determine the likelihood of a successful attack on a system. Afterwards, it returns a probabilistic value of how secure the system is. However, determining the initial values for the probability of an attack is quite cumbersome, and so far, the propagation does not consider dynamic access policies like ABAC [5]. Approaches such as [37], [38] or Deloglos et al. [39] reuse existing vulnerability classifications to calculate the attack paths. However, so far they do not explicitly consider access control policies. In contrast, Aksu et al. [10] and Yuan et al. [11] consider, additionally, the privilege of the attacker. However, the access control model in both cases are simple and they do not use a fine-grained access control model together with an architectural model.

Overall so far, to our knowledge, there exists no approach which considers architectural attack propagation based on the dynamic access control policies such as ABAC and vulnerabilities.

V. Approach Overview

Overall, our approach should consider existing vulnerabilities and access control policies to generate attack paths based on a starting point in the software architecture. Applying our approach can be split into 6 activities: 1) **Create Software Architecture:** Here, we reuse the PCM. The architect can either model the architecture manually, for instance, during the development phase or use existing reengineering approaches such as SOMOX [40] or [41]. 2) **Create Attacker Model:** The architects use our newly developed attacker model to specify which types of attacks should be investigated. For specifying the attack types, they could reuse existing resources about attackers such as OWASP [42]. 3) **Create Access Control Model:** In this step, the architects can specify for each architectural element the access control policy by using our access control policy model. 4) **Create Vulnerability Model:** Here, architects can define the concrete vulnerabilities of architectural elements by using our vulnerability model. Architects can model these manually or, in case of existing software, reuse existing automated vulnerability analyses such as snyk [43] or vulnerability databases such as the National Vulnerability Database (NVD) [44]. 5) **Run Analysis:** The attacker propagation calculates attack paths based on the access control decisions, the vulnerabilities, and the attacker model. It then returns a list of compromised architectural elements. 6) **Evaluate Result:** In the last step, architects can evaluate the results. Based on these results, they can change the models from steps 1-4 to mitigate certain attack paths. This could be stricter access control rules, changing vulnerable architectural elements or introducing mitigation approaches for certain vulnerabilities. Otherwise, they can decide the results are sufficient, for instance, if the system can only be compromised by a very rare attacker type. Steps 2-4 can also be executed in parallel since they are mostly independent of each other. Therefore, the architect could be split into multiple experts for vulnerabilities, access control and the attacker model.

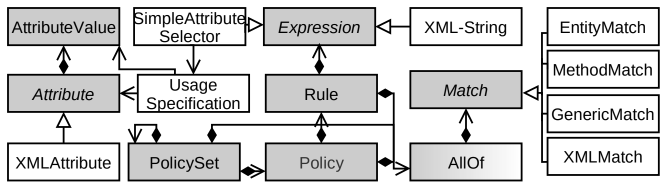

Fig. 2. Excerpt of the access control policy metamodel

VI. Modelling Access Polices & Attackers

During an attack, attackers can exploit the different vulnerabilities and access control policies. These vulnerabilities and access control policies are tied to architectural elements. Therefore, it is necessary to model these inside an ADL, so that later the analysis can use this information to calculate attack paths. However, modelling only vulnerabilities or only attacker policies is not sufficient for the analysis since they are interdependent. Access control policies on their own cannot describe how attackers exploit weakness in components such as the Heartbleed [45] bug. On the other hand, some exploits only work for authorized users. Therefore, we extended the PCM to support access control policy modelling and vulnerability modelling.

A. Modelling Access Control Policies

For considering access control in the attack propagation, we need to model access control policies. These policies can be used to determine whether an attacker could access an architectural element. Our access control model follows the ABAC [5] model. This allows us to model access control policies for dynamic environments. ABAC uses the attributes of the subject, environment, operation, and resource (see section II) to determine the authorisation. We reused this idea by developing a metamodel, which closely resembles the OASIS standard XACML 3.0 [46] for ABAC. This allows architects to easily learn our modelling approach since they might be already familiar with it. Additionally, our modelled policies can not only be used for our analysis but also be reused during the runtime in any XACML compatible PDP, which saves effort and avoids failures by specifying new runtime access control policies. While we adapted and extended some parts for easier modelling of access control policies in PCM, the attack propagation analysis transforms the adapted model into a valid XACML model. Figure 2 provides an excerpt of the access control policy metamodel. The grey elements are similar to the original XACML metamodel, while the white elements are new elements, which are not in the XACML metamodel. The `AllOf` element is a combination since we changed here the target definition of XACML for easier modelling. The complete metamodel can be found in our dataset [47]. An access control model then consists of one root `PolicySet`. A `PolicySet` can contain multiple `Policy` elements, `PolicySet` elements and has an optional target for which it applies. The target is here directly the `AllOff` element in contrast to the XACML metamodel. A `Policy` contains multiple `Rule` elements and also an optional target. The `Rule` element stores the actual access decision such as permit or deny. These decisions are then propagated upwards to the root `PolicySet`. Therefore, each element (Policy, PolicySet) also contains combination algorithms. These describe how the access control decisions are combined. Additionally, the `Rules` contains again an optional target and the condition for the access decision, which is stored in the `Expression`. The `Expression` is a function with arbitrary parameters, which returns a boolean value. The parameters themself can also be functions. For an elaborate list of the available functions, see the XACML standard [46]. Besides the functions of the XACML standard, we also added the `SimpleAttributeSelector`, which is a wrapper for multiple XACML functions to provide a simple comparison of attributes. Additionally, we allow the architects to specify with the `XML-String` element valid XACML statements. These are later embedded in the XACML file for the analysis. The target definition describes on which elements the `PolicySet`, `Policy`, `Rule` should be applied. We define for this the `AllOf` elements. The `AllOf` consist of multiple `Match` elements.

Each `Match` element returns a boolean value and they form a conjunction. If this conjunction is true, it can be applied. Each of the elements (`PolicySet`, `Policy`, `Rule`) can have multiple `AllOf` elements. The semantics of these elements are later in the analysis a logical disjunction. This means that only one `AllOf` element needs to be true, so that it can be applied. We provide custom match operations specific to the PCM. The `EntityMatch` is for selecting an architectural element of the PCM. The `MethodMatch` is for selecting a system service. Here, we also introduced a new architectural element that links a PCM `AssemblyContext` or `Connector` to a PCM `Signature` to identify called services. This was necessary because there are currently no elements on the system level in Palladio to identify called services. The `GenericMatch` is used to describe the regular XACML match operation. Additionally, we provide with `XMLMatch` again an element that enables the security experts to directly write an XACML statement. Each `UsageSpecification` contains a reference to a `Attribute` and its concrete value as `AttributeValue`. For instance, the role of a user is the attribute role and the concrete role such as technician from our running example is the attribute value. The `UsageSpecification` is always used, when we want to reference a concrete attribute in our policy model. The reason for this is the XML extension with custom attributes provided by the `XMLAttribute`. Security experts can here define custom XACML attributes and custom datatypes, which are not included in our metamodel. They are then later embedded in the XACML file. However, architects are not required to use our adapted access control model. They also can provide their own XACML policy file. Using their own file, architects have to make sure that the policy definitions match the system architecture. In case architects use our model, this is only required for the custom XML extensions.

The access control policy of the technician from our running example looks like in Listing 1. The actual `Rule` is contained by Policy and PolicySet elements. Each contains the same combining algorithm, which denies all access unless there is a permit. The Rule then contains the actual decision (permit) and the expression that is the access condition. In our example, we check for two attributes: the role technician and the failed state of the machine. Additionally, the Rule contains an target, represented by AllOf. There we select the machineController.

```
PolicySet{
  combining:Deny_Unless_Permit
  Policy{
        combining:Deny_Unless_Permit
    Rule{
          name:"Technician with Machine  failure"
          decision:permit
          Expression{
           and{
                SimpleAttributeSelector{
                 UsageSpecification{
                  attribute:role
                  value:technician}
                }
                SimpleAttributeSelector{
                 UsageSpecification{
                  attribute:machineState
                  value:failure}
                }
           }
          }
          AllOf{
                entityMatch{
                 entity:machineController
                }
          }
        }
  }
}
```

Listing 1. Textual representation of technican policy

This modelling approach allows us to see UsageSpecifications as credentials in the system. For instance, the attribute describing a role can also symbolise being able to be authenticated as a user with such a role. Therefore, we can also annotate for each architectural element which set of attributes they provide. These attribute providers can be used to model central authorisation components or data storage where credentials are stored, such as in [48]. However, this could also apply to attributes, which are considered for the access control decision, for instance, in our running example, the AssemblyContext of the machine could be annotated with a attribute provider containing the context attribute for a broken machine.

B. Modelling Attacker Capabilities

To analyse attack paths, we model the attacker's capabilities and the vulnerabilities of the system. This can then be used to determine the potential weak spots after a new vulnerability is identified or to find attack paths to old vulnerable legacy elements such as essential components without security support. Our modelling approach is based on the commonly known vulnerability classifications *Common Weakness Enumeration* (CWE) [49], *Common Vulnerabilities and Exposures* (CVE) [50], and *Common Vulnerability Scoring System* (CVSS) [51]. These classifications are commonly used to describe and rate vulnerabilities and are for many systems publicly and freely available in databases such as the NVD [44]. Additionally, there exists automatic security analysis, that can provide these properties. This enables architects to determine the necessary properties and reuse existing knowledge.

CWEs provide an abstract description of a weakness. For instance, CWE-312 [52] describes that sensitive information is stored in cleartext. Additionally, each CWE can have children or parents CWEs. In contrast, CVEs describe concrete vulnerabilities, for instance, CVE-2021-28374 [53] for the authlib package in Debian. However, CVEs can be grouped by CWEs, such as in our case, the CVE-2021-28374 belongs to the CWE-312 group [53]. We use this mechanism, on the one hand, to describe different vulnerabilities types. Here, architects can specify either a more general vulnerability with CWEs or very precise vulnerabilities with CVEs. On the other hand, we use this also describe the capabilities of the attacker. Attackers can have the ability to exploit concrete vulnerabilities such as CVE-2021-28374 or they could have the ability to exploit a group of vulnerabilities such as with CWE-312. For instance, the capability CWE-312 can also exploit CVE-2021-28374. Our metamodel in Figure 3 also illustrates this with the different attack types (CWEAttack, CVEAttack) and vulnerability types (CVEVulnerability, CWEVulnerability). Each of these stores an id of its type and in the case of the CVEs, also for the corresponding CWE type. These are used in the analysis to match attacks with vulnerabilities. Additional to the capabilities, the attackers can also have credentials (here UsageSpecification), which they can use for gaining access to protected assets (see Figure 2). With these properties, Attackers can compromise different architectural elements (see Figure 3 dashed lines). Furthermore, an attacker can gather data during the propagation, which is stored as a list of CompromisedData. In Figure 3, the grey elements are preexisting Palladio elements, and the white elements are the newly added elements.

Additional to the type of vulnerability, we also modelled properties that describe in more detail when a vulnerability can be exploited and its impact. This is used later in the analysis for a more fine-grained decision about the propagation. However, identifying properties and estimating the impact of vulnerabilities can be challenging. Therefore, we partially reuse properties (marked with *) from the *Base Metric Group* of the CVSS calculation [54]. These values from the CVSS are also publicly available in attack databases and, therefore, easy to gather. Also, this type of information can be extracted from other scoring systems that use similar values, such as the CWSS [55]. Additionally, these values are commonly used to rate and describe vulnerabilities. Therefore, it is beneficial to reuse them for our vulnerability modelling since an architect can use them without needing detailed knowledge about the vulnerability.
a) AttackVector*: This describes whether the attacks needs to be local or not. For instance, if there is a local vulnerability on the storage server in our running example, an attacker would need to be already in the deployed components on this server. However, in case of a network vulnerability, the attacker could attack the hardware resource from any element in the system.

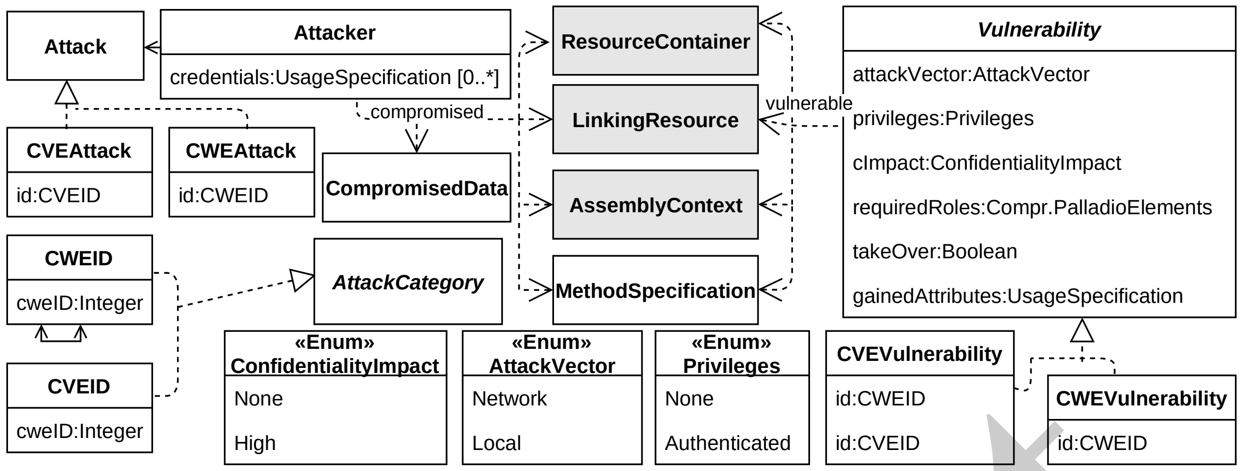

Fig. 3. Simplified metamodel for attackers and system vulnerabilities

b) `Privileges*`: This describes whether certain credentials are necessary to exploit the attack: `None` would require no credentials, `Authenticated` would require the ones necessary to access the element c) `ConfidentialityImpact*`: This describes the impact on confidentiality for exploiting this element: `None` would leak no additional data, and `High` would indicate that data is leaked. In the CVSS scoring, there is also an intermediary value available, which we combined with high, since we cannot differentiate it with Palladio. d) `RequiredRoles`: A list of required architectural elements, which need to be compromised for the attack. This is useful, for instance, if a client application can only be compromised by a specific server instance. e) `TakeOver`: A boolean flag, which indicates that vulnerability enables an attacker to compromise the attacked element. For instance, applied to a component, this would give the attacker full control over the component. Information about this can be found usually in the description of an exploit. f) `GainedAttributes`: Provides information about whether exploiting this vulnerability can give new credentials. This can be used similar to the attribute providers. However, it provides an additional means to model gained credentials based on certain attacks.

Besides the credential part (part of b, f), these elements are also independent of the modelled system and can be reused for other systems.

VII. Attack Propagation Analysis

The attack propagation analysis uses the access control modelling and the vulnerability modelling to calculate the propagation of an attacker. It will return a list of compromised architectural elements and a list of potentially compromised data. We first describe the propagation in general and afterwards describe important parts of the analysis process in more detail.

A. Attack Propagation

The first step of the attack propagation is to transform the access control policy model to a valid XACML file. This file is loaded into a PDP, which is later queried for access decisions. In our analysis, we used the ATT PDP implementation [56] since it is publicly available. However, any other PDP with support for XACML 3.0 should be possible.

After initialising the PDP and declaring a start point, the analysis checks iteratively every connected vulnerable architectural element, whether it is attackable or not. In every step, we first check for new credentials from the last step by using the attribute providers. Then, we calculate for each compromised architectural element the connected architectural elements. We then analyse whether we have already the necessary credentials to access them. Here, we create a request with the available attributes to the PDP. The PDP then determines, based on the loaded policies, the access decision. If access is permitted, we add them and their data to the list of compromised elements. We then analyse whether the rest of the connected elements have vulnerabilities that our attacker model can exploit.

If the attacker can exploit the vulnerability, depending on the impact description and whether the vulnerability allows a hostile takeover, we add the data of the element to the compromised data and the architectural element to the compromised architectural elements. It can be that an attacker can exploit multiple vulnerabilities for one element, but they might have different effects. In our analysis, we chose to let the attacker always exploit the vulnerabilities with the highest confidentiality impact. This process is then repeated until no further credentials are added or no new architectural element is compromised.

B. Data Extraction

For the analysis, we assume that we get complete control over the data handled by a compromised architectural element because the underlying architectural data model does not allow us to differentiate. This assumption might not always hold in reality where the data is, for instance, encrypted.

In future, we might extend the analysis here with a more fine-grained data model. As the data model, we chose the typical Palladio data model, which is parameters and return values of services. However, this also restricts data only to services and indirectly to components and hardware resources since components provide services and components are deployed on hardware resources. Therefore, components have only the data provided by their services, and hardware resources have only the data of their deployed components. Linking resources haves no data at all. While in reality, network devices or hardware devices might store additional data, in general, most of the critical data should at least be stored within dedicated components or managed by a service. The data of a service is calculated by the parameters with which it is called and the return values of external calls the service calls. In case only a service is compromised, this is calculated only for the service. In case of a component, it is calculated for all services of the component and in case of a compromised hardware resource, it is calculated for each deployed component on the hardware resource.

C. Analysing the Vulnerability

Attackers can only propagate through the system if they have the correct abilities to exploit the system's vulnerabilities or authorization. For the vulnerability part, we check this by comparing the `CWEID/CVEID` of the `Vulnerability` to the ids of the `Attack`. Here the analysis considers also the hierarchical dependencies of CWEs with parent and child elements. Also, a `CVEVulnerability` can be either exploited by a matching `CVEAttack` or a matching `CWEAttack`. This type of modelling enables to analyse groups of attacks or reproduce a specific attack, where the attacker's concrete capabilities are known. Additionally, if the vulnerability needs authorisation, the analysis checks whether the attacker has already gained the necessary authorisation by querying the PDP. Besides the authorisation and the id (CWE or CVE), the analysis also compares the `AttackVector`. Here the analysis considers the start point of the propagation step and whether the element is connected locally, such as a component deployed on a hardware resource or by a network such as different hardware resources or components. Additionally, we assume that by taking control over a hardware resource, the attacker can control the components, which are allocated on it and also access the managed data.

Now applying these concepts to our running example, we could model that the Terminal hardware resource is a Ubuntu server with the vulnerability CVE-2021-28374 [53]. As stated in the attack description, this attack allows an attacker to gain in certain circumstances the credentials for this component by exploiting the vulnerability in the courier-authlib package. Additionally, we assume that the storage server and the terminal server share the same credentials for the admin account in our running example. The other components and resources have no vulnerability. We could now add to the vulnerability the option that the gained credentials are the credentials for the Storage Server. As an example, we choose to give the attacker the capability to exploit all attacks based on CWE-312 [52] and as a start point, we choose the Terminal component, which means that the component is under full control of the attacker. The attacker has so far no credentials. The analysis then checks whether it can compromise any connected architectural element (StorageServer, MachineController, TerminalServer, Machine, LocalNetwork) based on its credentials. Since we have not gotten any credentials so far, it cannot compromise any elements this way. Afterwards, it checks all connected elements. Only the TerminalServer is vulnerable to our capabilities since CVE-2021-28374 belongs to CWE-312. However, the attack returns only the credentials to it, and it cannot comprise the TerminalServer with this attack. In the next step, the analysis then uses the credentials gained in the previous step to access all the connected elements from the already compromised component. This time it could compromise the StorageServer and the TerminalServer since it has their credentials. The next step then automatically compromises the deployed components of the storage server because of our assumption based on the resource containers. The result are then the compromised components (Terminal, Product-Storage, ProductionDataStorage), the compromised hardware resources (StorageServer, TerminalServer), the services of the components, and, additionally, all the data of the compromised components.

VIII. Evaluation

The evaluation follows the Goal Question Metric [57] approach. We first describe the goals, evaluation questions, and metrics. Afterwards, we describe the study design and case studies, discuss the results, threats to validity, and the assumptions and limitations.

A. Evaluation Goals, Questions and Metrics

The first goal is to evaluate the accuracy of the analysis. Our question is: **Q1:** Can the attacker access propagation analysis accurately identify affected architectural elements? An affected architectural element indicates that an attacker could compromise this element. The metrics for evaluating are precision and recall [58] since they are used in different approaches to describe accuracy [21], [22], [59]. We calculate them by comparing the results of each scenario with a manually created reference output as described in the following section. Each wrongly classified output element is a false positive f_p, and each correct classified element is a true positive t_p. If an element should be classified in the output and it is missing, this is a false negative f_n. The accuracy is then calculated by precision $M1.1 = \frac{t_p}{t_p + f_p}$, and recall $M1.2 = \frac{t_p}{t_p + f_n}$.

The second goal is to evaluate the effort reduction in comparison to manually checking the propagation. The evaluation question is: **Q2:** What is the effort reduction of using the automatic attack propagation compared to manually checking all the architectural elements? We define the effort as the architectural elements an architect has to analyse to identify the propagation. Architects need to analyse each element and identify the next possible propagation step and decide if it is possible. Each element they do not need to consider, because of our analysis, saves effort. Additionally, after a breach the architect must manually check each architectural element for compromising changes such as changed credentials or malware. Each element the architect does not need to analyse saves effort. We measured these effort types by counting the architectural elements (Components, LinkingResources, ResourceContainers) and comparing them. This enables a direct comparison without the influence of the experience of architects. The effort reduction is then calculated by the two metrics $M2.1$ and $M2.2$.

The first metrics is: $M2.1 = \frac{e_a}{e_a + e_c}$, where e_a is the number of affected elements and e_c is the number of connected elements to the affected elements. This ratio describes the effort reduction if the architects have to investigate an additional propagation step. This is the first effort type we described.

Additionally, we calculated $M2.2 = \frac{e_a}{e_n}$, where e_n is the number of all elements. This is the ratio between the affected elements to the overall elements. This calculates the effort reduction for our second effort type, where the architect analyses the compromised elements for changes.

B. Study Design

According to van den Berghe et al. [60], security approaches performed during the design-time often use only illustrative examples as an evaluation. They argue that using a case study might be a better approach since case studies might provide better insights, show the applicability and increase the comparability between different approaches. Therefore, we chose 3 case studies to evaluate the accuracy and effort reduction.

Accuracy: We created for each case study the architectural model, based on the literature sources [16], [17], [61], [62], if there was no model available. Afterwards, we determined for every case study the reference output of affected architectural elements. These included the affected components, linking resources, hardware resources and compromised services. We used for creating the reference output if available the literature source or online resources such as OWASP [42] or the NVD [44]. Afterwards, we added the vulnerabilities and access policies to each case study model and ran our analysis. The result was then compared to the reference output. Based on this comparison, we calculated the precision and recall for each case study or in case of the last case study for each scenario.

Effort Reduction: For the effort reduction, we reused the results of the accuracy evaluation. For e_a we used the size of the list of affected components, linking resources and hardware resources. We identified e_c by calculating the next potential propagation step manually for each compromised element. This propagation step is, of course, not possible in the original analysis, but we assumed here that the attacker would have all the requirements for compromising the next element. If the next elements were not already compromised, we added it to our set of connected elements. Afterwards, we calculated the size of this set. For e_n we calculated for each case study the list of components, linking resources and hardware resources.

C. Case Study Design

Here we describe our three case studies.

Target Breach: Our first case study is based on the data breach of the retail store Target in 2013. Personal data, including credit cards or private addresses from about 70 million customers, were stolen during this incident. We created a simplified case study based on Shu et al. [61] and Plachkinova et al. [16], which both summarise the breach. It is publicly believed that the attackers gained access to the target network by compromising the network of a supplier. There, they could steal the credentials to access the Target business backend, such as services for billing. In our case study, we modelled this by two connected components, one for the business backend and one for the supplier. The latter one contains credentials, which can be used to access the business backend. However, these credentials could not be used to control the business component or gain access to the credit card data. Therefore, some privilege escape action is needed. In our case study, we modelled this by annotating a CWEVulnerability for privilege escape to the business component. We used the CWEVulnerability here since our sources mention only the vulnerability type and no specific vulnerability. In [61] it is also described that the attacker compromised the Point of Sale (POS) devices, which contained the unencrypted credit card data. Besides the POS devices, the attackers also compromised ftp storage servers. We modelled these aspects by adding components representing POS devices and storage devices. These components are not directly linked with the same network as the supplier component since the original network used some network segregation. Additionally, there exist components representing databases. A cited report in [15] states that often weak passwords or default passwords were used. Additionally, the network contained outdated software. In our case study, we used this by annotating to the POS, storage and database components CWEvulnerabilities for default passwords or weak passwords. Each component is allocated at least once and uses its own hardware resource.

Ukrainian Powergrid Breach: Our second case study is based on the cyberattack on the Ukrainian Powergrid at the end of 2015. We have based our case study on the reports of [17], [62]. They provide a detailed attack walkthrough and information about the attacked software architecture. Here, an attacker gained access to an internal network to manage the power grid's circuit breakers (here ics). The attack was archived by first gaining access via phishing to one office computer. Afterwards, they compromised other office computers till they found credentials to the ics network. With these credentials, they could then access the ics network and, in the end, could activate the circuit breakers. We investigated in our case study the propagation part, from the moment the attacker got access to the first office computer till the attackers could activate the circuit breakers. Our architecture is modelled by two separate networks within the PCM. The first one is for the office network and the other one for the ics. As in the original case study, both networks are connected by a component, which acts as a VPN gateway. The office network contains different components. One is the initial compromised component without the credentials to the ics network. Another component contains the credentials to the ics network. We annotated the CVE-2014-1761 [63] for this component, which the attacker had to exploit to get the credentials. This CVE is representative since the used worm (BlackEnergy) used this vulnerability. The ics then contains components, which provide services, which represented the activation of the ics breaker.

TravelPlanner: The previous two case studies are based on real-world scenarios. In these specific cases, nearly the complete system was compromised. In our third case study, we want to focus on the aspects that only parts of the systems are compromised. Therefore, we chose a third confidentiality research case study. The case study is the TravelPlanner [18] case. It was originally developed for analysing confidentiality, and it was already applied to evaluate other confidentiality analyses such as [12], [19]. It describes a simple mobile application to book flights. It consists of four entities: customer, credit card centre, travel agency and airline. We used the model from Kramer et al. [19], who transformed it into a PCM model. We created different scenarios based on every vulnerability-based propagation step. Additionally, we created scenarios for edge cases in our analysis, such as empty attacker models. A complete list including the propagation description can be found in the dataset [47]. We selected the CWE vulnerabilities based on the OWASP Top Ten [42] and their corresponding CWE classes. OWASP describes the most common security issues in real web applications.

D. Results and Discussion

We have investigated 17 scenarios (2 real-world case studies and 15 TravelPlanner scenarios). For each case study and scenario regarding the accuracy (Q1), we have a precision and recall of 1.00. These are perfect results for accuracy. The reasons for the perfect results are that the system in both real-world case studies were highly compromised and in the TravelPlanner case study the scenarios are small. Both reasons simplify the outcome, whereas, in other scenarios the results might differ. However, these scenarios allow identifying functional correctness more easily. Therefore, the results indicate that the implementation is at least functional correct.

The results for Q2 are shown in Table I, where T is the Target case study, P is the Powergrid case study and the TPs are the scenarios from the TravelPlanner. For $M2.1$ higher numbers are better. As expected, the results for $M2.1$ in the Target and Power Grid case study are lower since nearly all the elements were affected. Therefore, the number of unaffected elements is lower and the security expert must look at more elements. Additionally, when we looked at the results in more detail, the unaffected elements were mostly external elements such as outside network elements. In the TravelPlanner scenarios, the results are better since not all elements are affected. This is usually the case when only a smaller part of the system is affected. The value of TP1 cannot be calculated since no architectural element is compromised. Overall for $M2.1$ the TravelPlanner scenarios show our analysis reduces the architectural elements architects needs to check between 44% and 86%. This indicates that for smaller breaches, our analysis could save the architects effort. The result for $M2.2$ shows basically the same behaviour as $M2.1$. Here lower numbers are better. The TravelPlanner scenarios show in all scenarios lower numbers and in most cases much lower numbers. As expected, this is based on the different number of affected elements. This indicates that in certain scenarios, architects can save significant work. For instance, in TP14, they only have to analyse 10% of the architectural elements manually.

Overall our results indicate that the analysis produces accurate result and can save effort.

E. Threats to Validity

We categorize the threats to validity in 4 categories as suggested by Runeson and H öst [64].

Internal Validity describes that only the expected factors influence the results. Our analysis uses different models as input and is evaluated based on these. Therefore, the results highly depend on the selected models and scenarios. Especially since we also created the reference output manually. We lowered here the risk by deriving the reference output based on real-world examples and existing literature. While the overall models are quite small, they contain the important parts of the attack. Adding more architectural elements might increase the number of compromised elements but would not change the outcome regarding the compromised elements. Also, regarding the analysis, the important aspects of it are already covered: Gaining of new credentials, exploiting vulnerabilities based on CWEs/CVEs (authenticated and not authenticated), propagation from different compromised components, identifying potentially affected data. Therefore, we assume the risk to be low. We did not cover in our effort cases the effort for creating the models. While this might be a significant effort, there also exist automatic tools that might help here. Therefore, we did not consider it.

External Validity describes how generalizable and useful for other researchers the results are [64]. While a case study might increase the insights into the problem, it may also not be a representative example of the problem. By using external case studies, we tried to lower the risk. However, the results so far only indicate the functional accuracy of the analyses and not the accuracy of the approach in general. Here especially, the potential overestimation of compromised data and the hardware resource containers is not investigated. This is partly based on the fact that in our real-world case studies nearly all relevant elements are affected and that finding case studies for this is quite challenging. We plan to address this in the future by investigating the approach with further case studies and a more detailed exchange with security experts.

Construct Validity discusses whether the investigated properties are relevant for the intended goal. Here, the properties are the different metrics. For accuracy, we chose precision and recall, which is also used to evaluate the accuracy of the KAMP approach [21], [22], which we used as a structure for our propagation algorithm. Using the same metric lowers this risk. $M2.1$ is simply the ratio between counted model elements. This also applies to $M2.2$. Also, in the original KAMP evaluation [21] $M2.2$ is used for answering a similar

TABLE I
EVALUATION RESULTS FOR EFFORT REDUCTION

	Scenarios																
Metrics	T	P	TP1	TP2	TP3	TP4	TP5	TP6	TP7	TP8	TP 9	TP 10	TP11	TP 12	TP 13	TP14	TP15
$M2.1$	0.29	0.22	-	0.86	0.63	0.78	0.78	0.78	0.67	0.70	0.71	0.67	0.44	0.60	0.80	0.83	0.88
$M2.2$	0.63	0.87	0.00	0.10	0.30	0.20	0.20	0.20	0.30	0.30	0.20	0.30	0.50	0.40	0.20	0.10	0.10

question. We think in conjunction, both metrics can be used to describe the effort reduction.

Reliability describes whether other researchers can reproduce the results later. Using statistical metrics avoids subjective interpretation and therefore increases reproducibility. Additional, we prepared a dataset [47] containing all the models and a description of how to apply them and made the source code of the analysis publicly available. This will allow other researchers to verify the results.

F. Assumptions & Limitations

Some attacks need the interaction of a third party which is not the attacker. This is most often a regular user. So far, we cannot differentiate between these kinds of attacks.

The attack propagation based on the vulnerability only works if the vulnerability and the attacks or at least their category is known. Unknown attacks or vulnerabilities are not considered. Therefore, the analysis cannot be used to identify unknown vulnerabilities. However, it allows identifying unknown attack paths. Also, based on the CWE categories, it is possible to consider some unknown vulnerabilities in a component by assigning a common CWE vulnerability, such as provided by OWASP [42].

We do not consider advanced mitigation strategies of attacks in our approach. While simple mitigation approaches such as access control or network segregation are partially supported, more advanced concepts like Intrusion Detection Systems are not supported. However, our approach could be used to identify locations for advanced mitigation strategies. For instance, an architect can review the attack paths and install mitigation strategies in critical paths. Additionally, the mitigated vulnerabilities could be removed from the model and then the architect could run the analysis to see how the attack paths change. Nevertheless, this solution comes with additional drawbacks, such as an inconsistency between the modelled vulnerabilities and the vulnerabilities in the system. One solution for this limitation might be the combination with approaches such as in [65].

Additional, our assumption about the automatic compromising of components based on the hardware resource container might limit this approach in areas where trusted execution environments [66] or other approaches such as [67] are used to mitigate the impact of compromised hardware or operating systems. While these approaches can circumvent these confidentiality leaks, they are in general not commonly applied and there also exist attacks such as [68], which circumvent these security measures.

The scalability of the approach might be problematic since for every newly gained credential or compromised element all connected elements need to be checked. This is because new credentials provide new authorisation and the compromised element might be used in a dependable vulnerability (see requiredRoles). While this is for smaller attack paths under 1000 steps no problems, early experiments show that the calculation time increases drastically for bigger attacks paths. The calculation time could be potentially decreased by using more efficient data-structures or parallel calculations.

Currently, the data extraction is only considered from the control flow. Aspects like unencrypted data, which flows over network nodes and could easily be seen by compromised network nodes, are so far not considered.

IX. CONCLUSION

In this paper, we presented an approach for an architectural attack propagation analysis. We proposed a metamodel extension for modelling access control policies and system vulnerabilities. We integrated the metamodel into the PCM, allowing system architects to specify access policies and vulnerabilities to different architectural elements, such as services or hardware elements.

This system model then can be analysed by an attack propagation to identify compromised architectural elements. The evaluation shows that our approach works accurately in the used scenarios and indicates that it saves effort by suggesting relevant architectural elements.

Using our approach is beneficial for identifying possible weak spots in the software architecture. This identification might prohibit attacker propagations. Additionally, the approach can be used during an active attack to identify compromised resources and reduce further propagation steps of attackers. The generated XACML file can be used during runtime as the access control policy file. This might reduce errors introduced by manually defining one from the specification. The model can also serve as documentation of the access control or vulnerabilities. This eases the communication between different stakeholders.

In the future, we plan to investigate the automatic creation of vulnerability models by combining existing vulnerability analysis with approaches such as [41]. Regarding the evaluation, we plan to apply our approach to different real-world scenarios and investigate the scalability.

References

[1] J. Greig. “96% of third-party container applications deployed in cloud infrastructure contain known vulnerabilities: Unit 42.” (Oct. 5, 2021), [Online]. Available: https : / / www . zdnet . com / article / 96 - of - third - party - container - applications - deployed - in - cloud - infrastructure- contain- known- vulnerabilities- unit- 42/ (visited on 09/29/2021).

[2] HP. “Hp study reveals 70 percent of internet of things devices vulnerable to attack.” (Jul. 29, 2014), [Online]. Available: https://www.hp.com/us- en/hp- news/press-release.html%3Fid=1744676 (visited on 10/05/2021).

[3] I. Mori, “Cyber security breaches survey 2021: Statistical release,” p. 66,

[4] E. Cole, *Advanced persistent threat: understanding the danger and how to protect your organization*. Newnes, 2012.

[5] V. Hu *et al.*, “Attribute-Based Access Control,” *Computer*, vol. 48, no. 2, pp. 85–88, Feb. 2015.

[6] D. Verma, E. Bertino, G. de Mel, and J. Melrose, “On the impact of generative policies on security metrics,” in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, Jun. 2019, pp. 104–109.

[7] E. Bertino, A. A. Jabal, S. Calo, D. Verma, and C. Williams, “The challenge of access control policies quality,” *Journal of Data and Information Quality*, vol. 10, no. 2, pp. 1–6, Sep. 7, 2018.

[8] [Online]. Available: https : / / bloodhoundenterprise . io/ (visited on 10/05/2021).

[9] [Online]. Available: https://www.darpa.mil/program/cyber-grand-challenge/ (visited on 10/05/2021).

[10] M. U. Aksu, K. Bicakci, M. H. Dilek, A. M. Ozbayoglu, and E. I. Tatli, “Automated generation of attack graphs using nvd,” in *ODASPY ’18*, Tempe, AZ, USA: Association for Computing Machinery, 2018, pp. 135–142.

[11] B. Yuan, Z. Pan, F. Shi, and Z. Li, “An attack path generation methods based on graph database,” in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, vol. 1, 2020, pp. 1905–1910.

[12] S. Seifermann, R. Heinrich, and R. Reussner, “Data-driven software architecture for analyzing confidentiality,” in *ICSA’19*, Hamburg, Germany: IEEE, pp. 1–10.

[13] D. Basin, M. Clavel, J. Doser, and M. Egea, “Automated analysis of security-design models,” *Information and Software Technology*, vol. 51, no. 5, pp. 815–831, May 1, 2009.

[14] R. Reussner, S. Becker, J. Happe, *et al.*, *Modeling and Simulating Software Architectures – The Palladio Approach*. Cambridge, MA: MIT Press, Oct. 2016, 408 pp.

[15] B. Krebs. “Inside target corp., days after 2013 breach.” (Sep. 21, 2015), [Online]. Available: https : //krebsonsecurity.com/2015/09/inside-target-corp-days-after-2013-breach/.

[16] M. Plachkinova and C. Maurer, “Security breach at target,” *Journal of Information Systems Education*, vol. 29, no. 1, pp. 11–20, 2018.

[17] B. A. Hamilton, “Industrial cybersecurity threat briefing,” Tech. Rep., p. 82.

[18] K. Katkalov, “Ein modellgetriebener ansatz zur entwicklung informationsflusssicherer systeme,” doctoralthesis, Universität Augsburg, 2017.

[19] M. Kramer, M. Hecker, S. Greiner, K. Bao, and K. Yurchenko, “Model-driven specification and analysis of confidentiality in component-based systems,” KIT-Department of Informatics, Tech. Rep. 12, 2017.

[20] “XACML,” [Online]. Available: https://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (visited on 10/25/2021).

[21] K. Rostami, R. Heinrich, A. Busch, and R. Reussner, “Architecture-based change impact analysis in information systems and business processes,” in *ICSA’17*, pp. 179–188.

[22] R. Heinrich, S. Koch, S. Cha, K. Busch, R. Reussner, and B. Vogel-Heuser, “Architecture-based change impact analysis in cross-disciplinary automated production systems,” *JSS 146*, vol. 146, pp. 167–185, 2018.

[23] R. Al-Ali, R. Heinrich, P. Hnetynka, A. Juan-Verdejo, S. Seifermann, and M. Walter, “Modeling of dynamic trust contracts for industry 4.0 systems,” in *ECSA ’18*, ACM.

[24] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, “Verification and change-impact analysis of access-control policies,” 2005, p. 196.

[25] F. Alberti, A. Armando, and S. Ranise, “Efficient symbolic automated analysis of administrative attribute-based rbac-policies,” in *ASIACCS ’11*, 2011, p. 165.

[26] D. Ferraiolo, J. Cugini, and D. R. Kuhn, “Role-based access control (RBAC): Features and motivations,” in *ACSAC’95*, 1995, pp. 241–248.

[27] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone, “Analysis of xacml policies with smt,” in *Principles of Security and Trust*, Springer, 2015, pp. 115–134.

[28] J. Jürjens, “Umlsec: Extending uml for secure systems development,” in *UML 2002*. Springer Berlin Heidelberg, 2002, vol. 2460, pp. 412–425.

[29] T. Lodderstedt, D. Basin, and J. Doser, “Secureuml: A uml-based modeling language for model-driven security,” vol. 24, Springer, Berlin, Heidelberg, 2002, pp. 426–441.

[30] K. Tuma, R. Scandariato, and M. Balliu, “Flaws in flows: Unveiling design flaws via information flow analysis,” in *ICSA’19*, pp. 191–200.

[31] C. Gerking and D. Schubert, “Component-Based Refinement and Verification of Information-Flow Security Policies for Cyber-Physical Microservice Architectures,” in *ICSA’19*, IEEE, Mar. 2019, pp. 61–70.

[32] K. Katkalov, K. Stenzel, M. Borek, and W. Reif, “Model-driven development of information flow-secure systems with iflow,” in *SOCIALCOM’13*, pp. 51–56.

[33] N. Boltz, M. Walter, and R. Heinrich, "Context-based confidentiality analysis for industrial iot," in *SEAA'20*, IEEE.

[34] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer Science Review*, vol. 13–14, pp. 1–38, Nov. 2014.

[35] B. Schneier, "Attack trees," *Dr. Dobb's journal*, vol. 24, no. 12, pp. 21–29, 1999.

[36] T. Sommestad, M. Ekstedt, and H. Holm, "The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures," *IEEE SYSTEMS JOURNAL*, vol. 7, no. 3, p. 11, 2013.

[37] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, and H. Mouratidis, "From product recommendation to cyber-attack prediction: Generating attack graphs and predicting future attacks," *Evolving Systems*, vol. 11, no. 3, pp. 479–490, Sep. 2020.

[38] N. Polatidis, M. Pavlidis, and H. Mouratidis, "Cyber-attack path discovery in a dynamic supply chain maritime risk management system," *Computer Standards & Interfaces*, vol. 56, pp. 74–82, 2018.

[39] C. Deloglos, C. Elks, and A. Tantawy, "An attacker modeling framework for the assessment of cyber-physical systems security," in *Computer Safety, Reliability, and Security*, Cham: Springer International Publishing, 2020, pp. 150–163.

[40] S. Becker, M. Hauck, M. Trifu, K. Krogmann, and J. Kofroň, "Reverse engineering component models for quality predictions," in *CSMR '10*, pp. 194–197.

[41] Y. R. Kirschner, "Model-driven reverse engineering of technology-induced architecture for quality prediction (short paper)," in *ECSA 2021 Companion Volume, Virtual (originally: Växjö, Sweden), 13-17 September, 2021*, ser. CEUR Workshop Proceedings, vol. 2978, CEUR-WS.org, 2021.

[42] [Online]. Available: https://owasp.org/www-project-top-ten/ (visited on 10/25/2021).

[43] [Online]. Available: https : / / snyk . io/ (visited on 11/02/2021).

[44] "NVD," [Online]. Available: https://nvd.nist.gov/vuln (visited on 10/25/2021).

[45] [Online]. Available: https://heartbleed.com/ (visited on 11/06/2021).

[46] [Online]. Available: https : / / www . oasis - open . org / committees/tc home.php?wg abbrev=xacml (visited on 10/10/2021).

[47] [Online]. Available: https://publikationen.bibliothek.kit.edu/1000141655 (visited on 11/04/2021).

[48] C. Osborne. "Misconfigured, old airflow instances leak slack, aws credentials." (Oct. 5, 2021), [Online]. Available: https : / / www. zdnet . com / article / misconfigured - airflow-instances-leak-slack-aws-credentials/ (visited on 10/24/2021).

[49] "Cwe," [Online]. Available: https : / / cwe . mitre . org/ (visited on 10/25/2021).

[50] "Cve," [Online]. Available: https://cve.mitre.org/ (visited on 10/25/2021).

[51] "Cvss sig," [Online]. Available: https://www.first.org/cvss/ (visited on 10/25/2021).

[52] "Cwe-312," [Online]. Available: https://cwe.mitre.org/data/definitions/312.html (visited on 10/25/2021).

[53] "Cve-2021-28374," [Online]. Available: https : / / nvd . nist . gov / vuln / detail / CVE - 2021 - 28374 (visited on 10/25/2021).

[54] "Cvss 3.1," [Online]. Available: https://www.first.org/cvss / v3 - 1 / cvss - v31 - specification r1 . pdf (visited on 10/25/2021).

[55] "Cwss," [Online]. Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html (visited on 10/25/2021).

[56] [Online]. Available: https://github.com/att/xacml- 3.0 (visited on 10/25/2021).

[57] G. Basili, V. R. Caldiera, and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.

[58] C. Van Rijsbergen and C. Van Rijsbergen, *Information Retrieval*. Butterworths, 1979.

[59] S. Seifermann, R. Heinrich, D. Werle, and R. Reussner, "Detecting violations of access control and information flow policies in data flow diagrams," *The Journal of Systems and Software*, 2021, accepted, to appear.

[60] A. van Den Berghe, R. Scandariato, K. Yskout, and W. Joosen, "Design notations for secure software: A systematic literature review," *Softw. Syst. Model.*, vol. 16, no. 3, pp. 809–831, 2017.

[61] X. Shu, K. Tian, A. Ciambrone, and D. Yao, "Breaking the target: An analysis of target data breach and lessons learned," *arXiv:1701.04940 [cs]*, Jan. 17, 2017.

[62] E. I. Sharing and A. C. (E-ISAC), "Analysis of the cyber attack on the ukrainian power grid, defense use case," Tech. Rep., 2016, pp. 1–29.

[63] "Cve-2014-1761," [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2014-1761 (visited on 10/25/2021).

[64] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, Dec. 19, 2008.

[65] E. Taspolatoglu and R. Heinrich, "Context-based architectural security analysis," in *2016 13th Working IEEEIFIP Conference on Software Architecture (WICSA)*, Venice, Italy: IEEE, Apr. 2016, pp. 281–282.

[66] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, pp. 57–64.

[67] J. Criswell, N. Dautenhahn, and V. Adve, "Virtual ghost: Protecting applications from hostile operating systems," ser. ASPLOS '14, Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, pp. 81–96.

[68] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, *SGAxe: How SGX fails in practice*, https://sgaxeattack.com/, 2020.