A Mobility Case Study for Validating Attack Propagation Analyses

Bachelor's Thesis of

Yakup Ensar Evli

at the Department of Informatics
Institute for Information Security and Dependability

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

27th June 2022 – 27th October 2022

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Yakup Ensar Evli)

Abstract

The “Architectural Attack Propagation Analysis for Identifying Confidentialty Issues” proposed by Walter et al. [25] considers vulnerability analysis in software architecture. The analysis is using access control policies together with the vulnerabilities and their combinations to propagate through the system. This phenomenon has to be investigated thoroughly in a real-life context to be able to make conclusions about metrics, e.g. accuracy. However, a concrete approach to achieve the investigation of Attack Propagation Analyses in a real-life context is missing. This work aims to fill this gap with “A Mobility Case Study for Validating Attack Propagation Analyses”. In order to achieve validity, conventional properties of case studies in software engineering were identified. Afterward, the end result, in form of a software model, was reviewed according to these properties. This review has revealed that all properties were fulfilled, however not in the highest degree of fulfillment. A discussion about this is held in this paper.

Zusammenfassung

Die “Architectural Attack Propagation Analysis for Identifying Confidentialty Issues” von Walter et al. [25] befasst sich mit der Analyse von Schwachstellen in der Softwarearchitektur. Die Analyse verwendet Zugriffskontrollrichtlinien zusammen mit den Schwachstellen und deren Kombinationen, um sich im System auszubreiten. Diese Analyse muss in einem Echtwelt Kontext untersucht werden, um Rückschlüsse auf Metriken, z. B. die Genauigkeit ziehen zu können. Es fehlt jedoch ein konkreter Ansatz zur Untersuchung von Angreiferausbreitungsanalysen in einem Echtwelt Kontext. Diese Arbeit zielt darauf ab, diese Lücke mit einer “Fallstudie zur Angreiferausbreitungsanalyse in Mobilitätssystemen” zu schließen. Um Validität zu erreichen, wurden konventionelle Eigenschaften von Fallstudien in der Software-Entwicklung identifiziert. Anschließend wurde das Endergebnis in Form eines Softwaremodells auf diese Eigenschaften hin überprüft. Die Überprüfung hat ergeben, dass alle Eigenschaften erfüllt worden sind, jedoch nicht im höchsten Grad. Eine Diskussion darüber wird in dieser Arbeit geführt.

Contents

List of Figures

List of Tables

1 Motivation

The Architectural Attack Propagation Analysis [25] proposed by Walter et al. considers vulnerability analysis in software architecture. With this analysis, software architects are enabled to determine confidentiality violations as early as in the design process of the software in development. However, the practical use has yet to be investigated thoroughly, first and foremost in a real-life context. A reliable method to connect theory and practice in scientific fields is case studies. A case study in software engineering is defined by Runeson et al. as "an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified" [21]. As the boundary between theory and practical use of the analysis cannot be determined, a bachelor thesis shall be proposed to conduct a case study in the real-life context of mobility systems for attacker propagation analyses. Although case studies for the analysis have been done, it is important to carry out case studies in many different fields to generate a more in-depth insight. Mobility systems can provide a wide variety of entities, e.g. providers for mobility systems, mobility devices in use, and users themselves. Thus, resulting in an adequate environment to research the boundary between theory and practical use. Every entity can be a source of data hence insight, regarding the analysis. Further, ethical considerations for case studies are important in terms of consent, especially when conducting a case study that exploits vulnerability issues. Since case studies demand a real-life context, it is challenging to find an organization that consents to this regarding that their system could be exploited. So, the open-source mobility system provided by the “Open Mobility Foundation” [14], that will be an example for this study, delivers a solution in terms of accessibility and consent because the information is unconditionally available and their tools are open-source. In addition, the required properties for case studies in software engineering have to be outlined. An example of one of these properties is “Flexibility”. These properties are extracted from various academic literature dealing with empirical software engineering and case study research in software engineering. One of the most important sources is provided by the work of Runeson et al. in “Guidelines for conducting and reporting case study research in software engineering”. Afterward, the result has to be evaluated according to these required properties. In the following, the important foundations for the proposed thesis will be given in Chapter 2. Further, Chapter 3 gives an overview of the case study itself, the properties required for the case study, and the design decisions for the upcoming case study, including key goals. In Chapter 4 models of the resulting case study are presented, design decisions are explained and expected results are outlined. Additionally, Chapter 5 will give an overview of lessons learned during the work for readers that may profit from such. It will also include threats to the validity and limitations of the case study. With regard to giving a broader picture of the work, Chapter 6 includes related work about the topic in general and earlier studies. Finally,

a conclusion in Chapter 7 will give a summary of the achievements in this work as well as a discussion about the future of the case study.

2 Foundations

Before being able to conduct the case study some important topics, which I will refer to as foundations, have to be researched and understood. This chapter will give a brief summary of the foundations that are needed to carry out and understand this case study. These foundations include the Palladio Component Model that is used, the Architectural Attack Propagation Analysis that is the context and Mobility Systems which are the domain of the case study.

2.1 Palladio Component Model

The Palladio Component Model (PCM) [20] is an approach to model, simulate and analyse the performance of software architecture. For this, PCM offers its own framework and an own Architecture Description Language (ADL). As the name suggests, PCM supports the component-based development process. In this process, PCM provides a *component repository*. In this repository, developers can describe the components themselves, the interfaces of components, the component behaviour and required resources for external calls. In our case, I will be using *BasicComponent*s since no compositions or subsystems are needed. Components can then require or provide interfaces. *ServiceEffectSpecifications* can be defined in PCM to specify the performance of the components hence describing resource demand. Moreover, data types can also be defined in the repository. Components in the component repository can then be composed into a system in the *system model*. The system model in PCM includes different components that are instantiated and connected to each other through the provided and required interfaces. It also makes it possible to specify which interfaces are provided or required outwards the system. PCM further allows hardware modelling in the *resource environment*. Here, resource containers can be defined that are used to run the components but also linking resources which are needed to connect these resource containers. Next, the *allocation context* defines which allocations of the components should be placed in which resource container. Lastly, PCM can model user behaviour in the *usage model* which is not relevant to the scope of this work. To sum it up, the component-based development process will take an important part in the architectural modelling for this case study.

2.2 Architectural Attack Propagation Analysis

In terms of Architectural Attack Propagation Analysis, this work is setting the focus on the Architectural Attack Propagation Analysis proposed by Walter et al. [25]. It simulates an attacker that is propagating through the system and is using vulnerabilities in the system to propagate further. The analysis can also take place during design time. This was made possible through an extended PCM. In the analysis, attack paths are extracted and further analysed

concerning the combination of different vulnerabilities. If not detected, these combinations can allow attackers to propagate through the system by exploiting vulnerabilities thus exploiting access control policies. Moreover, Walter et al. state that "the increase in connected elements also adds new vulnerabilities and allows additional attack vectors" [25] stressing the importance of analysing the dependency between access control policies and vulnerabilities. What makes this approach unique is "a fine-grained access control model together with a vulnerability modelling based on commonly used attack classifications" [25]. The analysis extends PCM by access control models, attacker and vulnerability specifications. The access control model that is used is the Attribute Based Access Control (ABAC) [24]. ABAC uses different attributes that are taken into consideration for granting access to users. Subject attributes represent attributes which request the resource in the model. Environment attributes formalize properties like time. Requested Operation Attributes can be operations like read, write or calls in an architecture. Finally, an example of Requested Resource Attributes are filenames or in context: Components. The attributes are transformed in conjunctions of boolean expressions, building a policy and granting access whenever the expression is fulfilled. The attacker propagation of the analysis is built upon the Karlsruhe Architectural Maintainability Prediction (KAMP) [19]. KAMP is an approach to evaluate the maintainability of software architecture. In KAMP propagation was also considered since architectural changes can lead to follow-up changes through references and uses-relations. KAMP uses kinds of relations to systematically detect affected components during the propagation process.

2.3 Mobility Systems

The context of this case study is Mobility Systems that support providers of Mobility as a Service (MaaS) as well as cities in which these providers operate. MaaS "is a service concept that integrates public transport with other mobility services, such as car sharing, ride sourcing, and bicycle sharing" [6]. Mobility services from public and/or private mobility providers are managed through a single instance, the mobility system in use. Users are being offered mobility solutions, which they can use based on their needs. An important aspect of MaaS is the environment in which they are used, for example, cities. Cities can have different requirements and policies for such providers considering traffic, safety for residents, taxes, etc. In this work, mobility systems can be seen as mediators between MaaS providers and their operational areas. The demand for MaaS is increasing such that the worldwide market volume for the mobility as a service market is forecasted to be 372,1 billion US-Dollars in the year 2026 [27]. As a comparison, the market volume in the year 2018 was 42,3 billion US-Dollars [27]. Widely-known examples, especially in the U.S., of these mobility systems in taxi contexts are Uber [23] and Lyft [5].

2.3.1 Open Mobility Foundation

The Open Mobility Foundation is a non-profit organization that supports the development of open-source standards and tools that provide scalable mobility solutions for cities [14]. One of their open-source tools is the Mobility Data Specification (MDS) [7]. MDS is a tool that was developed to support cities to better manage transportation. At the core, MDS manages communication and data-sharing between cities and private mobility providers, thus

standardizing the communication. Examples of private mobility providers are e-scooter or bike-share companies. The advantages of this standardization are on one hand that cities can share and validate policies through MDS. On the other hand, mobility service providers are given a framework that they can easily implement and reuse in new markets.

2.3.2 Mobility Data Specification Architecture

In this section, a reference architecture of mobility systems based on the MDS of the Open Mobility Foundation is outlined. MDS provides Application Programming Interfaces (APIs) enabling standardized two-way communication. Firstly, a communication channel for cities that collect data through the API and further publish regulations. Regulations can affect traffic management and can include public policy decisions. Secondly, a communication channel for private companies that can share information about their operations, f.ex. availability status of their mobility solution, but implement the regulations. Three primary APIs provided by MDS are the following as stated by MDS [7]:

- Provider: "Provider allows private mobility companies to report data to cities on the number, location, status, and ride history of devices in use" [7]. The Provider API is hosted by a provider and allows an agency to pull data from it.
- Agency: "Agency allows real-time updates and collaboration between city officials and providers when complex city transportation problems demand dynamic solutions" [7]. The Agency API is hosted by an agency and allows a provider to push data to it.
- Policy: "Policy allows cities to set rules for how and where different vehicles can operate, how many can operate, and other high-level policy initiatives" [7]. The Policy API is also hosted by an agency and allows a provider to pull policy information from the agency.

A city using the MDS standard can adopt one or both of these APIs. More detailed information on the APIs can be found in the documentation about "Understanding MDS APIs" [11]. An example architecture is presented in Section 3.2.2. MDS does not contain and obtain any information about users of the mobility solution. Private information about users like name, contact information, payment information and history are in the charge of the company providing the mobility solution. However, MDS does collect information about the mobility solution devices for example the cars in use. In detail, MDS collects vehicle and trip data from providers, which is forwarded by the mobility solution company through a secure channel. After, this data is sent to cities in the same way.

3 Overview

Case studies are conducted in various fields. They provide an in-depth study of a phenomenon in a real-life environment. Thus, case studies can lead to findings for phenomena that are not easy to separate from their environments. Insightful findings for propagation analyses cannot be generated without a real-life context, since it is used in security-relevant contexts. Runeson et al. define a case study in software engineering as "an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified" [21]. This case study has the aim to be positivist. A positivist case study searches evidence for formal propositions, measures variables, tests hypotheses and draws inferences from a sample to a stated population [21]. Moreover, the case study will follow the three characteristics of case studies proposed by Runeson et al. in [21] (Qualitative, Exploratory, and Flexible). As explained in Chapter 1 the case study is intended to deliver insights about the Architectural Attack Propagation Analysis for Identifying Confidentiality Issues by Walter et al. [25] in the domain of mobility systems. Specifically, in the environment of the Mobility Data Specification by the Open Mobility Foundation (OMF) [14]. During the case study, an architectural model is designed. The Attack Propagation Analysis is used on this model and the case study will, later on, be analyzed regarding the properties.

3.1 Case Study Properties

There are some properties a case study should fulfill and some that will determine the success. By success, I refer to the insight and knowledge generated by the case study. The properties will be used to evaluate each version of the case study. In the following, these properties will first be explained. After, methods to measure the fulfillment of the properties will be defined by us, as well as the graduation of fulfillment. Three main characteristics proposed by Runeson et al. [21] that case studies should fulfill are the following:

- Qualitative: A primarily qualitative approach to the subject will be helpful in achieving the required results. Per se, this property is not measurable.
- Exploratory: the case study aims to be exploratory, thus resulting in new insights to the analysis and in the best case generating ideas and hypotheses for further research. In order to measure this property, I will break it down into three grades which reflect the degree of fulfillment of this property.
    1. Analysis delivers results. The case study is feasible.
    2. Analysis delivers results in the domain of mobility systems.

3. Analysis delivers results in the domain of mobility systems and expected results are documented in such a way that conclusions can be made (e.g. accuracy).

- Flexibility: The research subject of mobility systems and solutions is a real-world phenomenon and features a complex and dynamic characteristic. This raises the need for a flexible design of the case study as given by a qualitative approach. In our case, modeling properties of mobility systems that are not relevant will be treated as risks for the flexibility of the case study. Non-relevant properties in this context are defined as properties that go beyond the three core APIs in MDS since they are not needed for the core functionality of mobility systems. Further, non-relevant properties include beta functionality as they expose a risk of change if they are modified or are not continued. This property will be broken down into two grades that should reflect the degree of fulfillment of this property.
  1. Model does not include non-relevant functionality.
  2. Model is extendable. This will be measured through a comparison of changes with a version after the measured version. For example, the first version will be measured by changes in elements in the second version.

Further characteristics that this case study set as goals, besides the properties before, to achieve are:

- (Data-)Triangulation: Runeson et al. list Triangulation as one of the main characteristics of case studies. In particular, Data Triangulation is defined as "using more than one data source or collecting the same data on different occasions." [21]. As a consequence, (Data-)Triangulation will take a part in our case study in form of different viewpoints and aspects of the subject of mobility systems and solutions. For measurement purposes, I divided this property into three degrees of fulfillment
  1. Model includes the viewpoint of one of the three APIs (Agency or Policy or Provider).
  2. Model includes the viewpoint of two of the three APIs.
  3. Model includes all viewpoints of the three APIs.
- Real-World Context: I want to embed the case study in a real-world context to show the behavior of the analyzed method in the real world. Since real-world scenarios come with challenges, restrictions, and boundaries, a closer connection to the real world is more likely to conclude in meaningful results. This property also includes non-triviality. In a real-world context, the case study cannot be tailored such that the results reflect the expectations of the researcher and will likely reflect a complex issue. By outlining characteristics of case studies, Runeson et. al also describes that case studies are "coping with the complex and dynamic characteristics of real-world phenomena" [21]. As I want to have a metric for this property, I defined the following degrees of fulfillment.
  1. Idea of a real-world system is represented.
  2. Idea of a real-world system is represented and concepts are modeled so that they could probably occur in the real world.
  3. A simplified real-world system is modeled.

4. A real-world system is modeled.

- Reproducibility: As already brought forward by Runeson et al. "conclusions [of case studies] are based on a clear chain of evidence, whether qualitative or quantitative, collected from multiple sources in a planned and consistent manner" [21]. It should be possible to reproduce the case study so that other people can reconstruct the case study. A case study that is not reproducible would only offer unprovable claims for external researchers. An important part of reproducibility is also comprehensibility. In the end, I want to present a clear chain of evidence of our conclusions and ideally add to existing knowledge about the Architectural Attack Propagation Analysis for Identifying Confidentiality Issues [25]. Thus, a comprehensible case study including comprehensible proceedings and results is unavoidable. It should be possible for other people to understand what and why something happens in the case study, thus making it accessible for modifications and extensions. I transfer the Artifact Review and Badging Policy of the Association for Computing Machinery [1] to our criteria and evaluate them based on the following degrees.
  1. All models of the case study are included.
  2. Artifacts are relevant to the subject of this work and contribute to the generation of the main result.
  3. Documentation for available artifacts is available. This contributes to Reproducibility as well as Comprehensibility.
  4. Models can be used, executed with the analysis by Walter et al. [25] and modified. The model is proven to produce results and is feasible.
  5. Case study is publicly available
  6. Case study has been validated by another person except for the author.
- Vulnerability: Since the case study will be used for "Validating Attack Propagation Analyses" the model needs vulnerabilities, resulting in attack paths that can be found through the analysis. This property was defined by us specifically for this case study and it is divided into four degrees of fulfillment for measurement purposes.
  1. Model includes vulnerabilities. There are vulnerabilities in the system, but these cannot be assigned to components or architectural elements and accuracy cannot be calculated.
  2. Model includes vulnerabilities such that accuracy can be calculated. Meaning that vulnerabilities are security relevant and can lead to attack paths.
  3. Model includes vulnerabilities such that accuracy can be calculated and vulnerabilities appear in the real-world.
  4. Model includes vulnerabilities such that accuracy can be calculated and vulnerabilities appear in the domain of mobility systems.
- Accuracy calculability: It should be possible for Attack Propagation Analyses to measure their accuracy through the case study. Further, it should be possible for them to measure

their accuracy when performing the analysis in different domains. As an example, the accuracy could be the ratio between found attack paths by the analysis and the number of attack paths that were modeled. This property was defined by us specifically for this case study and is measured by the following metric.

1. There are attack paths and expected results of attacker propagation analyses available.

3.2 Case Study Design

The superior objective of the case study is to determine the viability of the Architectural Attack Propagation Analysis for Identifying Confidentiality Issues [25]. As discussed earlier, I will be using the Open Mobility Foundation and their MDS as our case. The whole theory will be based on the frame of reference given by the Architectural Attack Propagation Analysis. Method collection is done in the following steps. Firstly, I have to develop an architectural model representing mobility systems, in our case the MDS. Since the Architectural Attack Propagation Analysis is based on the framework of the Palladio Component Model (PCM) [20], the architectural model has to be developed in PCM. In the development, I face questions that have to be researched and for which suitable answers have to be found:

- Which level of abstraction will the architectural models have? Meaning, which parts of the architectural model will be granular and which parts of the model should only deliver a rough picture of the real-world system.
- Which functionalities of the real-world example should be covered?

After finishing the architectural model in PCM, I will outline the expected results of the Attack Propagation Analysis. After, the case study will be evaluated on the required properties. The whole case study will be built in an iterative process. I will define a scope for the iteration, design a model covering functionalities in this frame, analyze the results of the propagation analysis and evaluate the iteration on the evaluation criteria. In the next iteration, a model will be designed for another scope and analyzed in the end. In Figure 3.1 you can see an example of this iterative process. In the first iteration, there is only one scenario. This is the scope which was defined. It consists of only one module. In the second iteration, another scope, here scenario, is defined. Thus the model could be extended by another module. Further, it could be possible that there are different versions of an iteration which you can see in the third iteration. It is planned to do iterations by time described by the timeline “Time t”.

In the following, input and output artifacts for the case study will be worked out and the implementation process of the case study will be explained.

3.2.1 Inputs

Input artifacts that are needed for the case study are the following:

- Analysed method: In our case, this will be the Architectural Attack Propagation Analysis [25].

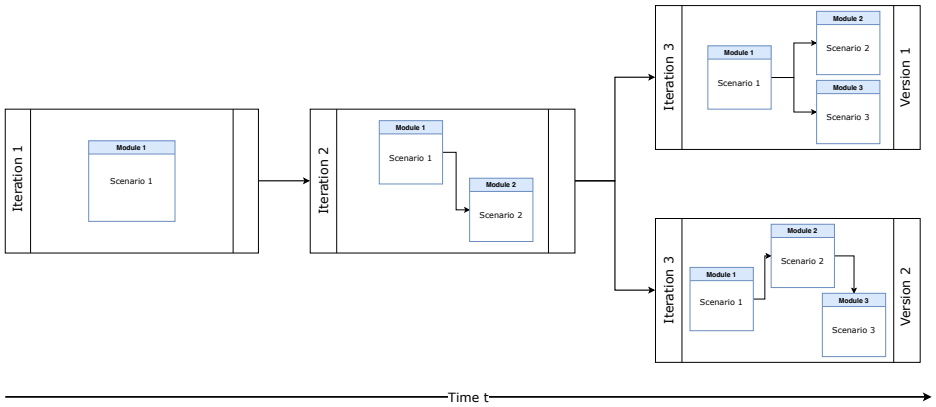

Figure 3.1: Example of the iterative process in case study building

- Mobility system: For a viable case study information about its context is inevitable. I will focus the research on the Open Mobility Foundation.
- Architectural model: Since the analyzed method is generating results on architectural models, I will need a model as an input. This model is not available beforehand, it will be modeled iteratively during the case study. Specifically, the MDS by the Open Mobility Foundation will act as a reference for the architectural modeling.
- Expected results: Before applying the Attack Propagation Analysis I have to define a set of expected results that the Attack Propagation should deliver. This will be done for each iteration.

3.2.2 Implementation Process

The following steps must be carried out during the case study to obtain results from the analysis:

- Model software architecture: In this step, an architectural model of the system to be analyzed has to be modeled in PCM. In Figure 3.2 you can see what an example architecture of MDS could look like before the first iteration. There are three interfaces, Provider, Agency and Policy, that represent the three APIs of MDS. Every interface has its functionality as methods. The provider interface has methods to create new devices, update device location and statuses, is able to get updates, and send updates to the agency. The agency interface allows updating of providers or policy, whereas the policy can define new rules, update rules, get updates or send updates to the agency. Every interface has components that implement these interfaces.
- Create attacker model: Through the attacker model of the Architectural Attack Propagation Analysis types of attacks have to be specified that should be modeled. So, the attack types to be analyzed have to be taken into consideration during the case study. A helpful resource with existing attack types is the Open Web Application Security Project (OWASP) [15]. OWASP lists security risks, including OWASP10 with the top ten web application security risks such as A01 (Broken Access Control) [16]. A01 describes a

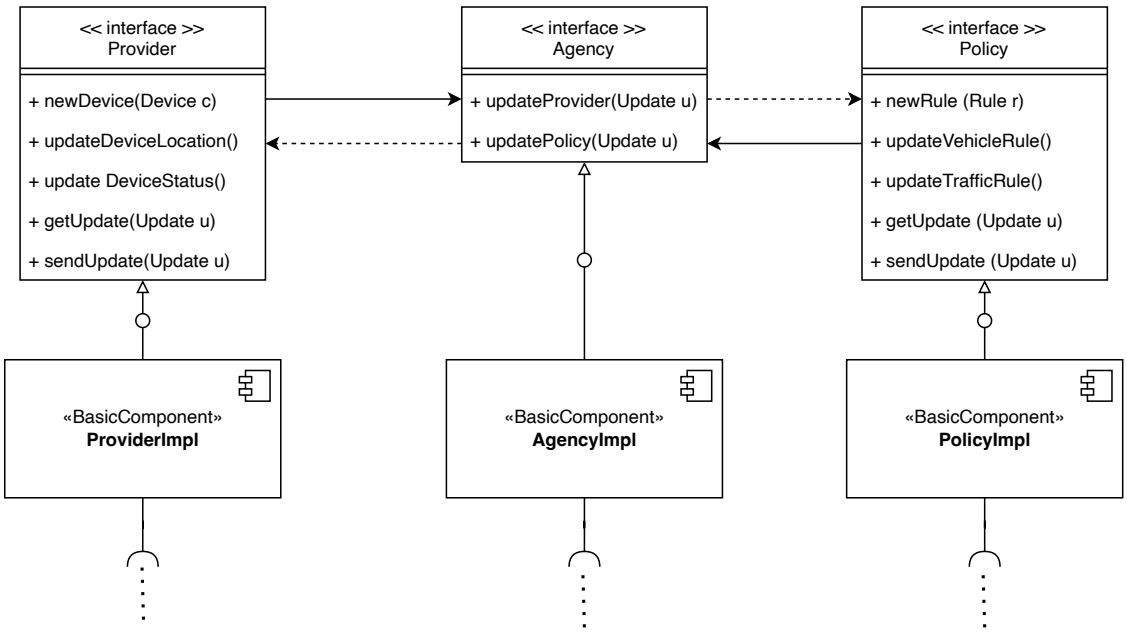

Figure 3.2: Example architecture model for mobility systems

failure in access control such that users can act outside of their intended permissions leading to e.g. unauthorized information disclosure.

- Create Vulnerability Model: In this step, vulnerabilities of the architectural elements are determined. These vulnerabilities can be modeled manually or through vulnerability databases, e.g. National Vulnerability Database (NVD) [13].
- Create Access Control Model: Access Control Policies that are wished to be applied in the model can be specified in the access control policy model. To achieve this, roles have to be defined, for example, a provider role. After, it has to be specified which access rights these roles have. Finally, elements have to be specified that inherit these roles.
- Expected results: Results that are expected from an attack propagation analysis have to be documented using the vulnerabilities.

After these steps, the analysis can be run and obtained results can be documented.

3.2.3 Outputs

Output artifacts that are generated during the case study are the following:

- Documentation: During the Case Study I have to create a documentation for our models, f.ex. our architectural models. This is also needed to fulfill the property “Reproducibility” as discussed in Section 3.1.
- Architectural model: At the end of the case study, I will be left with architectural models that will represent a mobility system, at least partly, and are oriented around the MDS.

- Results of the analysis: After the analysis of our architectural model by the Attack Propagation Analysis, I will receive the results of the analysis.

4 Case Study Modelling

This chapter provides an overview of the case study models. As described in Section 3.2 I worked iteratively and the modeling process resulted in two iterations. The first iteration has one version, and the second iteration has three versions in total. For each version, the repository, resource environment, allocation, access control policy, and attacker models are described and explained. The system model is an instantiation of the components which are connected together. No further description is needed and explanation can be given. During the description of the models, explanations for design decisions are given. Further, the expected result for attacker propagation analyses is outlined. At the end of each version, the model is evaluated regarding the case study properties. The evaluation does also include a concrete result for an attacker propagation analysis using the analysis [25] by Walter et al.

4.1 Minimal Working Example (v1.0)

The aim of this iteration was to represent a minimal working example of the mobility system. It includes core functionalities of the Mobility Data Specification (MDS). The implementation consists of one provider and one agency, each with its own applications. The agency includes core functionalities of the Agency API and Policy API since both can be managed through the same user entity as specified in MDS. The agency and the provider are connected through a single-point network which allows them to send and receive data.

4.1.1 Repository

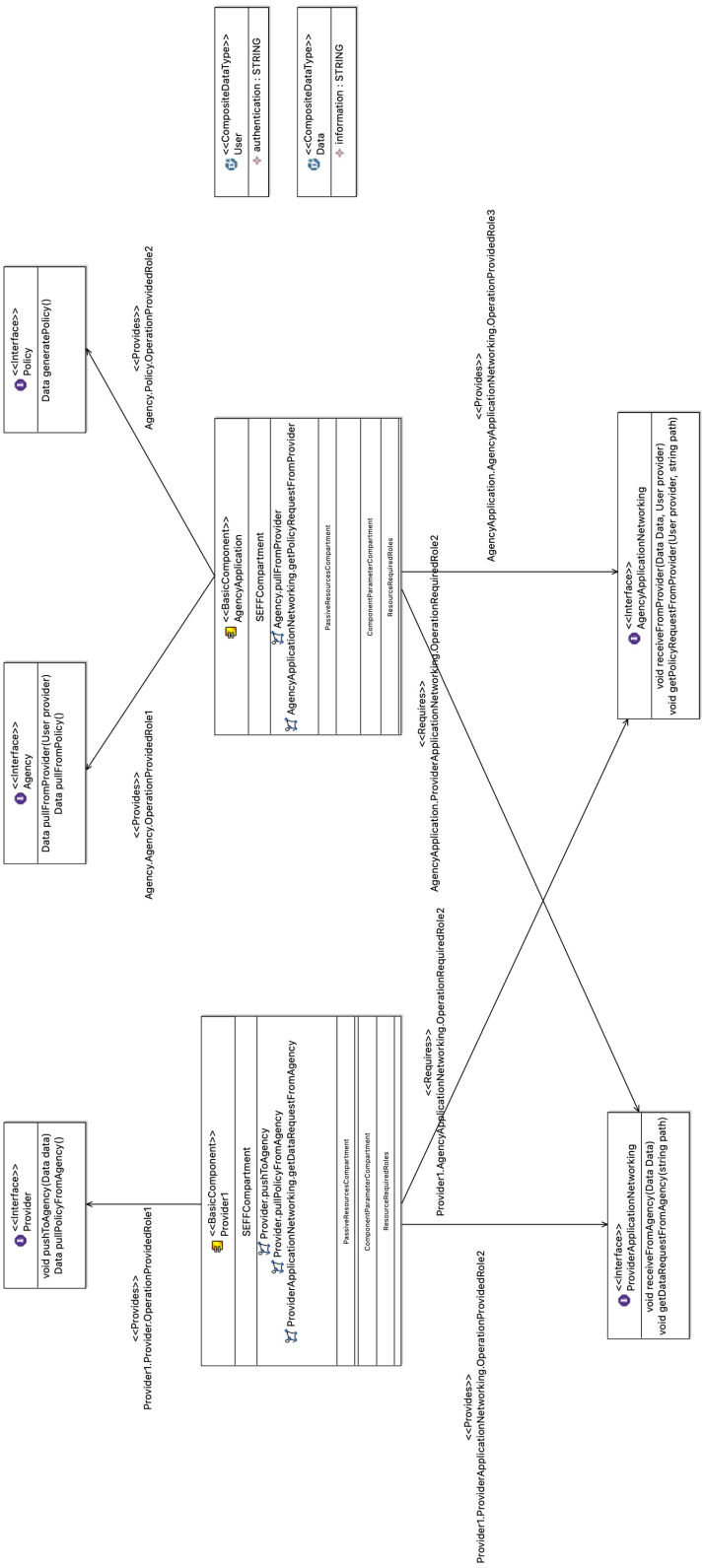

Figure 4.1: Iteration 1: Repository

A visual representation of the repository can be seen in Figure 4.1. In the repository, I defined two different datatypes. Firstly, `Data` is an abstraction for data that is being transferred over the network. Secondly, `User` represents data that is needed to identify users in the network, e.g. a JSON-Token (see [10] for further information). The Mobility Data Specification is defining in their documentation [11] that the Agency API is hosted by an agency and allows a provider to push data to it. The Provider API is hosted by a provider and allows an agency to pull data from it. According to MDS, a city using their standard can adopt one or both of these APIs [11]. As brought up, the Policy API is also hosted by an agency and allows a provider to pull policy information from the agency. As already stated, this iteration has one provider and one agency instance, modeling both APIs.
The provider was modeled in the repository through the `Provider Interface` and the `Provider1 Component`. The Provider Interface defines methods `pushToAgency(Data data):void` and `pullPolicyFromAgency():Data`. The first method should send data to the agency whereas the second method gets policy rules from an explicit agency. Using the assumption that a provider has at most one agency, a parameter to identify an agency is not needed for both methods. Provider1 is the basic component that implements the Provider Interface. Provider1 represents the main application used by a provider.
On the other hand, the agency was modeled in the repository through two interfaces and a single component. The `Agency Interface` represents agency functionalities, defining methods `pullFromProvider(User provider):Data` and `pullFromPolicy():Data`. The method `pullFromProvider` gets data from a provider, whereas `pullFromPolicy` gets policy data. Since the agency can easily have multiple providers, the providers are identified through a User type `provider` parameter. The `Policy Interface` represents policy functionalities, defining the method `generatePolicy():Data` which is a method for general purpose policy creation. `AgencyApplication Component` is the component that implements the provider and the policy interface. It represents the application used by the agency.
In order to represent a network data exchange two networking interfaces were introduced. The `ProviderApplicationNetworking Interface` is being provided and implemented by Provider1 but is required by AgencyApplication. It consists of two methods. The method `receiveFromAgency(Data data):void` determines what should happen when Data is received from the agency. On the other hand, the method `getDataRequestFromAgency(String path):void` determines what should happen when the agency sends a data request, for example, a get data request, to Provider1. The parameter `path` of type String directs to a specific GET endpoint, f.ex. /trips as required by the Provider API in MDS [9]. Next, the `AgencyApplicationNetworking Interface` is being provided and implemented by AgencyApplication but required by Provider1. Likewise the interface consists of the two methods: `receiveFromProvider(Data data, User provider):void` and `getPolicyRequestFromProvider(User user, String path):void`. Symmetrically, they define what should be done if data is received from the provider or the latter what should happen when the provider sends a data request to the agency.
To represent example activities, SEFFs for Provider1 and AgencyApplication were defined. Provider1 has three SEFFs:

1. `Provider.pushToAgency` has an external call on `AgencyApplicationNetworking.receiveFromProvider`.

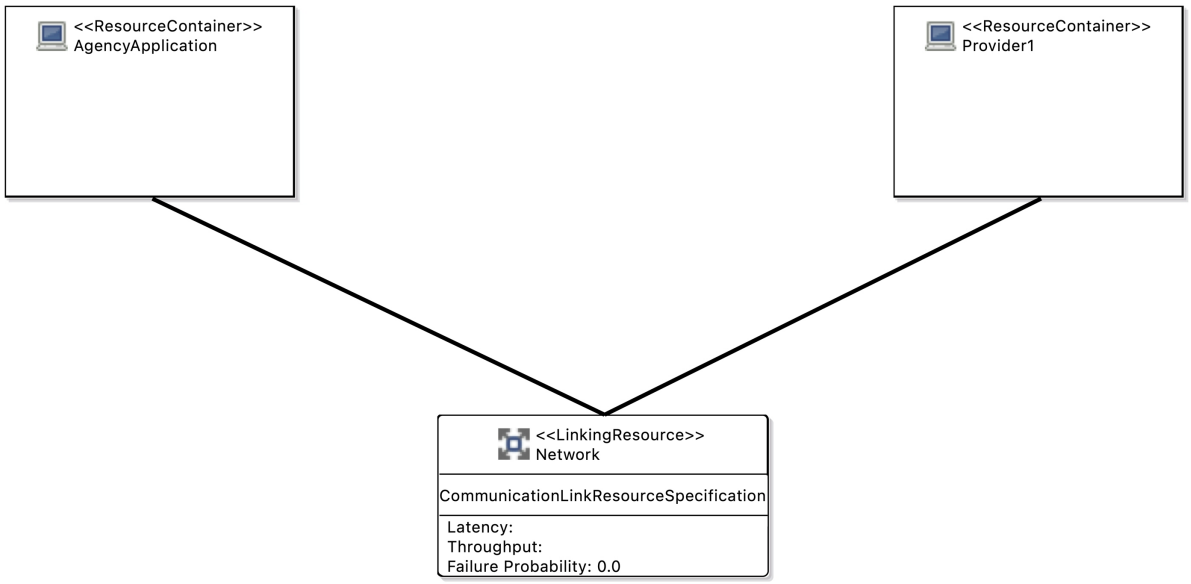

Figure 4.2: Iteration 1: Resource Environment

2. `Provider.pullPolicyFromAgency` has an external call on `AgencyApplicationNetworking-.getPolicyRequestFromProvider`.

3. `ProviderApplicatioNetworking.getDataRequestFromAgency` has an external call on `Agency-ApplicationNetworking.receiveFromProvider`.

AgencyApplication has two SEFFs:

1. `Agency.pullFromProvider` has an external call on `ProviderApplicationNetworking.get-DataRequestFromAgency`.

2. `AgencyApplicationNetworking.getPolicyRequestFromProvider` has an external call on `ProviderApplicatioNetworking.receiveFromAgency`.

4.1.2 Resource Environment

The resource environment of the first iterations is held minimal and Figure 4.2 is a visual representation. It consists of two resource containers and one linking resource. On the first resource container `Provider1`, the provider application will be running. The second resource container `AgencyApplication` will be used by the agency application. The linking resource `Network` connects both resource containers to represent a network between them.

4.1.3 Allocation

In the allocation model, I placed the `AgencyApplication Allocation` in the AgencyApplication resource container as can be seen in Figure 4.3. On the other hand, the `Provider1 Allocation` was placed in the Provider1 resource container.

Figure 4.3: Iteration 1: Allocation

4.1.4 Access Control Policies.

In our access control policies, I defined three roles. Firstly, `ProviderAdmin`, which represents the administrative user of Provider1. That's why I provided this role to the assembly context of Provider1 through a specification container. The role `AgencyAdmin` represents the administrative user of AgencyApplication and is provided to AgencyApplication for the same reason. Lastly, the management of the network is represented by a more abstract role of a `RestAPI`. This role was assigned to the linking resource `Network` because management has to be done in the network between the provider and the agency. In total, three access control policies were defined with these roles. An agency policy describes that `AgencyAdmin` has access to the AgencyApplication resource container because an administrative user should have access to all resources of an agency. A provider policy grants a `ProviderAdmin` access to the Provider1 resource container for the same reason. Finally, a RestAPI policy grants the RestAPI role access to Provider1 and AgencyApplication resource containers, so it can manage the data transfer between the two components and has access to the part of the model where the data is sent.

4.1.5 Attacker Model

Through a category specification the `CWE 284` [3] vulnerability was specified. The vulnerability represents an improper access control vulnerability that is mapped to A01 [16]. A01 has the first place in the OWASP10 [15] thus chances for the occurrence in a real-world setting are high. In a vulnerability container it is specified that `CWE 284` implies a takeover of the affected element. After this, `CWE 284` was placed in an attack container to represent an attack by the vulnerability. A system specification container was used to integrate the vulnerability into

the system by a vulnerability system integration. This integration was done by using a PCM element that holds the linking resource Network. The Network linking resource was chosen for this since access control is crucial for data transfer, for example in a Rest Server. In a container an attacker with the created attack container was specified. The attacker specification was done by using a PCM element that holds the assembly context of Provider1.

4.1.6 Expected result

It is expected that the Attacker can propagate to the linking resource Network without using any attacks, just by using the defined access control policies. However, by the modeled vulnerability in Network there should be a takeover of this linking resource from the attacker. The attacker should get control over the agency since Network has the role RestAPI which grants access to the provider and agency. Finally, the attacker should have control over the whole system.

4.1.7 Evaluation Regarding Properties

In this chapter, I evaluate this iteration in regard to the desired properties for our case study that are outlined in Section 3.1. In Table 4.1 an overview of the fulfilled and not fulfilled properties of this iteration is given.

Exploratory This iteration delivers a result for an attack propagation analysis in the domain of mobility systems that can be found in our published models [12]. Further, the expected results are documented in Section 4.1.6. Thus, the highest degree of the property Exploratory is fulfilled. It is expected that insights can already be extracted from this iteration for example a statement about the accuracy of an analysis.

Flexbility This iteration includes high-level models of a RestAPI, Provider, and an Agency such that further functionality or fine-granular modeling can be adapted easily. Moreover, non-relevant functionality or beta functionality from the current state of MDS was not modeled. That's why Flexibility is given in our system. Since, I defined that flexibility can be measured with the next iteration, a comparison with the versions in Section 4.2 shows that this iteration has been adopted without any (major) changes. Flexibility is fulfilled in the highest degree.

(Data-)Triangulation The property (Data-)Triangulation is not fulfilled. This iteration includes a very general and high-level approach. The possibility of analyzing different viewpoints is not given yet. To achieve this, further iterations will need to reflect the different viewpoints and their access control policies.

Real-World Context In the course of this iteration, the lowest degree of a Real-World Context is given. Iteration one reflects the core qualities and ideas of a mobility system. However, it is held too general to occur like this in the real world. For this, I need more granularity and a distinct coverage of the current MDS functionality in interfaces.

Fulfilled	Degree	Property	Explanation
✓	2/3	Exploratory	Analysis delivers results in the domain of mobility systems. Expected results are documented in such a way that conclusions can be made.
✓	2/2	Flexibility	Model does not include non relevant functionality. Model is extendable.
✗	0/3	(Data-)Triangulation	High-level model, no detailed viewpoint given.
✓	1/4	Real-World Context	Idea of a real-world system is represented.
✓	5/6	Reproducibility	Documentation available. Artifacts are relevant, contribute to the main result, and are available. The model is proven to produce results. Not validated yet.
✓	2/4	Vulnerability	Model includes vulnerabilities such that accuracy can be calculated. Vulnerabilities appear in the real world.
✓	1/1	Accuracy calculability	There are attack paths and expected results of attacker propagation analyses available.

Table 4.1: Iteration 1: Properties

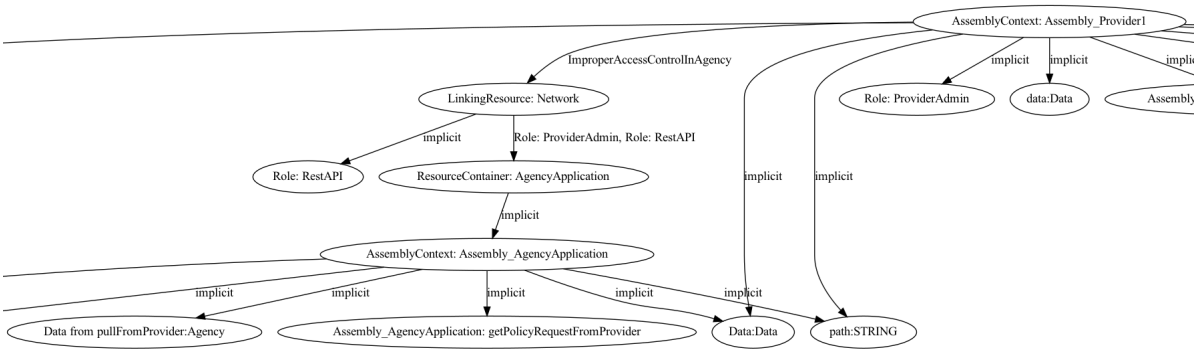

Figure 4.4: Iteration 1: Extract from Result

Reproducibility Documentation for the artifacts of this iteration was given in this chapter. The artifacts with all important models as well as their documentation are publicly available [12]. This iteration is used as a pillar for further iterations, thus they are relevant and contribute to the generations of the main results. As already stated, the model is proven to produce results and it is further feasible. As a result, the case study fulfills Reproducibility. However, the case study has to be validated by another person to reach the highest degree of Reproducibility.

Vulnerability The model includes CWE 284 [3] as a vulnerability. This vulnerability appears in the real world and accuracy can be calculated. However, it has to be further researched if this vulnerability occurs in the context of mobility systems. Vulnerability is fulfilled in the third degree.

Accuracy Calculability There are attack paths in this iteration and expected results of attacker propagation analyses are available in Section 4.1.6. Accuracy Calculability is fulfilled.

4.1.7.1 Result

The returned result was exactly the same result that was expected. The analysis was correct. In Figure 4.4 you can see an extract of the attacker propagation graph. The attacker uses the vulnerability, namely `ImproperAccessControlInAgency`, to take over the network linking resource. By using this vulnerability, the attacker gains the role RestAPI with which it has implicit access to the agency and all its elements.

4.2 Detailed Modelling of Entities (v2.x)

As the first iteration was a minimal working example, this iteration is aiming to provide a detailed perspective for each entity (Policy, Agency and Provider). This detailed view allows a better understanding of the structure of the Mobility Data Specification (MDS). It has three versions in total for each entity so that they can be focused on separately, allowing one to analyse access control policies and vulnerabilities without regard to other entities. Further, it was in our interest to analyse the behaviour of the analyses for the three entities given by the API considering the (Data-)Triangulation property (see Section 3.1). The second iteration builds upon the first iteration and extends the model for each entity. That's why, differences and extensions, in comparison to the first iteration, are described for each model. The models include all required API features for the MDS and access control policies were extended accordingly. The attacker model was only changed for the versions concerning the provider entity to observe the behaviour of the analysis, and whether it would propagate correctly. The attacker model concerning the agency and policy versions did not change since the attacker model from the first iteration already propagates through the whole system. No further SEFFs were implemented in this iteration since they were not crucial for the analysis yet.

4.2.1 Provider Entity (v2.1)

The first version of the second iteration includes a detailed model of the provider entity and its view of the Mobility Data Specification. This version of the second iteration extends the provider view with functionality and a database.

4.2.1.1 Repository

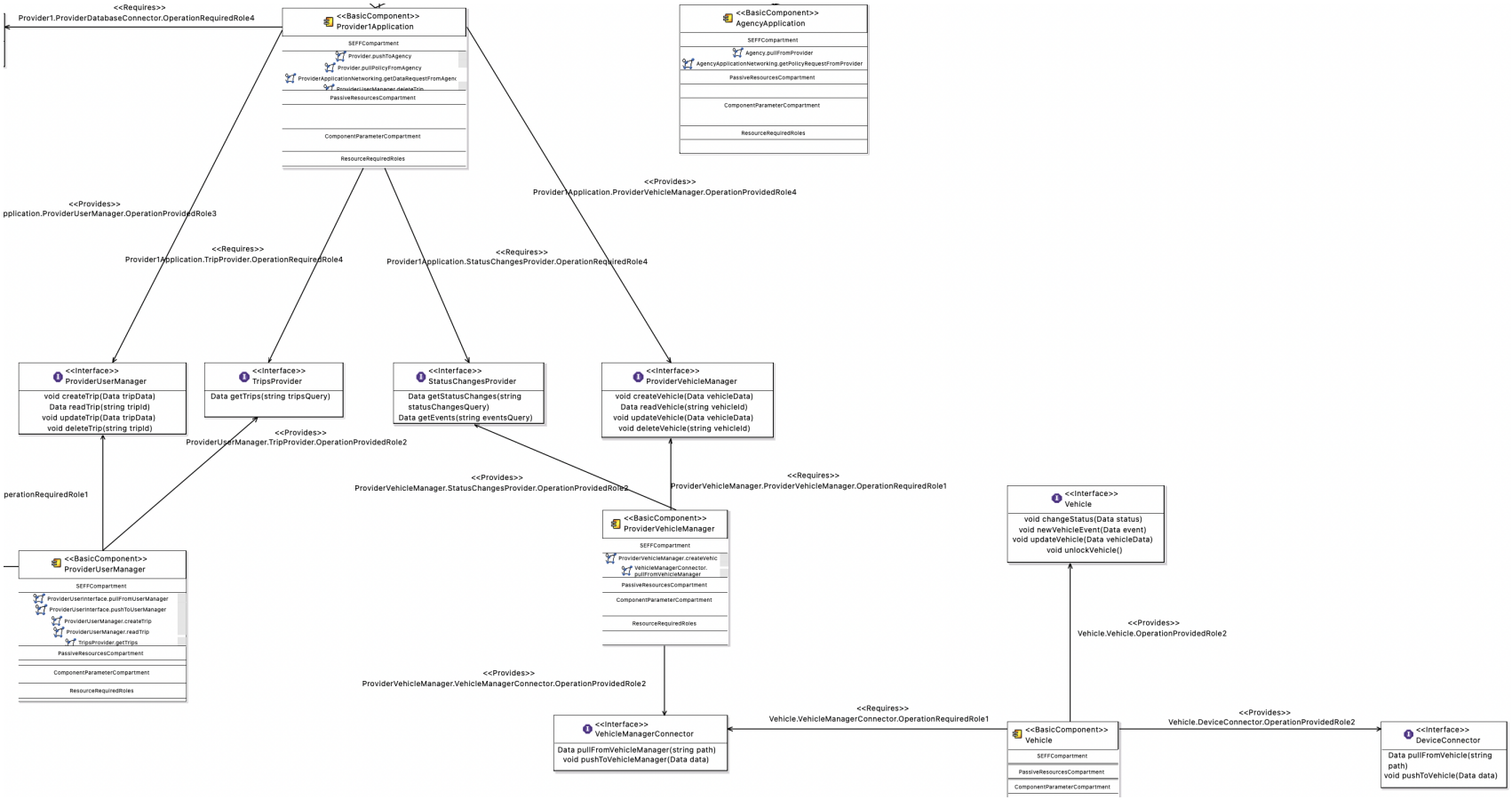

Figure 4.5: Iteration 2.1: Extract from Repository

An extract from the visual representation of the repository can be seen in Figure 4.5. The `Provider1 Component` was renamed to `Provider1Application` so that it is evident that the Provider1Application is the main application of the provider. Provider1Application was extended by two provided interfaces: `ProviderUserManager Interface` and `ProviderVehicleManager Interface`. In addition, the Provider1Application further requires three interfaces: `ProviderDatabaseConnector Interface`, `TripsProvider Interface` and `StatusChangesProvider Interface`. Firstly, since the provider is a mobility as a service provider, it has users of the service who have to be managed as outlined in Section 2.3. In order to fulfill this function, the `ProviderUserManager Component` was introduced to represent the management of interactions between users of the provider and the provider itself. The ProviderUserManager component provides a `ProviderUserInterface Interface` for users of the provider, with which they can send and receive data by using the methods `pullFromUserManager(String path):Data` and `pushToUserManager(User providerUser, Data data):void`. Further, the ProviderUserManager component requires the ProviderUserManager interface which is provided by the Provider1Application basic component. The ProviderUserManager interface allows the ProviderUserManager component to do Create, Read, Update and Delete operations concerning trips with the following methods: `createTrip(Data tripData):void`, `readTrip(String tripId):Data`, `updateTrip(Data tripData):void` and `deleteTrip(String tripId):void`. These operations are needed to manage trips that are initiated by users. As a result, the ProviderUserManager component can provide for the `getTrips(string tripsQuery):Data` method in the `TripsProvider Interface`. Thus, the Provider1Application is able to provide the `/trips` endpoint to agencies as required by the MDS Provider API in their documentation [9].
Secondly, the `ProviderVehicleManager Interface` is required by the `ProviderVehicleManager Component`. The component should encapsulate the management of provider vehicles, as providers have vehicles that do need management for example to change their availability status. To be able to manage vehicles, the ProviderUserManager has to communicate with vehicles in his inventory. Thus, the component provides the `VehicleManagerConnector Interface` that is needed to establish a communication channel between `Vehicle Components`. Effectively, this is possible with the provided methods `pullFromVehicleManager(String path):Data` and `pushToVehicleManager(Data data):void`. Vehicle components encapsulate vehicle behavior and provide meaning to the methods defined in the `Vehicle Interface`. As described in the MDS documentation [9], vehicles can change their states and can emit events. Thus, the methods `changeState(Data status):void` can change the state of a vehicle and the method `newVehicleEvent(Data event):void` can register a new event for a vehicle. For completeness purposes, the methods `updateVehicle(Data vehicleData):void` to update a vehicle and `unlockVehicle():void` to unlock a vehicle were introduced. Further, the vehicle component provides meaning to the methods defined in the `DeviceConnector Interface`. The DeviceConnector interface is needed to establish a communication channel between a provider vehicle and a device for a vehicle. A vehicle device can send geographical data to a vehicle which is needed for a vehicle as seen in the documentation [9]. As a result, the DeviceConnector interface declared the method `pullFromVehicle(String path):Data` to pull data from a vehicle to a device and `pushToVehicle(Data data):void` to receive data as a vehicle from a vehicle device. Symmetrically to the ProviderUserManager, the ProviderVehicleManager interface allows the ProviderVehicleManager component to do Create, Read, Update and Delete operations concerning vehicles with the following methods: `createVehicle(Data vehicleData):void`,

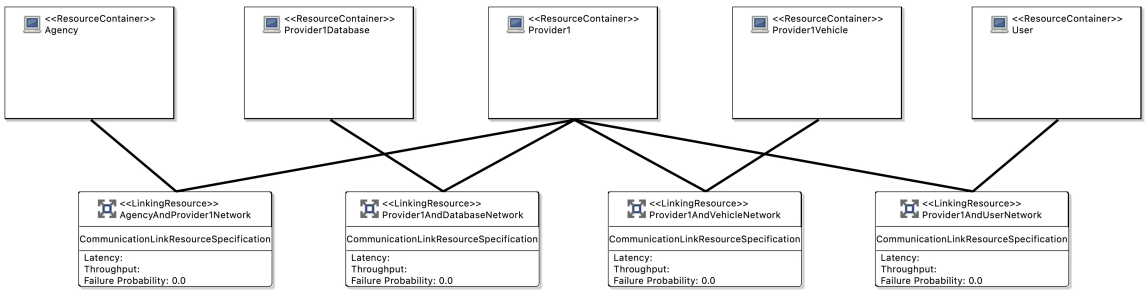

Figure 4.6: Iteration 2.1: Resource Environment

`readVehicle(String vehicleId):Data`, `updateVehicle(Data vehicleData):void` and `deleteVehicle(String vehicleId):void`. These operations are needed to manage the status changes of vehicles. As a result, the ProviderUserManager component can provide for the `getStatusChanges(string statusChangesQuery):Data` and `getEvents(string eventsQuery):Data` methods in the StatusChangesProvider interface. Thus, the Provider1Application is able to provide the `/status_changes` and `/events` endpoints to agencies as required by the MDS Provider API in their documentation [9].
Finally, a database for the Provider was needed to save provider data. The management of the database is done with the `DatabaseManager Component` to outsource the responsibility from Provider1Application. The DatabaseManager component was introduced to encapsulate management of data, f.ex. queries submissions, database connection establishment et cetera. The DatabaseManager provides `readData(string query):Data` and `writeData(string query)` methods in the `ProviderDatabaseConnector Interface`. However, the DatabaseManager requires an interface to the Database in use which is represented by the `ProviderDatabaseManager Interface` with its `readData(string query):Data` and `writeData(string query):Data` methods. As a result Provider1Application does not depend on a low-level database implementation.

4.2.1.2 Resource Environment

As a change to the first iteration, the resource container `AgencyApplication` was renamed to `Agency` as can be seen in the visual representation in Figure 4.6. The reason for this name change is that it does not make sense to name a resource container Application rather it is the element in which the application is running. In addition, the connecting linking resource between former AgencyApplication and Provider1 was renamed to `AgencyAndProvider1Network`. This change was done to emphasize that this linking resource connects specific elements as new linking resources are added in this version. Further, the resource environment was extended by a `Provider1Database`, `Provider1Vehicle` and `User`. The Provider1Database is the resource environment in which the database of the provider is running. It is connected by `Provider1AndDatabaseNetwork` linking resource to Provider1. The User resource environment should represent the user's device from which the user connects to the provider. The connection to Provider1 is represented by `Provider1AndUserNetwork` linking resource. Lastly, Provider1Vehicle is a resource environment for the provider's vehicles. Provider1 and Provider1Vehicle is connected through `Provider1AndVehicleNetwork` linking resource.

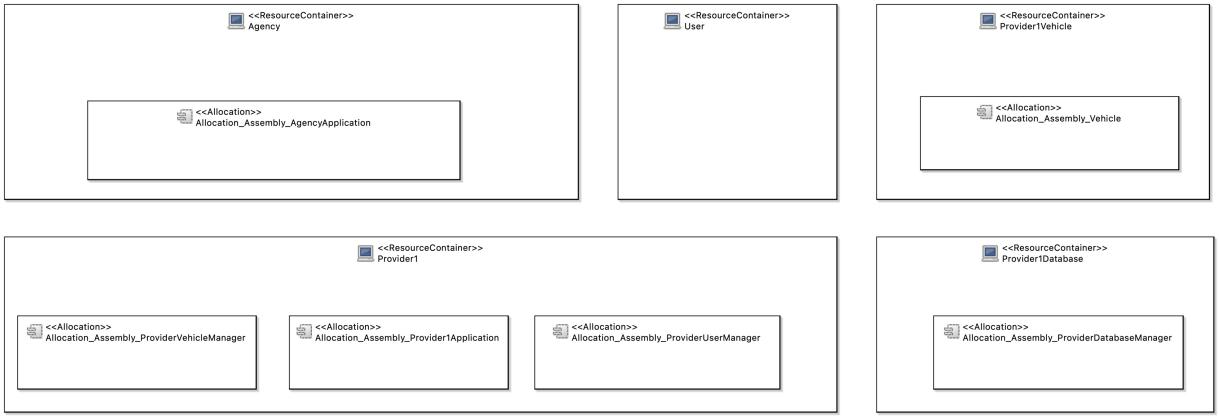

Figure 4.7: Iteration 2.1: Allocation

4.2.1.3 Allocation

In the allocation model, I placed the `AgencyApplication Allocation` in the Agency resource container as can be seen in Figure 4.7. The `Provider1Application Allocation`, `ProviderUser Allocation`, and `ProviderVehicle Allocation` was placed in the Provider1 resource container. The `Provider1DatabaseManager Allocation` was placed in the Provider1Database resource container to see how the attacker would behave when placing the DatabaseManager in a separate resource container from the resource container of the application itself. As a comparison, this was done differently in the other versions of this iteration. The `Vehicle Allocation` was placed in the Provider1Vehicle resource container since it should run on the vehicle itself. Moreover, it would be interesting to see if an attacker could propagate to the vehicle resource container, allowing him to manipulate the vehicle. The User resource container was left empty since no allocation for the user is available and is out of our scope.

4.2.1.4 Access Control Policies

The access control policy model was extended by two roles: `Provider1User` and `Provider1Vehicle`. Provider1User should represent a user of the provider, thus it is provided to the User resource container in a specification container. Provider1Vehicle should represent a user vehicle and is provided to the Provider1Vehicle resource container in a specification container. The provider policy was renamed to Provider1 policy to be consistent with the repository. Further, it is extended and has now access to: Provider1AndUserNetwork linking resource, Provider1Database resource container and Provider1AndVehicleNetwork linking resource as it needs access to them for data transfer. Two new policies were introduced. Firstly, a Provider1User policy grants a Provider1User access to Provider1User resource container and Provider1AndUserNetwork linking resource for data transfer purposes. Secondly, a Provider1Vehicle policy grants a Provider1Vehicle access to Provider1Vehicle resource container and Provider1AndVehicleNetwork linking resource for data transfer purposes.

4.2.1.5 Attacker Model

In this version of the second iteration, a new attacker model was defined to have an attacker in the user itself. I wanted to see if a provider user, which could be anyone using the provider's vehicle, could gain control over the whole system. There were no changes to the first iteration regarding the attack container, vulnerability container and category specification. In the attacker model, the attacker specification was done by using a PCM element that holds the resource container User. A system specification container was used to integrate the vulnerability to the system by a vulnerability system integration. The integration into the system was done by using a PCM element that holds the linking resource AgencyAndProviderNetwork, as in the first iteration. In addition, another integration was done by using a PCM element that holds the assembly context of ProviderUserManager.

4.2.1.6 Expected Result

It is expected that the Attacker, starting in the User resource container, can propagate to the Provider1AndUserNetwork by making use of the defined access control policy for the User role. Propagating further to the ProviderUserManager where a takeover should take place. By exploiting the vulnerability in the ProviderUserManager and taking the component over, the attacker should get access to provider components and roles. Therefore, using the same attack as in the first iteration, the provider should get access to the RestAPI role hence getting access to the agency role and components.

4.2.1.7 Case Study Properties

In this chapter, I evaluate this iteration in regard to the desired properties for our case study that are outlined in Section 3.1. In Table 4.2 an overview of the fulfilled and not fulfilled properties of this iteration is given.

Exploratory This iteration delivers a result for an attack propagation analysis in the domain of mobility systems that can be found in our published models [12]. Further, the expected results are documented in Section 4.2.1.6. Thus, the highest degree of the property Exploratory is fulfilled. It is expected that insights can already be extracted from this iteration.

(Data-)Triangulation In this version, the provider view is extended. It fulfills the property (Data-)Triangulation by the lowest degree since the model includes only the viewpoint of the provider API.

Flexbility As no major changes to the first iteration were made except for example name changes, this is evidence of the flexibility of the first iteration. Since, I defined that flexibility can be measured with the next iteration and this is the last iteration, a comparison is not possible. However, this iteration includes a detailed model of the provider such that further functionality or fine-granular modeling can be adapted easily. Moreover, non-relevant functionality or beta functionality from the current state of MDS was not modeled. That's why Flexibility by the first degree is given in our system.

Fulfilled	Degree	Property	Explanation
✓	3/3	Exploratory	Analysis delivers results in the domain of mobility systems. Expected results are documented in such a way that conclusions can be made.
✓	1/2	Flexibility	Model doesn't include non relevant functionality
✓	1/3	(Data-)Triangulation	Model includes provider viewpoint.
✓	2/4	Real-World Context	Idea of a real-world system is represented. Concepts are modeled so that they could probably occur in the real world.
✓	5/6	Reproducibility	Documentation available. Artifacts are relevant, contribute to the main result, and are available. The model is proven to produce results. Not validated yet.
✓	3/4	Vulnerability	Model includes vulnerabilities such that accuracy can be calculated. Vulnerabilities appear in the real world.
✓	1/1	Accuracy calculability	There are attack paths and expected results of attacker propagation analyses available.

Table 4.2: Iteration 2.1: Properties

Real-World Context In contrast to the first iteration, this iteration achieves a higher degree of a Real-World Context. That's because concepts, namely provider concepts, are modeled so that they could probably occur in the real world as they fulfill the API requirements given by MDS.

Reproducibility A documentation for the artifacts of this iteration was given in this chapter. The artifacts with all important models as well as their documentation are publicly available [12]. This iteration is used as a pillar for further iterations, thus they are relevant and contribute to the generations of the main results. As already stated, the model is proven to produce results and it is further feasible. As a result, the case study fulfills Reproducibility. However, the case study has to be validated by another person to reach the highest degree of reproducibility.

Vulnerability The model includes CWE 284 [3] as a vulnerability. This vulnerability appears in the real world and accuracy can be calculated. However, it has to be further researched if this vulnerability occurs in the context of mobility systems. Vulnerability is fulfilled in the third degree.

Accuracy calculability There are attack paths in this iteration, including the provider view, and expected results of attacker propagation analyses are available in Section 4.2.1.6. Thus, Accuracy calculability is fulfilled.

4.2.1.8 Result

The returned result was exactly the same result that was expected. The analysis was correct. In Figure 4.8 a visual extract of the attacker propagation graph is available. The attacker uses the vulnerability, namely `ImproperAccessControlInAgency`, to take over the ProviderUserManager. The name may be confusing since it is the same vulnerability used for the agency attack in the first iteration. At the bottom of the figure, you can see that the attacker also gained control over the agency, including AgencyApplication and the AgencyAdmin role.

4.2.2 Agency Viewpoint (v2.2)

The second version of the second iteration includes a detailed model of the agency view of the Mobility Data Specification (MDS). This version of the second iteration extends the agency view with functionality and a database.

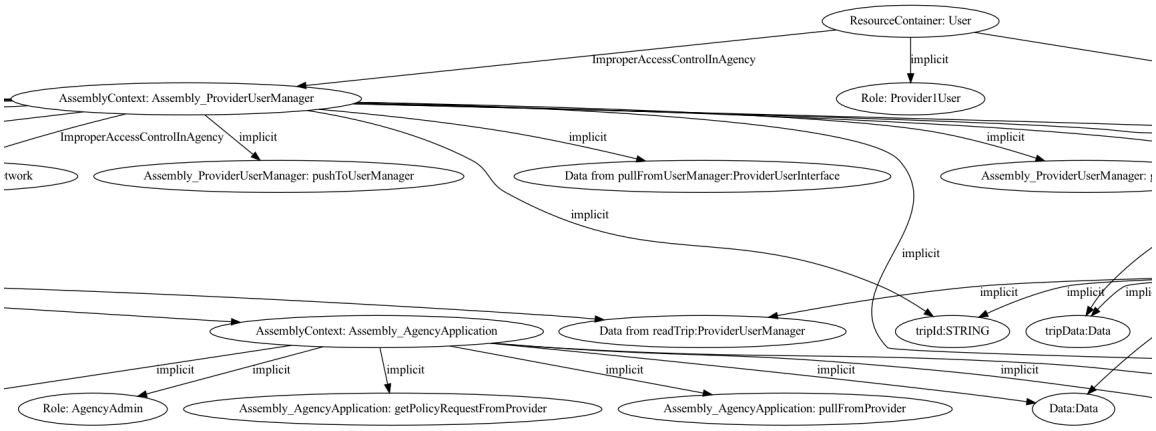

Figure 4.8: Iteration 2.1: Extract from Result

4.2.2.1 Repository

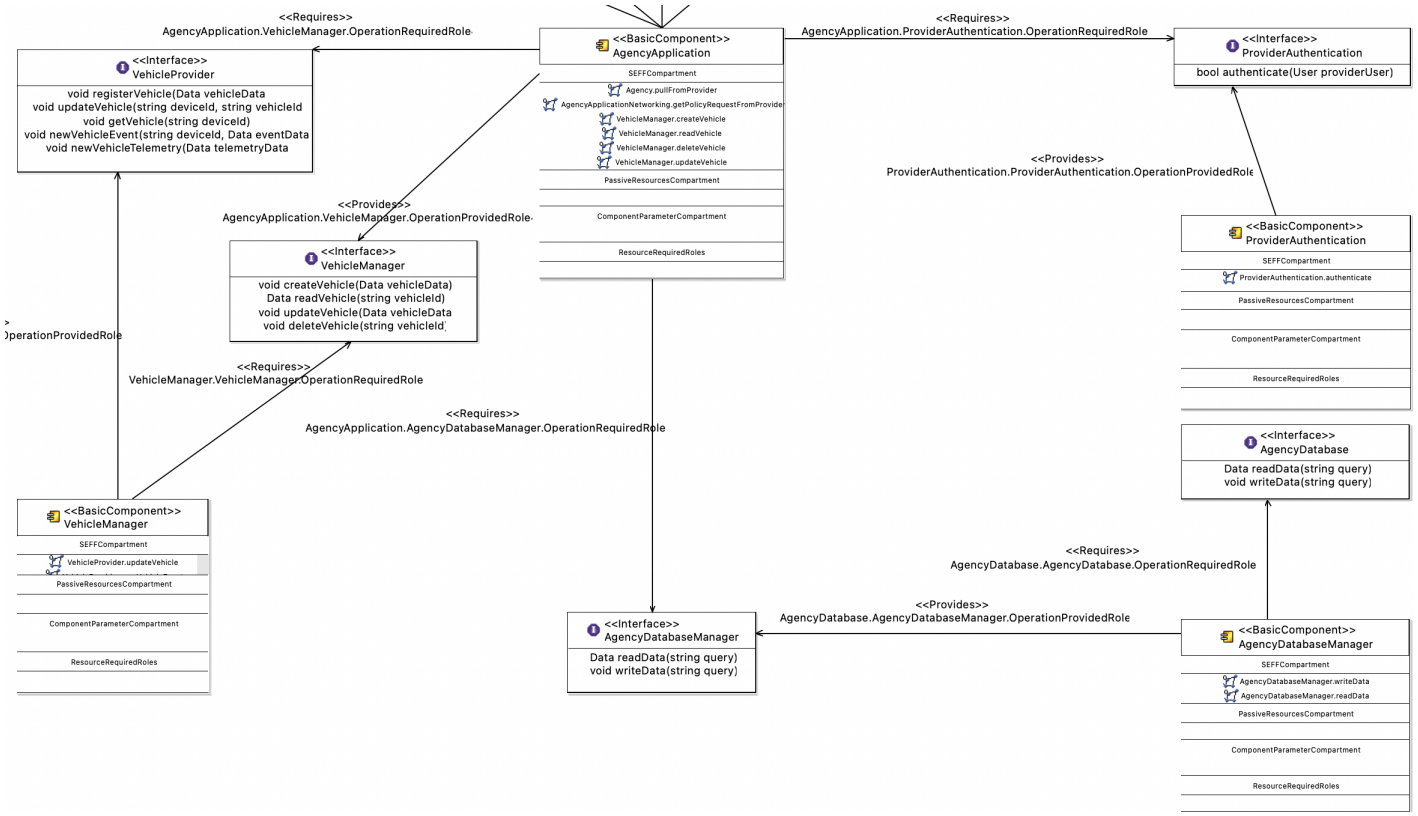

Figure 4.9: Iteration 2.2: Extract from Repository

An extract from the visual representation of the repository can be seen in Figure 4.9. The AgencyApplication component was extended by three required Interfaces: `ProviderAuthentication Interface`, `VehicleManager Interface` and `AgencyDatabaseManager Interface`. In addition, the AgencyApplication further provides the interface `VehicleProvider Interface`. Firstly, since the agency is a regulatory instance, it has providers who are managed through the data of the agency and thus consume the Agency API as outlined in Section 2.3. In order to fulfill this function, the `VehicleManager Component` was introduced to represent vehicle data management needed for providers. Further, the VehicleManager component requires the VehicleManager interface which is provided by the AgencyApplication component. The VehicleManager interface allows the VehicleManager component to do Create, Read, Update and Delete operations concerning vehicles with the following methods: `createVehicle(Data vehicleData):void`, `readTrip(String vehicleId):Data`, `updateTrip(Data vehicleData):void` and `deleteTrip(String vehicleId):void`. These operations are needed to manage the vehicle data of providers. As a result, the VehicleManager component can provide the VehicleProvider interface. The VehicleProvider interface allows the AgencyApplication component to register, update and get vehicle data. Moreover, the AgencyApplication is able to register vehicle events and vehicle telemetry data (for example location of the vehicle). This is represented with the following methods in the VehicleProvider interface: `registerVehicle(Data vehicleData):void`, `updateVehicle(string deviceId, string vehicleId):void`, `getVehicle(string deviceId):void`, `newVehicleEvent(string deviceId, Data eventData):void` and `newVehicleTelemetry(Data telemetryData):void`. Thus, the AgencyApplication is able to provide the `/vehicles` endpoint to providers as required by the MDS Agency API in their documentation [8].
Since data from providers have to be stored (for example vehicle data), the agency needs a database. The management of the database is done with the `AgencyDatabaseManager Component` to outsource the responsibility from AgencyApplication. The AgencyDatabaseManager component was introduced to encapsulate management of data, f.ex. queries submissions, database connection establishment et cetera. The AgencyDatabaseManager component provides to the AgencyDatabaseManager interface, allowing database operations through the provided `writeData(String query):void` and `readData(String query):Data` methods. However, the AgencyDatabaseManager component requires an interface of the database, which is named `AgencyDatabase` to read and write from the database. As a result AgencyApplication does not depend on a low-level database implementation.
Finally, in the documentation of MDS [9], it is specified that the provider should get authorized with a JWT-Token. In order to model this, the `ProviderAuthentication Component` is providing to the belonging `ProviderAuthentication Interface`, giving functionality to the `authenticate(User providerUser):bool` method.

4.2.2.2 Resource Environment

In addition to the first iteration, a new resource container `AgencyDatabase` was introduced as can be seen in the visual representation in Figure 4.10. It is used to run the database application on it. The AgencyDatabase resource container is linked with the AgencyApplication through the linking resource `AgencyAndDatabaseNetwork` so data change can be done between the database and the AgencyDatabaseManager therefore the AgencyApplication.

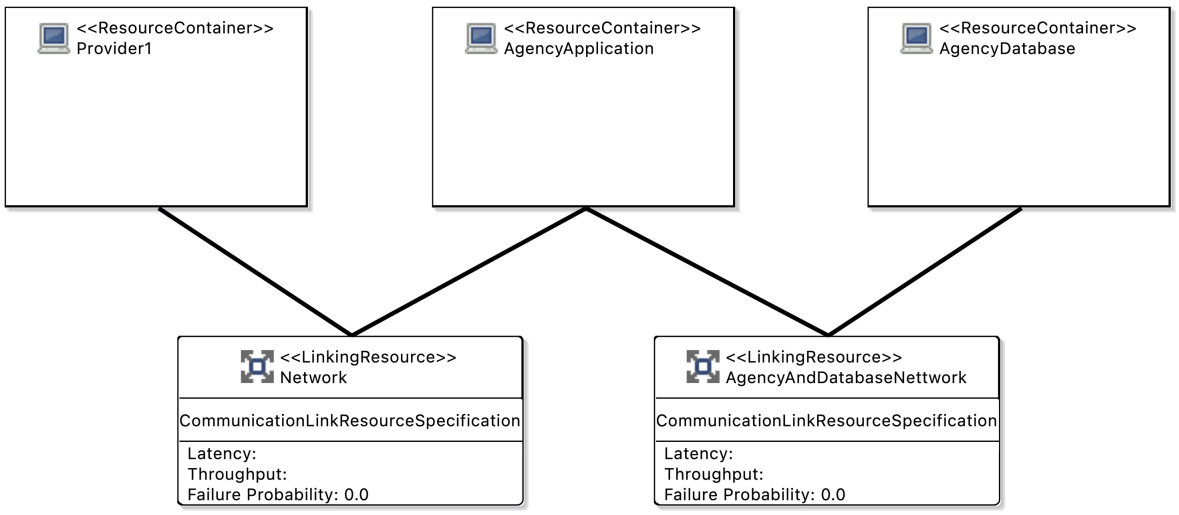

Figure 4.10: Iteration 2.2: Resource Environment

4.2.2.3 Allocation

A visual representation of the allocation model can be seen in Figure 4.11. All new introduced components or rather their allocations, `ProviderAuthentication Allocation`, `VehicleManager Allocation` and `AgencyDatabaseManager Allocation` were placed in the AgencyApplication resource environment. In this version I did not separate the AgencyDatabaseManager from the AgencyApplication resource environment since I do not need to introduce microservices, rather it is preferred this way in terms of overhead, performance, and latency. This leaves the AgencyDatabase resource environment without any allocations.

4.2.2.4 Access Control Policies

The access control policy was only changed in the Agency policy. The AgencyAdmin role has now access to the AgencyDatabase resource container since the agency has to access the database for read or write operations.

4.2.2.5 Attacker Model

There was no change in the attacker in comparison to the first iteration.

4.2.2.6 Expected Result

It is expected that the Attacker can propagate to the linking resource Network without using any attack just by the defined access control policies. However, through the modeled vulnerability in Network there should be a takeover of this linking resource to the attacker. The attacker should get control over the agency since Network has the role RestAPI which grants access to Provider and Agency. Finally, the attacker should have complete control over the whole system.

Figure 4.11: Iteration 2.2: Allocation

4.2.2.7 Case Study Properties

In this chapter, I evaluate this iteration in regard to the desired properties for our case study that are outlined in Section 3.1. In Table 4.2 an overview of the fulfilled and not fulfilled properties of this iteration is given.

Exploratory This iteration delivers a result for an attack propagation analysis in the domain of mobility systems that can be found in our published models [12]. Further, the expected results are documented in Section 4.2.2.6. Thus, the highest degree of the property Exploratory is fulfilled. It is expected that insights can already be extracted from this iteration.

(Data-)Triangulation In this version, the agency view is extended. It fulfills the property (Data-)Triangulation by the lowest degree since the model includes only the viewpoint of the agency API.

Flexbility As no major changes to the first iteration were made except for example name changes, this is evidence of the flexibility of the first iteration. Since, I defined that flexibility can be measured with the next iteration and this is the last iteration, a comparison is not possible. However, this iteration includes a detailed model of the agency such that further functionality or fine-granular modeling can be adapted easily. Moreover, non-relevant functionality or beta functionality from the current state of MDS was not modeled. That's why Flexibility by the first degree is given in our system.

Fulfilled	Degree	Property	Explanation
✓	3/3	Exploratory	Analysis delivers results in the domain of mobility systems. Expected results are documented in such a way that conclusions can be made.
✓	1/2	Flexibility	Model doesn't include non relevant functionality.
✓	1/3	(Data-)Triangulation	Model includes agency viewpoint.
✓	2/4	Real-World Context	Idea of a real-world system is represented. Concepts are modeled so that they could probably occur in the real world.
✓	5/6	Reproducibility	Documentation available. Artifacts are relevant, contribute to the main result, and are available. The model is proven to produce results. Not validated yet.
✓	3/4	Vulnerability	Model includes vulnerabilities such that accuracy can be calculated. Vulnerabilities appear in the real world.
✓	1/1	Accuracy calculability	There are attack paths and expected results of attacker propagation analyses available.

Table 4.3: Iteration 2.2: Properties

Real-World Context In contrast to the first iteration, this iteration achieves a higher degree of Real-World Context. That's because concepts, namely agency concepts, are modeled so that they could probably occur in the real world as they fulfill the API requirements given by MDS.

Reproducibility A documentation for the artifacts of this iteration was given in this chapter. The artifacts with all important models as well as their documentation are publicly available [12]. This iteration is used as a pillar for further iterations, thus they are relevant and contribute to the generations of the main results. As already stated, the model is proven to produce results and it is further feasible. As a result, the case study fulfills Reproducibility. However, the case study has to be validated by another person to reach the highest degree of reproducibility.

Vulnerability The model includes CWE 284 [3] as a vulnerability. This vulnerability appears in the real world and accuracy can be calculated. However, it has to be further researched if this vulnerability occurs in the context of mobility systems.

Accuracy Calculability There are attack paths in this iteration, including the agency view, and expected results of attacker propagation analyses are available in Section 4.2.2.6. Thus, Accuracy calculability is fulfilled.

4.2.2.8 Result

The returned result was exactly the same result that was expected. The analysis was correct. In Figure 4.12 a visual extract of the attacker propagation graph is available. The attacker uses the vulnerability, namely `ImproperAccessControlInAgency`, to take over the Network linking resource. The name may be confusing since it is the same vulnerability used for the agency attack in the first iteration. By using this vulnerability, the attacker gains the role RestAPI with which it has implicit access to the agency and all its elements. Because of this, the attacker has also access to the newly added VehicleManager.

4.2.3 Provider Viewpoint (v2.3)

The third version of the second iteration includes a detailed model of the policy view of the Mobility Data Specification (MDS).

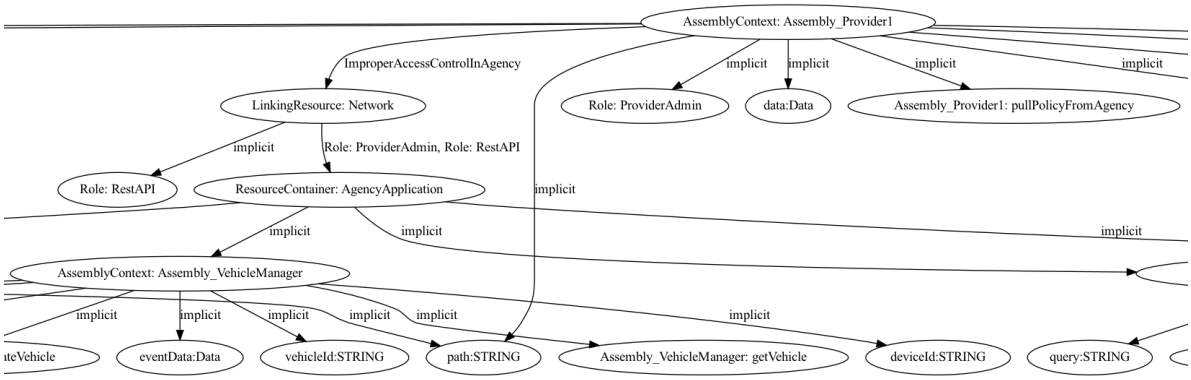

Figure 4.12: Iteration 2.2: Extract from Result

4.2.3.1 Repository

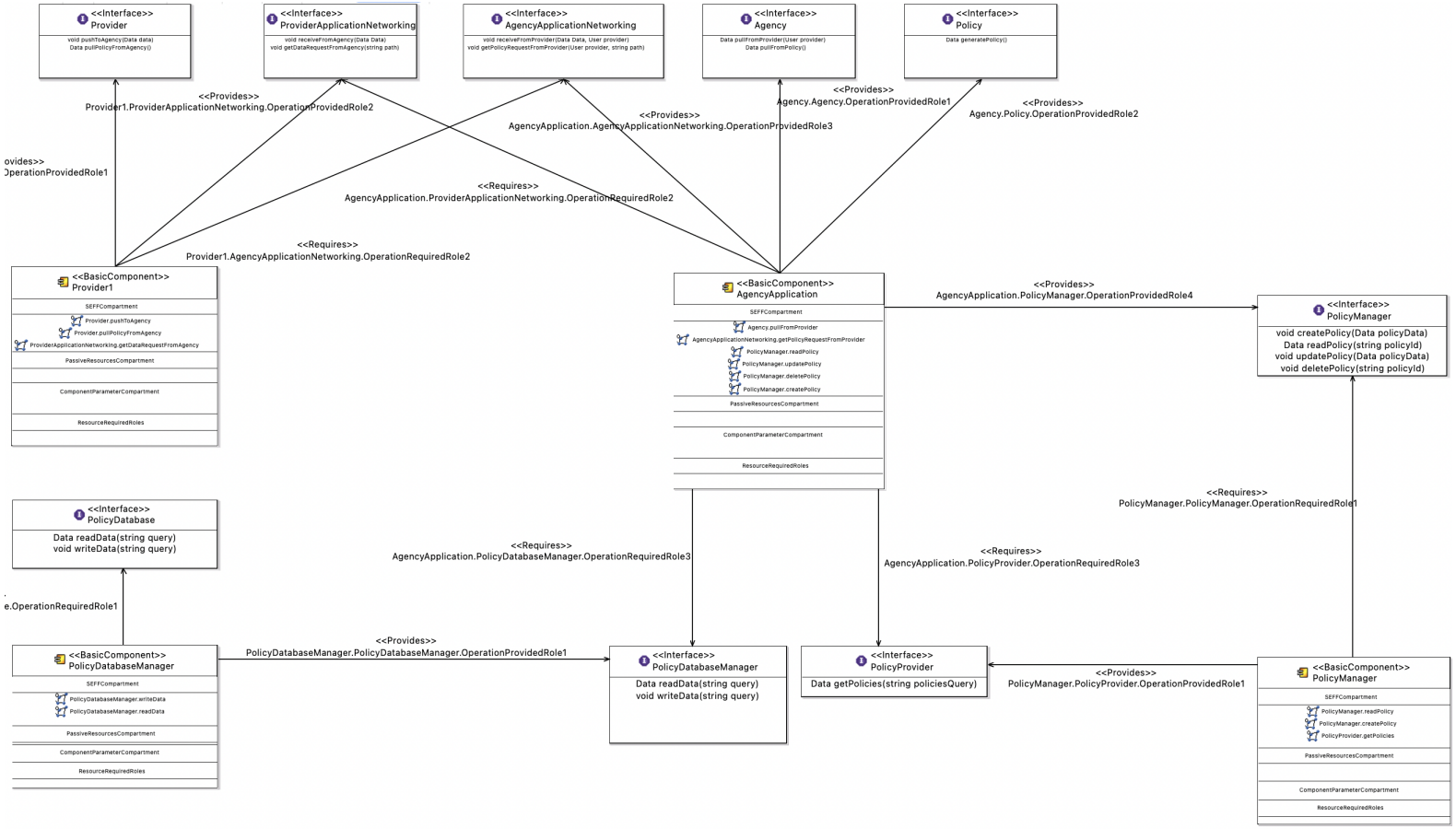

Figure 4.13: Iteration 2.3: Extract from Repository

An extract from the visual representation of the repository can be seen in Figure 4.13. The AgencyApplication component was extended by two required Interfaces: `PolicyProvider Interface` and `PolicyDatabaseManager Interface`. In addition, the AgencyApplication further provides the interface `PolicyManager Interface`. Firstly, since the agency is a regulatory instance, it is able to establish policies for providers who are managed through the data of the agency and thus consume the Agency API as outlined in Section 2.3. In order to fulfill this function, the `PolicyManager Component` was introduced to represent the policy management of the agency. Further, the PolicyManager component requires the PolicyManager interface which is provided by the AgencyApplication component. The PolicyManager interface allows the PolicyManager basic component to do Create, Read, Update and Delete operations concerning policies with the following methods: `createPolicy(Data policyData):void`, `readPolicy(String policyId):Data`, `updatePolicy(Data policyData):void` and `deletePolicy(String policyId):void`. These operations are needed to manage the policy data of the agency. As a result, the PolicyManager component can provide the PolicyProvider interface. The PolicyProvider interface allows the AgencyApplication get policy data with the method `getPolicies(string policiesQuery)` in the interface. Thus, the AgencyApplication is able to provide the `/policies` endpoint to providers as required by the MDS Agency API in their documentation [8].
Finally, this policy data has to be stored in a database. The management of the database is done with the `PolicyDatabaseManager Component` to outsource the responsibility from AgencyApplication. The PolicyDatabaseManager component was introduced to encapsulate management of data, f.ex. queries submissions, database connection establishment et cetera. The PolicyDatabaseManager component provides to the PolicyDatabaseManager interface, allowing database operations through the provided `writeData(String query):void` and `readData(String query):Data` methods. However, the PolicyDatabaseManager component requires an interface of the database, which is named `PolicyDatabase Interface` to read and write from the database. As a result AgencyApplication does not depend on a low-level database implementation.

4.2.3.2 Resource Environment

In addition to the first iteration, a new resource container `PolicyDatabase` was introduced as can be seen in the visual representation in Figure 4.14. It is used to run the database application on it. The AgencyDatabase resource container is linked with the AgencyApplication through the linking resource `AgencyAndPolicyDatabaseNetwork` so data change can be done between the database and the AgencyDatabaseManager as well as the AgencyApplication.

4.2.3.3 Allocation

A visual representation of the allocation context can be seen in Figure 4.15. All new introduced components or rather their allocations, `PolicyManager Allocation` and `PolicyDatabaseManager Allocation` were placed in the AgencyApplication resource environment. In this version I did not separate the PolicyDatabaseManager from the AgencyApplication resource environment since I do not need to introduce microservices, rather it is preferred this way in terms of overhead, performance, and latency. This leaves the PolicyDatabase resource environment without any allocation contexts.

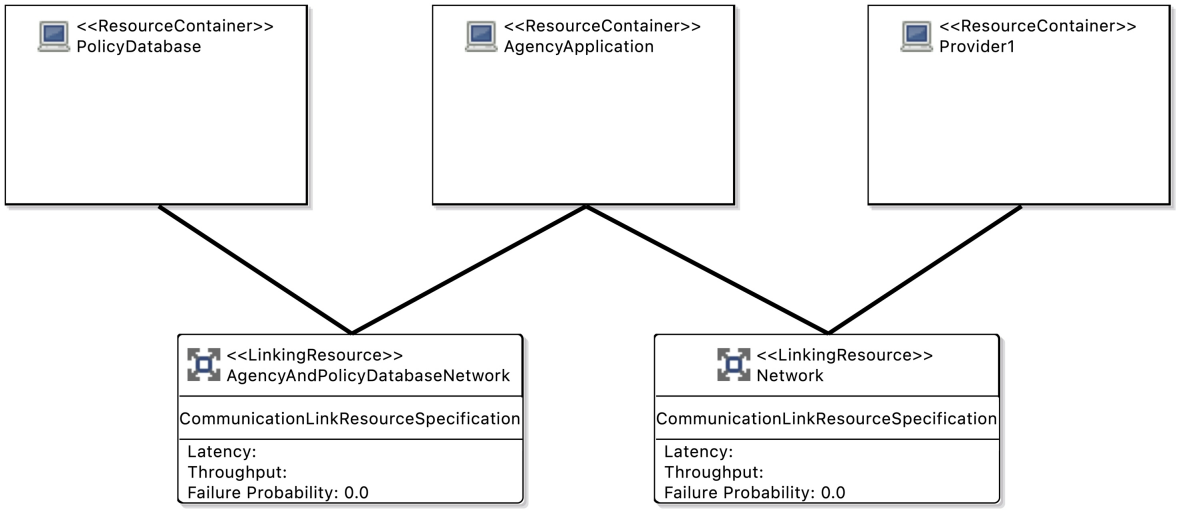

Figure 4.14: Iteration 2.3: Resource Environment

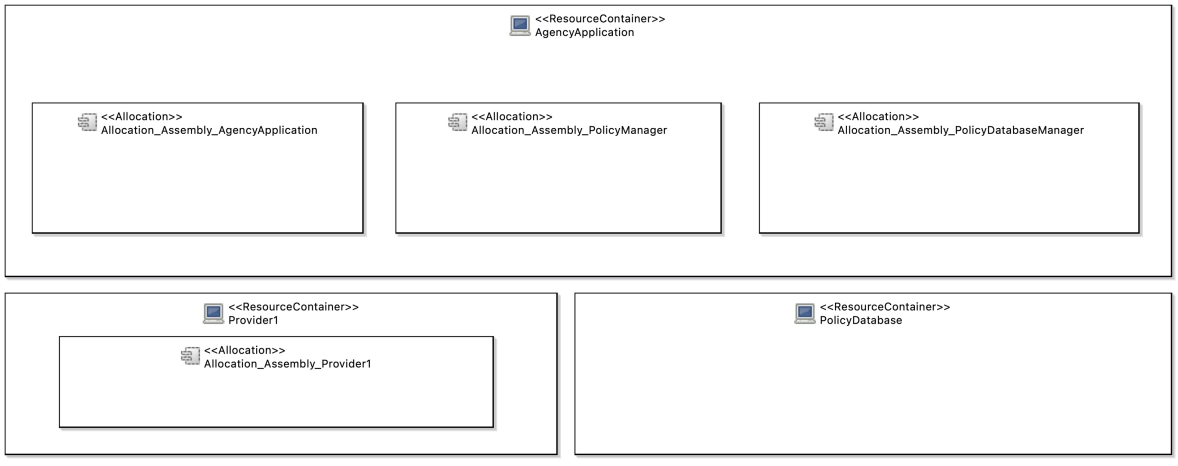

Figure 4.15: Iteration 2.3: Allocation

4.2.3.4 Access Control Policies

The AgencyAdmin role is provided to the AgencyApplication assembly context. The access control policy was only changed in the Agency policy. The AgencyAdmin role has now access to the AgencyDatabase resource container since the agency has to access the database for read or write operations.

4.2.3.5 Attacker Model

There was no change in the attacker in comparison to the first iteration.

4.2.3.6 Expected Result

It is expected that the attacker can propagate to the linking resource Network without using any attack just by the defined access control policies. However, through the modeled vulnerability in Network there should be a takeover of this linking resource to the attacker. The attacker should get control over the agency since Network has the role RestAPI which grants access to the provider and agency. Finally, the attacker should have complete control over the whole system since the role ProviderAdmin has also access to the PolicyDatabase resource container.

4.2.3.7 Case Study Properties

In this chapter, I evaluate this iteration in regard to the desired properties for our case study that are outlined in Section 3.1. In Table 4.4 an overview of the fulfilled and not fulfilled properties of this iteration is given.

Exploratory This iteration delivers a result for an attack propagation analysis in the domain of mobility systems that can be found in our published models [12]. Further, the expected results are documented in Section 4.2.3.6. Thus, the highest degree of the property Exploratory is fulfilled. It is expected that insights can already be extracted from this iteration.

(Data-)Triangulation In this version, the policy view is extended. It fulfills the property (Data-)Triangulation by the lowest degree since the model includes only the viewpoint of the policy API.

Flexibility As no major changes to the first iteration were made except for example name changes, this is evidence of the flexibility of the first iteration. Since, I defined that flexibility can be measured with the next iteration and this is the last iteration, a comparison is not possible. However, this iteration includes a detailed model of the agency such that further functionality or fine-granular modeling can be adapted easily. Moreover, non-relevant functionality or beta functionality from the current state of MDS was not modeled. That's why Flexibility by the first degree is given in our system.

Real-World Context In contrast to the first iteration, this iteration achieves a higher degree of Real-World Context. That's because concepts, namely policy concepts, are modeled so that they could probably occur in the real world as they fulfill the API requirements given by MDS.

Fulfilled	Degree	Property	Explanation
✓	3/3	Exploratory	Analysis delivers results in the domain of mobility systems. Expected results are documented in such a way that conclusions can be made.
✓	1/2	Flexibility	Model doesn't include non relevant functionality. Model is extendable.
✓	1/3	(Data-)Triangulation	Model includes agency viewpoint.
✓	2/4	Real-World Context	Idea of a real-world system is represented. Concepts are modeled so that they could probably occur in the real world.
✓	5/6	Reproducibility	Documentation available. Artifacts are relevant, contribute to the main result, and are available. The model is proven to produce results. Not validated yet.
✓	3/4	Vulnerability	Model includes vulnerabilities such that accuracy can be calculated. Vulnerabilities appear in the real world.
✓	1/1	Accuracy calculability	There are attack paths and expected results of attacker propagation analyses available.

Table 4.4: Iteration 2.3: Properties

Reproducibility A documentation for the artifacts of this iteration was given in this chapter. The artifacts with all important models as well as their documentation are publicly available [12]. This iteration is used as a pillar for further iterations, thus they are relevant and contribute to the generations of the main results. As already stated, the model is proven to produce results and it is further feasible. As a result, the case study fulfills Reproducibility. However, the case study has to be validated by another person to reach the highest degree of reproducibility.

Vulnerability The model includes CWE 284 [3] as a vulnerability . This vulnerability appears in the real world and accuracy can be calculated. However, it has to be further researched if this vulnerability occurs in the context of mobility systems.

Accuracy calculability There are attack paths in this iteration, including the policy view, and expected results of attacker propagation analyses are available in Section 4.2.2.6. Thus, Accuracy calculability is fulfilled.

4.2.3.8 Result

The returned result was exactly the same result that was expected. The analysis was correct. In Figure 4.16 a visual extract of the attacker propagation graph is available. The attacker uses the vulnerability, namely `ImproperAccessControlInAgency`, to take over the Network linking resource. The name may be confusing since it is the same vulnerability used for the agency attack in the first iteration. By using this vulnerability, the attacker gains the role RestAPI with which it has implicit access to the agency and all its elements. Because of this, the attacker has also access for example to the PolicyDatabase as seen in the figure.

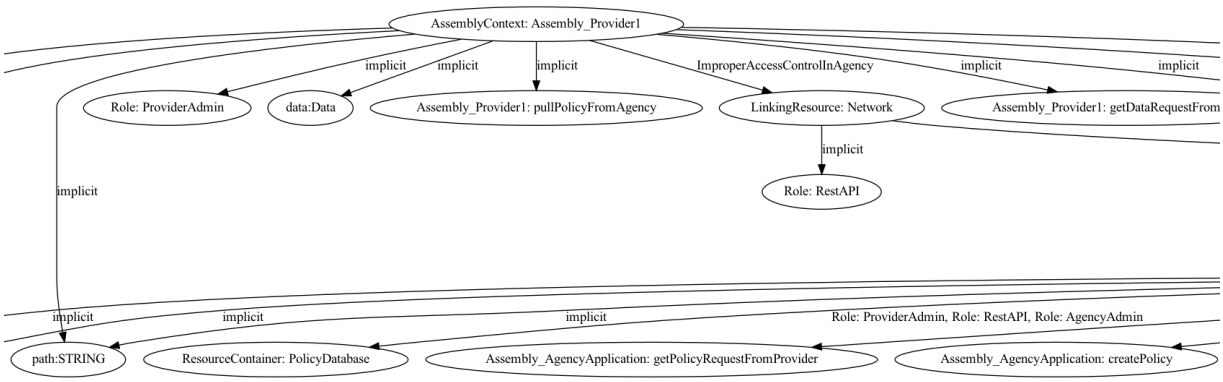

Figure 4.16: Iteration 2.3: Extract from Result

5 Lessons Learned

During the preparation of this work a deeper understanding of many subject areas was achieved and the lessons learned can be divided into the following groups: component-based modeling including the palladio component model, access control policy modeling and attacker modeling, case studies, especially in software engineering and academic work in general. This chapter gives an overview of the lessons that were learned during this work but aims to give useful information for further research in the context of this use case. It is worth noting that the useful information only represents the way of work that was pursued in this case study and does not represent the only way to solve a problem.
In the beginning, it was difficult to model a system and stay in the component-based modeling context. As I did not use the theoretical knowledge of components-based modeling in practice an obstacle was holding the abstraction level of component-based modeling. It occurred multiple times that the modeling, especially the repository modeling, strived towards more low-level modeling. However, in the process of this work, I have learned what component-based modeling means in practice. It is important to keep the functionality of the software in the foreground and to subdivide these functionalities into different areas. Conversely, this means that implementation details, the how do not play a role. To give an illustrative example in the context of this work: The agency can already be seen as a component and in the sense of component-based software development, it is irrelevant how this component (here the agency) implements its functionality. It is sufficient to know that the agency has a functionality, which in this case can be distinguished from the functionality of the provider. This does not mean that modeling ends here; instead, the functionality of the agency can be subdivided into further components.
In the course of component-based modeling, I had to use the Palladio Component Model. As already mentioned in Chapter 2, the Palladio Component Model provides a framework for component-based modeling and uses Eclipse for this. While using the Palladio Component Model in combination with the attack propagation analysis extension, I learned a lot about their practical use, which I will summarise below. Firstly, for the repository, resource environment, allocation and system there are representations on which component-based modeling can be done with a UI. For allocation modeling, a resource environment is needed beforehand to assign allocations to resource environments. In the system, it is possible to provide interfaces to the outside of the systems. In our work, this was done for the provider, agency, and policy application interfaces. Access control policies and attacker specifications don't have a representation in form of a diagram. Thus, elements had to be added in a specific way. For the access control policies, I learned that the root element is a `Confidential Access Specification`. This element then consists of three children. Firstly, `Attributes` in which a `Simple Attribute Role` child element can hold different roles of the system. Secondly, a `Set` in which multiple `Policy` child elements can be placed. These policy elements represent different access control policies for the system and include `Rule` child elements. In this rule element, the

type of conditions could be specified. In our case, a `Simple Attribute Condition` was enough. Further, the access conditions were specified with `All of` child elements which specifies for which model elements the policy applies. These model elements can be specified with an `Entity Match` child element. Lastly, a `Specification Container` which is used to assign roles to different elements of the system. This can be done with a `PCM Attribute Provider` child elements which define the elements that should get the roles. The assigned roles are then defined in a `Usage Specification` child element of the PCM attribute provider. Concerning attack specifications the root elements is a `Specification` element. I used five children elements: `Container`, `Attack Container`, `Vulnerability Container`, `Category Specification` and `System Specification Container`. It is useful to first decide on a vulnerability that should be integrated into the system. In the category specification element a `CWEID` or `CVEID` child element can then be added where vulnerability information is stored. After, in the vulnerability container a `CVE Vulnerability` or `CWE Vulnerability` child element can be added where the effects of the vulnerability in the system are defined. This is needed in the system specification container where the vulnerability is integrated into the system with a `Vulnerability System Integration` where the vulnerability is stored. Further, the vulnerability system integration has a `PCM Element` child element in which it is defined where the vulnerability is placed. The attack container holds the `CWE Attack` or `CVE Attack` as a child element. This is needed to give the attack to an attacker in the `Container` element. The container element has an `Attacker` child element in which the attack is defined. Lastly, the attacker element has a `PCM Element` as a child in which the attack is assigned to a model element. Finally, to run the attack propagation analyses of Maximilian Walter et al. [25] a kamp4atttackmodificationmarks file is needed. It stores propagation details. However, the root element `Kamp4attack Seed Modifications` is needed in which the attacker is defined through an `Attacker Selection` child element. For the access policy context, attacker model, and the kamp4attackmodificationsmarks, it is sometimes necessary to load the other models manually to be able to assign their elements. This can be done through right-clicking and "Load Repository".
Besides the theoretical knowledge about case studies, I've learned that the evaluation process of the case study is one of the most important but also the most difficult parts. During the case study design there are many properties that one could think of. However, measuring these properties is the harder part. That's why measurement methods have to be discussed while defining properties for the evaluation. These measurement methods can be different from case study to case study and there is no general approach to this. Moreover, only with an evaluation and proven properties, are other researchers enabled to confirm their results. An example of this is the `Accuracy calculability` property. The measurement of this property was that

There are attack paths and expected results of attacker propagation analyses are available Section 3.1. Thus, the researcher is able to give an estimate of the accuracy of the attack propagation analyses. Finally, the most valuable lesson was that implementation work should be done in iterations whenever possible, in academic works and in general. By recommendation of my advisors, the implementation progress was done in iterations as described in Section 3.2. This gave me the possibility to better organize my time since nothing had to be planned for a long time period before. Furthermore, iterative modeling is fault tolerant. Faults in prior iterations can be corrected in post-iterations. In addition, it was better in terms of time management since I could decide when to stop with the implementation of the case study.

5.1 Threats to Validity

In this chapter, I want to discuss the validity of this case study and must therefore discuss potential threats to the validity. The discussion will be based on the four categories of validity as suggested by Runeson et al. [21].

Construct Validity describes whether the end result of this work is representing a mobility case study for validating attack propagation analyses. Firstly, it is of concern if this work fulfills the requirements to be a case study. If it is a case study, then it has to be clarified whether it can validate attack propagation analyses. In the progress of constructing this work, I examined properties that case studies in software engineering should fulfill from academic literature. These properties were then listed in Section 3.1 together with possibilities to evaluate them. The chapters about Evaluation Regarding Properties for our second iteration have revealed that all of them were fulfilled. Regarding these results, I can assume that this work can be categorized as a case study. This list of properties further included properties that were introduced by us specifically for attack propagation analyses, namely Vulnerability and Accuracy calculability. Accuracy calculability required that "There are attack paths and expected results of attacker propagation analyses are available". As this property is fulfilled in the second iteration and I could additionally verify the result of the attack propagation analyses of Walter et al. [25] in each version, I can also assume that this case study can validate attack propagation analyses.

Internal Validity discusses whether there are internal factors that could be a threat to the validity of which I am not aware. A major threat to internal validity is that I constructed the measurement methods for our properties myself. These measurement methods have to be validated by external researchers or already validated measurement methods have to be used to minimize the threat to internal validity. In detail, I have set the metrics in form of degrees for the following properties: Exploratory, Flexibility, (Data-)Triangulation, Real-World Context, Reproducibility, Vulnerability and Accuracy calculability.

External Validity deals with the question if the result can be generalized. Firstly, the Mobility Data Specification was our reference for mobility systems in this case study. The Open Mobility Foundation is providing a way to standardize "communication and data-sharing between cities and private mobility providers" [14] with the Mobility Data Specification. It is used already by many cities. Therefore, the aspect of the modeled mobility system should not be a threat to validity in this case study. However, the implementation of the mobility system in our case study offers a component repository, a resource environment, an allocation context, access control policies, and an attacker model as explained in Section 3.2.2. If there are attack propagation analyses that need other elements, a generalization would be restricted. As a result, this would represent a threat to the external validity of the case study and has to be researched.

Reliability describes if the results, in this context the case study, can be reproduced by other researchers. In the list of properties, the case study is evaluated, I defined the property Reproducibility. This property covers exactly this aspect. As in our second iteration, this property is fulfilled, I can assume that the risk for reliability in the context of the case study in

general is minimized. However, Reliability describes also if the evaluation can be reproduced by other researches. As already discussed for Internal Validity, I set the metrices for the case study properties. It is possible that other researchers would find different methods to measure the properties. To minimize this risk, I discussed the properties regularly with my advisors. The possibility that other researchers would use completely different properties is low since the properties were extracted from popular academic literature.

5.2 Limitations

The current version of the case study consists of two iterations and four versions in total. It was planned to do a third iteration nevertheless the time was not enough. This leaves the case study with properties in Section 3.1 that are not fulfilled in the highest grade. There is no version of the case study which has all three viewpoints of MDS in a single model, thus the highest degree of "(Data-)Triangulation" could not be achieved. Concerning the "Reproducability" the case study has to be validated by another person except for the author. Since the second iteration consists of three different versions, each representing one view of the three APIs in MDS, the property "Real-World Context" is fulfilled to the second degree. Third degree "Real-World Context" would be achieved if they were merged into one model which was planned for the third iteration. Lastly, the highest degree of "Vulnerability" couldn't be achieved since there is no evidence that the used vulnerabilities in this case study appear in the domain of mobility systems.

6 Related Work

This case study mainly relates to two works. First of all, the case study refers to the "Architectural Attack Propagation Analysis" [25] which is being investigated here. Secondly, the "Guidelines for conducting and reporting case study research in software engineering" [21], which is referred to by many papers about case studies in software engineering and which I'll be using as a reference work. Wohlin published a paper namely "Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study?"[26] in which she refers to the already mentioned Guidelines and clearly shows the boundaries of case studies. It is useful for this work since it's easy to cross these boundaries as shown in the statistics of the work. Runes et al. have published a book, namely "Case Study Research in Software Engineering - Guidelines and Examples" [17] with guidelines and valuable examples for case studies in software engineering. Another important work about case study research is found in "Guide to Advanced Empirical Software Engineering" by Schull et al [22]. It concentrates mainly on empirical software engineering in which case study research takes a role. Furthermore, it deals with philosophical stances about empirical truth which must be kept in mind for academic research. However, these academic works concentrate mainly on the creation of case studies. This case study was done specifically in the domain of mobility systems. Concerning the "Architectural Attack Propagation Analysis", the Palladio Component Model (PCM) [20] is needed to understand architectural models and to model themselves. However, important aspects can be found in the introduction paper about PCM [4]. In addition, the Common Component Modeling Example (CoCoME) [18] delivers an approach to model the component-based part of the architecture.
Concerning earlier studies, there are three case studies that have been conducted for the Attack Propagation Analysis in [25]. Firstly, a case study that simulates in its real-world context a cyber attack on the retail store Target in the year 2013. They have modelled a component for a business backend and a component handling the billing. In addition, the model contained Point of Sale devices, storage devices and a database. All of them include a CWEVulnerability, same for the business backend component. Secondly, a case study with a cyber attack on the modelled Ukrainian Power Grid as context. In that case study, two networks were modelled and connected through a component that acted as a VPN gateway. Further, CVE-2014-1761 [2] was included as a vulnerability. Lastly, a case study with the context of a mobile application to book flights. Customer, credit card centre, travel agency and airline were modelled, and CWE vulnerabilities based on the OWASP Top Ten were included and analyzed.

7 Conclusion

In this work, I built a case study in the domain of mobility systems that can be used for attack propagation analyses. In our case study, I defined properties for evaluation purposes. Furthermore, all concepts of the Mobility Data Specification were modeled. The case study was evaluated according to the properties for case studies in software engineering in general and for the mobility system domain in particular. The evaluation showed that all properties are fulfilled to some degree. This allows to analyze attack propagation works in the mobility system domain. Attack propagation analyses can use this case study and are enabled to present references and proofs for their evaluation.
I want to discuss the future of this work including a third iteration that was planned but could not be implemented. It was planned that the third iteration is a merge of all three versions in the second iteration. So, the third iteration could be a single model of a complete mobility system. To achieve this, the policy and agency repository should be merged together. In this merge progress, the `AgencyDatabaseManager` and `PolicyDatabaseManager` component could be unified to one single `AgencyDatabaseManager` component. It is useful to separate a policy database from the agency database since both datasets have different concerns and it is easier to scale one of both. However, both databases should be managed by the same DatabaseManager component and should have the same databases to minimize work expenditure and for better maintainability. No changes are required when merging the provider repository. In addition, more datatypes could be introduced for example `PolicyData` to have a distinction in the data flow regarding the attack propagation analyses. Further, the database allocations are also placed in these resource containers. An interesting extension in the third iteration would be to have multiple User allocations and resource containers as well as multiple vehicle allocations and resource containers. This could be useful for the analyses of the attack propagation analyses in terms of scalability. The case study could be used to measure performance and quality with many resource containers, and allocations, thus a big system. Generally, future work concerning this case study should strive to accomplish the highest grade of the properties in Section 3.1. This includes “(Data-)Triangulation” where the highest grade requires a view of all three APIs. Further, “Real-World Context” where an idea for future work could be to work with companies or instances who are using the Mobility Data Specification or at least third degree with the proposals for the third iterations. Lastly, in terms of “Vulnerability” it would be interesting to research which vulnerabilities occur in real-world mobility systems to integrate them into this case study. This would reach the highest degree of “Vulnerability” and improve the quality of the case study tremendously.

Bibliography

[1] *Artifact Review and Badging by ACM*. Last accessed 3 October 2022. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current.

[2] *CVE-2014-1761 Vulnerability*. Last accessed 23 June 2022. URL: https://nvd.nist.gov/vuln/detail/CVE-2014-1761.

[3] *CWE 284: Improper Access Control*. Last accessed 7 October 2022. URL: https://cwe.mitre.org/data/definitions/284.html.

[4] Heiko Koziolek et al. "Evaluating Performance of Software Architecture Models with the Palladio Component Model". In: *Model-Driven Software Development: Integrating Quality Assurance* (Jan. 2008). DOI: 10.4018/978-1-60566-006-6.ch005.

[5] *Lyft*. Last accessed 13 June 2022. URL: http://lyft.com.

[6] Miloš N. Mladenović. *Mobility as a Service*. 2021, pp. 12–18. ISBN: 978-0-08-102672-4. DOI: https://doi.org/10.1016/B978-0-08-102671-7.10607-4.

[7] *Mobility Data Specification (MDS)*. Last accessed 13 June 2022. URL: https://openmobilityfoundation.org/about-mds/#.

[8] *Mobility Data Specification (MDS), Agency*. Last accessed 12 October 2022. URL: https://github.com/openmobilityfoundation/mobility-data-specification/blob/main/agency/README.md.

[9] *Mobility Data Specification (MDS), Provider*. Last accessed 3 October 2022. URL: https://github.com/openmobilityfoundation/mobility-data-specification/blob/main/provider/README.md.

[10] *Mobility Data Specification (MDS), Provider Authorization*. Last accessed 7 October 2022. URL: https://github.com/openmobilityfoundation/mobility-data-specification/blob/main/provider/auth.md.

[11] *Mobility Data Specification (MDS), Understanding APIs*. Last accessed 3 October 2022. URL: https://github.com/openmobilityfoundation/governance/blob/main/technical/Understanding-MDS-APIs.md.

[12] *Models for A Mobility Case Study for Validating Attack Propagation Analyses*. Last accessed 27 October 2022. DOI: 10.5281/zenodo.7257735. URL: https://doi.org/10.5281/zenodo.7257735.

[13] *National Vulnerability Database (NVD)*. Last accessed 12 June 2022. URL: http://nvd.nist.gov/vuln.

[14] *Open Mobility Foundation (OMF)*. Last accessed 30 May 2022. URL: http://openmobilityfoundation.org.

[15] *Open Web Application Security Project (OWASP)*. Last accessed 12 June 2022. URL: http://owasp.org.

[16] *OWASP Top 10, A01 (Broken Access Control)*. Last accessed 20 June 2022. URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

[17] Per Runeson and Martin Höst and Austen Rainer and Björn Regnell. *Case Study Research in Software Engineering – Guidelines and Examples*. Feb. 2012. ISBN: 9781118104354. DOI: 10.1002/9781118181034.

[18] Ralf Reussner and Heiko Koziolek et al. "The Common Component Modeling Example". In: vol. 5153. Jan. 2007, pp. 16–53. ISBN: 978-3-540-85288-9. DOI: 10.1007/978-3-540-85289-6_3.

[19] Ralf Reussner and Robert Heinrich et al. "Architecture-based change impact analysis in cross-disciplinary automated production systems". In: 146 (Dec. 2018), pp. 167–185. DOI: 10.1016/j.jss.2018.08.058.

[20] Ralf Reussner, Anne Koziolek, and Erik Burger et al. *Modeling and Simulating Software Architectures - The Palladio Approach*. 2016.

[21] Per Runeson and Martin Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empir Software Eng* 14 (2008), pp. 131–164. DOI: https://doi.org/10.1007/s10664-008-9102-8. URL: http://dx.doi.org/10.1007/s10664-008-9102-8.

[22] F. Schull, Janice Singer, and Dag Sjøberg. *Guide to Advanced Empirical Software Engineering*. Jan. 2008, pp. 1–388. ISBN: 978-1-84800-043-8. DOI: 10.1007/978-1-84800-044-5.

[23] *Uber*. Last accessed 13 June 2022. URL: http://uber.com.

[24] Vincent Hu and D. Richard Kuhn and David Ferraiolo. "Attribute-Based Access Control". In: *Computer* 48 (2015), pp. 85–88. DOI: http://dx.doi.org/10.1109/MC.2015.33.

[25] Maximilian Walter, Robert Heinrich, and Ralf Reussner. "Architectural Attack Propagation Analysis for Identifying Confidentiality Issues". In: (2022), pp. 1–12. DOI: 10.1109/ICSA53651.2022.00009.

[26] Claes Wohlin. "Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study?" In: *Information and Software Technology* 133 (Jan. 2021), p. 106514. DOI: 10.1016/j.infsof.2021.106514.

[27] *Worldwide market volume of the mobility-as-a-service market in the years of 2018 and 2026*. Last accessed 13 June 2022. URL: https://de.statista.com/statistik/daten/studie/984840/umfrage/weltweites-marktvolumen-des-mobility-as-a-service-marktes/.