Architectural Generation of Context-based Attack Paths

Master's Thesis of

Jonathan Schenkenberger

at the Department of Informatics
Institute of Information Security and Dependability (KASTEL)

Reviewer:	Prof. Dr. Ralf H. Reussner
Second reviewer:	Prof. Dr.-Ing. Anne Koziolek
Advisor:	M.Sc. Maximilian Walter
Second advisor:	M.Sc. Tobias Walter

25. October 2021 – 25. April 2022

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Abstract

In industrial processes (Industry 4.0) and other fields in our lives like the energy or health sector, the confidentiality of data becomes increasingly important. In order to protect confidential information on critical systems, it is crucial to be able to determine whether compromisation of a critical system is possible. Therefore, relevant attack paths in different access-control contexts need to be found in order to compare different architectures regarding this security aspect. In order to minimize costs, it is important to already consider potential attack paths in the design phase of the software architecture. There are already approaches considering the topic of attack path generation. However, often they do not consider software architecture modeling, which renders the analysis for the purpose of component-based software modeling more difficult. Moreover, other approaches do not consider both vulnerabilities and access control mechanisms. For that purpose, this thesis presents an approach for finding all potential attack paths in a software architecture model considering access control and vulnerabilities. This helps software architects and security experts to find relevant and critical attack paths to a critical element more easily. However, all attack paths are often to many, so the approach presented here introduces and utilizes meaningful filter criteria based on wide-spread vulnerability classification standards. The purpose of these filters is to enable the software architect to restrict the resulting attack paths on only relevant ones. The evaluation for the thesis indicated that the model and the implemented approach can be mostly used in small scenarios derived from real-world case studies. Furthermore, it also indicated an effort reduction from 35% up to 80% for the software architect. However, a larger scalability of the approach could not be shown due to an exponential runtime behavior of the implemented analysis. However, mitigating the scalability issue is one of the reasons for the usage of filter criteria.

Zusammenfassung

In industriellen Prozessen (Industrie 4.0) und anderen Bereichen unseres Lebens wie dem Energie- oder Gesundheitssektor wird die Vertraulichkeit von Daten zunehmend wichtig. Um vertrauliche Informationen auf kritischen Systemen zu schützen, ist es wichtig zu bestimmen ob die Kompromittierung dieser kritischen Systeme möglich ist. Deshalb müssen relevante Angriffspfade in verschiedenen Zugriffskontrollkontexten gefunden werden, um verschiedene Softwarearchitekturen bezüglich dieses Sicherheitsaspekts zu vergleichen. Um Kosten zu sparen, ist es wichtig potentielle Angriffspfade bereits in der Entwurfsphase der Softwarearchitektur zu betrachten. Es gibt bereits Ansätze, die das Thema der Angriffspfadgenerierung adressieren. Allerdings betrachten sie es oft nicht auf einer Softwarearchitekturmodellierungsebene, was die Analyse für den Zweck der komponentenbasierten Softwaremodellierung erschwert. Des Weiteren, betrachten andere Ansätze oft nicht sowohl Verwundbarkeiten als auch Zugriffskontrollmechanismen. Deshalb stellt diese Arbeit einen Ansatz vor, um alle potentiellen Angriffspfade in einem Softwarearchitekturmodell bezüglich Verwundbarkeiten und Zugriffskontrolle zu finden. Das hilft Softwarearchitekten und Sicherheitsexperten relevante und kritische Angriffspfade zu einem kritischen Element leichter zu finden. Jedoch sind alle Angriffspfade oft zu viele, sodass der hier präsentierte Ansatz sinnvolle Filterkriterien einführt und verwendet, welche auf verbreiteten Verwundbarkeitsklassifikationsstandarts beruhen. Der Grund für diese Filter ist es, dem Softwarearchitekt zu ermöglichen, die resultierenden Angriffspfade auf die relevanten zu begrenzen. Die Evaluation der Arbeit deutete an, dass das verwendete Modell und der implementierte Ansatz in kleinen Szenarien, die aus Fallstudien aus der echten Welt extrahiert wurden, meistens angewendet werden kann. Außerdem deutete die Evaluation ebenfalls eine Aufwandsreduktion von 35% bis zu 80% für den Softwarearchitekt an. Allerdings konnte keine größere Skalierbarkeit des Ansatzes gezeigt werden, da ein exponentielles Laufzeitverhalten festgestellt wurde. Allerdings ist das Abmildern des Skalierbarkeitsproblem einer der Hauptgründe für das Verwenden der Filterkriterien.

Contents

List of Figures

List of Tables

1 Introduction

As digitization progresses further in several areas in our lives like industrial processes (Industry 4.0), the energy or health sector, the confidentiality of data-exchange becomes increasingly important.

1.1 Motivation

The reason for this is that malicious attackers may steal sensitive, confidential or even secret data stored on systems. For that purpose, it is critical to be able to determine relevant attack paths in different contexts in order to compare different architectures regarding this security aspect. In order to minimize costs, it is important to already consider this analysis in the design phase. During the attack path search, the implemented approach considers the interplay of vulnerabilities and access control. An attack path is the path an attacker utilizes to reach a certain target point from an entry point in the system. The overall domain of this thesis is confidentiality in component-model-based software architectures. The benefit of generating attack paths is enabling the software architect to compare different architectures with each other regarding the aspect of potentially threatened critical architectural elements.

State-of-the-art access control strategies like *Attribute-based access control* (ABAC) [25] are often utilized in order to increase the context-based confidentiality. However, ABAC alone hamper attack propagation analysis due to complex access control policies, rendering the estimation of confidentiality properties more difficult [7]. Especially, finding the propagation of an attacker through the system is a rather complicated task. However, a very important one because a stronger density of connections often means more possible vulnerabilities and attack vectors. Therefore, it is crucial to consider the question of attack paths to a critical element. However, a manual search for these paths is often time-consuming, difficult and error-prone on complex architectural models. There are already approaches considering the topic of attack path generation described in Chapter 3. However, they do not consider a specific critical element in a software architecture model or the interplay of vulnerabilities in the system and access control mechanisms. Hence, the goal of this thesis is to design and implement an automatic attack path generation considering access control and vulnerabilities. Since the approach utilizes software architecture models, the analysis helps software architects and security experts to find relevant and critical attack paths to a critical element more easily.

1.2 Approach and Running Example

This thesis presents an approach for an attack path search to a critical software architectural element taking account of the interplay of vulnerabilities and access control. Moreover, the approach enables the software architect to utilize different filter criteria like complexity of the attack or different attack vectors in order to restrict the resulting attack paths to the ones that are more crucial for the purpose of analyzing confidentiality.

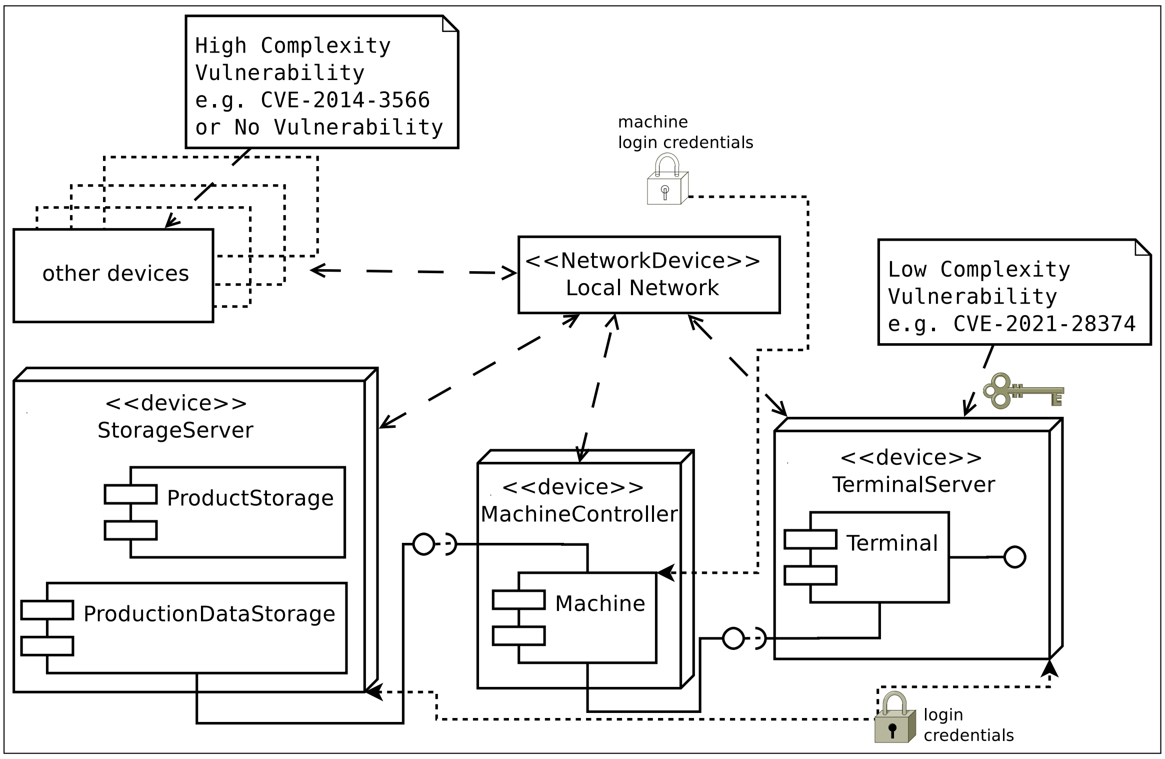

High Complexity Vulnerability e.g. CVE-2014-3566 or No Vulnerability
machine login credentials
other devices
<<NetworkDevice>> Local Network
Low Complexity Vulnerability e.g. CVE-2021-28374
<<device>> StorageServer
ProductStorage
ProductionDataStorage
<<device>> MachineController
Machine
<<device>> TerminalServer
Terminal
login credentials

Figure 1.1: Component and Deployment View of the Running Example.

Figure 1.1 depicts the component and deployment view of the running example inspired by a scenario from Al-Ali et al. [3] and adapted by Walter et al. [55]. The scenario takes place in an environment with two companies (A and B) in an Industry 4.0 setting. The external maintenance service organization B repairs machines of manufacturing company A. The access control to that machine is secured with login credentials for the machine, depicted by the white lock. Furthermore, the system contains a `Terminal` as an user interface, deployed on a `TerminalServer`. Additionally, there are the `ProductionDataStorage` for storing the data of the machine and the `ProductStorage` for storing drafts. These components are deployed on the `StorageServer`. All the beforementioned devices and other devices inside company A are connected to the local network. Moreover, the `TerminalServer` is vulnerable to a low complexity vulnerability enabling an attacker to obtain the login credentials, depicted by the gold-colored key. In this example, these credentials are needed for the admin account of the terminal and the `StorageServer` as depicted by the gold-colored lock. Some of the other devices have only high complexity vulnerabilities or no vulnerabilities at all. Starting from the terminal, an attack path search with a filter for only low complexity attacks finds a path consisting of the terminal server and the storage server and the respective contained components as resulting compromised elements.

1.3 Contribution

The contribution of this thesis consists of two parts. The first one is the extension and adaption of an attacker metamodel in order to enable the analysis approach to a critical element. The attacker can also be customized with various filter criteria.

The second contribution of the thesis consists of the design and implementation of an approach for an automatic attack path search using the aforementioned filter criteria. These are there in order to enable software architects and security experts to focus upon relevant and critical attack paths. Moreover, the automatic search of these paths can diminish the effort in comparison to a manual examination on more complex architectural models.

1.4 Structure

This thesis is structured as follows. Chapter 2 describes the foundational matters for the thesis. It is followed by Chapter 3 that explains state-of-the-art approaches that are not directly used in the approach of the thesis. Thereafter, Chapter 4 gives an overview about the approach. The next two Chapters 5 and 6 describe respectively the metamodel adaptions and the actual analysis approach in more detail. Moreover, there is also Chapter 7 describing an overview over the implementation of the prototype tool for the approach presented in this thesis. Finally, the Chapters 8 and 9 describe respectively the evaluation and the conclusion and future work of this thesis. The evaluation considered 3 case studies and a research example in order to evaluate accuracy, effort reduction and scalability of the approach.

2 Foundation

This chapter presents the foundation of the approach presented in this thesis. Section 2.1 presents the classification schema for vulnerabilities used in this thesis. Thereafter, the Palladio Component Model (PCM) [5] is presented in Section 2.2 because it is the foundation for the metamodel on which the automatic search for attack paths will be based. In Section 2.3 attribute based access control (ABAC) is described. The last Section 2.4 explains the attacker metamodel that is the foundation of the analysis designed and implemented in this thesis.

2.1 Classification of Vulnerabilities

In order to be able to classify vulnerabilities, this section explains the *Common Weakness Enumeration* (CWE) [16] and the *Common Vulnerabilities and Exposures* (CVE) [14] classes. CVE vulnerabilities represent concrete attacks like a vulnerability in the *courier-authlib library* (*CVE-2021-28374*) depicted in the running example. In contrast, CWE vulnerabilities can also be used represent whole attack categories like the usage of weak passwords. All vulnerabilities have an unique ID in order to render a clear identification possible. Additionally, CWE IDs can also be hierarchical giving the CWE vulnerabilities the capability of constituting a categorization of attacks. The CVSS is used in order to assign a scoring from 0.0 to 10.0 to a vulnerability. The scoring assumes the attacker to have already discovered the vulnerability and represents the criticality of the exploit. There are databases that are used for storing information about vulnerabilities and how they are classified. One such database is the widely utilized *National Vulnerability Database* (NVD) [36] containing classifications for many known vulnerabilities.

Moreover, the classification information from the *Common Vulnerability Scoring System* (CVSS) [15] is also used in order to be able to rate vulnerabilities. Inside the CVSS specification [11], there are two categories of classifications of vulnerabilities that are especially interesting for the approach presented in this thesis. The first one is the category *Exploitability* with values for *Attack Vector*, *Attack Complexity*, *Privileges Required* and *User Interaction.* The second category is the *Impact* category with values for *Confidentiality Impact*, *Integrity Impact* and *Availability Impact.*

Table 2.1 depicts the values for all the beforementioned classification types. For the *Attack Vector* exploitability rating, the lowest value is when the attacker is somewhere in the network (N). The second lowest rating, the attacker needs to be in the same network (A). The *Local* (L) rating requires the attacker to be accessing the target system locally or remotely (e.g. via ssh). The maximum value is P where the attacker needs to be physically manipulating the vulnerable component. The attack complexity can be low for an attack that does not require special access conditions, or high if specialized preparations

Classification	Values from Minimum to Maximum
Attack Vector	Network (N), Adjacent Network (A), Local (L), Physical (P)
Attack Complexity	Low, High
Privileges Required	None, Low, High
User Interaction	None, Required
Confidentiality Impact	None, Low, High
Integrity Impact	None, Low, High
Availability Impact	None, Low, High

Table 2.1: Overview of the Categories in the CVSS [11].

are necessary such as for a man-in-the-middle attack. The required privileges rating can be using no privileges to using default user privileges (Low) upto requiring high administrative privileges. The last exploitability rating is about whether user interaction is required or not. The impact values range from none to high equating to no impact up to a maximum impact, loosing all confidentiality or integrity or availability of the affected component. For instance, according to the NVD [36] the vulnerability *CVE-2021-28374* depicted in the running example has a *Low* attack complexity, an attack vector value *Network* and it requires no privileges and no user interaction leading to values *Low* for these two exploitability metrics. The impact values consist of a *High* confidentiality impact value but no integrity and availabilty impact, rendering these values to the value of *None*.

2.2 Palladio Component Model

This section briefly describes the Palladio Component Model (PCM) [40]. The PCM is part of the Palladio simulator for software architecture. It is able to predict several quality properties of software like performance and reliability. Initially, it has been developed by Karlsruhe Institute of Technology (KIT), FZI Research Center for Information Technology, and the University of Paderborn.

The Palladio Component Model (PCM) is a detailed EMF-based (Eclipse Modeling Framework based) metamodel for examining component-based software [4]. The model consists of components, interfaces and connectors between those and can be used for describing software and simulating performance, reliability and confidentiality aspects of the system [40, 45].

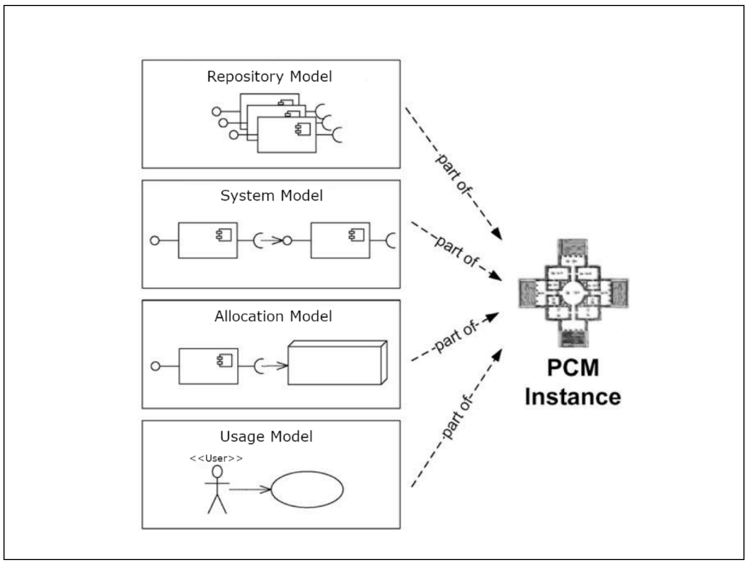


Figure 2.1: Composition of a Palladio Component Model Instance (Inspired by [5]).

Figure 2.1 depicts the four submodels of the PCM. First, the *Repository Model* represents the available components and interfaces. The so-called *Service Effect Specifications* (SEFFs) are there in order to represent inner behavior of components as an abstraction from the actual control flow. Second, the *System Model* describes the combination of components defined in the repository, i.e. the structure of the software architecture system. It defines *Assembly Contexts* representing instantiated repository components in the system. Third, the *Allocation Model* represents the allocation of assembly contexts to different *Resource Containers*. These represent the devices on which assembly contexts can be allocated. The contexts for this allocation are called *Allocation Context*s. Moreover, the allocation model also defines a *Resource Environment* describing the connection of resource containers to *Linking Resource*s. These represent links in a network, connecting different devices with each other. A resource container may be connected to no, one or several linking resources. Fourth, there is the *Usage Model* in order to represent the users' engagement with the system [40].

2.3 ABAC - Attribute Based Access Control

This section describes the foundation of attribute based access control (ABAC). It uses attributes in the form of name-value pairs and gives these attributes to different objects and subjects. These attributes are then used in access control policies to determine whether access should be granted [26].

A wide-spread implementation of the ABAC concept is the *eXtensible Access Control Markup Language* (XACML). The basic structures of the XACML policy language model for modeling access control are depicted in Figure 2.2. A `PolicySet` consists of zero or more `Policy` instances. Each policy contains one or more `Rules` combined using a `Rule Combining Algorithm` whereas policies are combined with a Policy Combining Algorithm. Matches are included inside `AllOfs` that are themselves contained inside `AnyOfs`. An `AllOf` behaves like a logical *AND* and an `AnyOf` behaves like a logical *OR*. The `Target` consists of an arbitrary amount of `AnyOfs` and decides about whether the rule can be applied. If it can and the optional `Condition` is met, then the `Effect` takes place. Moreover, there is also the concept of *Obligations* in XACML accomplished by the policy enforcement point (PEP) that is described later in this Section [49].

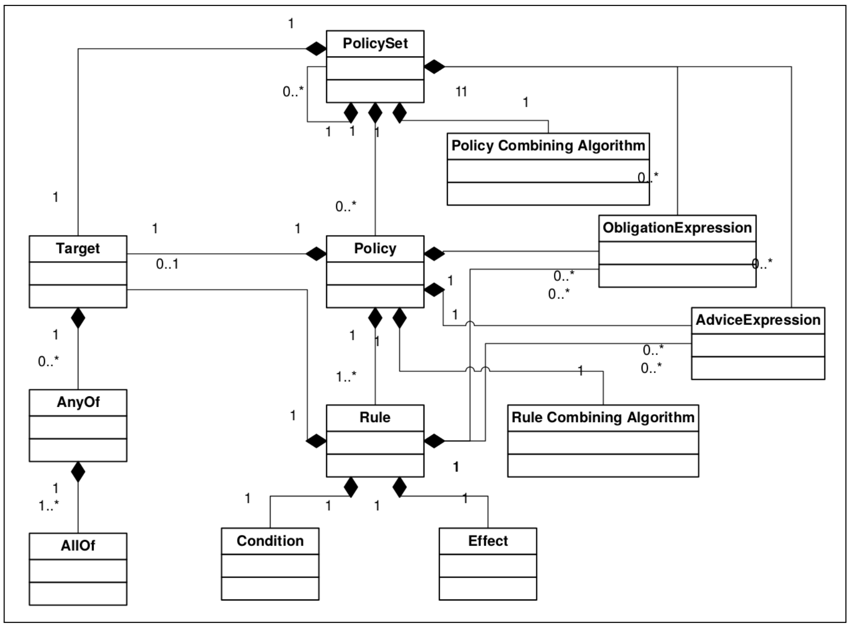


Figure 2.2: XACML Policy Language Model [49].

A request in the XACML framework consists of attributes from the categories *subject*, *resource*, *environment* and *action*. As Figure 2.3 depicts, the *Policy Administration Point* (PAP) controls the *Policy Repository* used by the *Policy Decision Point* (PDP). In order to compute the access control decision, the PDP also utilizes the *Policy Information Point* which exists for loading attributes. Furthermore, there is the *Policy Enforcement Point* (PEP) actually enforcing the access control decision on the subject and object. The PEP uses the PDP in order to decide whether access should be granted [49].

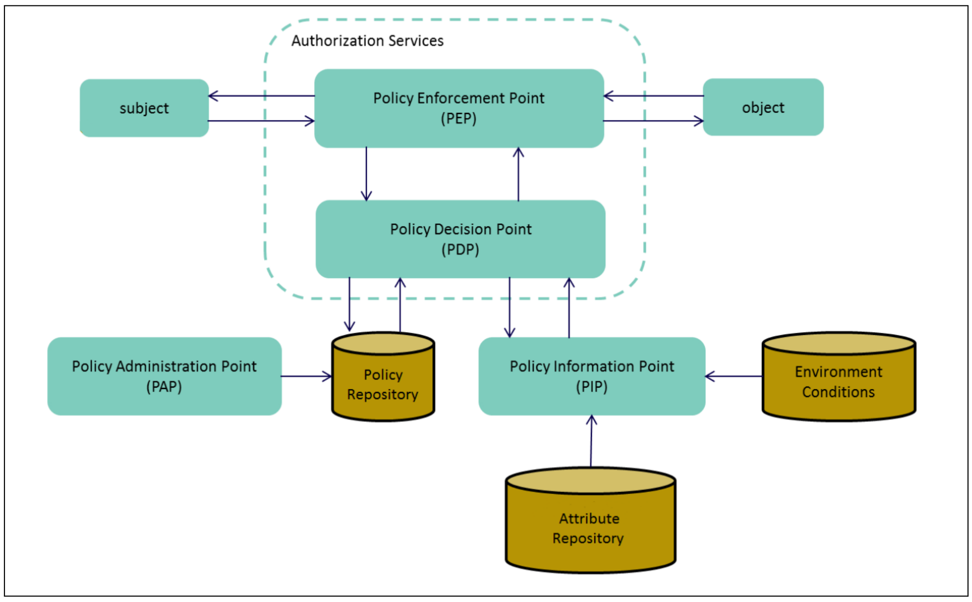

Authorization Services
subject
Policy Enforcement Point (PEP)
object
Policy Decision Point (PDP)
Policy Administration Point (PAP)
Policy Repository
Policy Information Point (PIP)
Environment Conditions
Attribute Repository

Figure 2.3: XACML Access Control Decision Structure [25, p. 15].

2.4 Attack Metamodel

This section describes the metamodel [54] that is the basis for the automatic attack path generation considered in this thesis.

In order to model access control used in the attack propagation the metamodel provides a context-based modeling of access control policies. The policies are used for determining whether a certain system element can be successfully attacked by an attacker. The concept of the metamodel is based on using context-based description of access control policies [8] in the form of ABAC (see Section 2.3).

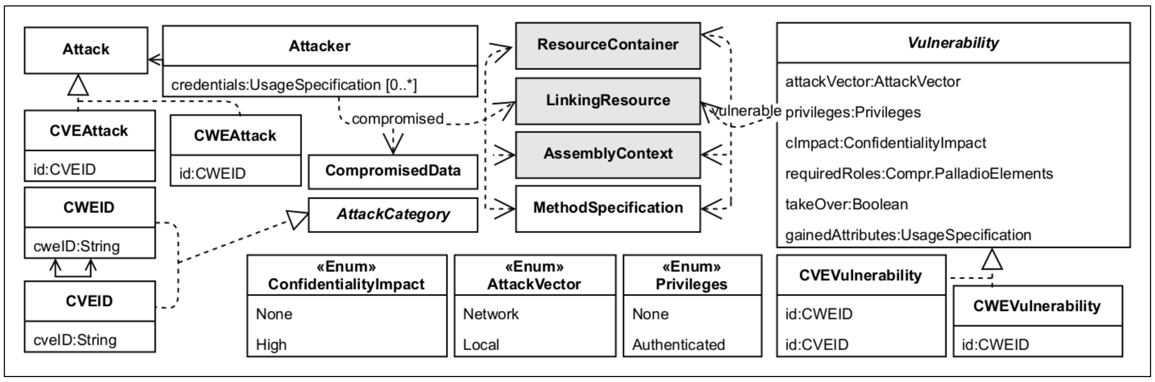


Figure 2.4: Simplified Metamodel for Attackers and System Vulnerabilities [55].

Additionally, there is also an attacker model contained in the metamodel [54]. As depicted in Figure 2.4, there is a metamodel for an attacker's capabilities and the system vulnerabilities. Every attacker has a list of `UsageSpecifications` being used as login details in the system. Moreover, there is also for each attacker a list of `CompromisedData` representing the potentially stolen data. An attacker can compromise the following Palladio elements: `ResourceContainers`, `LinkingResources`, and `AssemblyContexts` by the access credentials or by exploiting a `Vulnerability` in an `Attack`. The used classification system for vulnerabilities (CVSS) were already described in Section 2.1 and are modeled as `Vulnerability`, `AttackCategory` and `Attack` subclasses. Each vulnerability has a unique id represented by the `CVEID` and `CWEID` classes. Since a `CWEID` represents a category of attacks, it may also contain other IDs of vulnerability IDs. Both ID classes are subclasses of the class representing the attack category. Only the CVSS scoring values inside the metric are reused, not the actual scoring. For the purpose of describing exploitability, these are attack vector, attack complexity, required privileges and necessary user interaction. In order to describe the impact, the used metrics are confidentiality, integrity and availability. The scoring values are described in Table 2.1. There are also further model elements. First, an `AttackVector` that describes a location in the system, i.e. local or somewhere inside the network. Second, the `Privileges` enum that describes necessary login details, this means either `None`, the one for the element itself (`Low`) or a `Special` set of required credentials (specified by: `requiredCredentials`). Third, the `ConfidentialityImpact` that represents the impact regarding the leaked data: `None` means no data is leaked, `Low` means only directly affected data is leaked, `High` means all data is leaked. Fourth, there is a `gainedPrivilege` set describing the potentially newly gained privileges by exploiting the vulnerability. Lastly, there is a boolean flag in order to model whether an attacker can use the vulnerability to take over and compromise an element entirely, for example a compromised component can then be completely controlled by the attacker. Most parts of the system are reusable in other systems, only the credentials must be system-specifically defined.

3 State of the Art

This chapter briefly addresses the state of the art for managing access control in Section 3.1. Thereafter, model-driven confidentiality analysis and alternatives to the classification system are described in Section 3.2. Finally, different attacker modeling approaches are discussed in Section 3.3.

3.1 Access Control

This section describes different approaches than ABAC (see Section 2.3) for access control utilized in the field.

Two access control methods often used are Discretionary Access Control (DAC) [53] and Mandatory Access Control (MAC) [17]. DAC and MAC associate access rights directly to specific users and MAC additionally requires a passphrase. However, there are also other state-of-the-art access control strategies which are described in the following paragraphs.

Role-based access control (RBAC) extends a simple user-based access control system with roles. This allows the system administrator to define user groups. Each user group consists of different users that have the same role in the system and can therefore be granted the same access rights to perform certain operations. Furthermore, RBAC also allows the definition of role hierarchies in order to make sub-groups possible. RBAC also ensures that all users have exactly the access rights defined by their roles [19]. For example SecureUML [32] uses this access control strategy (see Section 3.2). One problem with RBAC is its fixed roles that are too static [56] to be able to react to dynamic context changes often seen in cloud services. At the moment, the approach presented by this thesis considers a subset of ABAC. This subset is close to role-based access control. However, there is also the considering of the interplay of vulnerabilities and access control in the attack path search presented in this thesis.

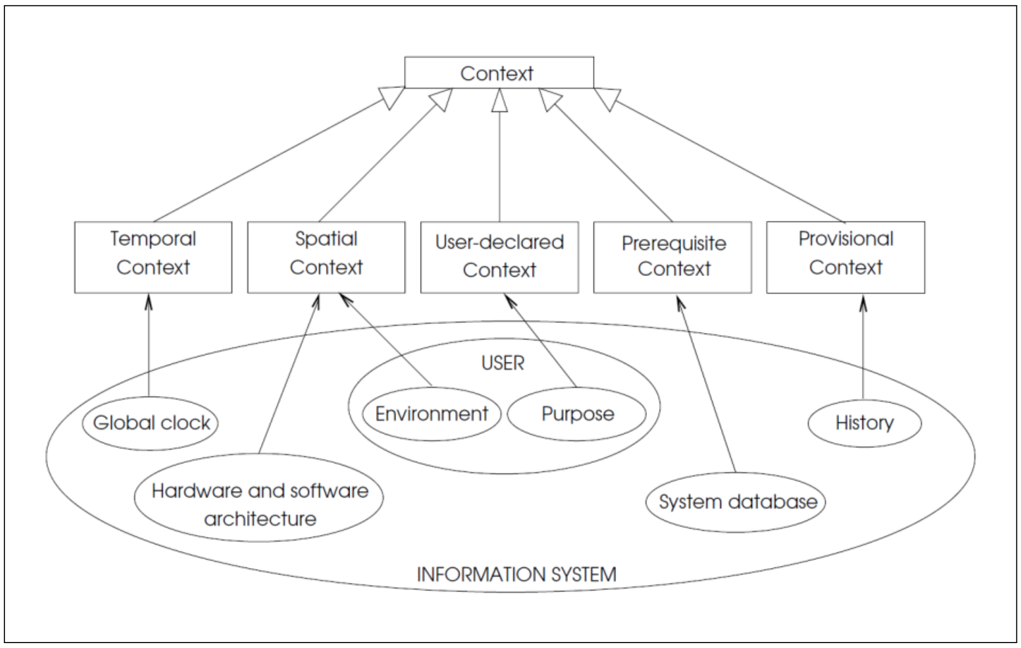


Figure 3.1: Structure of Different Context Classes in OrBAC [13].

Organization-based access control (OrBAC) is a context-based access control strategy that defines users and organizations as subjects and roles and contexts for defining access control policies. At the time OrBAC was presented, the context were yet uncategorized [28]. However, a classification of contexts was added later to the approach. As it can be seen in Figure 3.1, there are subclasses for temporal, spatial, user-declared, prerequisite and provisional contexts. The temporal contexts represent a certain time and spatial context a certain location defined by e.g. the user's IP address. Furthermore, the prerequisite context provides a possibility for granting access only if a given precondition is met. The provisional context is defined by an action that was triggered by an earlier executed action. At last, the user-declared context is used to enable users to define further purposes [13]. OrBAC is an alternative to ABAC also defining abstract elements like attribute but OrBAC accomplishes this in a hierarchical way (as can be seen in the figure) instead of a flat structure like attributes in ABAC [6].

3.2 Model-driven Confidentiality Analysis

As Nguyen et al. [35] state, there are many state-of-the-art techniques with the topic of model-driven confidentiality analysis. In this section, a selection of these approaches are described and briefly compared to the approach of this thesis.

UMLSec is an UML extension for modeling confidentiality concerns on the control path. It ensures secure communication and uses RBAC. Four security requirements are part of the UMLSec extension. These are fair exchange, confidentiality, secure information flow and a secure communication link. A fair exchange avoid cheating by a communication partner. Confidentiality means, that only a receiver that is allowed to obtaining a confidential information may receive this information. Secure information flow ensures that no information is leaked, not even partially. A secure communication link represents a link that is secure regarding a given attacker model [27].

SecureUML is also an UML extension allowing modelers to define permissions, roles and users in an UML diagram. For that purpose, annotations are used. These annotations can be added to any UML element. So, every element may be used as an resource containing security information. An automatic generation of access control policies is also possible. The SecureUML approach can also be seen as an extension of a RBAC system with OCL in order to add more dynamic ways of defining accesss control policies [32].

The following paragraphs describe the confidentiality analyses concerning information flow security. Katlakov et al. present the IFlow [29] approach. It enables model developers to automatically generate code and a formal model from an UML model of an application where a secure information flow is relevant. The generated code may be adapted but can always be checked against the formal model so that the user is not obliged to trust the programmer.

Tuma et al. [51] present an approach based on a Security Data Flow Diagram (SecDFD). The idea of this approach is to add security-relevant information to flow diagrams using labels. Each node represents a part of code which takes some input and generates output from it. The nodes have different types with different contracts. There are *Join*, *Copy*, *Encrypt* and *Decrypt* contracts. The contracts result respective in the highest of the input, the same, a `low` and the same labels on the output as on the input flow.

Gerking and Schubert [21] present another approach for information flow security. It is component-based and can be used to refine a macro-level security model to more fine-grained micro-level policies used in microservice architectures.

Kramer et al. [31] present an approach that models confidentiality information on a component-based architecture. The approach also provides and attacker intrusion analysis. For the purpose of representing the attack, the approach uses confidentiality definitions and adversary models. It can then infer whether confidential information was leaked. All this happens on a high-level architectural modeling level, which improves the usability and reusability of certain components in other systems.

As opposed to the approach presented in this thesis, the approaches mentioned in this section do not examine the attacker’s propagation with respect to the interplay of vulnerabilities and access control in the system.

3.3 Attacker Modeling

Schneier [44] presented the general idea of attack trees already in 1999. Mauw and Oostdijk [34] provided the formal foundation of attack trees and their semantics and analysis. This paragraph briefly presents the idea and semantics of *attack tree*s. The most important reasons for using attack modeling in general and attack trees especially are recognizing attack goals and probable attacks in order to be able to protect a system more efficiently. Attack trees typify attacks and defenses in a tree-like structure. The root node is the attacker's goal whereas each leaf node represents an attack. It is also possible to combine goals with an `AND` so that both goals must be fulfilled or an `OR` so that one of the goals must be fulfilled. Moreover, there is also the possibility to enhance the nodes with boolean values like *possible* or *impossible*. Continuous node values are also possible [44]. Nevertheless, attack trees only model the actions an attacker performs but they do not consider interconnection of different systems or even attacker propagation. An example for an approach using an extended version of attack trees is presented by Gadyatskaya et al. [20] who provide a modeling approach for attack-defense trees. This means, trees that not only model the attacker but also the defender. In their approach they also use timed automata in order to be able to apply model checking.

Kordy et al. [30] present in their survey DAG-based approaches for modeling attack and defense scenarios. There are two different classification models for attack and defense modeling. The first dimension represents whether an approach models only the attack, only the defense or both aspects. The second dimension represents whether an approach models only static aspects or also sequential time-based or order-based information. An example for static attack modeling is an *attack tree*. In order to model more complex attacks and defenses, different more complex DAGs are used like cryptographic DAGs or Bayesian networks for security. The rest of this section explains approaches directly based on DAGs.

This paragraph is about the *Cyber Security Modeling Language* (CySeMoL) [48]. It is a modeling language for describing computer systems in order to be able to calculate probabilities for successful attacks. The CySeMoL approach is based upon *probabilistic relational models* (PRM) defining attributes being discrete random variables and reference slots which represent relationships to other classes. The PRM is used in order to be able to compute the different probabilities of certain element properties inside an architectural instance. CySeMoL utilizes a PRM framework for obtaining probabilities of different attack path. Thereby, it does not only cover attacks against but also defenses of the system. However, there are some limitations to the defense modeling due to the construction of the PRM which should be as complete as possible whilst remaining usable for an average security manager. For instance, the PRM has limitations concerning threats against the confidentiality.

Polatidis et al. [39] present an approach for generating attack graphs in order to predict future attacks. The attack graphs used in this paper represent every path that can be used by attackers to increase the amount of privileges they have. Before the approach can be applied, four activities must be done. First, the user identifies starting points from where the attack begins. Second, the user identifies target points by selecting critical target systems. Third, attacker profiles need to be identified in order to define the attacker's potential.

Last, the approach generates vulnerability chains in order to enable the possibility for attacks with multiple steps. The actual attack prediction consists of a filtering method using parameters and the sorting of a certain amount of nearest neighbors. However, the filtering only considers the vulnerabilities whereas the approach presented here also contains a filter about access control and one for defining a maximum path length.

Aksu et al. [1] present approaches for rule-based and machine learning-employed models used for automatic attack graph generation. The general approach utilizes a graph containing pairs of a device and the associated privilege information as nodes. These information can be used as pre- and postconditions, which eases the generation of the attack graph. The database used for obtaining possible vulnerabilities is the NVD [36] already described in Section 2.1. However, this database does not provide the necessary information for deriving the privileges directly, so the approach presents two kinds of automatic generation of privileges, namely a rule-based one and a machine-learning-based one.

Yuan et al. [57] propose an approach based on attack graphs using a graph database storing the network's topological information. Then, the approach utilizes the graph database for generating a BFS-based algorithm in order to create all attack paths that are possible. The structure of the graph database and the used query language facilitates a fast implementation of the approach. However, the relatively new concept of graph databases has some disadvantages regarding usage and security in comparison to the more time-tested relational databases. In contrast to the approach of this thesis, the approach does not consider access control or more precisely the interplay of vulnerabilities and access control, e.g. obtaining credentials for other architectural elements by using a vulnerability.

Deloglos et al. [18] present an attacker modeling approach for *cyber-physical systems* (CPS) in order to make a prediction about the attacker's probability of succeeding. For that purpose, the approach utilizes a simulation of common attack behaviors derived from vulnerability databases. Additionally to the already described classification databases in Section 2.1, the approach also uses the *Common Attack Pattern Enumeration and Classification* (CAPEC) [10] and the *Common Platform Enumeration* (CPE) [12] database. In order to model the attack propagation, an attack state is used containing knowledge about the CPS nodes. Once a node is compromised the attack can continue with attacking adjacent nodes. Thus, the approach considers the attack propagation but it takes place in a cyber-physical system and not in a more general component-based system on an architectural level. Moreover, the approach does not consider access control an hence the effects of this aspect on the attack propagation are not examined.

To sum up, the different other state-of-the-art approaches do not consider the interplay of access control and vulnerabilities or they are not set in an model-based environment. The approaches that use filtering mechanisms do not consider filtering based on initial access control information.

4 Concept Overview

This chapter presents an overview of the concept of this Master's Thesis. The analysis takes place in the software architecture modeling environment *Palladio* [40] and the respective model *PCM* and the *Architectural Attack Propagation Metamodel* [54]. The purpose of the analysis is the finding of all relevant attack paths to a fixed critical architectural element, such as an assembly context or a resource container.

The attack path search considers all paths to a certain critical element. At the moment, it is possible to use a resource container or an assembly contexts as a critical element. There is always exactly one critical element during one search. The attacker inside the analysis references the critical element. Moreover, the analysis can make use of filter criteria for which there is an example in this chapter and more detailed descriptions in the following chapters.

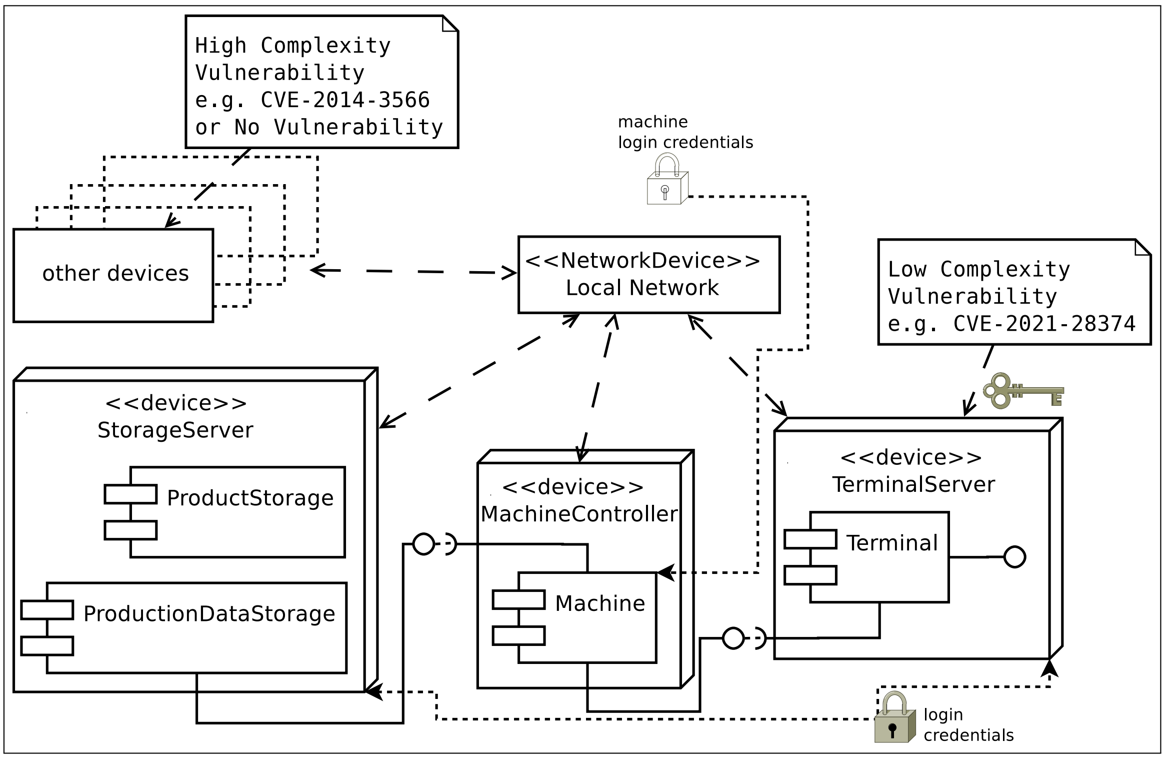

High Complexity Vulnerability e.g. CVE-2014-3566 or No Vulnerability
machine login credentials
other devices
<<NetworkDevice>> Local Network
Low Complexity Vulnerability e.g. CVE-2021-28374
<<device>> StorageServer
ProductStorage
ProductionDataStorage
<<device>> MachineController
Machine
<<device>> TerminalServer
Terminal
login credentials

Figure 4.1: Running Example.

Figure 4.1 depicts the running example described in the introduction. If one considers the assembly `ProductionDataStorage` as the critical assembly, the attack path search finds paths to that critical assembly. One of these paths is the one using the vulnerability annotated to the `TerminalServer` gaining the login credentials that can be reused in this example for the `StorageServer` to finally also the critical assembly due to its allocation on the newly compromised resource container. Without the usage of filters, there are also other attack paths including ones utilizing the high vulnerability.

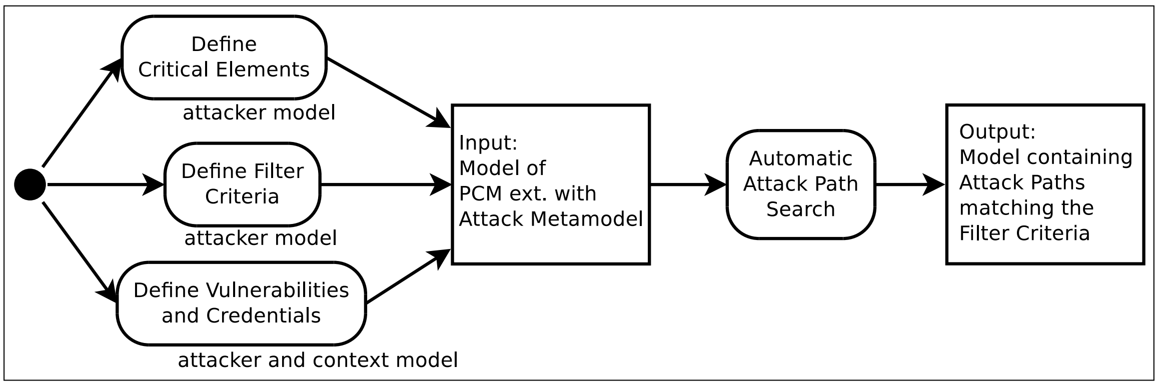

Define Critical Elements
attacker model
Define Filter Criteria
attacker model
Define Vulnerabilities and Credentials
attacker and context model
Input: Model of PCM ext. with Attack Metamodel
Automatic Attack Path Search
Output: Model containing Attack Paths matching the Filter Criteria

Figure 4.2: Overview of the Attack Path Search.

Figure 4.2 depicts an overview of the approach. For the purpose of starting the analysis, it is necessary that the software architect defines critical elements and annotates vulnerabilities and necessary credentials to the architectural model. Optionally, filter criteria can be added. The resulting output consists of the fitting found attack paths.

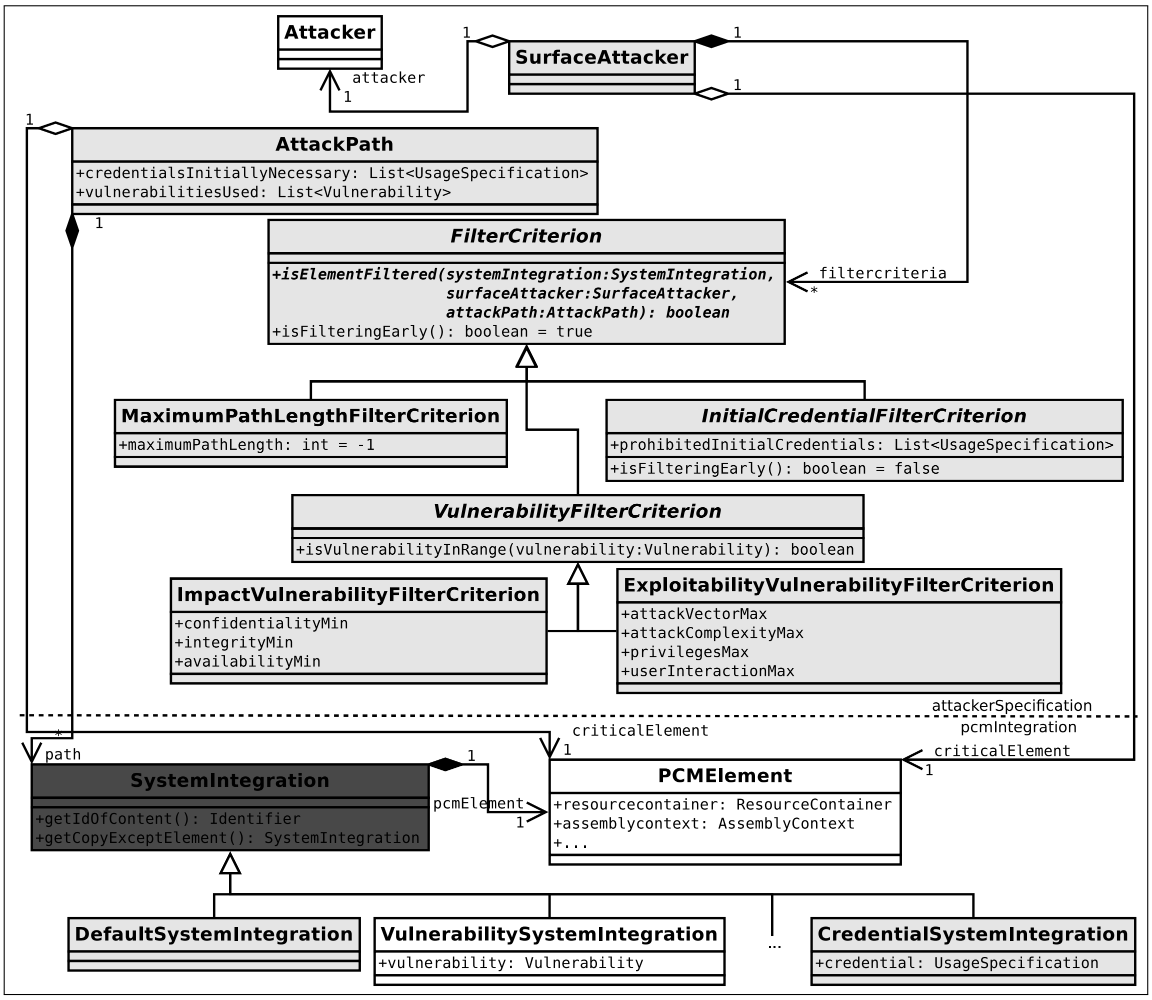
For instance, in the context of the running example (see Figure 4.1) one could decide to only search for low complexity attacks, which would result in finding only the relevant path to the `StorageServer`. The other devices without low complexity vulnerabilities do not need to be regarded in this example. For the purpose of finding the attack path, the simulated attacker would utilize the vulnerability on the terminal server in order to obtain the login credentials. In the next step, the attacker could try to use the login credentials on other devices and is able to compromise the storage server as well due to the same login data on it. Other vulnerabilities would not be looked at because of the filter criterion of only low complexity vulnerabilities.

Since a naive brute-force approach of finding all possible attack paths does not scale and is often not really meaningful, a selection based on certain filter criteria can take place in order to render the attack path generation useful. For that purpose, filter criteria are derived from properties of vulnerabilities, credentials and attack paths. In order to achieve this, the metamodel needs to be adapted as the following Chapter 5 describe in more detail. The Chapter 6 describes the concept of the attack path search.

The attack surface of a system is defined as the possible ways an attacker can gain access to the system in order to cause damage [33]. The approach presented in this thesis is called *Surface Attacker Analysis* because it exists in order to find all possible attack paths to a critical element. This can be considered an approach in order to minimize the attack surface to that critical element when the critical element is seen as the final exit point and all components as possible entry points.

5 Surface Attacker Modeling

This chapter describes additions and adaptions to the metamodel. In order to be able to perform a successful attack path search matching the requirements of the considered use cases and in order to add more extensibility in the future, there need to be adaptions to the metamodel. As a basis for selecting relevant paths there needs to be an extension to the metamodel describing the target elements considered critical in a given scenario. Especially, the filtering is important for more complex future usages of the metamodel and the analysis. Moreover, there is also an addition of the CVSS ratings (see Section 2.1) that are not present in the former metamodel.



Attacker
attacker
1
SurfaceAttacker
1
1
1
AttackPath
+credentialsInitiallyNecessary: List<UsageSpecification>
+vulnerabilitiesUsed: List<Vulnerability>
1
FilterCriterion
+isElementFiltered(systemIntegration:SystemIntegration, surfaceAttacker:SurfaceAttacker, attackPath:AttackPath): boolean
+isFilteringEarly(): boolean = true
filtercriteria
*
MaximumPathLengthFilterCriterion
+maximumPathLength: int = -1
InitialCredentialFilterCriterion
+prohibitedInitialCredentials: List<UsageSpecification>
+isFilteringEarly(): boolean = false
VulnerabilityFilterCriterion
+isVulnerabilityInRange(vulnerability:Vulnerability): boolean
ImpactVulnerabilityFilterCriterion
+confidentialityMin
+integrityMin
+availabilityMin
ExploitabilityVulnerabilityFilterCriterion
+attackVectorMax
+attackComplexityMax
+privilegesMax
+userInteractionMax
attackerSpecification
pcmIntegration
*
path
criticalElement
1
criticalElement
1
SystemIntegration
+getIdOfContent(): Identifier
+getCopyExceptElement(): SystemIntegration
1
pcmElement
1
PCMElement
+resourcecontainer: ResourceContainer
+assemblycontext: AssemblyContext
+...
DefaultSystemIntegration
VulnerabilitySystemIntegration
+vulnerability: Vulnerability
...
CredentialSystemIntegration
+credential: UsageSpecification

Figure 5.1: Additions and Adaptions of the Attacker Metamodel.

The class diagram in Figure 5.1 depicts the additions and adaptions to the metamodel. The light-gray classes are the added metamodel elements and the dark-gray class is an adapted one. The classes shown in white already existed in the base attacker model described in Section 2.4.

Additionally to the `Attacker`, there needs to be given a `SurfaceAttacker` as input. It references a critical element which is contained inside a `DefaultSystemIntegration`. Moreover, it contains different filter criteria which are explained in more detail in the next paragraph. The output model references the different `AttackPaths` that contain a path, i.e. a list of `SystemIntegrations` in order to also output the causes of compromisation of the elements on the path. A `SystemIntegration` exists for integrating vulnerabilities into the system and enabling outputting of architectural elements. For the purpose of having an easier overview over the used vulnerabilities and the initially necessary credentials each `AttackPath` also stores a list of vulnerabilities and credentials (`UsageSpecification`). The `pcmIntegration` package contains the different `SystemIntegrations`. The base class was adapted in order to enable copying `SystemIntegration` instances and getting the `Identifier` of the cause, i.e. the `Vulnerability` or `UsageSpecification`. Furthermore, the metamodel now also contains a `CredentialSystemIntegration` for annotating necessary credentials to model elements and integrating usage of credentials in the attack path.

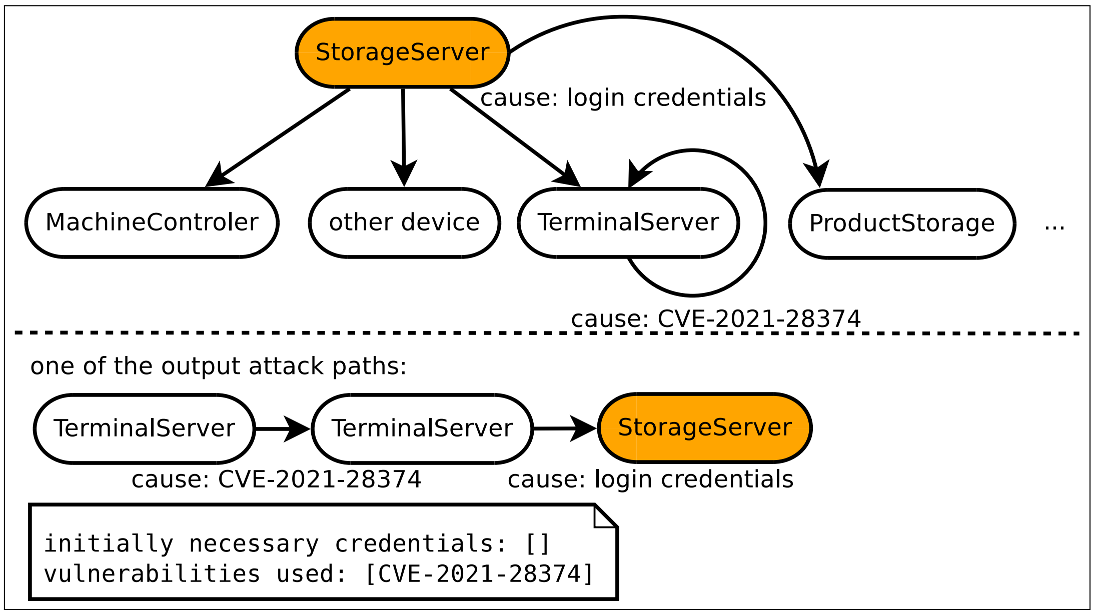
Another important aspect to the surface attack analysis is the usage of filter criteria. Each `FilterCriterion` is explained here in detail. The abstract base class defines two operations: the `isElementFiltered` and the `isFilteringEarly` operation, both returning a `boolean` value. The first operation determines whether an element is filtered, considering the actual element inside a `SystemIntegration`, the `SurfaceAttacker` itself and the temporary attack path at the moment of the filtering. The second operation defaults to `true` and determines whether the filtering can take place early in the analysis or only at the end. At the moment only the `InitialCredentialFilterCriterion` can only be used at the end. It is used to prohibit credentials to be used in the beginning, for example the root credentials. Another important filter criterion is the `MaximumPathLengthFilterCriterion` that filters a path if its length is higher than the positive `maximumPathLength` value. If this value is negative, no path is filtered. It is important that the path contains also the attack start. This means that attack path length is one element longer than just the path itself. For the purpose of restricting the usable vulnerabilities, there is an abstract `VulnerabilityFilterCriterion` with subclasses for filtering with respect to impact and exploitability of the vulnerability. The base class defines another operation named `isVulnerabilityInRange` determining whether the vulnerability is in range. A `VulnerabilitySystemIntegration` is filtered iff its vulnerability is not in range. For the `ImpactVulnerabilityFilterCriterion` the minimum value for confidentiality, integrity and availability are given and each vulnerability with higher values in each category is considered in range. Whereas for the `ExploitabilityVulnerabilityFilterCriterion` the maximum value for the attack vector, the attack complexity, the privileges and the user interaction is set in the filter. Each vulnerability with lower values in all categories is considered in range. For an exact overview over the different values for classifying vulnerabilities and which ones are lower and higher, see Section 2.1.

The different added vulnerability filter criteria are added using a wide-spread standard CVSS [11] so that a software architect can easily rely on a standardized approach when modeling instances of the metamodel. It is also possible during the modeling process that open databases are used in order to determine classifications for vulnerabilities (see Section 2.1). The filter criterion for initial credentials is used in order to prohibit certain initial credentials for the attacker to have because it is unrealistic that an attacker has for example root access from beginning on. The maximum path length filter is added in order to restrict the search to shorter paths, which is meaningful in order to prohibit too complex paths that are considered not realistic because they are too long. The filter criteria metamodel is designed to be extensible in the future with further filter criteria. Moreover, the usage of all the elements in the metamodel is possible at design time because the software architect can know all the relevant information or use wide-spread online databases, for instance to look up the concrete exploitability and impact values for vulnerabilities.

6 Design of the Surface Attacker Analysis

This chapter describes how the surface attacker analysis is designed. First, a conceptual overview about the general idea of the surface attacker analysis is presented. Thereafter, there are detailed descriptions of the most important aspects of the analysis.

The analysis makes use of a directed graph in order to represent the architecture model in a more abstract way. The reason for this is that it allows the usage of considering each architectural element as a node independently of its concrete type. The following example is used to describe how the graph is structured and how the attack paths are found inside it.



```
initially necessary credentials: []
vulnerabilities used: [CVE-2021-28374]
```

Figure 6.1: Example of a Graph and a Resulting Attack Path from the Analysis for the Running Example.

In the upper half, Figure 6.1 depicts an important part of the graph generated by the analysis for the running example. Moreover, it depicts one of the output paths in the lower half of the figure.

The graph has nodes representing the different architectural elements like resource containers and assembly contexts. The edges of the graph represent the possible ways an attack can take place. The attack may come from other resource container in the same network (i.e. connected to the same linking resource) or from local or remote (allocated in connected resource containers inside the same network) assembly contexts. The edges are directed against the direction of the attack and can store information about the causes of the attack. The reason for the direction to be against the attack direction is that the graph has only one root node, namely the critical element, and is therefore more easily to traverse. In the example depicted here, the critical element `StorageServer` node is connected to the resource container nodes `MachineControler`, `other device`, `TerminalServer` and the node for the local assembly context `ProductStorage` allocated on the critical element. Moreover, the `TerminalServer` has a self-edge representing the attack with the vulnerability `CVE-2021-28374`. There are further connections in the graph which are left out here so that the figure remains well-arranged. Additionally, there is an edge with a cause compromising the storage server with the gained login credentials. The analysis adds this cause after having gained the credentials due to the attack to the terminal server using the vulnerability.

The lower half of Figure 6.1 depicts an attack path resulting from the graph. It is the attack path already described in the last paragraph inside the graph. The attack path consists of the `TerminalServer` as the attack start, compromising itself with the vulnerability, thereby gaining the login credentials. These are then used in the next step in order to take over the `StorageServer` which is the critical element in this scenario. The causes are stored in the output model using the respective `SystemIntegrations`, i.e. a `CredentialSystemIntegration` for credentials or a `VulnerabilitySystemIntegration` for vulnerabilities. Furthermore, there are two more lists in the attack path output to the output model. The first one is the *initially necessary credentials* of which there are none in this example because no credentials are initially needed for this attack. It outputs the credentials that are necessary at the beginning of the attack path in order to successfully use this path. The second one is the list storing the vulnerabilities used on the attack path. In this example this list only contains the vulnerability `CVE-2021-28374` because only this vulnerability is used on the path. This list helps the software architect to see the utilized vulnerabilities at first sight.

Algorithm 1 Overview of the Main Procedure $surfaceAttackerAnalysis()$

Require: correct PCM, context and attacker models with a SurfaceAttacker containing a critical element e_{crit} and a correct attacker, [optionally filter criteria]

Ensure: output model containing all attack paths $list_{paths}$ to e_{crit}

```
1: $g : GraphWithARootElement$ storing the attack status of elements
2: $g \leftarrow g + e_{crit}$                    ▷ add the critical element as the root of the graph
3: $list_{element} \leftarrow \emptyset$
4: $changed \leftarrow true$
5: while $changed$ do
6:     $unvisitAll(g)$                          ▷ Set all nodes in the graph unvisited
7:     $e \leftarrow e_{crit}$
8:     $changed \leftarrow false$
9:     while $e \neq null$ do
10:        $c \leftarrow null$
11:        if $\neg isFiltered(e)$ then
12:            $list_{connected} \leftarrow getUnvisitedConnectedElements(e)$
13:            $list_{element} \leftarrow list_{element} + list_{connected}$
14:            for all $e_c \in list_{connected}$ do
15:                $changed \leftarrow changed \vee attackIfUnfiltered(g, e_c, e)$
16:            end for
17:            $list_{element} \neq \emptyset \rightarrow (c \leftarrow list_{element}.remove(0))$
18:        end if
19:        $visit(e)$
20:        $e \leftarrow c$
21:    end while
22: end while
23: $list_{paths} \leftarrow findAllAttackPaths(g)$
24: return $list_{paths}$
```

The pseudo-code Algorithm 1 describes an overview of the surface attacker analysis. The approach utilizes a directed graph with a root element that is the critical element. The attacker tries to propagate towards that critical element until no further attacks can be accomplished on the way to this element (see line 5). The graph is built during the attacker propagation by adding edges, silently adding also nodes to the graph. In each iteration the nodes in the graph are visited again as one can see in line 6. Each iteration searches the graph determined by the input models starting from the critical element (see line 7). The element e can be filtered if at least one of the filter criteria is filtering early and the element is filtered by this filter criterion. An attack and further propagation does not take place in this case (see line 11). However, if the element is not filtered the unvisited connected elements, i.e. all elements allocated on it or connected via connectors, are added at the end of the list of elements next to be visited (see line 13). These unvisited connected elements are used as attack sources for attacking e (see lines 14-16). Note that the *or* inside the formula in line 15 is non-short-circuiting, this is the case for all following pseudo-codes. The next element is determined as the first element of the element list if available (see line 17). If none is available, the inner iteration ends. If the propagation does not yield any changes anymore, the outer iteration ends and in the end the attack paths are found using the built graph `g` (see line 23). The pseudo-code algorithms for the procedures `attackIfUnfiltered` and `findAllAttackPaths` are described in the following paragraphs in more detail.

Algorithm 2 The Procedure $attackIfUnfiltered(g, e_{source}, e_{target})$

```
Require: $g : GraphWithARootElement$
Require: $e_{source} : Element$
Require: $e_{target} : Element$
Ensure: $\neg isFiltered(e_{source}) \rightarrow edge_{target,source}$ added or updated to / in $g$ with information about the attack if it is possible
1: $changed \leftarrow false$
2: if $\neg isFiltered(e_{source})$ then
3:     for all $attackHandler \in \{VulnerabilityHandler, ContextHandler\}$ do
4:         $changed \leftarrow changed \lor attackHandler.attack(g, e_{source}, e_{target})$
5:     end for
6: end if
7: return $changed$
```

The pseudo-code Algorithm 2 describes the `attackIfUnfiltered` procedure in more detail. The procedure exists for the purpose of attacking an element from an attack source element and in doing so adding an edge from the attack target to the attack source containing the cause of the attack. Note that the edge is reversed in respect to the attack paths generated in the end. This is implemented this way because it eases the traversal of the graph which has in this case exactly one root element, namely the critical element for the analysis. As one can see in line 2 an attack only takes place if the source element is not filtered. If the source element is not filtered there are two possible ways of attacking it. These methods are by using a vulnerability or by using credentials obtained by a former attack on the path. For each of this methods there is an attack handler that takes care of the actual attack (see lines 3-5). The procedure of the actual attack is explained in more detail in the next paragraph.

Algorithm 3 The Procedure $attackHandler.attack(g, e_{source}, e_{target})$

```
Require: $g$ : $GraphWithARootElement$
Require: $e_{source}$ : $Element$
Require: $e_{target}$ : $Element$
Ensure: $edge_{target,source}$ added or updated to / in $g$ with information about the attack if it
    is possible
 1: $changed \leftarrow false$
 2: $edge_{target,source} \leftarrow getOrCreateEdge(g, target, source)$
 3:                 ▷ Edge from target element to source element since graph is inverted
 4: $attackCause \leftarrow vulnerability \lor credentials \lor null$
 5:                      ▷ for respective handler or $null$ iff no attack is possible
 6: if $attackCause \neq null \land \neg edge_{target,source}.contains(attackCause)$ then
 7:     $edge_{target,source}.addCause(attackCause)$
 8:     $changed \leftarrow true$
 9: end if
10: $g \leftarrow g + edge_{target,source}$
11:   ▷ add or update edge to graph, silently adding potentially missing node for source
    element
12: return $changed$
```

The pseudo-code Algorithm 3 describes the `attack` procedure of the different attack handlers. As one can see in line 2 and 3, an edge from the target to the source is taken from the graph or created if it is not yet contained in the graph. Thereafter, the attack cause is identified. It can be a vulnerability or a usage of credentials formerly obtained contingent upon the attack handler type (see lines 4-5). If there is a new attack cause found it is added to the edge and the *changed* value is set to `true` (see lines 6-9). Afterwards, the edge is added to the graph if not existing or updated otherwise (see lines 10-11). The approach differentiates an attack in order to steal credentials form a complete taking over of a node. There are three possibilities of taking over a node. The first one takes place if a vulnerability is used that enables the attacker to take over the annotated element, i.e. the vulnerability must have the `takeOver` flag set. The second one uses already gained credentials to take over a node annotated with a `CredentialSystemIntegration` allowing the takeover with the gained credentials. The third one is the taking over of allocated assembly contexts if the containing resource container is already taken over. Attacks that do not take over a node but only steal credentials from a node are stored anyway inside the attack cause in order to be able to use them later on the attack path. Finally, the *changed* value is returned in order to inform the caller whether a new attack actually took place.

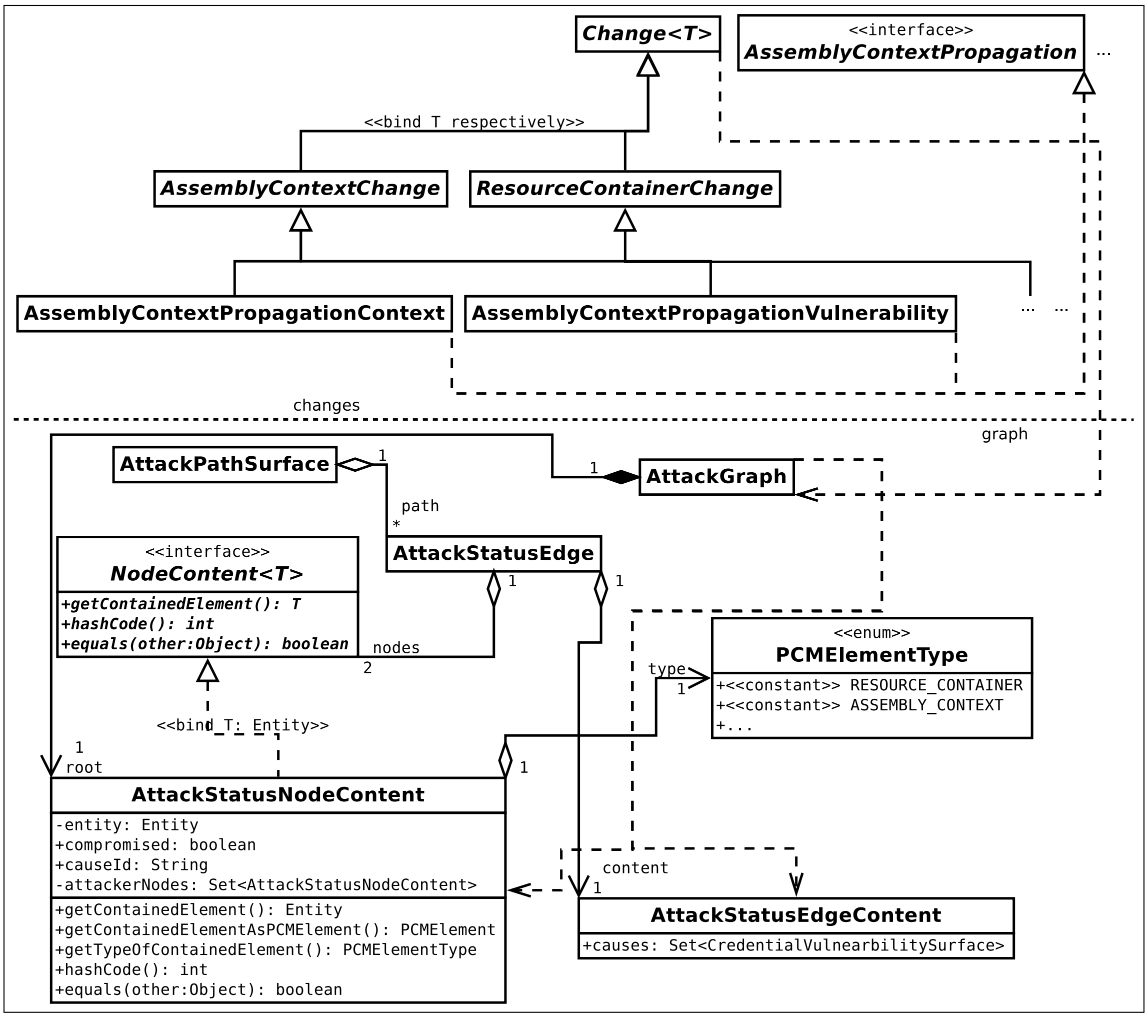
Algorithm 4 The Procedure $findAllAttackPaths(g)$

```
Require: $g : GraphWithARootElement$
Require: $g$ already filled with nodes and edges from the analysis and critical element $e_{crit}$
Ensure: all attack paths $list_{paths}$ to $e_{crit}$
 1: $unvisitAll(g)$                                    ▷ Set all nodes in the graph unvisited
 2: $list_{paths} \leftarrow \emptyset$
 3: for all $e \in g$ do
 4:     $attackElementWithInitiallyNecessaryCredentials(e)$ ▷ attack all nodes in graph
    with initially necessary credentials
 5: end for
 6: for all $e \in g.nodeIterable()$ do                ▷ sorted by edge/attack relevancy
 7:     if $\neg isVisited(e) \wedge isAttacked(e)$ then
 8:         for all $c \in children(e)$ do
 9:             if e.isAttackedBy(c) then
10:                 $list_{newAttackPaths} \leftarrow$ all fitting add. paths incl. the new attack edge excl.
    paths that are filtered out by the maximum path length filter
11:                 $list_{paths} \leftarrow list_{paths} + list_{newAttackPaths}$
12:             end if
13:         end for
14:     end if
15:     $visit(e)$
16: end for
17: $list_{paths} \leftarrow filterResult(list_{paths})$
18: return $list_{paths}$
```

The pseudo-code Algorithm 4 describes the `findAllAttackPaths` procedure mentioned in the overview pseudo-code. This procedure has the purpose of finding the attack paths in the attack graph and handle the propagation with initially necessary credentials. Therefore, to begin with, all elements in the graph are tried to be attacked with initially necessary credentials before actually creating the attack paths (see lines 3-5). Thereafter, the nodes are iterated in an order determined by firstly closeness to the critical element and secondly attack relevancy of the respective attack edge, i.e. reverse edge (see line 6). An attack is considered more relevant if there are more attack causes stored in the respective edge. Of course, new paths are only added if the considered element is not yet visited and if it is actually attacked, i.e. it is either taken over or at least one of the credentials are extracted (see line 7). In this case, all children `c` of the element `e` attacking the element `e` are considered as new attack edges (`c -> e`) yielding new attack paths (see line 9). For this edge, all paths fitting with this edge in the beginning are added to the list of possible paths (see l.10-11). An path and an edge are fitting when the resulting path is not too long and the first element of the path matches the element `e`, i.e. the target of the attack edge. In the end, there are some filters applied for filtering out duplicate and invalid paths which may be created by the algorithm (see line 17). Finally, the list of distinct valid paths are returned to the caller (see line 18) for the purpose of writing them to the output model.

7 Implementation

The implementation of the concept is an Eclipse plugin written in Java 11 because it is a widespread language and compatible with the EMF metamodel. During the process of the implementation, the version control system `git` was used. Moreover, the test framework `JUnit5` was used in order to test the functional correctness of the implementation. The final version of the metamodel and the implementation of the analysis prototype tool is available here[1]. The following paragraphs of this chapter describe an overview over the architecture of the implementation of the prototype tool implemented for the approach.



Change<T>
<<interface>> AssemblyContextPropagation
...
<<bind T respectively>>
AssemblyContextChange
ResourceContainerChange
AssemblyContextPropagationContext
AssemblyContextPropagationVulnerability
... ...
changes
graph
AttackPathSurface
1
path
*
1
AttackGraph
AttackStatusEdge
<<interface>> NodeContent<T>
+getContainedElement(): T
+hashCode(): int
+equals(other:Object): boolean
nodes
2
1
1
<<enum>> PCMElementType
+<<constant>> RESOURCE_CONTAINER
+<<constant>> ASSEMBLY_CONTEXT
+...
type
1
<<bind T: Entity>>
1
root
1
AttackStatusNodeContent
-entity: Entity
+compromised: boolean
+causeId: String
-attackerNodes: Set<AttackStatusNodeContent>
+getContainedElement(): Entity
+getContainedElementAsPCMElement(): PCMElement
+getTypeOfContainedElement(): PCMElementType
+hashCode(): int
+equals(other:Object): boolean
content
1
AttackStatusEdgeContent
+causes: Set<CredentialVulnearbilitySurface>

Figure 7.1: Overview of the Architecture of the Implemented Prototype Tool.

[1] `https://doi.org/10.5281/zenodo.6475682`

Figure 7.1 depicts an overview of the architecture of the implementation of the approach. The class structure is similar to the tool implemented by Walter et al. [55]. The difference is that an internal attack graph is used already described in the chapters before. There are `Change` classes for representing attack propagations coming from the respective architectural elements, i.e. an `AssemblyContextChange` for assembly contexts and `ResourceContainerChange` for resource containers. Each of these classes has subclasses for context and vulnerability propagations in order to add extensibility in the future. Every implemented propagation change implements a propagation interface, in the case of assembly contexts an `AssemblyContextPropagation`. For handling the actual attacks, there are attack handlers for different architectural elements and subclasses of them for the different attack kinds, namely attacks via contexts (credentials) and via vulnerabilities. The changes and the attack handlers utilize the `graph` package depicted in the lower half of the class diagram. It contains the `AttackGraph` that consists of nodes and edges. Internally, the *Google Common Graph*[2] framework is used. The nodes use instances of `AttackStatusNodeContent` as content in order to ease comparison of nodes. It implements the `NodeContent` interface, binding the generic parameter to be an `Entity`. Moreover, it has a `PCMElementType` that stores the type of the wrapped entity. As already described in Chapter 6, the edges contain the information about the attack. These information are stored in the `causes` set inside the `AttackStatusEdgeContent`. For the purpose of creating the attack paths, instances of `AttackStatusEdge` are used to create an `AttackPathSurface` instance that can then be used in order to create the actual output `AttackPaths` for the output model.

[2]`https://guava.dev/releases/30.1-jre/api/docs/com/google/common/graph/package-summary.html`

8 Evaluation

This Chapter describes the outline of the design of the evaluation in the form of a *Goal-Question-Metric-Plan* (GQM-Plan) [9] in Section 8.1. Thereafter, the design and use cases for the evaluation are described in Section 8.2. The following Section 8.3 presents the results of the evaluation. Moreover, the threats to validity are discussed in Section 8.4 and the limitations of the approach are discussed in Section 8.5.

8.1 Evaluation GQM-Plan

In this Section the GQM-Plan of the evaluation for accuracy, effort reduction and scalability are presented. Hence, there are these evaluation goals:

- **EG-1:** Evaluate the accuracy of the attack path search in order to consider functional correctness.
- **EG-2:** Evaluate the effort reduction of the automatic attack path search to a manual search.
- **EG-3:** Evaluate the scalability of the implemented approach.

In order to evaluate the accuracy of the approach, scenarios from the Subsection 8.2 are used to answer the following questions.

(Q-1.1) How accurate is the attack path search able to accurately find a critical attack path that could be found with a manual analysis?

(M-1.1 & M-1-2) As a metric the widespread *precision and recall* (PR & RC) [50] metric is used, consisting of the formulae:

$$PR = \frac{TP}{TP + FP}$$

$$RC = \frac{TP}{RPA}$$

where TP is the number of true positives and FP the one of false positives. RPA is the total number of expected positives. Here, a true positive means if an actual attack path is also found by the automatic generation. A false positive is a wrongly detected alleged attack path, that is not really one, wrongly found by the automatic generation. In order to determine which paths are considered a true positive, the expected paths are described in the next Section 8.2. The PR & RC metrics are used due to their widespread usage for evaluating the accuracy in other similar investigations such as [41], [23] and [3]. For **(M-1.1)** and **(M-1.2)** higher values are better.

For the evaluation scenarios from the Section 8.2 are used to answer the following question:

(Q-2.1) How much does the attack path search reduce the effort in comparison to a manual analysis?

In order to evaluate the effort reduction of the approach, the effort needs to be defined independently of the architect's experience. Walter et al. [55] present such an evaluation metric. Hence, this metric is used for evaluating effort reduction. In this metric, the effort is defined as the number of elements that need to be considered. Each element the analysis already detected inside an attack paths saves effort for the software architect. Moreover, for the purpose of calculating another propagation step, the software architect is also obliged to check each connected elements to the affected elements on the attack path.

(M-2.1) As a first metric for determining the amount of reduced effort (ER_1), the evaluation examines the ratio of affected elements (e_a) in each path to the number of connected elements (e_c):

$$ER_1 = \frac{e_a}{e_a + e_c}$$

(M-2.2) As a second metric for determining the amount of reduced effort (ER_2), the evaluation examines the ratio of affected elements (e_a) in each path to all elements (n):

$$ER_2 = 1 - \frac{e_a}{n}$$

For **(M-2.1)** and **(M-2.2)** higher values are better.

The scalability is evaluated using a larger scenario in order to estimate the runtime requirements of the implemented approach. Therefore, this questions is considered:

(Q-3.1) How does the runtime behavior of the implementation of the approach develeop with respect to the number of resource containers?

(Q-3.2) How does the runtime behavior of the implementation of the approach develeop with respect to the number of assembly contexts?

(M-3.1) In order to evaluate the time scalability of the approach, the time (t) is considered in relation to the number of architectural elements (n) as the following formula shows.

$$S_{time} = \frac{t}{n}$$

The architectural elements considered are assembly contexts and resource containers. The number of assembly contexts is used due to the component-based nature of the metamodel and assembly contexts are instantiated components. The resource containers are considered because assembly contexts are often allocated on many different resource containers. Therefore, scaling a model with respect to resource containers is also an important question to investigate. The metric **(M-3.1)** is used for answering both questions and lower value are better.

8.2 Evaluation Design

This Section presents the design of the evaluation including scenarios from case studies used during the evaluation process. However, some of the following scenarios cannot be directly used and therefore need to be adapted in order to enable modeling the scenario in Palladio. The evaluation uses three case studies and one research example. These are described here in detail. The research example is based on the TravelPlanner [45] architecture for the purpose of evaluating some of the added functionality not evaluated in the other scenarios. The other three scenarios are taken from real-world attacks described in case studies. As van den Berghe et al. [52] describe case studies are superior to using illustrative examples because it evaluates an approach in a real-world environment. This is helpful because it indicates usability and accuracy in reality and not only in constructed examples.

The first case study is a cloud storage scenario described by Alhebaishi et. al [2]. The study describes an overview of the considered architecture and also some attack paths described as text. The paper also presents an attack path analysis. However, it does not consider filter criteria and is not located in a software architecture modeling environment. In order to enable the evaluation of the implemented approach, the architecture was partially modeled in Palladio and relevant attack paths were determined. Only paths relevant for the scope of this thesis are considered. The attack paths are extracted manually from the information presented in the case studies. The case studies describe paths the attacker probably used in order to propagate through the architecture. These paths are then expected to be found by the analysis.

Figure 8.1 depicts the architecture of the cloud system as it is modeled for the use cases presented here. The network architecture consists of three network layers represented by the three network clouds, i.e. linking resources, with connections to several devices. The virtual machines are modeled as components and the actual servers are represented by resource containers. The networks are connected with network bridges represented as resource containers connected to two linking resources. There are three devices connected to the first network layer namely the users, the cloud tenant and an “OpenStack” authentication server. The second network layer consists of servers for ftp, http, an application VM and a database VM. Layer three contains a management device for the storage device and the storage device itself. For the purpose of analyzing different attack paths the database VM server and the storage device are modeled in more detail. The database VM server contains a hypervisor and a source VM with vulnerabilities. The `DB-VM` is connected to the source VM and the target VM. The vulnerabilities in the scenarios are `CVE-2012-3515` and `CVE-2013-4344` which enable stealing the credentials for the server. The next paragraphs describe the different expected attack paths in the cloud storage attack scenario.

The first two paths are described in the example 1 in the original case study. The first considered path is the path from the data base virtual machine (`DB-VM`) via the hypervisor using `CVE-2013-4344` taking over the `DB VM Server` with the gained hypervisor credentials. In the next step the target VM can then be compromised. The second path uses `CVE-2012-3515` on the `Source VM` to gain the credentials in order to compromise first the `DB VM Server` and the `Target VM` afterwards.

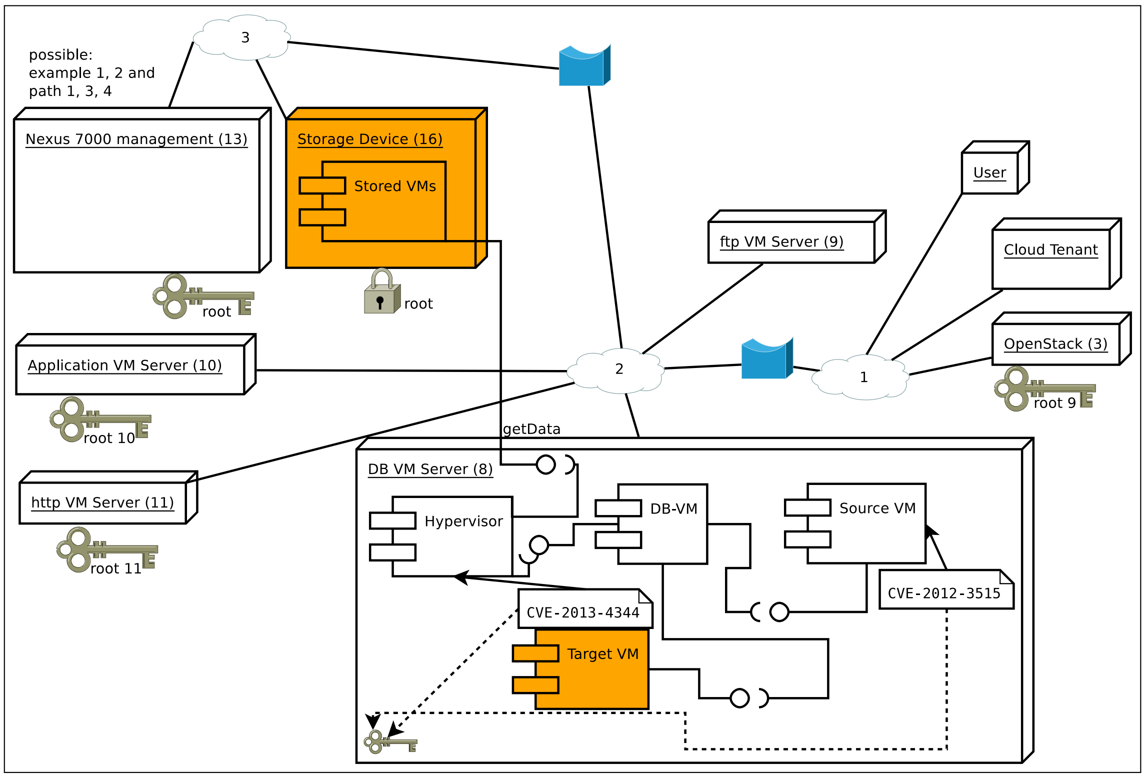


Figure 8.1: Overview over the Modeled Architecture of the Cloud Storage System.

Example 2 describes a path taking over the storage device with root credentials attacking from the Nexus 7000 management machine additionally also to the contained Stored VMs assembly as a variation in the critical element.

Path 1 is adapted because compromising linking resources was not implemented. That is why, there are now two paths considered inside the evaluation. The first one is an instance of path 3 described in the next paragraph and example 1. Thereafter, the stored VM data is obtained using the getData interface. The second one is a path from the Bridge 2-3 to the the storage device. Since compromising linking resources was not implemented, it is assumed that this bridge is already compromised. The path represents then the attacking of the storage device starting from the Bridge 2-3 instead of attacking the stored VMs from the hypervisor.

Path 3 is also slightly adapted and splitted into two path searches in order to compromise the Application VM Server explicitly as an critical element. The first path compromises the http VM Server and then the application server, the second one starts from the application server and attacks the ftp VM Server. The reason for this adaption is because both elements themselves can be considered critical element, so a software architect would also examine paths to both elements.

Path 4 simply consists of getting root for the ftp VM Server. With the use of example 1 and calling getData via FTP from the stored VMs the critical data of the VMs can be obtained.

For the following two case studies there already were Palladio models from the analysis of Walter et al. [55]. I adapted the models in order to fit the metamodel changes and enable the attack surface analysis to run.

The second case study takes place in an Ukrainian power distribution company in 2015. Booz Allen Hamilton and E-ISAC reviewed the attack in reports [22, 46]. The attack used the BE3 malware in order to steal credentials that were then used to attack the corporate network and the *industrial control systems (ICS)* network.

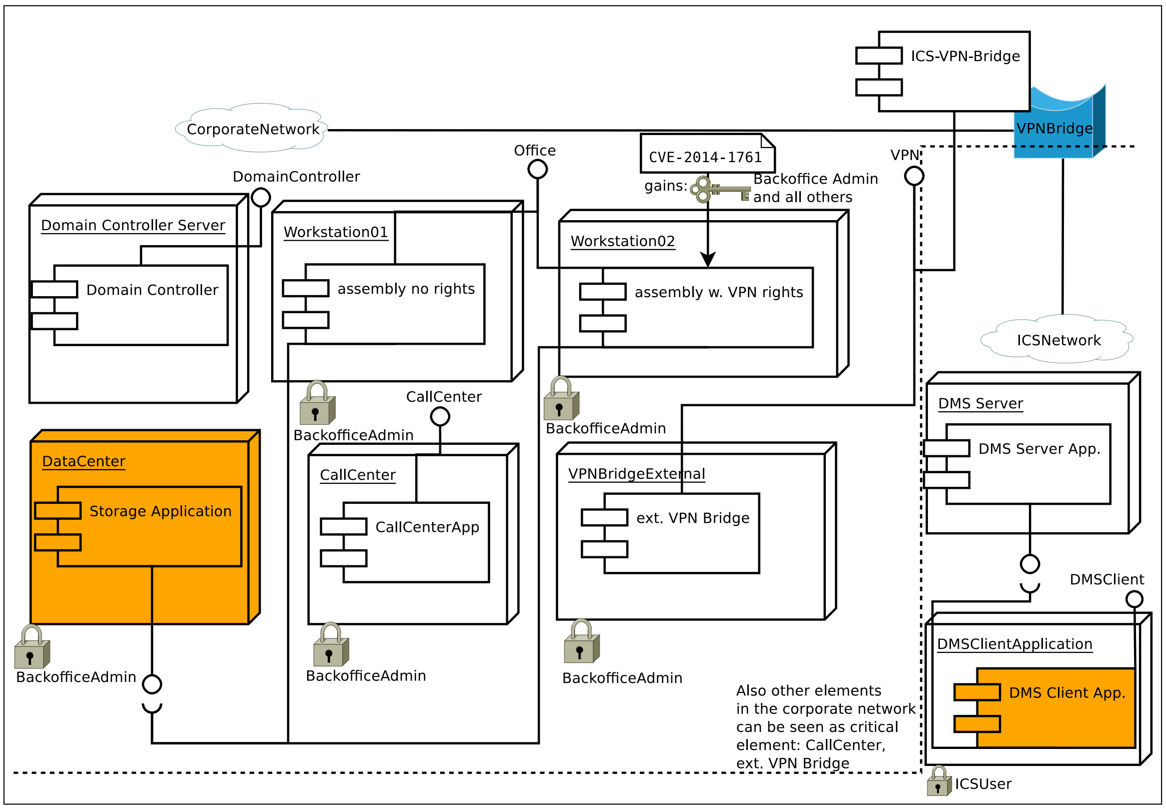


Figure 8.2: Overview over the Modeled Architecture of the Power Grid Corporate and ICS Network.

Figure 8.2 depicts the architecture modeled in order to analyze relevant attack paths in this scenario. The corporate network contains six modeled resource containers and a VPN bridge to the ICS network. The servers are a `DomainControllerServer`, two workstations, a `CallCenter`, a `DataCenter` and an external VPN bridge. Each resource container contains an application modeled as a component. The different resource containers are secured with a credentials of the `BackofficeAdmin`. This credential and also other relevant credentials can be obtained using the vulnerability `CVE-2014-1761`. Moreover, there is the ICS Network containing the document management server (`DMS Server`) and the respective client (`DMSClientApplication`). The application resource container is secured with credentials of the `ICSUser`. Several elements can be considered as a critical element in different attack path searches. There are 4 paths considered for analysis and one path with two variants. The next paragraphs describe these expected attack paths in the power grid attack scenario.

The first attack path considers the storage application as the critical element and has two variants. The first step of the attack path consists of attacking the assembly allocated on the `Workstation02` from the `Workstation02` resource container with the annotated vulnerability. The first variant does not use the stolen credentials on the `Workstation02` itself whereas the second variant also compromises the workstation. Thereafter, the stolen credentials are used in order to compromise the `DataCenter` and with that the attacker compromises the allocated `StorageApplication` too. The reason for the two variants is that the attacker can either also attack the workstation or do not use the credentials on it. The variant described by the case study is the one where the credentials are also used on the workstation itself. However, the other variant is also important because it puts the focus rather on the actual attacking of the critical element rather then the compromised elements on the path.

The second attack path considers the call-center application as the critical element whereas the third path finally attacks the external VPN bridge. These paths differ from the first path by the resource containers and assemblies that are attacked in the end of the path. The case study also describes and depicts these paths too, so they need to be considered here as well.

The fourth attack path models the propagation of the attack to the ICS network. In this path, the attacker first uses the vulnerability attached to the assembly allocated on `Workstation02` in order to steal the `ICSUser` credentials and the ones for the `VPNBridge` to the ICS network. Thereafter, the attacker propagates to the `DMSClientApplication` resource container, compromising it with the stolen credentials. The taking over of the container also triggers the taking over of the allocated assembly that is considered the critical element in this path because it can be used to compromise data inside the DMS via calling the respective service. This path is also described in the case study and it is especially important because the attacker also propagates here across boundaries of a network via a network bridge.

The third case study considers the Target Data Breach which happened in the end of 2013. The scenario is based on the case studies by Shu et al. [47] and Plachkinova et al. [38]. The scenarios considered in this use case contain CWE vulnerabilities related to weak and default passwords and improper privilege management. These vulnerabilities can be used to compromise the elements to which they are annotated. For future analyses there are also explicitly modeled credentials.

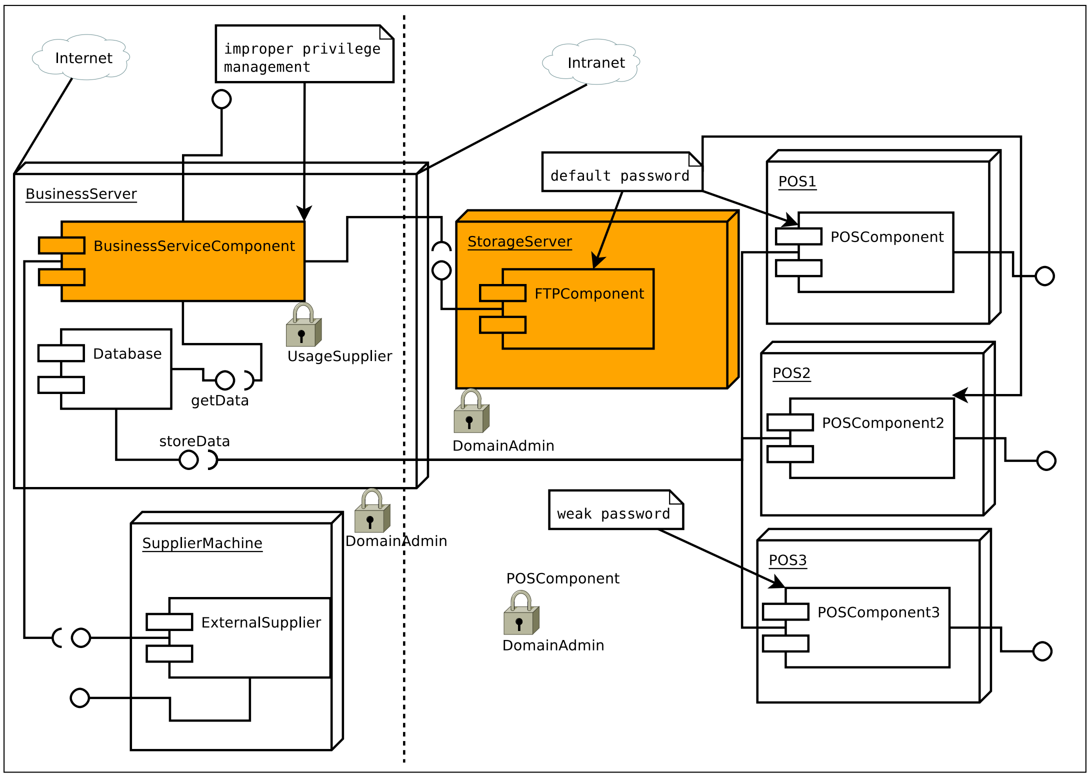


Figure 8.3: Overview over the Modeled Architecture of the Target Business Network.

Figure 8.3 depicts the part of the Target business network necessary for modeling attack paths extracted from the case study. The `Intranet` contains three Point of Sale (PoS) terminals (`POS`) and the `StorageServer` hosting an ftp server. On each POS, a `POSComponent` is deployed. The first and the second PoS component use a *default password*, the third one a *weak password* CWE vulnerability. The `SupplierMachine` and the `BusinessServer` are connected via the internet. However, the `BusinessServiceComponent` has a vulnerability of the category *improper privilege management* and the server is also connected to the intranet and especially to the `FTPComponent` allocated on the `StorageServer` via an assembly connector. Furthermore, the business service component has an assembly connection to the `ExternalSupplier` assembly allocated on the `SupplierMachine`. The following paragraphs describe the expected attack paths to the business service component and the ftp component in order to extract critical data from it.

The first attack path consists of the use of the vulnerability annotated to the business service component coming from the external supplier. The initial breach into the system of Target came from an attacker that compromised a machine of the external supplier Fazio. This path represents the initial entry path of the attacker described into the intranet of Target. That is why, this path is part of the expected path when examining the `BusinessServiceComponent` as a critical element. It is crucial to examine this element as a critical element because it is the connection to the internet and requires a service from an external supplier on the internet. Therefore, it is a possible entry point to the intranet in order to perform more complex attacks inside it.

The variants of the second attack path consider attacks from the business service components via the PoS components to the ftp component using the respective vulnerabilities. Since there are three PoS components, there are three attack paths considered as variants of this path. Once, the business service component is compromised, the attacker spreads inside the intranet network as described in the case study. The reason for examining this path is that it depicts approaches of attacking the critical storage element once the attacker has initial access to an assembly on a resource inside the network.

The third attack path is also examined as three variants, one for each PoS terminal. The attack path starts from the external supplier assembly and then attacks the business service component in order to compromise the POS component and finally also the FTP component. These paths represent the complete attack path to the critical storage component, starting from the external supplier. The reason for this is that this path represents the approach the actual attacker took described in the case study.

The fourth case study is the research example *TravelPlanner* [45] used for evaluating the functionality of adding filter criteria to the surface attacker and also different combinations of filter criteria.

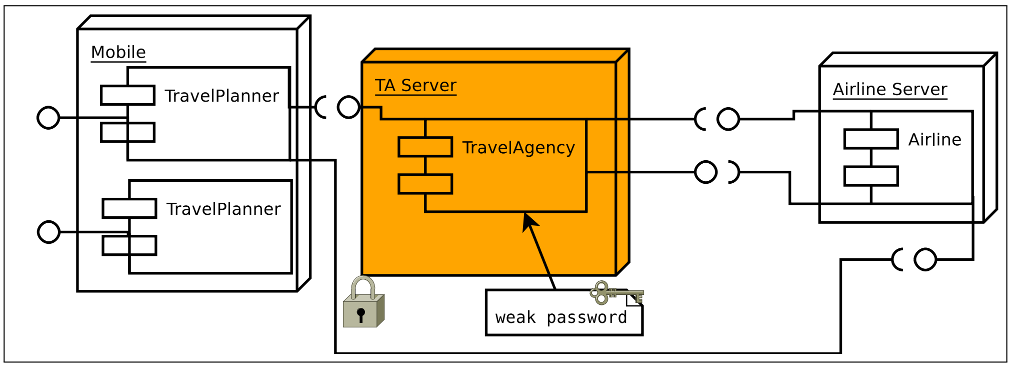


Figure 8.4: Overview of the TravelPlanner Architecture.

Figure 8.4 shows the architecture of the *TravelPlanner* case study. It consists of a `Mobile` device, a travel agency server (`TA Server`) and an `Airline Server` with the depicted assembly contexts and connectors. Furthermore, the `TA Server` is considered the critical element. The `TravelAgency` has a vulnerability of the category *weak password* that enables the attacker to obtain the root credentials for the TA Server, depicted by the golden key.

v Filter / Filter >	Root Credentials usable	Root credentials unusable
Vulnerabilities usable	no filters, max. path length 2/3/∞	max. path length 2/3/∞
Vulnerabilities unusable (avail. impact)	max. path length 2/3/∞	all paths filtered
Vulnerabilities unusable (attack vector)	max. path length 2/3/∞	all paths filtered

Table 8.1: The Evaluation Design Overview for the Evaluation of the Filter Criteria.

In order to evaluate the additional functionality, not investigated in the other case studies used for the evaluation, combinations of filter criteria are investigated on the *TravelPlanner* system annotated with vulnerabilities and credentials. Table 8.1 shows the overview about the evaluated combinations. There are 15 scenarios for the combinations of root credentials usable or unusable combined with vulnerabilities usable and unusable due to availability impact or attack vector. For each combination there are three scenarios, respectively with path length filters set to 2,3 or deactivated (∞) except for the cases in which all paths are already filtered out by the vulnerability and credential filters. Additionally, there is a scenario with no filters activated.

Fur the purpose of evaluating effort reduction, the values necessary for applying the metric described in the Section 8.1 are obtained by counting the elements that fulfill the described properties. The evaluation uses the aforementioned models extracted from the first three case studies described in this section. An affected element is an element on an examined attack path. A connected element to an affected element, is an allocated assembly context or a resource container in the same network in the case of a resource container element. In the case of an assembly context, only assemblies connected to the assembly via an assembly connector and the resource container the assembly is allocated on is considered a connected element. For counting all elements, only assembly contexts and resource containers are considered because linking resources were not directly incorporated in the analysis. Furthermore, only inaffected elements are counted as connected elements and no element is counted multiple times.

In order to evaluate scalability, a synthetic example is used and more attackable resource containers and assembly contexts are added to it automatically and then the analysis is run. It is then possible to see how the runtime behavior of the analysis is developing for larger models with more attackable elements. For the purpose of simulating larger models, a simple model is adapted by copying an element. The simple model consists of a `initial` resource container containing a resource container selectable as the critical element and an `initial` assembly context with a provided interface allocated on it, also selectable as a critical element. Moreover, another resource container `middle` is linked via a linking resource with the initial resource container. An assembly context named `middle` is allocated on the resource container with the same name. That assembly context has a provided and a required interface in order to be able to be chained. In the case of the assembly scalability evaluation the assemblies are chained, made attackable by a test vulnerability and then allocated with pooled resource containers, i.e. on different resource containers connected to the same linking resource. For resource container scalability evaluation, there was also another kind of chaining, namely chaining the resource containers with a new linking resource for each added resource container. For both kinds of evaluation, there are measurements for running the complete analysis and for running only the respective propagation, i.e. resource to resource propagation or assembly to assembly propagation. Since paths longer than the number of elements are not meaningful a path length filter of this length is added.

The results were obtained with a warmup value of two and ten repetitions. Except for the evaluation of 20 added elements, there were only two repetitions for time reasons. The resulting time value was then determined as the average of the measured time values for the number of repetitions. The system used for the scalability evaluation is the notebook `NHx0DB,DE` running Linux Ubuntu 20.04.1LTS with an `Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz` CPU with 4 cores and 8 threads and 32 GB `SD-4` RAM. The Java and Eclipse versions are Java 11 (`openjdk 11.0.14.1 2022-02-08`) and `Eclipse Modeling Tools 2021-12 (4.22.0)`.

8.3 Evaluation Results and Discussion

This section presents and discusses the results of the evaluation of the analysis. First, there will be a summary of the results for the accuracy evaluation and a discussion about these results. Second, the results of the evaluation for effort reduction are described. Finally, the scalability of the approach is examined and discussed.

Table 8.2 show the results for accuracy the implemented prototype for the presented approach achieved. It summarizes the results for each case studies. The first column shows the different case studies and one last summary row for the whole accuracy evaluation. The *TP* column shows the true positives, i.e. the correctly found paths. The next column *RPA* shows the expected positives, i.e. the paths expected by the respective reference set. The fourth column shows the false positives. The next column depicts the result for precision and the last column for recall. All path variants were considered in the evaluation. All expected paths were found except one path variant in the Target Breach case study, probably due to an issue related to node order in attack path finding. Overall, the accuracy evaluation resulted in a precision (*PR*) value of 1.0 and a recall (*RC*) value of 0.97.

Case Study	TP	RPA	FP	**Result PR**	**Result RC**
Cloud Storage	8	8	0	1.0	1.0
Power Grid	5	5	0	1.0	1.0
Target Breach	6	7	0	1.0	0.86
Travel Planner	15	15	0	1.0	1.0
Q-1.1	34	35	0	**1.0**	**0.97**

Table 8.2: The Accuracy Evaluation Result for the Implemented Prototype Tool.

The next investigated evaluation question is about the effort reduction. Table 8.3 depicts the results of this evaluation. The first column shows the different examined paths. The next three columns depict the values necessary for the calculation of the metrics and the last two columns show the results of the metrics. Although, the first metric with an overall result value of 34% is not very high, the second metric of the effort reduction evaluation achieved a value of 80%. This means that effort can be rather reduced in relation to the complete number of elements rather than the number of connected elements for affected elements on the path. Since the approach searches a path to a critical element this is an acceptable result because it is more important to reduce the effort of manually checking all model elements than to reduce it with respect to connected elements to affected elements on the path. However, also for this metric an effort reduction of 34% can be achieved regarding the considered scenarios.

Case Study and Path	e_a	e_c	n	ER_1	ER_2
Cloud Storage example 1.1	4	8	16	0.33	0.75
Cloud Storage example 1.2	3	9	16	0.25	0.81
Cloud Storage example 2 container	2	2	16	0.50	0.87
Cloud Storage example 2 assembly	3	1	16	0.75	0.81
Cloud Storage path 1.2	3	7	16	0.30	0.81
Cloud Storage path 3.1	2	5	16	0.29	0.87
Cloud Storage path 3.2	2	5	16	0.29	0.87
Cloud Storage path 4	1	5	16	0.16	0.94
Power Grid path 1a	4	5	18	0.44	0.78
Power Grid path 1b	4	5	18	0.44	0.78
Power Grid path 2	4	5	18	0.44	0.78
Power Grid path 3	4	5	18	0.44	0.78
Power Grid path 4	5	9	18	0.36	0.72
Target Breach path 1	2	7	13	0.22	0.85
Target Breach path 2a	3	8	13	0.27	0.77
Target Breach path 2b	3	8	13	0.27	0.77
Target Breach path 2c	3	8	13	0.27	0.77
Target Breach path 3a	4	6	13	0.40	0.69
Target Breach path 3b	4	6	13	0.40	0.69
Q-2.1	60	114	296	**0.34**	**0.80**

Table 8.3: The Effort Reduction Evaluation Result for the Approach.

The last evaluation goal examined the scalability of the approach. The following diagrams show the results of the different scalability evaluation setups. The first Figure 8.5 depicts the runtime scalability with respect to assembly contexts whereas the second Figure 8.6 depicts the runtime scalability with respect to resource containers. The third Figure 8.7 depicts the runtime behavior of the chained resource containers scalability evaluation. The x-axis shows the number of added assembly contexts or resource containers whereas the y-axis depicts the consumed runtime for running the analysis and is a logarithmic scale. The blue dots always represent the runtime of the analysis only for the respective element type whereas the red dots present the results for the complete analysis. The black and orange lines are the respective exponential reference lines.

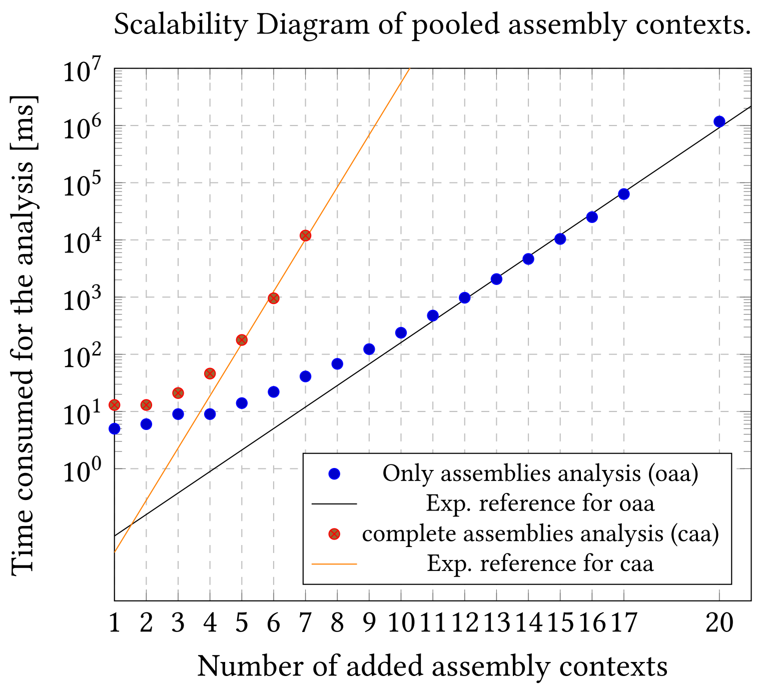


Figure 8.5: Results of the Analysis for Assembly Contexts Runtime Scalability Evaluation.

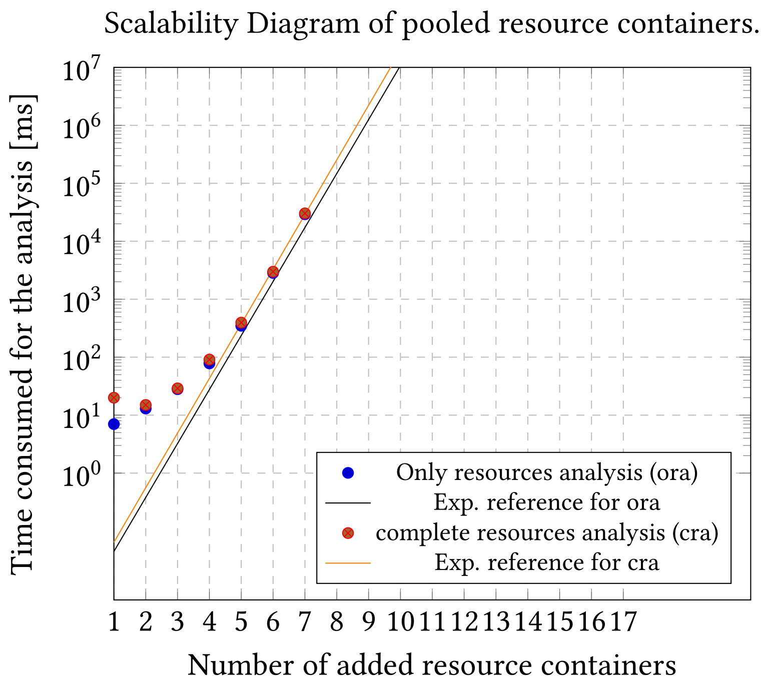


Figure 8.6: Results of the Analysis for Pooled Resource Containers Runtime Scalability Evaluation.

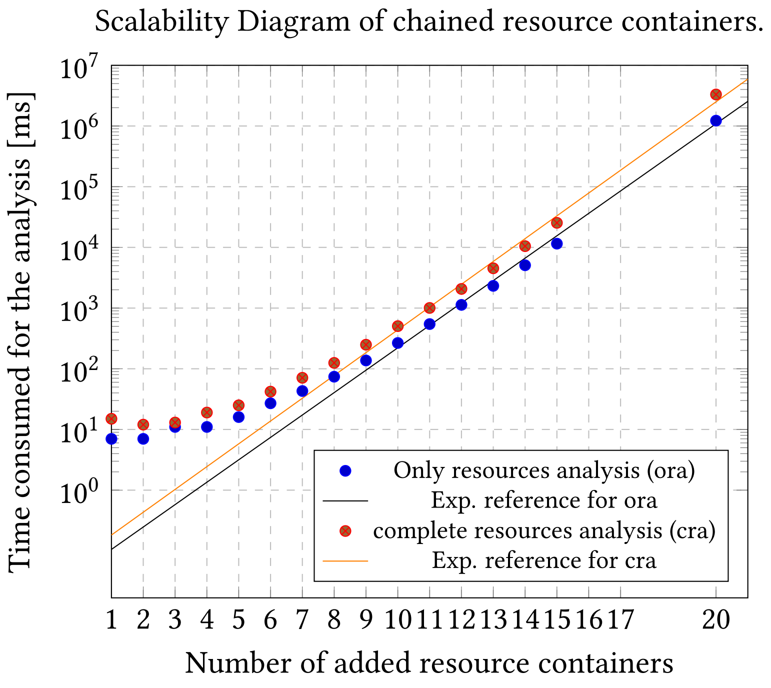


Figure 8.7: Results of the Analysis for Chained Resource Containers Runtime Scalability Evaluation.

As the results indicate, the runtime scalability behavior is exponential in the number of added architectural elements. The reason this is the case, is because the generated models lead to an exponentially increasing number of possible attack paths because it is possible for an attacker to move in both directions along a assembly connector or a link. Another reason for the exponential runtime behavior is the properties of the added elements. Since each element is attackable, there is no direction in which an attack stops propagating. Moreover, no filters except a path length filter for the number of elements are applied. However, for the scalability scenario for chained resource containers and the one for pooled assembly contexts regarding only the assembly propagation are usable for a number of added elements of 15 or for assemblies 17 elements runtime values of under 70 seconds are achieved. For the other cases a value of seven added elements achieved a runtime from under 40 seconds. Nevertheless, the scalability of the approach is the main weakness of it and path finding in general as the evaluation results for 20 added elements show in the cases this evaluation was done. Therefore, it is necessary that the software architect use a wise set of filters in order to obtain a result in a reasonable time. Moreover, rendering the analysis more scalable for larger scenarios needs to be examined in future work. In general, graph algorithm for finding even all paths between two nodes have exponential runtime behaviors because the number of paths are exponential. In the case of the analysis presented in this thesis it is even required to find all paths to the critical element from arbitrary start nodes, which worsens the situation even more. However, a meaningful selection of filter criteria can render the approach more scalable and allows the software architect to restrict the output paths to relevant ones.

8.4 Threats to Validity

This section describes the concept of how to lower the risk of threats to validity. The categorization and description of these threats were described by Runeson and Höst [43] and consist of *Construct Validity*, *Internal Validity*, *External Validity* and *Reliability*. The following subsections describe these four categories and how these threats are addressed.

The *Construct Validity* is about whether the intended investigation by the goal and questions is operationally well represented. This means that the examined characteristics, in this case the metrics are relevant for achieving the respective goals. For that purpose, for the accuracy evaluation similar metrics like precision and recall are used as in other architectural model-based approaches. For example, the KAMP approach [24, 42] and the approach presented by Walter et al. [55] use these metrics. For the evaluation of the effort reduction there is also usage of a similar metric. Using similar metrics decreases the overall risk. Thereby, the threat to *Construct Validity* is reduced. For scalability, a higher number of tested elements could lower the risk of obtaining wrong results but the considered number of element already indicated the issues with scalability rendering filtering necessary.

The facet of *Internal Validity* is about causal relations and influences of one factor on another. If this is the case and not regarded by the research, there is a high probability that other factors are also not considered and hence, the internal validity is threatened. Since the attack path search is very dependent upon the selected scenario models and the manually created output for comparison, there should be a strategy to reduce the risk for threats to *Internal Validity*. For that purpose, the scenarios are selected such that all aspects of the implemented approach are examined. Especially, this is accomplished with the usage of the TravelPlanner evaluation (see Sections 8.2 and 8.3) for examining newly implemented functionality not examined by the other case studies. Moreover, each model element type is considered in at least one of the scenarios used in the evaluation of the approach. Furthermore, the considered attack paths for the reference output are selected as described inside the considered case studies. Therefore, the threat to internal validity can also be assumed to be reduced.

The facet of *External Validity* considers the portability to other use cases in order to universalize the research results. Moreover, it is also investigated whether the results are useful to other researchers. In order to reduce the risk of threats to the *External Validity*, external case studies are used. However, there are some limitations to the approach that could be a threat to this kind of validity, for example the scalability of the approach. These threats should be considered in future work by the investigation of further case studies and the consultation of security experts. The limitations are described and discussed in the next Section 8.5.

Reliability is about the reproducibility of the results. Clearly defining and providing the scenario models and using well-defined statistical metrics enables future researchers to reproduce the results and thereby reduces the risk of threats to *Reliability*. Moreover, there are automatic tests for the accuracy and scalability evaluation enabling other researchers to reproduce the results. Furthermore, the code, the metamodel and the models are also published in order to ease reproduction of the results.

8.5 Limitations

This section presents the limitations of the approach and its implementation. There are some conceptional limitations regarding the kind of attacker considered by the approach. For instance, an attacker requiring user interaction of a regular user during the attack cannot be analyzed. Moreover, it is also crucial that the vulnerabilities in the system are known so that they can be modeled by the software architect and then considered by the analysis. However, it is often possible to at least know attack categories and use a CWE vulnerability, for example provided by utilizing OWASP [37]. Another conceptional limitation of the approach is that it does not consider complex mitigation strategies or defense mechanisms. Nevertheless, the approach is usable in order to enable the software architect to find elements where additional security measures could be meaningful in order to render a found attack path impossible.

Due to insufficient time, some initially planned functionality could not be implemented. The first limitation is the attack propagation to other architectural elements than assembly contexts and resource containers such as linking resources and services. Therefore, some more detailed attack scenarios cannot be considered, for example if an attack takes place via a service rather than a whole assembly context. Another limitation is that at the moment only a simplified version of the initially planned access control mechanisms is implemented. Namely, only simple attributes, i.e. login credentials are considered at the moment. Moreover, there are limitations to the scalability of the approach that should be considered in future work.

9 Conclusion and Future Work

This chapter concludes the findings of this thesis. Furthermore, it points out possible future work.

To conclude, the thesis adapted the attack metamodel (see Chapter 5) in order to then present and implement the approach for an attack path search described in Chapter 6. The adaptions to the metamodel consists of adding filter criteria and an attacker with a reference to a critical element. The analysis utilizes the adapted metamodel in order to analyze instances of the metamodel with respect to finding all attack paths to a critical element. The filter criteria presented are derived from widespread standards and are therefore easily capable of being integrated.

The evaluation indicated a high accuracy of 97 % for the implemented approach in scenarios taken from real-world case studies. Moreover, it indicated an effort reduction of 34 % up to 80 %. The scalability of the implemented approach was also considered but indicated a exponential runtime. However, adding filter criteria to the analysis can render the approach more scalable and helps to restrict the output paths to more relevant ones.

The main contribution of this thesis is an approach for finding all attack path in models inside a component-based architectural environment with filter criteria. It enables software architects and security experts to model software architectures also with respect to confidentiality and find relevant attack paths to a critical element in these architectures. This can lead to the addition of additional defensive mechanisms in the considered architectures rendering them more secure.

This paragraph describes ideas for future work concerning the subject of this thesis. The first necessary future work is the implementation of the unimplemented features described in Section 8.5. Additionally, if an attacker performs data extraction is also not yet considered. Adding this feature to the analysis may take place in the future. Moreover, it is important to render the approach more scalable in the future. This could be done by adding further filter criteria in order to restrict the search-space. Additionally, this is also meaningful in order to restrict the output attack paths to match the requirements of software architects even more. Other important future work consists of examining larger scenarios and use cases derived from case studies.

Bibliography

[1] M. U. Aksu et al. “Automated Generation of Attack Graphs Using NVD”. In: *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*. CODASPY ’18. Tempe, AZ, USA: Association for Computing Machinery, 2018, pp. 135–142. ISBN: 9781450356329. DOI: `10.1145/3176258.3176339`. URL: `https://doi.org/10.1145/3176258.3176339`.

[2] N. Alhebaishi et al. “Threat modeling for cloud data center infrastructures”. In: *International symposium on foundations and practice of security*. Springer. 2016, pp. 302–319.

[3] R. Al-Ali et al. “Modeling of dynamic trust contracts for industry 4.0 systems”. In: *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*. 2018, pp. 1–4.

[4] S. Becker, H. Koziolek, and R. Reussner. “Model-Based Performance Prediction with the Palladio Component Model”. In: *Proceedings of the 6th International Workshop on Software and Performance*. WOSP ’07. Buenes Aires, Argentina: ACM, 2007, pp. 54–65. ISBN: 1-59593-297-6. DOI: `10.1145/1216993.1217006`. URL: `http://doi.acm.org/10.1145/1216993.1217006`.

[5] S. Becker, H. Koziolek, and R. Reussner. “The Palladio Component Model for Model-driven Performance Prediction”. In: *Journal of Systems and Software* 82 (2009), pp. 3–22. DOI: `10.1016/j.jss.2008.03.066`. URL: `http://dx.doi.org/10.1016/j.jss.2008.03.066`.

[6] M. Belhaouane, J. Garcia-Alfaro, and H. Debar. “Evaluating the comprehensive complexity of authorization-based access control policies using quantitative metrics”. In: *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*. Vol. 4. IEEE. 2015, pp. 53–64.

[7] E. Bertino et al. “The challenge of access control policies quality”. In: *Journal of Data and Information Quality (JDIQ)* 10.2 (2018), pp. 1–6.

[8] N. Boltz, M. Walter, and R. Heinrich. “Context-Based Confidentiality Analysis for Industrial IoT”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2020, pp. 589–596. DOI: `10.1109/SEAA51224.2020.00096`.

[9] V. R. B. G. Caldiera and H. D. Rombach. “The goal question metric approach”. In: *Encyclopedia of software engineering* (1994), pp. 528–532.

[10] *CAPEC*. URL: `https://capec.mitre.org/` (visited on 04/22/2022).

[11] *Common Vulnerability Scoring System version 3.1.* Tech. rep. FIRST.Org, Inc. (FIRST). URL: `https://www.first.org/cvss/v3-1/cvss-v31-specification_r1.pdf` (visited on 04/22/2022).

[12] *CPE.* URL: `https://nvd.nist.gov/products/cpe` (visited on 04/22/2022).

[13] F. Cuppens and A. Miege. "Modelling contexts in the Or-BAC model". In: *19th Annual Computer Security Applications Conference, 2003. Proceedings.* Dec. 2003, pp. 416–425. DOI: `10.1109/CSAC.2003.1254346`.

[14] *CVE.* URL: `https://cve.mitre.org/` (visited on 04/22/2022).

[15] *CVSS.* URL: `https://www.first.org/cvss` (visited on 04/22/2022).

[16] *CWE.* URL: `https://cve.mitre.org/` (visited on 04/22/2022).

[17] S. De Capitani di Vimercati and P. Samarati. "Mandatory Access Control Policy (MAC)". In: *Encyclopedia of Cryptography and Security.* Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 758–758. ISBN: 978-1-4419-5906-5. DOI: `10.1007/978-1-4419-5906-5_822`. URL: `https://doi.org/10.1007/978-1-4419-5906-5_822`.

[18] 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.* Ed. by A. Casimiro et al. Cham: Springer International Publishing, 2020, pp. 150–163. ISBN: 978-3-030-54549-9.

[19] D. Ferraiolo, J. Cugini, D. R. Kuhn, et al. "Role-based access control (RBAC): Features and motivations". In: *Proceedings of 11th annual computer security application conference.* 1995, pp. 241–48.

[20] O. Gadyatskaya et al. "Modelling Attack-defense Trees Using Timed Automata". In: *Formal Modeling and Analysis of Timed Systems.* Ed. by M. Fränzle and N. Markey. Cham: Springer International Publishing, 2016, pp. 35–50.

[21] C. Gerking and D. Schubert. "Component-based refinement and verification of information-flow security policies for cyber-physical microservice architectures". In: *2019 IEEE International Conference on Software Architecture (ICSA).* IEEE. 2019, pp. 61–70.

[22] B. A. Hamilton. *When the lights went out: A Comprehensive Review of the 2015 Attacks on Ukrainian Critical Infrastructure.* Tech. rep. 2016. URL: `https://www.boozallen.com/content/dam/boozallen/documents/2016/09/ukraine-report-when-the-lights-went-out.pdf` (visited on 04/22/2022).

[23] R. Heinrich et al. "Architecture-based change impact analysis in cross-disciplinary automated production systems". In: *Journal of Systems and Software* 146 (2018), pp. 167–185.

[24] R. Heinrich et al. "Architecture-based change impact analysis in cross-disciplinary automated production systems". In: *Journal of Systems and Software* 146 (2018), pp. 167–185.

[25] C. T. Hu, D. F. Ferraiolo, and D. R. Kuhn. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations.* NIST, US, NIST SP, 2019. URL: `https://doi.org/10.6028/NIST.SP.800-162`.

[26] V. C. Hu et al. "Guide to attribute based access control (abac) definition and considerations (draft)". In: *NIST special publication* 800.162 (2013), pp. 1–54.

[27] J. Jürjens. "UMLsec: Extending UML for Secure Systems Development". In: *UML 2002 — The Unified Modeling Language.* Ed. by J.-M. Jézéquel, H. Hussmann, and S. Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 412–425. ISBN: 978-3-540-45800-5.

[28] A. A. E. Kalam et al. "Organization based access control". In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks.* June 2003, pp. 120–131. DOI: `10.1109/POLICY.2003.1206966`.

[29] K. Katkalov et al. "Model-driven development of information flow-secure systems with IFlow". In: *2013 International Conference on Social Computing.* IEEE. 2013, pp. 51–56.

[30] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer. "DAG-based attack and defense modeling: Don't miss the forest for the attack trees". In: *Computer Science Review* 13-14 (2014), pp. 1–38. ISSN: 1574-0137. DOI: `https://doi.org/10.1016/j.cosrev.2014.07.001`. URL: `https://www.sciencedirect.com/science/article/pii/S1574013714000100`.

[31] M. E. Kramer et al. "Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems". In: (2017).

[32] T. Lodderstedt, D. Basin, and J. Doser. "SecureUML: A UML-Based Modeling Language for Model-Driven Security". In: *UML 2002 — The Unified Modeling Language.* Ed. by J.-M. Jézéquel, H. Hussmann, and S. Cook. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 426–441. ISBN: 978-3-540-45800-5.

[33] P. K. Manadhata and J. M. Wing. "An attack surface metric". In: *IEEE Transactions on Software Engineering* 37.3 (2010), pp. 371–386.

[34] S. Mauw and M. Oostdijk. "Foundations of Attack Trees". In: *Information Security and Cryptology - ICISC 2005.* Ed. by D. H. Won and S. Kim. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 186–198.

[35] P. Nguyen et al. "An extensive systematic review on the Model-Driven Development of secure systems". In: *Information and Software Technology* 68 (Dec. 2015), pp. 62–81. ISSN: 09505849. DOI: `10.1016/j.infsof.2015.08.006`.

[36] *NVD.* URL: `https://nvd.nist.gov/vuln` (visited on 04/22/2022).

[37] *OWASP.* URL: `https://owasp.org/www-project-top-ten/` (visited on 04/22/2022).

[38] M. Plachkinova and C. Maurer. "Security breach at target". In: *Journal of Information Systems Education* 29.1 (2018), pp. 11–20.

[39] N. Polatidis et al. "From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks". In: *Evolving Systems* 11.3 (2020), pp. 479–490.

[40] R. H. Reussner et al. *Modeling and Simulating Software Architectures: The Palladio Approach.* MIT Press, Oct. 2016. ISBN: 9780262034760.

[41] K. Rostami et al. "Architecture-based change impact analysis in information systems and business processes". In: *2017 IEEE International Conference on Software Architecture (ICSA).* IEEE. 2017, pp. 179–188.

[42] K. Rostami et al. "Architecture-based change impact analysis in information systems and business processes". In: *2017 IEEE International Conference on Software Architecture (ICSA).* IEEE. 2017, pp. 179–188.

[43] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical software engineering* 14.2 (2009), pp. 131–164.

[44] B. Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[45] S. Seifermann, R. Heinrich, and R. Reussner. "Data-driven software architecture for analyzing confidentiality". In: *2019 IEEE International Conference on Software Architecture (ICSA).* IEEE. 2019, pp. 1–10.

[46] E. I. Sharing and A. C. (E-ISAC). *Analysis of the cyber attack on the Ukrainian power grid.* Tech. rep. 2016, pp. 1–23.

[47] X. Shu et al. "Breaking the target: An analysis of target data breach and lessons learned". In: *arXiv preprint arXiv:1701.04940* (2017).

[48] T. Sommestad, M. Ekstedt, and H. Holm. "The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures". In: *IEEE Systems Journal* 7.3 (2012), pp. 363–373.

[49] O. Standard. *eXtensible Access Control Markup Language (XACML) Version 3.0.* Jan. 2013. URL: `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[50] K. M. Ting. "Precision and recall". In: *Encyclopedia of machine learning.* Springer, 2011, pp. 781–781. ISBN: 978-0-387-30768-8.

[51] K. Tuma, R. Scandariato, and M. Balliu. "Flaws in flows: Unveiling design flaws via information flow analysis". In: *2019 IEEE International Conference on Software Architecture (ICSA).* IEEE. 2019, pp. 191–200.

[52] A. Van den Berghe et al. "Design notations for secure software: a systematic literature review". In: *Software & Systems Modeling* 16.3 (2017), pp. 809–831.

[53] S. D. C. d. Vimercati. "Discretionary Access Control Policies (DAC)". In: *Encyclopedia of Cryptography and Security.* Ed. by H. C. A. van Tilborg and S. Jajodia. Boston, MA: Springer US, 2011, pp. 356–358. ISBN: 978-1-4419-5906-5. DOI: `10.1007/978-1-4419-5906-5_817`. URL: `https://doi.org/10.1007/978-1-4419-5906-5_817`.

[54] M. Walter. *Context Confidentiality Metamodel.* 2021. URL: `https://github.com/FluidTrust/Palladio-Addons-ContextConfidentiality-Metamodel`.

[55] M. Walter, R. Heinrich, and R. Reussner. “Architectural Attack Propagation Analysis for Identifying Confidentiality Issues”. In: *IEEE International Conference on Software Architecture*. ICSA 2022. accepted, to appear. IEEE, 2022.

[56] B. für Wirtschaft und Energie (BMWi). *Zugriffssteuerung für Industrie 4.0-Komponenten zur Anwendung von Herstellern, Betreibern und Integratoren*. Diskussionspapier. Nov. 2018. URL: `https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/zugriffssteuerung-industrie40-komponenten.html`.

[57] B. Yuan et al. “An Attack Path Generation Methods Based on Graph Database”. In: *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*. Vol. 1. IEEE. 2020, pp. 1905–1910.