

The present work was submitted to  
the RESEARCH GROUP  
SOFTWARE CONSTRUCTION

of the FACULTY OF MATHEMATICS,  
COMPUTER SCIENCE, AND  
NATURAL SCIENCES

MASTER THESIS

**Towards a software  
engineering view of security  
for microservice-based  
applications**

presented by

**Brian Sinkovec**

Aachen, November 25, 2022

EXAMINER

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr.-Ing. Ulrike Meyer

SUPERVISOR

Alex Sabau, M.Sc.



# Danksagung

Zunächst bedanke ich mich aufrichtig bei meinem Betreuer und Mentor, Alex Sabau. Du hast es mir ermöglicht, an diesem spannenden und fesselnden Thema zu forschen. Ich schätze und bedanke mich sehr für die gemeinsame Zeit und das Engagement, das du für diese Thesis aufgebracht hast. Die lehrreichen und humorvollen Diskussionen mit dir werde ich sehr vermissen.

Ich möchte mich ebenfalls bei meinen beiden betreuenden Professoren bedanken. Durch Prof. Dr. rer. nat. Horst Lichter erhielt ich wertvolles Feedback sowie die Anerkennung des wissenschaftlichen Beitrags dieser Arbeit. Bei Prof. Dr.-Ing. Ulrike Meyer bedanke ich mich für die interessanten Gespräche zu ihren Vorlesungen und ihre Unterstützung während meiner Thesis. Beiden Professoren möchte ich danken, dass sie meine Leidenschaft für die Informatik, konkret in der IT-Sicherheit und Softwareentwicklung, gefestigt haben.

Weiterhin möchte ich mich bei allen Teilnehmern der Evaluation für ihre gespendete Zeit und Mühen bedanken. Durch ihr Feedback erhielt ich wertvolle Einsichten und Verbesserungsvorschläge, die ich gerne in zukünftigen Arbeiten realisieren möchte.

Schließlich, und von ganzem Herzen, bedanke ich mich bei meiner Familie und meinen Freunden. Auf eure Unterstützung konnte ich mich, auch in schwierigen Zeiten, immer verlassen. Besonders bedanke ich mich bei Elena, Robin, Christian, Björn, Manuel, Max, Florian und Marc. Die gemeinsame Zeit während unseres Studiums werde ich nie vergessen und die wahren Freundschaften, die dadurch entstanden sind. Meinen größten Dank richte ich an meine Eltern und meine Geschwister. Robin und Jacqueline, ihr habt mir stets euer offenes Ohr geschenkt und mich immer unterstützt. Papa, du hast mir gezeigt, was im Leben wichtig ist woran ich festhalten kann. Leider wurdest du uns zu früh genommen, ich vermisse dich sehr. Mama, durch dich konnte ich meine Leidenschaft für die Informatik entdecken und zu meiner Aufgabe machen. Du bist immer für mich da und hast mich in allen Lebensphasen unterstützt. Ich bin dankbar, dass ich durch dich meinen Weg gefunden habe. Danke.

*Brian Sinkovec*



# Abstract

Nowadays, security is one of the most important minimum requirements of software applications. In order to meet these requirements, a deep understanding of the software architecture, the security mechanisms the architecture contains, and the effect of these solutions on security and other quality attributes is required. Thus, modeling security in software architectures is a central component of the software development lifecycle.

Modeling approaches so far illuminate the security of a software system only from an attacker's perspective. Threat modeling techniques identify the existing threats and risks of a system. However, software architects and engineers need tools and methods to express and ensure the security of a software system from a constructive side.

In this thesis, we explore which security mechanisms exist for microservice-based applications and how modeling these techniques is possible. For the identification and collection of security mechanisms, we conduct a Systematic Literature Review (SLR). From the gathered information we compile a set of catalogs that classify and describe the security mechanisms based on different properties. Based on the insights of the SLR and other related work, we propose a security metamodel for the creation of architectural documentation. This metamodel divides the description of security into six different views, each of which defines its own set of requirements that must be addressed by the respective models. To validate our contributions, we performed semi-structured interviews with research and industry experts who reviewed the modeling approach using security views.

With these efforts, we propose a methodology for modeling and describing security in software architectures. We show how a software engineering perspective can be employed to conclude the effectiveness of security controls in a software architecture. Through the proposed definitions and models, we create a basis for a new research direction targeting the analysis of the security quality attribute from a software development point of view.



# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Research questions . . . . .	3
1.2. Goals and contributions . . . . .	3
1.3. Structure of this thesis . . . . .	4
<b>2. Foundations</b>	<b>7</b>
2.1. Software architecture modeling . . . . .	7
2.2. Microservice architectural style . . . . .	11
2.3. Security in software engineering . . . . .	12
2.4. Security design concept . . . . .	14
<b>3. Related work</b>	<b>17</b>
3.1. Systematic Literature Reviews and Mapping Studies on MSA security . .	18
3.2. Views of security in software engineering . . . . .	20
<b>4. Catalog of security design concepts for microservices</b>	<b>27</b>
4.1. Systematic Literature Review report . . . . .	29
4.2. Review result . . . . .	40
<b>5. Defining and modeling a software engineering view of security</b>	<b>51</b>
5.1. Research method . . . . .	52
5.2. Software engineering security metamodel . . . . .	52
5.3. Security views and concerns for microservices . . . . .	55
5.4. Definition of security . . . . .	62
<b>6. Evaluation</b>	<b>65</b>
6.1. Evaluation method . . . . .	65
6.2. Evaluation results . . . . .	72
<b>7. Discussion</b>	<b>79</b>
7.1. Research question findings . . . . .	79
7.2. Evaluation discussion . . . . .	81
7.3. Threats to validity . . . . .	82
<b>8. Conclusion and future work</b>	<b>85</b>
8.1. Summary . . . . .	85
8.2. Future work . . . . .	86

<b>A. Results of SLR trial searches</b>	<b>89</b>
<b>B. Catalog of security design concepts</b>	<b>91</b>
B.1. Catalog of security design principles . . . . .	91
B.2. Catalog of security activities . . . . .	94
B.3. Catalog of security tactics . . . . .	98
B.4. Catalog of architectural security patterns . . . . .	105
B.5. Catalog of security protocols . . . . .	113
B.6. Catalog of IDS/IPS approaches . . . . .	118
<b>C. Case study materials</b>	<b>121</b>
<b>D. Evaluation transcription</b>	<b>125</b>
<b>Bibliography</b>	<b>139</b>
<b>Acronyms</b>	<b>157</b>



# List of Definitions

2.1.	Software architecture [ISO11b]	8
2.2.	Concern [ISO11b]	9
2.3.	Architecture view [ISO11b]	9
2.4.	Architecture viewpoint [ISO11b]	9
2.5.	Quality attribute (QA)	10
2.6.	Quality model	10
2.7.	Architectural style [GS94; Cle+10]	11
2.8.	Microservice architecture (MSA) [FL14; New15]	11
2.9.	Asset	12
2.10.	Threat [Shi07]	12
2.11.	Security goals [ISO11a; Shi07; CH13; SB15]	13
2.12.	Security design concept [KK21]	15
4.1.	Security design principle [Kan03]	41
4.2.	Security activity	42
4.3.	Security tactic [BCK21]	44
4.4.	Architectural security pattern [BCK21]	46
4.5.	Security protocol	47
4.6.	Intrusion Detection/Prevention System (IDS/IPS)	49
5.1.	Security (in Software Engineering)	63



# List of Tables

4.1. The list of candidate scientific databases that were considered during the trial searches. . . . .	31
4.2. The results of the SLR's final search. . . . .	33
4.3. The exclusion criteria considered during the SLR's study selection phase. . . . .	34
4.4. The data extraction form used during the data extraction phase of the SLR. . . . .	36
6.1. Evaluation questionnaire. . . . .	66
A.1. The results of SLR trial searches. Each row depicts the number of publications retrieved from the selected scientific databases. . . . .	89
D.1. Interview transcriptions, participants 1-4. . . . .	126
D.2. Interview transcriptions, participants 5-7. . . . .	134



# List of Figures

2.1. A conceptual model of the definitions proposed in the ISO/IEC/IEEE 42010 standard for architecture descriptions [ISO11b]. . . . .	8
4.1. An overview of our applied SLR method. . . . .	28
4.2. The applied data synthesis procedure of the SLR. . . . .	37
5.1. The metamodel for software architecture security. . . . .	53
6.1. The authentication view case study, including the initial Spring Petclinic architecture (a), frontend (b) and backend (c) authentication models. . . .	69
C.1. The initial Spring Petclinic architecture model. . . . .	121
C.2. Illustration of trust and unauthenticated communication paths in the MSA.121	
C.3. First architectural context, the frontend. . . . .	122
C.4. The frontend authentication model. . . . .	122
C.5. Sequence diagram indicating authentication method of clients. . . . .	123
C.6. Second architectural context, the backend. . . . .	124
C.7. The backend authentication model. . . . .	124



# 1. Introduction

The journey of a thousand miles  
begins with a single step.

---

LAO TZU

## Contents

---

1.1. Research questions . . . . .	3
1.2. Goals and contributions . . . . .	3
1.3. Structure of this thesis . . . . .	4

---

In a world that is becoming increasingly digital, complex and interconnected, software applications are playing an ever more important role. The trend of the last decades shows that software applications have gained indisputable relevance in almost every field [Sim05; Sch06; Szo+20]. Furthermore, it is an essential part of software applications that they are resilient and maintainable [BDP06], manage complexity effectively [DD09], and meet constantly changing requirements [YZ10].

To meet these requirements, the discipline of software engineering emerged many years ago [Ham18]. At its core, software engineering deals with the systematic development of software applications. Among others, this discipline is confronted with questions like: how to effectively design maintainable software? How to test whether the application meets its requirements and works as expected? How to ship software products seamlessly to the customer? Is our application secure?

A concept that was established early in software engineering is the development of software architectures [SC06]. Software architectures are representations of systems. Similar to the blueprints of a building, software architectures represent the elements, structures, relations, and dependencies from different angles by creating multiple views of the software system [Kru95; PW92; BCK21]. In addition to describing a system, the architecture also serves to evaluate its quality.

Just as buildings can have different construction styles - such as that of a bungalow or a semi-detached house - software architectures can follow different architectural styles. One architectural style that has become popular in recent years, especially at large companies such as Amazon, eBay or Netflix [Hof15; Ngu20], is the *Microservice Architecture (MSA)* pattern. MSAs are characterized by the fact that the business logic of the application is divided into many small services. Each of these microservices is an isolated instance that runs in its process and communicates with other microservices over network interfaces to execute its functionality. In this way, complex business processes can be broken down into small units that are quickly developed and can be combined to form larger business logic.

The advantage of this architectural style lies in maintainability - since small modular services can be implemented more quickly - and in scalability - since corresponding microservices can be added easily when the load increases [FL14; Dra+16]. The spread of this style has been reinforced by trends such as DevOps, containerization and the shift to the cloud [Ham19]. DevOps is a set of practices and principles that promotes cross-functional collaboration of traditional development and operation teams to accelerate software development and provide increased quality [Kim+21].

Although it seems like the microservice pattern tackles many problems, it also introduces new challenges [STH18]. Designing and orchestrating hundreds or thousands of microservices quickly becomes a problem. Furthermore, evaluating the quality of such architectures has not been explored in-depth, neither in research nor in industry [Li+21]. In particular, it is unclear to what extent this style affects the security and other quality attributes of such applications, as well as what implications and constraints arise from it. There are quality models such as the ISO/IEC 25010 standard [ISO11a] which also cover security as a quality attribute. However, these models define security only on a rather abstract level and do not consider the choice of architectural style. For instance, various security-related techniques, such as secret management, tend to be more difficult in microservice-based applications than in their monolithic counterparts [Bil+22; KMM18]. We argue that this goes hand in hand with the high degree of distribution in MSAs. This perceived impact on the increased complexity of such activities is not reflected in the quality model of the ISO/IEC 25010 standard.

Besides the fact that existing quality models do not capture the influence of the microservice architectural style on several quality attributes, measuring and systematically describing security in such architectures is difficult. Different aspects of security that need to be satisfied are commonly known in terms of security goals, e.g., confidentiality [Shi07; Sta10]. However, the proposal of techniques to be used in such architectures rarely defines or evaluates how they influence these security goals, nor how it affects other quality attributes such as performance, maintainability, or complexity. Often the knowledge that is available to software architects in terms of available security measures is unstructured and thus makes it difficult to reason about the various techniques and how these affect the system.

Additionally, we perceive a gap when it comes to modeling security in software architectures. Modeling approaches and tools exist to describe the security of such an architecture from the attacker's perspective, e.g., *threat modeling* [Sho14]. These views clarify to what extent software applications are vulnerable, which weaknesses can be exploited, and what consequences this has on the security goals of the application. However, there is a lack of modeling concepts and notations that represent the security of an architecture from a *software engineering perspective*. A software engineering perspective of security takes into account the established design of the architecture to secure its assets, i.e., what design patterns and tactics are employed, what components are used for authentication and authorization, and how the system monitors and controls security-related events, among other things. Such a perspective is intended to support software architects and engineers to better understand and communicate the enforced



design decisions regarding the security of the application. Furthermore, with such a perspective we take a step towards a better understanding of security mechanisms and design decisions and their influence on the quality attribute security.

## 1.1. Research questions

In this work, we want to investigate what are the key aspects of security in software architectures and microservice-based applications, specifically. In particular, we aim to explore which security-related techniques and methods are used in such applications and to what extent they influence the quality attribute security as well as other quality attributes. Our goal is to identify and analyze the perceived gap and try to create a security model which takes the architectural and engineering point of view into account. Furthermore, we provide a clear definition of what security is and what it is not in this context. Thus, the core research question which drives this thesis is the following.

**(RQ0)** *How can we model the security of software architectures from a software engineering perspective?*

We pose the following research questions that need to be answered in this work to reason about our central question. First, we identify which security-related *design concepts* exist for microservice-based applications. In short, security design concepts are modeling building blocks embeddable into the architecture. We propose a clear definition of this term in Chapter 2. Furthermore, we restrict ourselves to this particular architectural style to limit the possible solution space of design concepts and to provide more precise results in this regard. The results we aim for are a structured description and classification scheme of these design concepts. These goals are formulated in our first research question.

**(RQ1)** *How can we describe and classify security-related design concepts for software applications in microservice architectures?*

With the former research question, we want to answer *what* solutions exist for a particular architectural style that can be modeled in the respective software architecture. The following research question tackles the modeling aspect of security, i.e., *how* can we model the requirements and solutions in an expressive and meaningful way. Specifically, we aim to identify which modeling *building blocks* are required to achieve this goal and to ultimately define security in the software engineering context.

**(RQ2)** *Which modeling building blocks can be used to describe and define the security of software architectures?*

## 1.2. Goals and contributions

The primary goal of this thesis is to shed light on the quality attribute security of software applications. We pursue to create a security model and, correspondingly, a new definition

of security to be used when creating, modeling, and evaluating the security of software architecture. This work is intended to provide developers and software architects with a guide to understanding which aspects of security are relevant to the development of their systems and to present the broad knowledge of security techniques in a structured format. The model and definition shall also serve as a basis for future research working towards a quality model in terms of security.

In this thesis, we provide three main contributions. Our first contribution is a catalog of security design concepts that apply to microservice-based applications. The purpose of this catalog is to structure the available measures that exist to harden such a software application from an architectural viewpoint, as well as to provide a first approach toward a searchable and navigable knowledge database to be used by software architects. To accomplish this, we perform an adapted version of Kitchenham et al.'s *Systematic Literature Review (SLR)* methodology [KC07]. In particular, we create a review protocol and define a search and selection strategy to identify and select the appropriate research studies to create such a catalog. Using this approach ensures that our results are transparent and reproducible.

Based on the insights from the conducted SLR as well as other related work, we propose a security metamodel for the description, modeling, and assessment of security in software architectures. The security metamodel illustrates the most important concepts and relationships when modeling the security of such an architecture. The core of this metamodel defines the notion of a *security view*. In short, a security view provides the link between related security concerns and requirements and corresponding security design concepts, as well as their influence on the security quality attribute. We consider a partition into different security views as necessary to cope with the complexity of the security-relevant design. Furthermore, we present six of those security views and describe what these views encompass and which security concerns each view needs to consider. These security views are supposed to be created during the design and engineering of an architecture and should capture certain security aspects of such a system, e.g., authentication.

Lastly, we contribute a new definition of security in the context of software engineering. This definition shall provide security researchers and architects a means of communicating the boundaries of what security in this context entails.

### 1.3. Structure of this thesis

The structure of the remaining thesis is as follows. Chapter 2 introduces the foundations of software architectures, architecture modeling, the microservice architecture pattern and security in software engineering in order to establish a common understanding of these concepts. In this chapter, we provide definitions for the terms that are relevant to this thesis. To get a clear picture of already existing approaches and contributions to MSA security, we review related work on this topic in Chapter 3. In particular, we cover other systematic reviews and mapping studies, as well as other models that provide a view of the security of software applications. In Chapter 4, we tackle our

first research question (**RQ1**) by describing the procedure of our adapted Systematic Literature Review method. We depict the individual steps of our SLR and the results of each step. Afterward, we present the final result of the SLR, namely the catalog of security design concepts for microservice-based applications. Specifically, in this chapter, we explain the rationale and the schematics of each security design catalog and refer to the corresponding catalog tables in Appendix B. In Chapter 5 we address our second research question (**RQ2**) by presenting and explaining the security metamodel as well as the six security views for MSAs we identified in this thesis. We also derive a definition of security to be used in the context of software engineering. To evaluate the developed security metamodel and the modeling approach based on security views, we performed semi-structured interviews with research and industry experts who reviewed our proposed model based on a case study. Chapter 6 describes the method and results of our evaluation. Chapter 7 discusses the various methods and results of this thesis. Concretely, we discuss to what extent we answered our three research questions as well as the limitations of our work. The definitions as well as the presented models shall serve as a basis for further development and research on modeling security in software architectures. Therefore, we summarize and conclude this thesis in Chapter 8 and provide a brief outlook for future work.



## 2. Foundations

The beginning of wisdom is the  
definition of terms.

---

SOCRATES

### Contents

2.1. Software architecture modeling . . . . .	7
2.1.1. Quality of software applications . . . . .	10
2.2. Microservice architectural style . . . . .	11
2.3. Security in software engineering . . . . .	12
2.3.1. Security in microservice architectures . . . . .	14
2.4. Security design concept . . . . .	14

In this chapter, we introduce essential definitions and terms for modeling software architectures and specifically for the security of MSAs. First, we define the term software architecture in general. The basis of definitions we choose is taken from the ISO/IEC/IEEE 42010 standard [ISO11b], a standard for the development of architecture descriptions. We introduce the substantial concepts of an architecture view, an architecture viewpoint and the associated concerns as well as the quality of a software application. We then discuss and define the core aspect of the microservice architectural style. Afterward, we review the relationship of security in software engineering, define the essential concepts and then give a brief overview of the security challenges that typically occur in MSAs. Finally, we propose a definition of *security design concepts* to frame building blocks of security design for software architectures.

### 2.1. Software architecture modeling

Software applications are used in certain application domains to solve domain-specific problems. Each of these software applications consists of many software elements, e.g., software components, which are related to each other in a certain way. The structure, which is formed by such a software application, is called *software architecture*. In addition to software architectures, there exist further architecture types, like hardware architectures, network architectures, or also enterprise architectures [PW92]. In the context of software architectures, software elements form the structure of these architectures [GS94; Cle+10].

Many definitions exist that describe what a software architecture is composed of [SEI10]. In this thesis, we define the software architecture term and its related con-

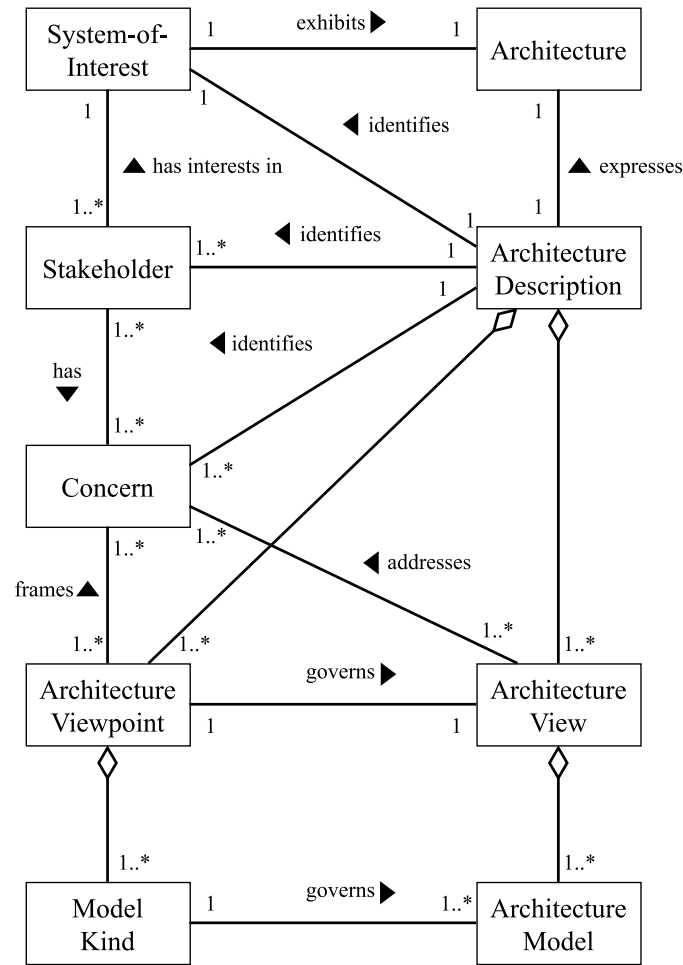


Figure 2.1.: A conceptual model of the definitions proposed in the ISO/IEC/IEEE 42010 standard for architecture descriptions [ISO11b].

cepts like architecture view and viewpoint as in the ISO/IEC/IEEE 42010 standard [ISO11b]. Figure 2.1 captures the essential concepts of a software architecture and its relationships according to this standard.

#### Definition 2.1: Software architecture [ISO11b]

A *software architecture* consists of the fundamental concepts and properties of a system in its environment embodied in its software elements, relationships, and in the principles of its design and evolution.

Thus, a software architecture describes not only the software elements and the interactions between them but also the collection of design decisions and principles that underlie the architecture. The form of the architecture is significantly influenced by the

decisions and developments of the software architect, but also by the *concerns* of other stakeholders. They represent the interest of several stakeholders, including the software architect, developers, and project managers but also customers and end-users.

**Definition 2.2: Concern [ISO11b]**

A *concern* is any form of interest of one or more stakeholders. They are stated as questions, functional or non-functional requirements to be addressed in the software architecture.

Many different types of concerns exist in many software architectures, e.g., functional, performance and security concerns. In order to formalize these decisions and concerns, *architectural descriptions* are used. An architecture description is formed by a collection of diverse *views*, which describe the architecture in each case from different perspectives. This subdivision into views is necessary since it is not possible to seize the complexity of software architectures in one model. Each view considers therefore the concerns relevant to the view.

**Definition 2.3: Architecture view [ISO11b]**

An *architecture view* expresses the software architecture from a specific perspective addressing related concerns of one or more stakeholders, using the conventions established by its viewpoint.

In order to address these concerns, an architecture view is composed of *architecture models*. These models are the concrete representations of a system, e.g., UML [OMG17] diagrams, tables, or instances of a domain-specific language. Architecture models are typically reusable across multiple views. *Viewpoints* are employed to specify exactly which concerns are considered and which modeling techniques are used to illustrate and address these concerns in such a view, thus being the perspective of an architecture view.

**Definition 2.4: Architecture viewpoint [ISO11b]**

An *architecture viewpoint* governs an architecture view. It sets conventions to create, interpret, and analyze one type of architectural view.

A viewpoint consists of several *model kinds*, each describing notations, modeling methods and languages and analyzing techniques to define architecture models. As an example, the UML meta-model [OMG17] is a model kind, as it defines the essential concepts and relationships of UML diagrams and how to create and interpret them. Concrete instances of this meta-model, such as class diagrams or sequence diagrams, are models that describe the architecture from a specific perspective. UML class diagrams are thus suitable, for example, for structural-related concerns, e.g., of a software engineer interested in building the application.

### 2.1.1. Quality of software applications

The quality of software applications is, among others, one of the reasons why it is important to design good software architectures [BM99]. The quality of a software application is measured by the system's fulfillment of its quality requirements. Quality requirements arise from the concerns of various stakeholders the software architect must consider. For instance, customers may require that the application performs its tasks in a reasonable amount of time, thus expressing their concerns regarding the performance of a system. Quality is measured based on various quality characteristics, or *Quality Attributes (QAs)*. Each QA describes under which aspect a software application is to be evaluated. For example, maintainability is one of the most important QAs of a software application [BWZ17]. The degree of maintainability expresses the effort required to adapt the architecture to new requirements. Maintainability influences the complexity and how comprehensible an architecture ultimately is.

#### Definition 2.5: Quality attribute (QA)

A *quality attribute* is a property of a software application. Quality attributes capture the quality of a software application from different perspectives.

Examples for QAs are, among others, maintainability as mentioned earlier, but also functional stability, performance, usability, and security [OMB07]. Quality models capture the various QAs that are of interest for software applications [SWC10; MMR14]. They define each QA, the sub-characteristics they are composed of and, if the corresponding QA is measurable, how it can be measured. Thus, most quality models hierarchically define the quality aspects of a software application.

#### Definition 2.6: Quality model

A *quality model* defines the QAs comprised in a software application. Each QA may be refined in terms of sub-characteristics. If applicable, a quality model may also define how to measure certain QAs and which measurements are required.

Many quality models have been proposed in the last decades [MMR14]. The most prominent quality model for software products is the ISO/IEC 25010 standard [ISO11a], which defines eight QAs with corresponding sub-characteristics. Although the standard also provides a quality measurement framework [ISO19], it does not provide specific measurements to assess the quality of each attribute quantitatively.

Security was not recognized as a QA until modern quality models. In older models, as in the software quality models of McCall [CM78] or Boehm [BBL76], this was not considered. The aspects, or sub-characteristics respectively, which belong to the security QA, are frequently framed in the literature under the term of *security goals* [Sta10; SB15; WM11]. The security goals which we consider in this work are further defined in Definition 2.11.



## 2.2. Microservice architectural style

Software architectures enable engineers and architects to build software applications to address the stakeholders' concerns and to structure the software elements to achieve business and quality goals. Nevertheless, making the right design decisions to create a good software architecture is difficult. Thus, software architects often embody sets of (proven) design decisions in *architectural styles* [BM99; Cle+10]. An architectural style defines a family of systems that follow a specific structural organization [GS94]. It is also used to compile a vocabulary for software architects to compare and reason about certain principles and design decisions embodied in those architectural styles.

**Definition 2.7:** Architectural style [GS94; Cle+10]

An *architectural style* is a collection of design decisions imposing certain structural or behavioral organization in the architecture. It specifies the elements, relation types, and constraints on how they can be used.

The microservice architectural style is a style that emerged in recent years [RS16; Dra+16; STH18; Bog+19]. Many sources cite the article by Fowler and Lewis [FL14] as one of the first to establish the concept of MSAs. At its core, the design approach of MSAs follows the idea that the software application is divided into a collection of microservices. Each microservice represents an independent unit in the architecture that can be developed and deployed in isolation from other microservices. In most cases, microservices implement a single business function. The entire business logic of the application is then composed of microservices that work together via lightweight communication protocols such as HTTP [Fie+99].

**Definition 2.8:** Microservice architecture (MSA) [FL14; New15]

*Microservice architecture* is an architectural style that is characterized by dividing the application into several, small, and autonomous software units called *microservices*. Each microservice runs independently on its own and communicates with other microservices using lightweight communication protocols.

The microservice style of architecture brings some benefits, but also some drawbacks. Microservices are scalable and resilient, as individual services can be replicated as needed and only parts of the application are affected when errors occur. Furthermore, they simplify and accelerate the deployment process due to their size and independence. Another capability of MSAs is that this architectural style enables polyglot architectures, i.e., each microservice can be implemented with different technologies and programming languages, depending on which tools are best suited for the use case. Thus, microservices seem to have a positive effect on certain QAs like scalability, resilience, and maintainability, among others. One of the primary disadvantages of MSAs is the organizational complexity introduced by this style. Managing hundreds or thousands of microservices

and their orchestration quickly becomes cumbersome. Microservices, due to their dynamic nature, can change their location and need to be located by other microservices accordingly. Maintaining data consistency across microservices becomes difficult when each microservice handles a database on its own [New15; STH18].

Nevertheless, the adoption and interest of the MSA style gained momentum in research and industry since Fowler and Lewis coined this term [FL14]. In particular, recent studies, as depicted in the next chapter, (including this one) explore how and to what extent MSAs affect the security of software applications employing this architectural style.

### 2.3. Security in software engineering

A software application consists of various software and hardware units. In a MSA, the primary software units are microservices, but databases, communication paths between microservices, and containers also appear. Data is processed, stored and exchanged in the software units of an application. Data is then either processed by business functions in order to fulfill the functionality of the software application or is required for application-specific purposes, e.g., dependency versions. Software units make use of hardware resources, e.g., by consuming computing power or memory for storing data. All these units, which occur in a software application, are called *assets*.

#### Definition 2.9: Asset

An *asset* is a resource of a system, either a software unit, business or application data, or a hardware resource such as CPU, memory, or bandwidth.

One primary factor affecting the security in a software application is *threats*. There are many types of threats, e.g., social engineering (phishing), threats that arise when internal employees purposefully want to cause damage or threats from natural disasters [WM11]. In this thesis, we limit the notion of threat to the kind of threats that arise from flaws in the design and implementation in an asset of a software application. Thus, each threat targets one or more assets specifically. Once a threat is exploited, it is referred to as an *attack* [Mie+10]. Such a realization eventually leads to an *impact*, e.g., that unauthorized actors get access to sensitive data or that the system has been compromised and is controlled by attackers.

#### Definition 2.10: Threat [Shi07]

A *threat* is a potential violation of security that can cause harm to one or more assets of a system when exploited.

Threats, therefore, arise from design or implementation flaws, also called weaknesses and vulnerabilities [McG06]. Since such flaws may always exist, the responsibility, therefore, lies with the software engineer to identify potential threats as early as possible and

to take appropriate countermeasures to detect or also prevent them [DJS19]. *Threat modeling* is a common modeling technique used to identify threats [Sho14]. Threat modeling is a process that is typically performed at the requirements or design phase of the *Software Development Life-cycle (SDLC)*. Threat modeling looks at the system from an attacker's perspective to determine what can go wrong. *Threat models* are the result of threat modeling that show which vulnerabilities exist in which assets in the system. There are various threat modeling approaches, including STRIDE [KG99], PASTA [MU15], and attack trees [Sch99]. We present some of these threat modeling views in more detail in Section 3.2.3. Threat models thus form a fundamental security view of the software architecture. They identify and describe the attack vectors that can be exploited by a potential attacker. This view thus addresses concerns that emanate from the perspective of an attacker, e.g., which threats and weaknesses exist in the system? How can these weaknesses be exploited? Which assets are affected by these threats?

The existence of a threat imposes a *risk* that potentially when realized by an attack leads to *damage*. Damage can occur in the form of stolen user data, damaged infrastructure, or even lost reputation of the company, to name a few. The existence of a threat, therefore, compromises the security of a software application. It is important to distinguish to what extent the security is impaired. For this reason, the security QA is divided into different sub-characteristics, also called *security goals* [Shi07; CH13; SB15]. In some quality models, e.g., the aforementioned ISO/IEC 25010 standard [ISO11a], security goals are represented as *quality factors*.

**Definition 2.11: Security goals [ISO11a; Shi07; CH13; SB15]**

A *security goal* is a sub-characteristic of the security QA which describes one aspect of the security of the overall system.

**Confidentiality** The ability of a system to protect its assets, especially its data, from unauthorized access.

**Integrity** The ability of a system to prevent unauthorized modifications to the assets of a system.

**Availability** The ability of a system to provide legitimate stakeholders access to its data and services.

**Authenticity** The ability of a system to successfully identify legitimate and unprivileged stakeholders.

**Authorization** The ability of a system to successfully determine the permissions of a stakeholder.

**Accountability** The ability of a system to trace the actions performed by a stakeholder or system component uniquely to that entity.

**Non-repudiation** The ability of a system to prove that actions have taken place so that these actions cannot be repudiated later.

Both threats and countermeasures can have an impact on certain security goals. Threats arise from software weaknesses. For instance, the storage of user passwords in plaintext is a software weakness, and thus the existence of the threat that the identity of these users could be compromised in the event of a data breach would negatively impact the security goal confidentiality. Countermeasures such as encrypting user passwords, on the other hand, would strengthen the confidentiality of a system.

### 2.3.1. Security in microservice architectures

Security in MSAs is a research and industry field that has become increasingly important in recent years, probably due to the trend created by microservices per se. In the following chapter, we will review some related work on this. Now, we discuss roughly what challenges can arise concerning the security in a MSA.

In principle, the increased degree of distribution and the use of different technologies, i.e., the polyglot architecture, create a broader attack surface in MSAs [Bil+22] than in monolithic applications. Overall, the risk and the amount of possible threats in an MSA increases, since each microservice is potentially vulnerable to individual attacks [HY20]. In addition to client-server communication, each service-to-service communication interface in a MSA must be considered in terms of its security [Per+21]. Furthermore, it may be necessary for each microservice to authenticate each other [YB18a; MCF21a]. Also, fine-granular rules about which microservice is allowed to access which *Application Programming Interface (API)* may increase the complexity of authorization [Per+19]. Finally, the overhead of monitoring each microservice individually and reacting to irregular events increases [Was+21a].

For this reason, security in an MSA must be carefully modeled to avoid potential weaknesses and design flaws and the resulting threats. One way to model this is through threat modeling, as described in the previous section. To the best of our knowledge, no specific threat modeling frameworks exist for MSAs. Furthermore, while threat modeling supports the software engineer in identifying threats, we claim that it is necessary to be able to create additional views and models that adequately reflect security-relevant design elements and decisions made in a software architecture.

## 2.4. Security design concept

In order to sharpen the view on security, we need a more precise understanding of methods and techniques related to the assurance of software security. A clear definition of the concepts we want to present is essential to be able to effectively discuss the design decisions made in a software architecture. For this reason, we now define the term security design concept.

The notion of a design concept was introduced by Kotov and Klein [KK21]. We adapt the definition in three ways. First, we apply the notion of a *security* design concept to the initial definition, to make the scope and purpose of these design concepts clear. Second, we rename the *externally developed components* to *integratable components*, since

we claim that these design concepts are not limited to being externally developed, but share the property to be integratable into the architecture. Third, we introduce the *protocol* design concept to the set of integratable components, since they represent a substantial role in the software architecture, especially in MSAs.

**Definition 2.12: Security design concept [KK21]**

A *security design concept* is an abstract building block from which the design of security of the software architecture is created. We refer to these categories of design concepts:

- Security design principles
- Security activities
- Security tactics
- Architectural security patterns
- Integratable component
  - Software products
  - Application frameworks
  - Technology families
  - Protocols
  - Platforms

Thus a security design concept forms the fundamental building block to model security in software architectures. Concrete design concepts represent the software elements or a set of software elements that are embodied in an architecture. Further, we differentiate between different categories of security design concepts. The difference lies in the application and the characteristics of the individual categories. This distinction is necessary to establish a clear separation of concepts, an effective description, and the possibility for comparative analysis.



## 3. Related work

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

---

MARTIN FOWLER

### Contents

---

3.1. Systematic Literature Reviews and Mapping Studies on MSA security . .	18
3.2. Views of security in software engineering . . . . .	20
3.2.1. Quality models . . . . .	21
3.2.2. Maturity models . . . . .	22
3.2.3. Threat models . . . . .	23

---

In this chapter, we survey related work in microservice security. More specifically, in the first part of this chapter, we look at what systematic literature search works exist on MSA security, i.e., *Systematic Literature Reviews (SLRs)* and *Systematic Mapping Studies (SMSs)*. Most of these studies investigated primary studies published between 2014 and 2021. We explore studies that deal exclusively with scientific articles, i.e., white literature, but also some that analyzed gray literature, i.e., professional articles, blogs, and white papers in more detail. Most of the papers either give an overview of the advantages and disadvantages of MSAs or examine which security-relevant techniques are used in MSAs. However, in contrast to this thesis, none of the SLRs and SMSs presented have created a catalog from the design concepts obtained. To the best of our knowledge, this work is the first attempt to propose such a catalog.

In the second part of this chapter, we examine existing views of security in software engineering. In the previous chapter, we set the fundamentals and defined what belongs to an architectural view. Existing views, which we introduce briefly in the context of this work, are, among others, quality models, maturity models, as well as threat models. Quality models and maturity models provide a view of the architecture that includes the quality and continuous improvement of a product or an organization. The concern of these views is primarily quality assurance itself. Threat models, on the other hand, provide a view of the architecture from the perspective of an attacker. A threat model is intended to help identify threats and risks in a software product in order to determine appropriate countermeasures. The concern of threat models is therefore the identification, and if necessary also the prevention, of threats.

#### 3.1. Systematic Literature Reviews and Mapping Studies on MSA security

Pahl and Jamshidi [PJ16] performed one of the first SMSs on MSAs. In their study, they focused on finding the primary motivation behind using microservices, which techniques, tools, and methods enable the development of this style, as well as exploring the main research issues that need to be tackled in future work. The mapping study selected 21 publications including non-peer-reviewed work such as books and theses. The study revealed a lack of research at that time, with only conceptual work available. The authors described this as a sign of immaturity, which is not surprising since this work could only give a temporal overview of studies published between 2014 and 2015. They also provided a reference model for MSAs based on patterns and principles, as well as a characterization framework to classify the identified primary studies according to methodological support, architectural support, platform/tool support, and application.

Alshuqayran et al. [AAE16] also conducted a SMS on the microservice architectural style. In contrast to Pahl and Jamshidi, they investigated what the architectural challenges of this style are, which diagrams and views are used to model microservices, and which QAs are related to MSAs in the literature. The study included 33 peer-reviewed publications from 2014 to 2016. Although security was identified as relevant QA, it received little attention compared to QAs like scalability and maintainability. Only 5 publications of the investigated population in the SMS mentioned security as an important QA.

Soldani et al. [STH18] performed a gray literature SLR to investigate the gains and pains of microservices in the industry. They collected 51 industrial studies published between 2014 and 2017. Pains and gains were categorized into different stages, e.g., design, development and operation, and into concerns, e.g., architecture, security, monitoring and others. The primary pains of developing microservice applications according to this paper are the size and complexity of MSAs, dealing with fine-grained access control policies and data consistency. However, microservices bring also gains such as bounded contexts (that are easier to develop), firewalling, loose coupling, and technology freedom.

Hannousse and Yahiouche [HY20] conducted a SMS on the security of MSAs. The study selected 46 peer-reviewed publications as primary studies from 2011 and onwards. The focus of their study was to explore and classify the security threats and risks specific to MSAs as well as mitigation techniques and methods and at which level of the architecture these techniques are applicable. The selected techniques were classified based on four aspects, namely the application layer at which the technique applies, the purpose of the technique, the threat it mitigates, and the target platform. Furthermore, the authors concluded the paper by proposing an ontology for securing MSAs. The ontology captures the various aspects of a security technique according to their classification scheme. However, it fails to establish the relations and dependencies between these aspects, e.g., that the purpose of a technique (for instance, authentication) is related to spoofing attacks.



Pereira-Vale et al. [Per+19] performed a SMS to explore security mechanisms used in microservice-based applications. The study obtained 26 primary studies which were published between 2015 and 2018. The authors reported that authentication, authorization and credentials were the primary security mechanisms proposed. Other security mechanisms they identified were TLS, access control, *Infrastructure-as-a-Service (IaaS)*, *Role-based Access Control (RBAC)*, and others. Further results of this study were that (1) most publications consider detecting or mitigating attacks, but not recovering from them, (2) most security mechanisms are validated by performing case studies and experiments, and (3) security patterns for microservice-based systems were not found.

The same authors [Per+21] also conducted a multivocal literature review on security in microservice-based systems, i.e., a SLR including white and gray literature articles. 34 white literature studies and 36 gray literature publications were selected in this study. The purpose of this study was to investigate and classify security mechanisms, the security scope, and the security context of studies in microservice-based systems. The authors explained that the rationale behind this was to explore whether some mechanisms were mature and to find potential research gaps in this area. A list of security mechanisms grouped according to their purpose, i.e., authorization, identity management, access control, secure communication, logging & monitoring, filtering, execution control, and their scope, i.e., application security, implementation security, security evaluation, threat modeling, and general security architecture, was provided. The researchers also pointed out that there is little work toward a systematic methodology to develop secure applications. Also, new proposals of security techniques need to be systematically categorized to prevent the re-publication of already proposed solutions.

Li et al. [Li+21] performed a SLR addressing the QAs of MSAs, selecting 72 peer-reviewed publications published until 2018. In contrast to earlier studies, this work identified that security is indeed a primary concern of microservice systems due to its distributed nature. The increasing complexity that is introduced by this style may be exploited by attackers. Establishing trust between microservices is another major concern that is not tackled in depth. In total, 8 of the 72 publications addressed security as QA of MSAs by proposing three security tactics that address the aforementioned problems. The security tactics proposed were Security Monitor, Authentication and Authorization, and Intrusion defender. Each of these tactics was described in terms of motivation for its purpose, a description, as well as constraints and dependencies introduced by this tactic.

Billawa et al. [Bil+22] conducted a gray literature SMS with 57 selected publications between 2011 and 2021 to explore MSA security challenges and recommendations or technologies to address these challenges. The main challenges they identified were trust between microservices, a larger attack surface, testing, container management, low visibility, secret management, and the polyglot architecture of MSAs. The proposed security solutions were classified into the categories best practices (defense in-depth, DevSecOps), methods (authorization frameworks like OAuth 2.0), deployment (real-time monitoring, certificate & configuration management), development (orchestration, static and dynamic security testing), and patterns (API gateway, circuit breaker).

A recent SLR on MSA security was performed by Berardi et al. [Ber+22]. In their work, the authors considered a large corpus of 290 peer-reviewed publications. Unlike most other SLRs in this research area, this work compiled a catalog of dichotomous research questions, i.e., with yes-or-no answers. Each of these research questions was categorized into one of four macro groups: threat model, security approach, infrastructure, and development. The threat model group addressed the adoption of STRIDE or other threat modeling approaches. Only a portion of the publications mentioned the usage of a threat model, and if they did, it was used tailored to the specific use case of their proposed solution. The authors claim that a possible explanation is that there is no common threat modeling approach for microservices, due to its difficulty in making it specific for microservices while avoiding the problem threat explosion, i.e. when the effort of considering and prioritizing all threats exceeds the benefits of managing them [Wuy+18]. The security approach group considered specific security solutions, e.g., *Intrusion Detection System/Intrusion Prevention System (IDS/IPS)* solutions, and the role that microservices play from a security perspective. The most common approach is to address specific problems, e.g., dealing with authentication, rather than following a general approach considering the whole application stack. The infrastructure group asked about infrastructure configurations considered, e.g., IaaS. The study found out that most orchestrating and monitoring solutions are centralized, and that IaaS and service discovery are emerging topics in the context of MSA security. The development group investigated established development practices and processes for security purposes. Agile approaches, such as DevOps or DevSecOps, *Continuous Integration/Continuous Delivery (CI/CD)*, as well as domain-driven or model-driven methods are primarily used to address MSA security concerns. Furthermore, the authors created a correlation matrix among the research questions to understand which aspects of MSA security most commonly appear together and which are mutually exclusive.

## 3.2. Views of security in software engineering

In addition to the presented secondary studies, a large number of different views of security in software engineering exist. In the following, we will discuss three of these different types of views, namely quality models, maturity models, and threat models.

Quality models serve the purpose of defining and quantifying quality in a software system. The quality of a system is often subdivided into several quality attributes, which in turn are composed of further sub-characteristics in a hierarchical structure. In this thesis, we present the standardized product quality model ISO/IEC 25010 [ISO11a], as well as the more recent Quamoco approach [Wag+12; Wag+16]. Before that, however, we briefly discuss the SQUID approach by Kitchenham et al. [Kit+97] which describes a method for defining quality models. Many other quality models have been developed so far. For further information, Miguel et al. [MMR14] provided an overview of the timeline and differences between developed quality models.

We then describe one established maturity model for assessing an organization's maturity in security. Maturity models offer companies the possibility to measure their

progress and possible improvement potential in a certain discipline by employing a maturity level. We briefly introduce the maturity model *OWASP Software Assurance Maturity Model (OWASP SAMM)* [OWASP20].

Finally, we describe some threat models. In contrast to the previous models, threat models take the perspective of the attacker on a software system. This should help software engineers to better identify and classify threats to their developed products. We describe in this work the threat models STRIDE [KG99] and attack trees [Sch99], as well as some established attack libraries.

### 3.2.1. Quality models

The SQUID approach proposed by Kitchenham et al. [Kit+97] is not a quality model per se but describes a method to define quality models. The authors claim, after evaluating existing quality models such as the McCall quality model [CM78] or the ISO/IEC 9126 [ISO91], that a quality model consists of a structural model and a content model. The structural model describes the essential aspects of a quality model: quality characteristics and sub-characteristics, as well as internal software properties and measurable properties. Content models, on the other hand, must be specifically adapted according to the products and quality requirements of a company. The authors, therefore, claim that a general one-size-fits-all quality model is not suitable for direct use. For this reason, companies and software developers themselves are responsible for designing suitable content models for their application purposes and products. Although we agree with the assertion that quality models must be customized according to the quality requirements of specific products, the lack of quality reference models for security is a problem. This makes it difficult for software developers to effectively measure the security of their products, especially if they lack expertise in software security.

As the successor to the aforementioned ISO/IEC 9216 standard, ISO/IEC 25010 [ISO11a] defines an international standard for software product quality. The standard consists of a hierarchical model, i.e., the quality of a software system is composed of several QAs which in turn consists of several sub-characteristics each, forming a tree-like structure. QAs defined in the standard include, among others, functional stability, performance, reliability, maintainability, compatibility, and security. The standard defines the security QA as *the degree to which a system protects data and information so that users and other systems have access according to their authorization rights*. The sub-characteristics the security QA is composed of are confidentiality, integrity, non-repudiation, accountability, and authenticity. The definitions of the individual sub-characteristics are similar to ours in Definition 2.11, although the standard assumes in each case the degree to which the product or system fulfills the security goal. In contrast to our definition, the QA availability in the ISO/IEC 25010 standard is not considered under the quality characteristic security, but under reliability. However, the standard mentions that availability, in the event of an attack, must also be considered under the aspect of security. In addition, there is the ISO/IEC 25020 [ISO19] standard which defines a reference model for the creation of quality measurements for software products. In order to measure quality characteristics or sub-characteristics, quality measurements

are necessary to determine the fulfillment of the QA. Nevertheless, it is also mentioned in the standard that not every QA is suitable to be measured. Instead, in this case, the quality model serves to define and communicate quality requirements more clearly. We also recognize, due to a lack of scientific or industrial work, that a gap exists when it comes to quality measurements in terms of security. We believe that the lack of quality metrics for security also leads to an inaccurate understanding of security in software applications. The lack of empirical data makes it difficult for software architects to determine how secure their architecture actually is.

The lack of an operationalized quality assessment method in the ISO/IEC 25010 standard was addressed by Wagner et al. [Wag+12; Wag+16] in their Quamoco approach. They identified the existing gap between abstract quality attributes in the ISO/IEC 25010 standard and concrete measurements. Reasons for this are the lack of a meta-model for quality models and a clearly defined, integrated quality assessment method. Furthermore, it is necessary to modularize a quality model, since a single quality model is too large and confusing. For this reason, the authors have created such a metamodel and the Quamoco base model. The metamodel describes the general concepts that such a quality model must contain. At the core of the model is the *factor*, which is an abstract property of an *entity*. The concept of the factor is reused in different levels of abstraction. *Quality aspects*, for example, are specializations of the factor and represent abstract quality goals. These quality objectives are comparable to the QAs from the ISO/IEC 25010 standard. *Product factors* are another, more fine-granular specialization of a factor. These types of factors represent the attributes of a part of the product. Quality aspects as well as product factors can be further refined by appropriate hierarchies. In addition, product factors have an *impact* on certain factors of quality aspects. Product factors in the leaf nodes of the hierarchy must then be quantified by appropriate *measures*. Measures in turn can be refined by other measures, with the most concrete measures using *instruments* to determine their value. The main difference to the ISO/IEC 25010 quality model is therefore the finer granular division of QAs into product factors, which form a hierarchy. The basic model presented by the authors primarily considers the quality aspects maintainability, functional suitability, and reliability. Security was also taken into account in the basic model as a quality aspect. However, there were only 17 product factors for this aspect. The authors justified this with the problem that a quality aspect such as security is based more on dynamic checks than other quality aspects. Furthermore, the Quamoco model was designed to be extended accordingly. Nevertheless, to the best of our knowledge, there are no established extensions of the Quamoco model that consider security in-depth.

#### 3.2.2. Maturity models

OWASP SAMM [OWASP20] is a model that focuses on the maturity level of the security discipline. At its core, this model also defines a hierarchical structure of business functions, security practices, and streams. Each stream defines activities that are necessary to achieve the corresponding maturity level in this discipline. The framework also provides questions and quality criteria for an organization to measure its maturity

level itself. For example, the design business function captures security practices such as threat assessment, among others. One stream under the threat assessment security practice is threat modeling. An organization reaches maturity level 3 in this stream when it can continuously improve and automate the threat modeling methodology. Through such a model, companies can capture which security disciplines and activities are necessary for their software development lifecycle.

### 3.2.3. Threat models

STRIDE [KG99] is a threat modeling approach developed by Microsoft in 1999. The acronym stands for spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege. Each of these threat classes covers a set of threats that harm the same security goal. Threats may be classified into one or more threat classes.

**Spoofing** includes all those attacks in which the attacker obtained any stakeholder's personal information to impersonate that person. This enables the attacker to perform actions on the user's behalf. This class of threats directly harms the *authenticity* of a system.

**Tampering** attacks modify the system or data by an unauthorized entity. An attacker stores malicious data or programs, inserts undetectable network packets, or makes changes to sensitive user information. This class of threats harms the *integrity* of a system.

**Repudiability** is the ability of an untrusted user to deny that he performed an illegal operation. This class of threats harms the *accountability* of a system.

**Information disclosure** threats deal with the leakage of personal or business information to those who are not supposed to have access to it. This class of threats harms the *confidentiality* of a system.

**Denial of service** attacks make the system temporarily or permanently unavailable to legitimate users. This class of threats harms the *availability* of a system.

**Elevation of privilege** deals with attacks that allow an unprivileged user to gain privileged access to certain assets or operations. This threat class harms the *authorization* capabilities of a system.

Dependencies or implications between these threat classes may exist, e.g., an attacker who successfully *spoofed* the identity of a system administrator also has gained an *elevation of privilege*, and may be able to *tamper* with privileged information. The model does not prescribe concretely how to identify threats in a system. It should rather help software engineers and architects view their software applications from an attacker's perspective and find and understand general sets of threats without expert knowledge. Typically, STRIDE is used in conjunction with *Data Flow Diagrams (DFDs)*. For

instance, various threat modeling tools like OWASP Threat Dragon [wwwg] or the Microsoft Threat Modeling Tool [wwwe] enable the modeling of DFDs with the STRIDE methodology as their basis.

Attack trees [Sch99] provide a formal yet simple approach to model threats in a software system. As the name indicates, this methodology suggests representing attacks against a system in a tree-like structure. The root node of the attack tree represents the goal of an attacker, e.g., gaining access to a user's personal information stored in a system. Then, sub-nodes are created that indicate how an attacker accomplishes that goal. Sub-nodes are AND nodes or OR nodes, i.e., either all of the AND nodes are required to accomplish the goal, or only one alternative of the OR nodes is required. This way, the attack tree is created iteratively where the attacker's goals and how to reach each goal are indicated by nodes and their sub-trees. Further steps using this methodology can be to label each node with a likelihood or impact or the costs of ignoring that risk. This can be used to calculate the overall risk or impact when such a threat is exploited. Another advantage of attack trees is that they can be composed to build attack tree libraries. Furthermore, STRIDE and DFDs are compatible with attack trees, i.e., they can be used in conjunction to find threats in an application.

STRIDE and attack trees provide software engineers with modeling tools to view their applications from the attacker's perspective. However, these approaches provide only an abstract view that may not cover all threats. For this reason, detailed lists and catalogs of vulnerabilities, weaknesses, and attack patterns have been created and maintained. The purpose of these catalogs is to provide a detailed and fine-grained, schematic view of software weaknesses and vulnerabilities in a searchable and navigable way. The most popular attack libraries are the following.

The *Common Attack Pattern Enumeration and Classification (CAPEC)* [wwwa] catalogs known attack patterns that are exploitable by adversaries. The concept for this catalog is derived from the paradigms of software design patterns [Gam+94]. Thus, "attack patterns are design patterns for attackers" [wwwb]. Each attack pattern in this catalog is described in terms of the execution flow, the likelihood and severity, mitigation techniques as well as relationships to other attack patterns. The perspective of an attack pattern is always from an attacker's point of view describing the exploration, experimentation and exploitation phases that an attacker performs which comprises the attack pattern. An example of a well-known attack pattern is *Cross-Site Scripting (XSS)*, uniquely identified as CAPEC-63<sup>1</sup>.

Each attack pattern is also related to a set of weaknesses which are cataloged in the *Common Weakness Enumeration (CWE)*. The CWE [STM17; wwwd] lists common software and hardware weakness types. In contrast to CAPEC, this view takes the perspective of the software engineer in terms of design or implementation faults that may be introduced by him. Each weakness is composed of a description, how and at which development stage it is typically introduced, consequences in terms of impacted security goals, and demonstrative examples. Potential mitigation techniques and when to apply these techniques are also provided. Attack patterns rely on an underlying software

---

<sup>1</sup><https://capec.mitre.org/data/definitions/63.html>

weakness that is exploitable, thus causing a threat to the system. In the aforementioned example, the XSS attack pattern is related to the *improper neutralization of input during web page generation* weakness (CWE-79)<sup>2</sup>.

Concrete instances of a weakness made public are vulnerability disclosures, which are cataloged in the *Common Vulnerabilities and Exposures (CVE)* database. The CVE [wwwc] is probably the most prominent list of software vulnerabilities. Each disclosed vulnerability is assigned a CVE-ID, a short description, a reference to the software and version in which the vulnerability is contained as well as a score that indicates the severity of the vulnerability. Following our example, an instance of the CWE-79 weakness would be the vulnerability CVE-2021-1879<sup>3</sup>.

CAPEC, CWE and CVE together form a set of related databases that help developers to assess the security of their software application from an attacker's and defender's point of view. In short, CWEs explains the root cause of a vulnerability, CAPEC describes how the weakness can be exploited by an attacker, and CVEs demonstrates specific instances of a weakness that are exploitable in existing software products.

*Adversarial Tactics, Techniques & Common Knowledge (ATT&CK)* [wwwf] is another attack library that lists specific tactics, techniques, and procedures that adversaries use. The techniques are categorized according to the technology domain, i.e., enterprise or mobile with domain-specific platforms, and the tactic representing the attacker's objective, i.e., the reason for acting. Gaining initial access, persisting access to a system, or privilege escalation are a few examples of these tactics. Similar to CAPEC, the ATT&CK library offers another view from the viewpoint of an attacker. However, they differ in their use cases. CAPEC focuses on application security, associates with CWEs, and is used for threat modeling and penetration testing. On the other hand, ATT&CK focuses on network defenses and defense against advanced persistent threats. It is rather used to compare defense capabilities and to find new threats.

Each of these attack libraries is a comprehensive and detailed database in itself. For instance, CAPEC lists 546 attack patterns, CWE contains 927 weaknesses, the CVE database covers 183528 vulnerabilities, and the ATT&CK library contains 191 techniques with 385 sub-techniques for the enterprise technology domain, with additional techniques for the mobile domain<sup>4</sup>. Although these fine-grained views are required to grasp the whole picture that entails security, knowing every entry of these databases is an impossible task for software engineers and architects. Thus, besides several "Top 25" views that these databases provide, the OWASP Top Ten [wwwh] list is a popular, "standard awareness document" for web application security. This view breaks the comprehensive catalogs that the aforementioned databases provide down to a graspable set of important risks and threats to be considered by every software engineer.

---

<sup>2</sup><https://cwe.mitre.org/data/definitions/79.html>

<sup>3</sup><https://nvd.nist.gov/vuln/detail/CVE-2021-1879>

<sup>4</sup>Numbers accessed on 2022-08-31





## 4. Catalog of security design concepts for microservices

When we learn something new,  
we don't go from 'wrong' to  
'right', rather we go from wrong  
to slightly less wrong.

---

MARK MANSON

### Contents

---

4.1. Systematic Literature Review report . . . . .	29
4.1.1. Need for a review . . . . .	29
4.1.2. Research question . . . . .	30
4.1.3. Developing and evaluating the review protocol . . . . .	30
4.1.4. Identification of research . . . . .	30
4.1.5. Study selection . . . . .	33
4.1.6. Data extraction . . . . .	35
4.1.7. Data synthesis . . . . .	37
4.1.8. Conflict of interest . . . . .	38
4.1.9. Excluded SLR steps . . . . .	39
4.2. Review result . . . . .	40
4.2.1. Security design principle . . . . .	41
4.2.2. Security activity . . . . .	42
4.2.3. Security tactic . . . . .	43
4.2.4. Architectural security pattern . . . . .	45
4.2.5. Security protocol . . . . .	47
4.2.6. Intrusion Detection/Prevention Systems . . . . .	48

---

One aspect of this thesis is to understand to what extent the microservice architecture pattern influences the security of software applications. An architecture pattern reflects a specific structural and behavioral composition of software elements of a software application. Software architectures are significantly shaped by design decisions and applied design concepts by the software architect and other stakeholders. To understand the impact of the MSA pattern on the security of a software application, we need to understand which design concepts exist for these applications and how they work. Therefore, we develop a catalog of security design concepts for microservices as a central research artifact in this thesis. The catalog provides a detailed and schematic description of the different design concepts. With the description and cataloging of security design concepts, we can evaluate the effect of microservices on the architecture's security.

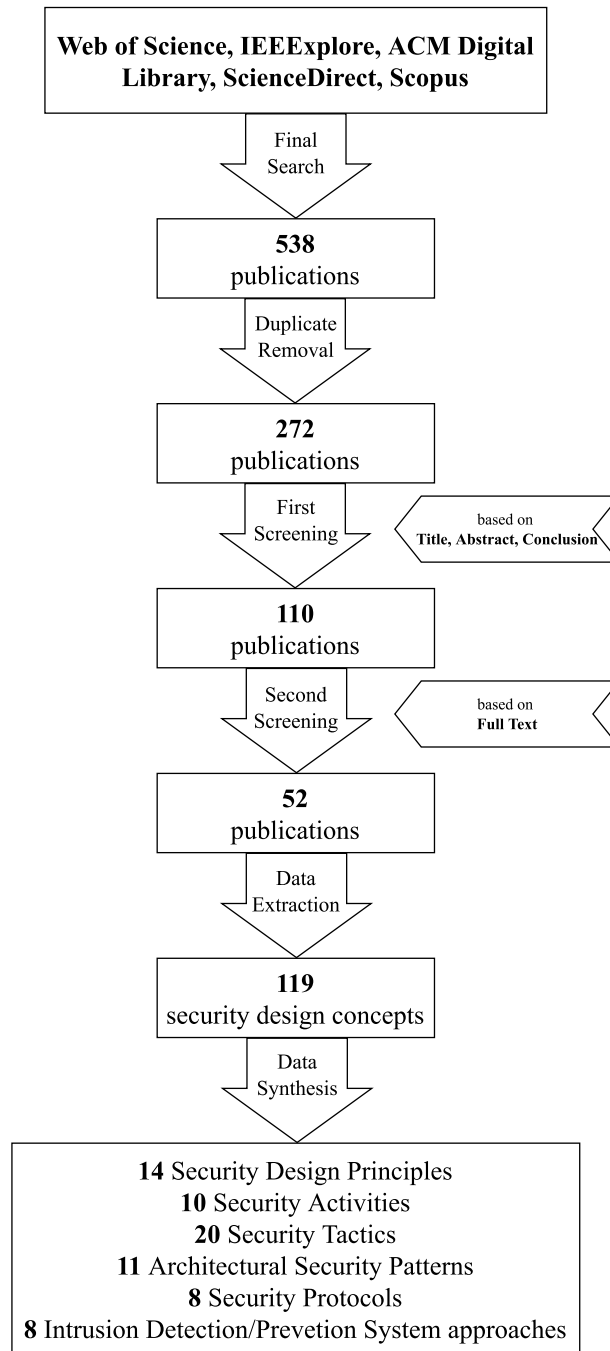


Figure 4.1.: An overview of our applied SLR method. Each step illustrates how many publications were identified/selected, and how many security design concepts were extracted and synthesized.

To create such a catalog, we decide to conduct a Systematic Literature Review (SLR) based on the methodology presented by Kitchenham et al. [KC07]. In the first part of this chapter, we first present the method of our work. We describe the procedure of our adapted SLR approach and present the results of each phase. In doing so, we first motivate the need for the review and introduce the research question we aim to answer with this SLR. We then describe how we identified research and selected the relevant studies. Last, we describe the process of data extraction and how we synthesized catalogs from the extracted data. Figure 4.1 illustrates an overview of the whole SLR method we applied.

In the second part of the chapter, we present the results of the SLR we conducted. Specifically, we address the schemas of each of the catalogs of security design concepts that we identified and also provide a definition and an example of each category of security design concept. The specific catalogs that emerged from this SLR can be found in Appendix B.

## 4.1. Systematic Literature Review report

In the following section, we report the process of our conducted SLR approach. A systematic review is a common method for *"[...] identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area [...]"* [KC07]. Although often conducted in the context of medical and healthcare sciences, Kitchenham et al. [KC07] proposed guidelines to pursue this approach in the context of software engineering research to gather and analyze given evidence according to a specific research topic. A SLR is characterized by the fact that its motivation and process are precisely specified and all (intermediate) results are presented. The reasons for conducting a SLR are that they are transparent, reproducible, and fair. Transparent, since we provide a detailed explanation of each conducted step, the reasoning behind this, and the corresponding results. Reproducible, since the well-defined method is documented such that other researchers can assess and repeat the process to validate its results. Fair, since all available research within the search strategy is considered and reviewed to minimize the selection bias.

### 4.1.1. Need for a review

As presented in Section 3.1, there are already several systematic reviews that have examined security threats and design concepts in microservice-based applications. In these papers, white literature and partly gray literature were investigated to identify these concepts and threats. However, none of these SLRs have yet performed a qualitative analysis of the design concepts explored. Classifications have been made, e.g., based on the security scope. However, we see a lack of contributions that try to describe the researched security design concepts in a detailed and schematic way. Cataloging security design concepts, similar to the existing attack libraries, is an essential tool for software engineers and architects to effectively design the architecture of a system. Only through

a detailed description of the applied design concepts is it possible to make statements about the quality and functionality of a system. Furthermore, a schematization of the design concepts offers the possibility of comparing the effectiveness of these design concepts with one another more clearly. Finally, we claim that such a knowledge base can help to improve the comprehensibility of security in software architectures.

##### 4.1.2. Research question

Thus, we aim to clarify which security design concepts exist, how they can be described and to which criteria they can be classified, specifically for microservice-based applications. Our goal is to propose a first draft of a navigable knowledge database for security design concepts applicable to software architectures. These catalogs should support software architects and engineers when creating new software architectures or reviewing existing ones. In terms of granularity, these catalogs are meant to be similar in detail to the presented attack libraries such as CAPEC [wwwa] and CWE [wwwd]. The goal of the described and applied SLR method is reflected in our first research question.

**(RQ1)** *How can we describe and classify security-related design concepts for software applications in microservice architectures?*

##### 4.1.3. Developing and evaluating the review protocol

The review protocol was developed and evaluated as part of this master thesis. In doing so, we elaborated an adapted, lightweight version of the SLR presented by Kitchenham et al. [KC07]. The following sections explain the steps performed and the results obtained from each phase. Furthermore, we describe at which points we deviated from Kitchenham's original SLR approach or omitted certain steps and provide reasoning for this. The results of this SLR, i.e., the security design catalog and corresponding catalog schemas, are presented in Section 4.2 and Appendix B. The review protocol was developed in collaboration with the supervisor of this thesis and continuously evaluated by the author and the supervisor.

##### 4.1.4. Identification of research

After we determined the need for a review and the research question we wanted to investigate, we began the search for relevant publications. We needed to select a suitable search strategy in order to exclude as many irrelevant papers as possible in advance. The search strategy determined which sources we used, as well as the search strings and filters we applied to each source. In specifying our search strategy, we oriented ourselves to the search strategies of related SLRs (cf. Section 3.1). Compared to these studies, however, our search needed to reduce the corpus of publications we wanted to sift through to a size that could be handled within the scope of this thesis. To determine a search strategy that meets these requirements, we first conducted a series of trial searches. In the following, we describe how we conducted the trial searches, which search strategy we finally chose, and present its results.

Table 4.1.: The list of candidate scientific databases that were considered during the trial searches.

DB#	Database	Included?	Exclusion reason
DB1	Web of Science	✓	
DB2	IEEEExplore	✓	
DB3	ACM Digital Library	✓	
DB4	Science Direct	✓	
DB5	Inspec	✗	Not accessible
DB6	SpringerLink	✗	Too many results
DB7	Scopus	✓	
DB8	CiteSeerx	✗	Limited filter capabilities
DB9	Wiley InterScience	✗	Limited filter capabilities
DB10	Ei Compendex	✗	Not accessible
DB11	IET Digital Library	✗	Limited filter capabilities
DB12	dblp	✗	Limited filter capabilities
DB13	Google Scholar	✗	Too many results

### Selection of scientific databases

First, we determined which sources to be included. Kitchenham et al. [KC07] suggest prospecting different sources for this, including journals, proceedings, as well as scientific databases. For this thesis, we limited ourselves to a few scientific databases. Table 4.1 lists the candidate databases that we initially considered. This list of candidates was generated by reviewing related SLRs as well as based on the recommendation of the supervisor of this thesis. In each of these scientific databases, a large number of scientific publications can be identified using a search string and, if necessary, a set of filters. In advance, we were able to exclude the *Inspec* (DB5) and *Ei Compendex* (DB10) databases for further use because we did not have access to search these databases. Furthermore, *SpringerLink* (DB6) and *Google Scholar* (DB13) were excluded due to their high number of search results. *CiteSeerx* (DB8), *Wiley Interscience* (DB9), *IET Digital Library* (DB11), and *dblp* (DB12) were excluded because of their limited filtering capabilities. Thus, the remaining scientific databases that we included for further trial searches and the final search were *Web of Science* (DB1), *IEEEExplore* (DB2), *ACM Digital Library* (DB3), *ScienceDirect* (DB4), and *Scopus* (DB7).

### Trial searches

After selecting the sources for this SLR, we began some trial searches. We first examined the search strategies of related works. Some studies developed rather generic search strings to find all relevant publications regarding MSA security. For instance, Pereira-Vale et al. [Per+19] retrieved 990 white and gray literature articles using the search string ("secure" OR "security") AND ("microservi"\* OR "micro servi"\*

OR "micro-servi"\*). Berardi et al. [Ber+22] gathered 1,704 publications in total in their initial dataset with the search string `Microservice AND Security` using a similar set of scientific databases that we chose. There were two reasons for us to refine these search strategies for our SLR. First, these amounts of publications are too large to handle in the scope of this thesis, since we are limited by time and capacity constraints. Second, and more importantly, to investigate our stated research question, we are only interested in finding specific software elements that are applicable to software architecture, i.e., security design concepts. Thus, we further refined the search string by adding specific security design concept terms. In our trial searches, we investigated which combination of terms yields an appropriately manageable set of publications. Furthermore, we additionally considered different spellings of the term `microservice`, since no generally accepted spelling exists for this term. The terms of categories for security design concepts that we considered in the trial searches were  $T = \{ \text{technique, pattern, best practice, tool, technology, design principle, activity, process, mechanism, service, design, development, tactic, protocol} \}$ . This list of terms was compiled based on discussions with the supervisor as well as our own experience. We then combined each of these terms, i.e., for each term  $t \in T$ , with the generic base search string and applied them to the selected databases. The corresponding search string is depicted below.

```
(security OR secure) AND (microservice OR microservices OR
"micro-service" OR "micro-services" OR "micro service" OR "micro
services") AND t
```

We examined how many results each database returned, and then determined a matching subset of terms. We applied the same filters to each trial search as we did to the final search (cf. Table 4.2, Applied filters). We also specified in the corresponding search strings of the databases that the terms may only appear in the title, abstract or keywords in order to filter for relevant papers as precisely as possible. The results of these trial searches are depicted in Appendix A.

#### Final search

From the previously mentioned trial searches, the following search string was selected for the final search of the SLR.

```
(security OR secure) AND
(microservice OR microservices OR "micro-service" OR "micro-services" OR
"micro service" OR "micro services") AND
(technique OR pattern OR "best practice" OR "design principle" OR
activity OR mechanism OR development OR tactic OR protocol)
```

As in the trial searches, the terms in this search string were searched for exclusively in the title, abstract and keywords of the publications. Table 4.2 lists the filters applied in

Table 4.2.: The results of the SLR’s final search, including the selected scientific databases, the applied filters, and the number of publications retrieved per database.

DB#	Applied filters	Number of publications
DB1	<ul style="list-style-type: none"> <li>• Languages: English</li> <li>• Document Types: Proceeding Paper or Article</li> <li>• Research Areas: Computer Science</li> </ul>	148
DB2	<ul style="list-style-type: none"> <li>• Filters Applied: Conferences or Journals</li> </ul>	111
DB3	<ul style="list-style-type: none"> <li>• All Publications: Proceedings or Journals<sup>1</sup></li> <li>• Content Type: Research Article</li> </ul>	37
DB4	<ul style="list-style-type: none"> <li>• Article type: Research articles</li> <li>• Subject areas: Computer Science</li> </ul>	31
DB7	<ul style="list-style-type: none"> <li>• Source type: Conference Proceeding or Journal</li> <li>• Document type: Conference Paper or Article</li> <li>• Publication stage: Final</li> <li>• Subject area: Computer Science</li> <li>• Language: English</li> </ul>	211
$\Sigma$		<b>538</b>

each of the selected databases and the number of publications each database returned. We restricted ourselves to peer-reviewed publications, i.e. only those that were published in research articles, e.g., in a proceeding or journal. If possible, we restricted the subject area to computer science only. Furthermore, the publications were restricted to be in English only, if possible. With this search string and the presented filters, we obtained a set of **538** peer-reviewed publications. The final search was performed on June 10, 2022.

#### 4.1.5. Study selection

After conducting the final search, we began with the selection of relevant studies. In this phase, irrelevant studies were sorted out using a defined set of exclusion criteria. In this SLR, we divided the study selection into two screening phases. In the first phase, we only screened the title, abstract, and conclusion of the paper for relevance. This already allowed us to exclude many publications based on simple criteria. In the second phase, we then considered the full text of each remaining publication. The exclusion criteria we considered in each of the two phases are shown in Table 4.3.

In the first phase, we first removed all duplicates (**EC-1**). A publication could occur more than once in the corpus according to our final search since the same publication can

<sup>1</sup>Only one publication type could be selected per query, so we performed two separate queries each with one corresponding publication type and merged the publication results.

Table 4.3.: The exclusion criteria considered during the SLR’s study selection phase, including the number of papers removed and remaining per criterion.

ID	Exclusion criteria	Removed papers	Remaining papers
<b>First screening (title, abstract, conclusion)</b>			
	Initial publication set		538
<b>EC-1</b>	Duplicate removal	-266	272
<b>EC-2</b>	False positive	-23	249
<b>EC-3</b>	Not English	-2	247
<b>EC-4</b>	Full text not available	-5	242
<b>EC-5</b>	Short paper, tutorial, etc.	-12	230
<b>EC-6</b>	Paper not discussing MSA security as their primary topic	-116	114
<b>EC-7</b>	Paper not discussing any design concept related to MSA security	-4	110
<b>Second screening (full text)</b>			
<b>EC-8</b>	Paper not discussing MSA security as their primary topic	-21	89
<b>EC-9</b>	Paper not discussing any design concept related to MSA security	-12	77
<b>EC-10</b>	Insufficient description of proposed security design concept	-10	67
<b>EC-11</b>	Proposed design concept is not security-related	-6	61
<b>EC-12</b>	Proposed design concept is platform or technology dependent	-7	54
<b>EC-13</b>	Proposed design concept is not applicable to MSAs	-2	<b>52</b>



exist in multiple sources. The removal of duplicates was automated using a literature management program and then checked for correctness afterward. False positives are those that incorrectly did not contain the necessary search terms in the title, abstract, or keywords and were excluded (**EC-2**). As in the applied filters in the final search, we also excluded non-English publications (**EC-3**). In addition, we sorted out publications for which we either could not access the full text (**EC-4**), or whose full text was a short paper of 2 pages or less, tutorial, video, editorial, book, or book chapter (**EC-5**). In terms of content, we excluded all publications that either did not primarily examine MSA security (**EC-6**) or it was already clear in the abstract or conclusion that no concrete design concept was proposed in this publication (**EC-7**). After the first screening phase, we excluded a total of 428 publications.

Subsequently, we considered the full text of the remaining publications in the second screening phase. As in the previous phase, we excluded all publications that did not address MSA security as a primary topic (**EC-8**) or did not present a concrete security design concept (**EC-9**). The reason for applying these exclusion criteria again in the second screening phase was that it was partly not clear from the abstract and the conclusion that the publication did not fulfill these requirements. Since we based our SLR on a qualitative analysis of the data used to create the catalog, we also excluded publications that had presented a concrete security design concept but either did not describe it sufficiently (**EC-10**) or it was not security relevant (**EC-11**). In addition, we are interested in general design concepts, and therefore excluded solutions that rely on a specific platform or technology (**EC-12**). Finally, we eliminated publications whose presented solutions were not applicable to our selected architecture style, i.e., microservices (**EC-13**). Through the second screening phase, we were able to exclude a total of 58 additional publications. Thus, a total of **52** studies were selected for further processing in this SLR.

#### 4.1.6. Data extraction

In the following, we describe the data extraction performed. In this phase, we reviewed the full texts of the selected publications and extracted the relevant information of the presented security design concepts using a predefined data extraction form. Table 4.4 captures the data extraction form used.

Specifically, we wanted to identify a unique *name* for each security design concept to establish a corresponding vocabulary of these concepts. Furthermore, we identified the *type* of each design concept, which represents its category. The type or category of the security design concept subsequently established the basis for creating different catalogs of concepts depending on the type. For each security design concept, we extracted a *description* that captures in prose form how the design concept functions. Furthermore, we were interested in the motivation and purpose of the security design concept. Therefore, we extracted for which *security requirements* the concept is applied and which *threats* are possibly mitigated by it. In addition to the *influence* on other *QAs*, we also wanted to find out how the security design concept is applied in concrete terms by determining the *security scope*, i.e., whether it detects, prevents, reacts to or recovers from concrete

Table 4.4.: The data extraction form used during the data extraction phase of the SLR.

Property	Description
Name	The name of the security design concept
Type	The type or category of the security design concept. Possible values: activity, best practice, design principle, pattern, process, security service, tool, mechanism, tactic, protocol
Description	A text description of the corresponding security design concept
Security requirement	A requirement statement that motivates the security design concept
Security goals	The security goals (cf. Definition 2.11) the security design concept targets
Security scope	The security scope of this design concept. Possible values: detect, prevent, react to, recover from
QA influenced	A list of QAs that the design concept influences, positively or negatively
Threats	A list of threats that the design concept mitigates, e.g., DDoS, code injection, lateral movement, remote code execution
Software scope	The scope where the design concept is applied, e.g., design, testing, CI/CD, development, network, API, container
Software components	A list of software components, if any, that the design concept either employs or depends on
Technologies	A list of technologies that the design concept uses, e.g., blockchain, machine learning
Domain	The application domain the security design concept was used in, e.g., IoT, cloud-native, healthcare
Publication	The publication, selected in the SLR process, the security design concept was established in

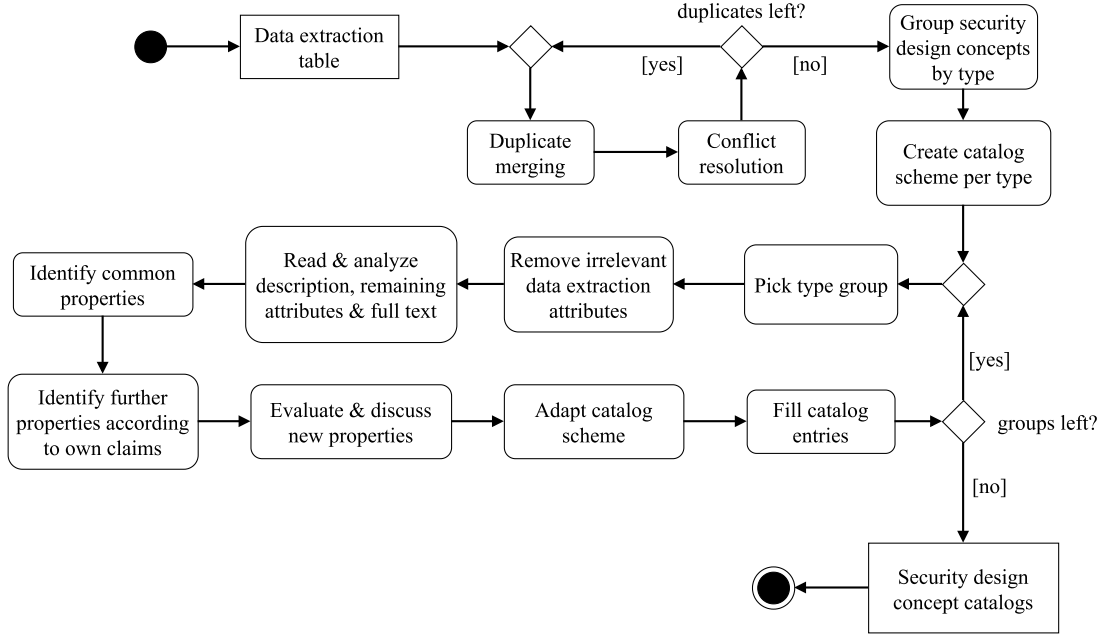


Figure 4.2.: The applied data synthesis procedure of the SLR.

attacks, the *software scope*, i.e., whether it is a design concept for APIs or on the network and communication level or others, and possible *technologies*, *domains*, or dependencies in the form of *software components*.

Through the data extraction phase, we identified a total of **119** security design concepts. However, it should be noted that this set of security design concepts also contained duplicates, e.g., if the same design concept has been described in multiple publications. Although Kitchenham et al. [KC07] explicitly advise against this practice, as that seriously biases the results, we chose to include duplicates in order to obtain a broad qualitative basis of data. This allows us to include details that have only been presented in single publications in the resulting catalog.

#### 4.1.7. Data synthesis

In the data synthesis phase, the task was to compile the corresponding catalogs of security design concepts from the raw data obtained in the data extraction phase. The procedure for the data synthesis was developed and discussed together with the supervisor and is described in more detail in the following. The activities of this procedure are depicted in Figure 4.2.

We first identified and merged duplicates of security design concepts. The identification of duplicates was primarily based on the names of the considered security design concepts, but also the description as well as the type and other properties from the data extraction form were used for this purpose. Conflicts such as conflicting types or ambiguous design concept names were also resolved in this step. After duplicates were merged,

we grouped each security design concept by its type. A separate catalog schema was then created for each security design concept using the attributes from the data extraction form. For each type of security design concept, we then proceeded as follows. First, we removed irrelevant attributes from the data extraction form for the corresponding security design concept type, e.g., if no or very little information was available for that attribute. Then, the descriptions, as well as the full texts and remaining attributes, were re-read and analyzed. Based on this, we tried to identify new common properties for this type. Furthermore, we added additional properties if necessary based on our assertions and experiences, if this information was not explicitly mentioned in the publications of the respective design concepts. The newly selected properties were then evaluated and discussed with the supervisor. After the relevant properties were identified, the catalog schema of the corresponding security design concept type was adapted and filled in. This procedure was performed for each type of security design concept.

Nevertheless, some types of security design concepts that were considered during the data extraction phase were removed. These include the types *best practice*, *mechanism*, *process*, and *tool*. These types were removed during the data synthesis procedure because either we could not establish a clear definition for them, or the concrete security design concepts of these types may also represent instances of other types. For instance, *best practices* can be employed as a design pattern, a tactic or as an activity. Security *mechanism*, on the other hand, is an established term in the security literature. However, we claim that its definition, "A method or process [...] that can be used in a system to implement a security service [...]" [Shi07], is rather unprecise. Such definitions apply to many design concept types, and rather increase ambiguity than clarification. Thus, we wanted to avoid those imprecise definitions and ambiguous design concepts reducing the quality of the catalogs. Furthermore, we included a new category of security design concepts, namely *Intrusion Detection/Prevention Systems (IDS/IPS)*, as many publications propose different methods and algorithms for this type of design concept. Regarding Definition 2.12, we refer to the IDS/IPS category as *technology family* and therefore as *integratable component*. Finally, we incorporated our understanding and definition of *security services* into the concept of *security views* which are presented and explained in detail in Section 5.3. We argue that this concept should not be placed on the same abstraction level as their security design concept counterparts. Instead, the identified *security views* play a fundamental and central role in the process of security modeling in software architectures and should be treated as such.

The results of the data synthesis phase, i.e., the definition and schemas of each security design concept type, are explained in detail in Section 4.2. The specific catalogs of security design concepts can be found in Appendix B.

##### 4.1.8. Conflict of interest

There are no conflicts of interest and no secondary interests of the authors of this SLR. All opinions presented are those of the author alone. No third-party organization or institution was involved. The results and progress of this work were reviewed and discussed with the supervisor at regular intervals. The supervisor of the master thesis and

SLR works as a research assistant at RWTH Aachen University. Neither the author nor the supervisor influenced the results of this thesis through other engagements. The thesis and the associated SLR was completed exclusively for scientific purposes as a requirement for the examination of the master's degree. The author declares *having no financial conflicts of interest, not using the company's (RWTH Aachen University) resources, time, or any other resources for this thesis (excluding having the thesis supervised by an RWTH Aachen University employee), not presenting patents, copyrights or royalties without an indicated citation, and finally not holding shares in any company that might benefit or be otherwise influenced by the paper.*

#### 4.1.9. Excluded SLR steps

We discussed our adapted SLR approach and the performed phases and steps of each phase. In the following, we discuss and reason about the steps that were omitted in this SLR but are typically performed in other SLRs as proposed by Kitchenham et al. [KC07].

##### Commissioning a review

An organization that requires information about a specific topic but that has neither the capacity nor the time to perform an investigative SLR commissions researchers to perform a SLR on this topic. In this case, a corresponding commissioning document should be created according to Kitchenham et al. [KC07]. However, since this SLR was performed on the researcher's own needs, as it was undertaken as part of the author's master studies, no third-party organization or institution was involved. Thus, this step was not required and therefore not included in this SLR.

##### Snowballing

Although not proposed by Kitchenham et al. [KC07] as a required step, snowballing is an activity that is often performed in similar SLRs [Woh14]. This phase is typically performed after the study selection phase. Snowballing indicates that from the set of selected studies, further research may be gathered by investigating the references of the corresponding selected study, i.e., *backward snowballing*, or by searching for studies that cite the study being examined, i.e., *forward snowballing*. As such, snowballing may serve as a good alternative to using scientific databases [Woh14], and is also often used in addition to that. However, the study selection phase of this SLR already yielded a considerable amount of publications to be examined. Further research gathering by snowballing would contradict our requirements to limit the set of publications to a manageable amount.

##### Study quality assessment

The study quality assessment phase is performed to analyze the systematic error of the selected publications, as well as their internal and external validity. It also provides a

means to weigh the importance of individual studies and to investigate quality differences between those [KC07]. Typically, quality assessments are conducted in the form of checklists that need to be evaluated for each selected study. Quality items may be assigned numerical scales, if applicable, for more comparative quality assessments. On the one hand, time and capacity constraints prevented conducting an in-depth quality assessment of the selected studies in this SLR. Thus, we assumed to retrieve little value or insights in performing an immature quality assessment of these studies. On the other hand, due to the heterogeneous group of security design concepts found, it was difficult to create a quality assessment checklist that was uniformly meaningful for all design concepts. Nevertheless, we recognize the value that an in-depth quality assessment strategy might add, and we consider preparing such a strategy in further planned SLRs.

#### **Dissemination strategy**

This SLR will be evaluated as part of the corresponding master thesis by other researchers and practitioners. The reported SLR was not assessed in form of a formal peer review at the time of writing. The thesis and the SLR were reviewed by the supervisor who is a research assistant at RWTH Aachen University and is qualified to supervise bachelor and master theses. Furthermore, the final paper will be assessed by the examiners of the thesis, who are two professors of RWTH Aachen University. Most publications cited in this paper were retrieved from scientific databases and as such mostly peer-reviewed.

### **4.2. Review result**

In the following section, we now present the results of this SLR. As already depicted in the report, we created several catalogs of security design concepts as part of this thesis. For each category, we first provide a description of the category and what purpose they serve in the context of software architectures. Furthermore, we propose for each of these categories a clear definition of what constitutes the category and to create an appropriate vocabulary. The provided definitions are compatible with the notion of a security design concept as defined in Definition 2.12. Then, we present the schemas of the developed catalogs in the context of this SLR by describing each property. Finally, we depict and explain briefly one element of the respective catalogs as an example.

Each of the presented catalogs of security design concepts contains the following three properties at minimum, namely a name, a list of security views, and a list of publications. We argue that clear naming is essential for the development of these design concepts. This facilitates the communication and description of an architecture. The concept of security views is another part of the contributions of this master thesis. In essence, security views represent different perspectives of security on the software architecture. A detailed description and definition of security views for microservices are given in Chapter 5. We argue that a separation of security views is necessary because the security of a software architecture is too multi-layered and complex to be represented in a single

view. The security views covered in this thesis are authentication, authorization, secure communication, secure storage, monitoring, and secure build & deployment. Each of these views is intended to help model one aspect of security at a time. As part of this SLR, we have developed a mapping that maps the concrete security design concepts to the corresponding security views in which they can be applied. This mapping should help to decide which security design concepts are useful in which context. For instance, *Token-based Authentication* is a security tactic that can be established as a modeling element in an authentication view but hardly makes sense as part of a monitoring view. Finally, we reference the publications in which the security design concept was presented and described.

#### 4.2.1. Security design principle

In software engineering, the quality of an architecture is significantly influenced by its design decisions. To establish a basic direction of good design, design principles are often employed in software engineering. A design principle establishes a concise guideline that can be used to guide design decisions based on an intent that motivates the principle. Deviations or violations of these design principles often lead to a reduction in software quality. However, some design principles might decrease specific aspects of the quality of a software architecture by nature. Principles may also be related to other principles, e.g., when they are commonly used in conjunction. Design principles are often recognized and identified by assigning a name or acronym to them, thus creating a vocabulary of commonly known principles. Among the best-known design principles in object-oriented software design are the SOLID principles [Mar08] which contribute to making software more modular and decoupled. *Security design principles* provide guidance towards more secure software design and help software engineers to make design decisions that improve the application's overall security.

Definition 4.1: Security design principle [Kan03]

A *security design principle* is a comprehensive and fundamental doctrine or rule that governs the creation of secure software design.

The catalog for security design principles is presented in Appendix B.1. The security design principles schema in our catalog is structured as follows.

**Name** The name of the security design principle.

**Security view** The security views the security design principle can be modeled in.

**Summary** A short description of what the security design principle entails.

**Intent** A motivation for following this principle.

**QA enhanced** QAs that this principle is about to improve.

**QA worsened** QAs that this principle is about to worsen.

**Related principles** A self-relation to principles that are related or commonly used in conjunction.

**Publications** A list of publications that mention or propose the security design principle.

As an example, *Zero Trust Networking* is a security design principle that guides the architecture towards a system where no implicit trust is granted between services or clients. When following this principle, it is always assumed that the microservice network is infiltrated and compromised. Thus, trust must be established explicitly by authentication and authorization techniques, and communication channels must be secured to prevent eavesdropping.

<b>Name</b>	Zero Trust Networking
<b>Security view</b>	Authentication, Authorization, Secure Communication
<b>Summary</b>	All actions must be verified and all data transfers should be encrypted. There is no implicit trust between services, and trust must be evaluated continuously.
<b>Intent</b>	Always assume that the microservice network is already compromised.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related principles</b>	Deny access by default, Principle of least privilege
<b>Publications</b>	[Mel21]

#### 4.2.2. Security activity

When developing a software system, we typically follow a process model or SDLC. Within this SDLC, the development of the application goes through different phases. For instance, fundamental architectural decisions are made at the beginning during the requirements engineering and design phase. In the implementation phase, the application is then developed in concrete terms, and its functionality and quality are subsequently checked in a testing phase. Finally, applications are distributed and rolled out during the deployment phase. During each of these phases, activities are performed, either automatically or manually by one or more stakeholders. An activity can also be specified by a concrete sequence of steps to achieve a certain goal or target state. *Security activities* are therefore activities that model or audit the security or security processes in a software architecture. Typical examples of security activities are *Static Application Security Testing (SAST)* and *Dynamic Application Security Testing (DAST)*, which check the software application for vulnerabilities, either statically or dynamically at runtime. *Dependency scanning* analyzes the *Software Bill of Materials (SBOM)* of software included as dependencies within the software application for existing vulnerabilities. Another central security activity in the design phase is threat modeling, which identifies threats through architectural flaws.

##### Definition 4.2: Security activity

A *security activity* is an activity within the SDLC, either executed once or repetitively, manually or automatically, by any stakeholder to model or audit security-relevant properties of a software architecture.



The catalog for security activities is presented in Appendix B.2. The security activity schema in our catalog is structured as follows.

**Name** The name of the security activity.

**Security view** The security views this security activity can be modeled in.

**Goal** A description of the motivation or target state this activity tries to achieve.

**Steps** A list of actionable steps that this activity is composed of.

**Automaticity** Whether the activity is performed manually or automatically.

**Stakeholders** A list of stakeholders who perform the given activity.

**Assets** A list of assets that are affected by this activity.

**SDLC phase** A list of SDLC phases in which the activity is performed. Possible values: Planning, Requirements Engineering, Design, Development, Testing, Deployment, and Operations.

**Publications** A list of publications that mention or propose the security activity.

*Dependency scanning* is a security activity usually performed automatically during the Testing and Deployment phases (CI/CD). This technique gathers software dependencies to identify existing vulnerabilities by comparing the software versions with the entries in a vulnerability database, e.g. CVE [wwwc]. Findings are reported in a human-readable format to the operators or developers for patching.

<b>Name</b>	Dependency scanning
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Due to the high mass of packages and cumulative dependencies that occur when using package managers, dependencies should be scanned for vulnerabilities in an automated way.
<b>Stakeholders</b>	CI/CD pipeline, Operator, Software engineer
<b>SDLC Phase</b>	Testing (CI), Deployment (CD)
<b>Automaticity</b>	Automated
<b>Assets</b>	Container, Source code
<b>Steps</b>	Upon code changes, the CI/CD pipeline pulls the repository changes and applies the usual integration and deployment steps. In addition, a new pipeline step, the Vulnerability Scan, scans dependencies in container images and application BOMs. If the vulnerability scanner identifies any active vulnerabilities, a report is created in a human-readable format and sent to the operator or developers.
<b>Publications</b>	[Thr+21; NL21]

### 4.2.3. Security tactic

A tactic [BCK21; KK21] is an atomic design decision to positively influence a specific QA. For this reason, specific sets of tactics can be defined for each QA. In particular, *security tactics* define design decisions that aim to strengthen an aspect of security. However, in addition to its primary QA, tactics can have side effects, both positive and negative, on other QAs. They are atomic because they represent the basic building blocks of the design. From their composition, structural patterns and architectural styles can be designed. They can also be used to augment existing design patterns. Tactics in general consist of three components, a stimulus, an environment, and a response. The stimulus

defines the action or event that derives the motivation to apply the specific tactic. In security tactics, the stimulus is often defined by one or a set of threats. The environment defines the context, e.g., involved assets, in which the tactic is to be applied. Finally, the response defines how the software application reacts to the stimulus in the context of the environment.

**Definition 4.3: Security tactic [BCK21]**

A *security tactic* is an atomic design decision that influences the achievement of a security quality response.

The catalog for security tactics is presented in Appendix B.3. The security tactic schema in our catalog is structured as follows.

**Name** The name of the security tactic.

**Security view** The security views this security tactic can be modeled in.

**Stimulus** An action or event that incites this tactic to be employed.

**Threat** A specific stimulus in which the action or event is a threat.

**Environment** The architectural context in which the tactic is employed.

**Response** An action or event that is performed in response to the given stimulus and environment.

**Depends on** A self-relation to dependent tactics.

**Implies usage** A self-relation to implied tactics.

**Patterns** A list of architectural design patterns that this security tactic can augment.

**Publications** A list of publications that mention or propose the security tactic.

*Token-based authentication* is an example of a security tactic. The stimulus of this tactic is that a user wants to access a protected resource. Additionally, there is a threat that an unauthorized user will gain access to data that is not intended for them. Thus, the environment of this tactic is authentication, specifically when an external client requests access to a resource. The response of token-based authentication is now that the system uses a cryptographic token to determine the identity of the user. The access token contains the relevant information to identify the user and a cryptographic signature to verify the validity of the token. Overall, the security tactic is employed to *prevent* the threat of *spoofing* and *elevation of privilege*. This token may contain sensitive information about the personal information of the owner, and may also be misused for impersonation when stolen. Therefore, the token needs to be exchanged over a secure communication channel. This requirement is represented in the security tactic dependency *Encrypted Communication*.

<b>Name</b>	Token-based authentication
<b>Security view</b>	Authentication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service or client performs a request to a restricted microservice. In order to process the request, the microservice needs to determine the identity and permissions of the requestor.
<b>Threat (Security Stimulus)</b>	A (malicious) client wants to have access to a service/data, which it is not allowed to have.
<b>Environment</b>	Authentication
<b>Response</b>	Cryptographic identity tokens are used to authenticate the requestor. The access token contains all relevant authentication and authorization information, as well as a cryptographic signature to verify its integrity.
<b>Depends on</b>	Encrypted Communication
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Access token
<b>Publications</b>	[JLE18; PS19; Zdu+22]

#### 4.2.4. Architectural security pattern

Software patterns are one of the most widely used design constructs in software engineering. The notion of design patterns was significantly established by the *Gang of Four (GoF)* who presented one of the first contributions in form of a catalog of different structural, behavioral, and creational design patterns [Gam+94]. In essence, software patterns provide concrete solutions to recurring problems in software development. They thus form a fundamental aspect in software engineering, since they encourage reusability and thus prevent the wheel from being reinvented over and over again for already known problems. Thus, software patterns form a relation between a problem, the context in which the problem occurs, and a solution. Cataloging software patterns is a substantial contribution of the last decades. The adoption and common acceptance of certain catalogs lead to the fact that we developed today a vocabulary for effective communication. Patterns are described in specific formats, also called *pattern languages*. Several different pattern languages have been developed so far [Fow06]. In this work, we follow the Alexandrian pattern language [Chr77] developed by Alexander Christopher in 1977. The Alexandrian form captures the most important aspects of a software pattern that forms the common ground and is contained in each pattern language, namely the problem, context and solution. Furthermore, it describes the solution of the pattern by providing a structure and instructions so the software engineer knows what is required to build the pattern. More exhaustive pattern languages like the one presented by the GoF [Gam+94] might be helpful to perform a comparative analysis of these patterns. However, it was rather difficult to create such a rich pattern catalog from the extracted data of our SLR, as most publications did not describe their patterns in such detail. In this work, we distinguish between design patterns and architectural patterns. The difference between these two types of patterns lies in the level of abstraction to which they are applied. While solutions of design patterns are concrete structures on the code level in the form of classes within a module or across modules, architectural patterns focus on the organization and composition on the component level. Thus, architectural patterns are more abstract than design patterns. *Architectural security patterns* focus specifically on security problems to solve.

**Definition 4.4:** Architectural security pattern [BCK21]

An *architectural security pattern* is a general, reusable solution to a common security problem. It commonly provides a structural or behavioral composition of components in a software architecture.

The catalog for architectural security patterns is presented in Appendix B.4. The architectural security pattern schema in our catalog is structured as follows.

**Name** The name of the architectural security pattern.

**Security view** The security views this architectural security pattern can be modeled in.

**Also known as** An alias or synonym of the architectural security pattern.

**Context** A description of the circumstances under which the pattern is applied.

**Problem** A description of relevant problems, constraints and issues that the pattern resolves.

**Solution** A description of how the pattern works, describing the static and dynamic relationships between components.

**Structure** A diagram that presents the structure or behavior of the architectural security pattern (if applicable).

**Related patterns** A self-relation to patterns that are related.

**Publications** A list of publications that mention or propose the architectural security pattern.

The *Security Gateway* pattern addresses the problem of ensuring and enforcing security policies of microservices on startup. It prevents microservices that violate any of these policies from registering with the Service Registry, a service that provides the dynamic location of microservices. On startup and before registration of the microservice, the Security Gateway probes the respective microservice by performing several security tests. The results of these security tests are reported and accessible through the *Security Health Endpoints* pattern. Failed security tests due to existing vulnerabilities are reported and the Security Gateway prevents the microservice from registration. The Security Gateway thus participates as a Security Enforcement Point (SEP) by enforcing the security policies in place.

<b>Name</b>	Security Gateway
<b>Security view</b>	Monitoring
<b>Also known as</b>	
<b>Context</b>	Within the microservice system, a Service Registry component handles the registration or de-registration of several microservices. This component provides information about the location (IP address and port) of the microservice as well as a description of the API the microservice offers. Before the registration of a new microservice instance, a collection of security tests, e.g., Dynamic Security Application Tests (DASTs), should be performed on the newly registered microservice instance.
<b>Problem</b>	How to effectively perform security tests before the startup and enrollment of a new microservice instance before its registration? A set of security policies should be enforced that satisfy the requirements that, for instance, a registering microservice might not be deployed if certain vulnerabilities are detected. How to deal with the discoverability problem, i.e., the capability to constantly detect the location of new microservices.
<b>Solution</b>	Before registering a new instance of a microservice, perform a set of security tests against it via a Security Gateway. In contrast to the API Gateway which is concerned with API composition and efficient routing of incoming and outgoing requests, the Security Gateway is a Security Enforcement Point (SEP), i.e., it is concerned with the enforcement of the employed security policies, e.g., that no microservice which contain a specific vulnerability or reach a vulnerability score is allowed to run in a productive environment. The Security Gateway is invoked by a Service Registry before the registration of the microservice instance begins. If the security tests fail, a test report is made accessible to the development team or other service, for instance by leveraging the Security Health Endpoints pattern for consumable security test results.
<b>Structure</b>	<pre> graph LR     Client[Client] --&gt; APIGateway[API Gateway]     APIGateway --&gt; ServiceRegistry[Service Registry]     ServiceRegistry --&gt; SecurityGateway[Security Gateway]     SecurityGateway -. probe .-&gt; MS1[Microservice 1]     SecurityGateway -. probe .-&gt; MSn[Microservice n]     ServiceRegistry --&gt; MS1     ServiceRegistry --&gt; MSn </pre>
<b>Related patterns</b>	Security Manager, Security Health Endpoints, Service Registry
<b>Publications</b>	[TSM17; Yan+22]

#### 4.2.5. Security protocol

In MSAs, communication between microservices forms one of the most central building blocks. Only through lightweight communication interfaces, the modularity of such an architecture is ensured. Microservices, therefore, use communication protocols to exchange data and request services. A protocol defines a concrete sequence of messages that are exchanged between two or more communication partners. Each protocol pursues a goal or target state, which is reached after the successful execution of the protocol. In the case of *security protocols*, the target state is typically the protection of the communication interface itself, e.g., through encryption, authenticating or authorizing the requesting entity, or any combination of these. Security protocols thus form part of the integratable components, since implementations are typically available for standardized protocols that can be incorporated directly into the architecture.

##### Definition 4.5: Security protocol

A *security protocol* is a procedure for components or services to communicate with each other with a specific security goal in mind.

The catalog for security protocols is presented in Appendix B.5. The security protocol schema in our catalog is structured as follows.

**Name** The name of the security protocol.

**Security view** The security views this security protocol can be modeled in.

**Standard/RFC** A standard or RFC that defines this protocol.

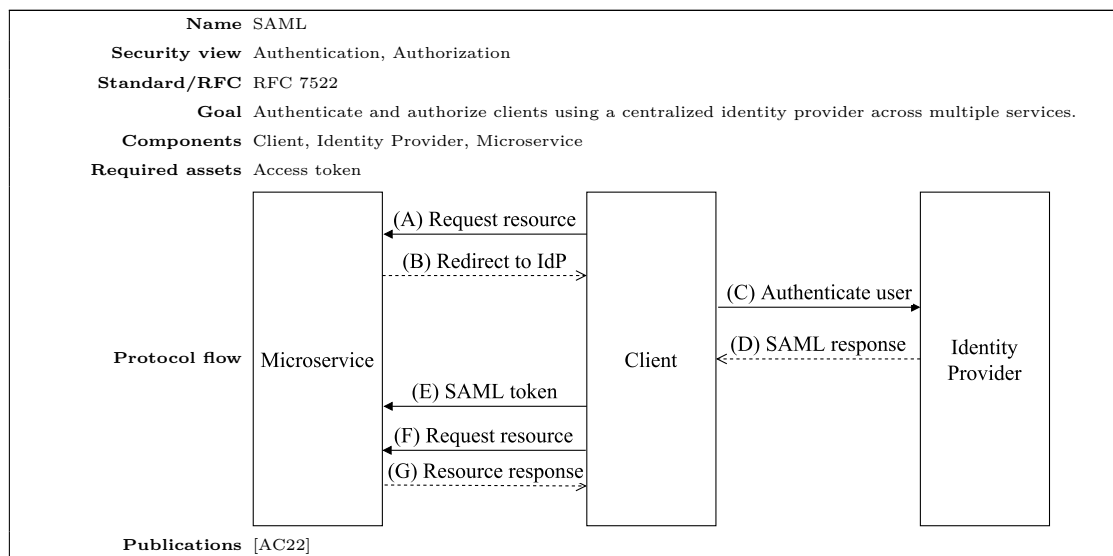
**Goal** A description of what this protocol wants to achieve.

**Components** A list of components that are usually involved in this protocol.

**Message flow** A diagram (usually a sequence diagram) that shows the messages exchanged between the components.

**Publications** A list of publications that mention or propose the security protocol.

*Security Assertion Markup Language (SAML)* is a security protocol and framework for the authentication and authorization of clients and microservices. The protocol assumes that a centralized identity provider holds the identity information of the requesting party. Whenever an unauthenticated client requests a protected resource, the microservice redirects the client to the identity provider to authenticate. After successful authentication, the client receives a SAML response with a SAML token. This access token is then used upon resource requests and validated by the microservice.



#### 4.2.6. Intrusion Detection/Prevention Systems

Especially in microservice-based software applications, the availability of individual microservices must be ensured. To check the availability, the architecture must take observability controls into account. Distributed logging systems and aggregators of runtime

metrics are typical techniques used for observability to check errors or the state of microservices. In the case of security, security-relevant events, such as actively executed attacks, must be detected and prevented or reacted to if necessary. One technology that is used to monitor security events is an IDS/IPS. These systems evaluate corresponding data at runtime. While network-based IDS/IPS monitor traffic into or within a microservice network, host-based systems check activities within a microservice or the container, e.g., system calls. A further distinction is made between signature-based and machine-learning-based IDS/IPS. While signature-based IDS/IPS look for concrete patterns, e.g., in messages or system calls, machine-learning-based approaches use data-driven algorithms like neural networks to detect anomalies. When a malicious activity or message is detected, IDS/IPS respond by reporting the incident to appropriate administrators. If necessary, these systems can also try to prevent attacks automatically, e.g., by blocking IP addresses in case of a denial of service attack.

**Definition 4.6: Intrusion Detection/Prevention System (IDS/IPS)**

An *Intrusion Detection/Prevention System (IDS/IPS)* is a family of technologies that monitors and/or prevents irregular or malicious behavior (intrusions) in a network or a system.

The catalog for IDS/IPSs is presented in Appendix B.6. The IDS/IPS schema in our catalog is structured as follows.

**Name** The name of the IDS/IPS approach.

**Security view** The security views this IDS/IPS approach can be modeled in.

**Algorithm approach** A description of the underlying algorithm, e.g., a machine learning-based approach or signature-based approach

**Evaluation asset** A list of assets that the IDS/IPS approach analyzes, e.g., traffic, message content, etc.

**Scope** Whether the IDS/IPS approach employs a network-based or host-based solution, whether internal or external content is monitored, i.e., within a microservice network or between the client and the microservice network.

**Placement** A list of assets where the IDS/IPS is placed on, e.g., microservice, network, API gateway etc.

**Threat** A list of threats the IDS/IPS approach detects or prevents, e.g., DDoS, code injection etc.

**Prevention** A description of how IDS/IPS approach responds to the intrusion (if any), e.g., block traffic, relocate microservice etc.

**Publications** A list of publications that mention or propose the IDS/IPS approach.

*Anomaly detection of distributed traffic* is an IDS/IPS approach proposed by Jacob et al. [JQL21; Jac+22]. Based on distributed traffic data, this approach uses a machine learning algorithm, namely Diffusion Convolutional Recurrent Neural Network (DCRNN), to identify anomalies in RPC traffic into a microservice network. Using this method, the authors were able to successfully identify several attacks performed on an experimental application such as password guessing brute force attacks and batch registration of bot accounts.

<b>Name</b>	Anomaly detection of distributed traffic
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	Diffusion Convolutional Recurrent Neural Network (DCRNN), machine learning
<b>Evaluation asset</b>	Distributed trace, Network traffic, RPC
<b>Scope</b>	External, Network-based
<b>Placement</b>	Per microservice
<b>Threat</b>	Batch registration attack, DDoS, Password Brute Forcing
<b>Prevention</b>	
<b>Publications</b>	[JQL21; Jac+22]



## 5. Defining and modeling a software engineering view of security

All models are wrong, but some are useful.

---

GEORGE BOX

### Contents

---

5.1. Research method . . . . .	52
5.2. Software engineering security metamodel . . . . .	52
5.3. Security views and concerns for microservices . . . . .	55
5.3.1. Authentication view . . . . .	55
5.3.2. Authorization view . . . . .	57
5.3.3. Secure communication view . . . . .	58
5.3.4. Secure storage view . . . . .	59
5.3.5. Secure build & deployment view . . . . .	60
5.3.6. Monitoring view . . . . .	61
5.4. Definition of security . . . . .	62

---

In the previous chapter, we performed a SLR based on the guidelines of Kitchenham et al. [KC07] to identify different security design concepts. These are the building blocks that can be used to design software architectures. From the findings of the SLR and the knowledge of existing related work, which we presented in Chapters 2 and 3, we now propose a modeling approach in order to be able to describe the security of software architectures from both, the software engineering and the attacker’s perspectives. We aim to unite the bases of software architecture modeling with the aspects of security in software applications.

Specifically, we now present the following models. First, we discuss the metamodel for software engineering security, which we developed in the context of this thesis. This metamodel is intended to represent and relate the most important elements between software architecture modeling and security in software applications. At the core of the model stands the concept of security views. In this work we have identified, from the findings of the SLR, six different security views for MSAs, which we specify in Section 5.3 concretely. We reason about the purpose of each view, and which concerns they should consider. Finally, we propose a new definition of security in the context of software engineering. This definition is intended to provide a basis for software engineers and researchers who want to explore and refer to these and potentially more security views in more detail.

## 5.1. Research method

In the following, we describe our research method for creating the proposed security model and security views. Essentially, we extract the findings from the SLR (cf. Chapter 4), related work (cf. Chapter 3) and the foundations presented (cf. Chapter 2) to create the model. Specifically, we draw on the ISO/IEC/IEEE 42010 standard [ISO11b] as the basis of our security model to define the fundamental concepts around security views (cf. Section 2.1). We then extend this model to include the aspects that we can retrieve from existing models for threat modeling. We see threat modeling as a type of security view that describes the security of an architecture from the attacker’s perspective. As a basis for the offensive part of our metamodel, we refer to the models proposed by Miede et al. [Mie+10] and Uzunov et al. [UF14]. These models concretely describe and model attack patterns in distributed systems. However, we argue that the defined concepts in these models also reflect the essential aspects of threat modeling. Finally, we extend our model to include aspects of the software engineering perspective. Specifically, there is a lack of concepts and methods to describe and model the defensive or constructive side of security. Based on the findings, we then develop the six concrete security views from the defensive, i.e., software engineering perspective. As a basis for this, the previously discussed mapping of security design concepts to the security views is developed (cf. Section 4.2). We create these mappings from the information and findings of the concrete security design concepts. Together with the supervisor of this thesis, these six security views are identified, discussed, and selected in an iterative process. Then, based on the SLR results and from our assertions and experiences, we develop the security concerns of the individual security views.

## 5.2. Software engineering security metamodel

In this thesis, a metamodel of software engineering security is developed which is depicted in Figure 5.1. The metamodel captures the most important concepts and their relations when modeling security in software architectures. It combines the efforts from existing models, i.e., the software engineering aspect from the ISO/IEC/IEEE 42010 standard [ISO11b], which we presented briefly in Chapter 2, and integrates the offensive security part developed by Miede et al. [Mie+10] and Uzunov et al. [UF14]. However, we do not claim that the model is complete, as further concepts may be introduced by future studies. The metamodel is structured hierarchically, with the most abstract concepts depicted on the top and increasing concreteness towards the bottom of the figure.

At the core of the model is the *security QA* which is composed of several security goals. The *security goals* (cf. Definition 2.11) define the different facets of security in a software architecture that need to be considered. Together they determine the degree of security a software application ultimately has. Each security goal is influenced by two forces, namely *threats* and *security design concepts*. Modeling those forces is key to describing and assessing the overall security of the software application successfully. Security goals are formulated by *security concerns* which are questions or specific security requirements

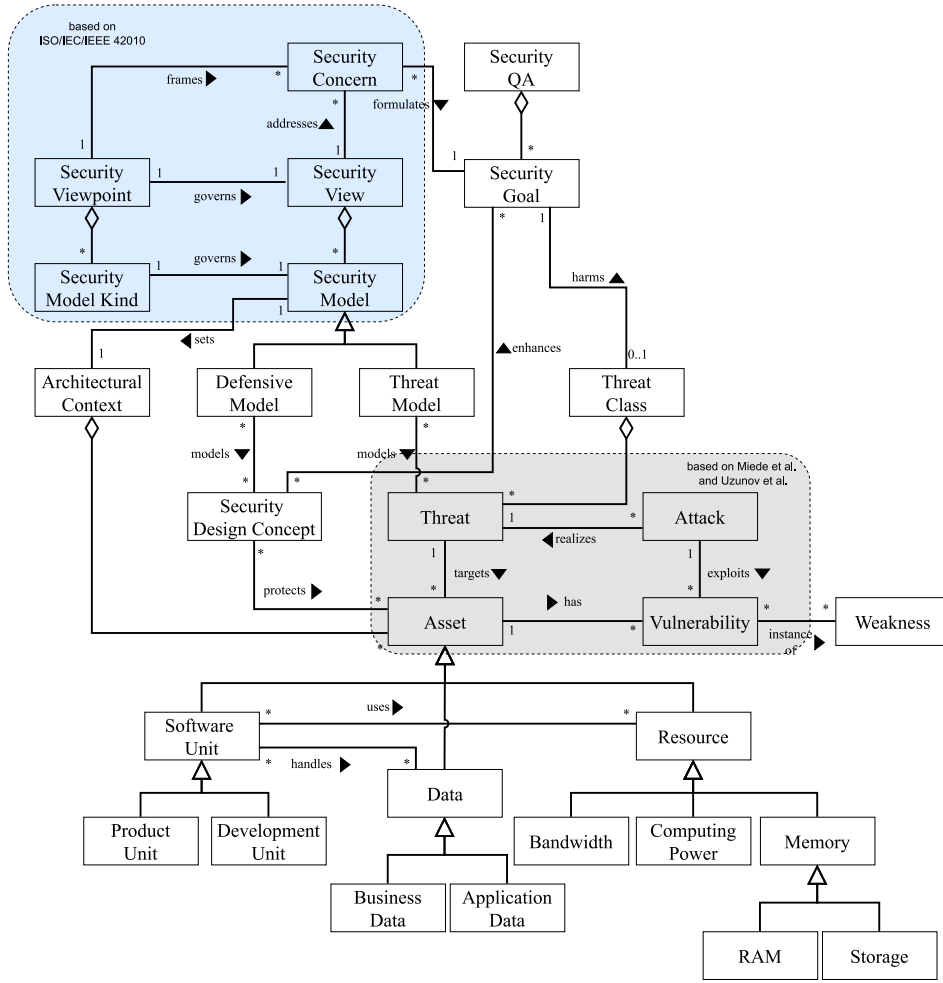


Figure 5.1.: The metamodel for software architecture security, capturing the essential concepts when architecting security in a software application, i.e., security views that describe one security aspect from a software engineering security perspective and its relations to other concepts such as security design concepts and security concerns, among others.

that need to be addressed in the software architecture. Related security concerns can be grouped to be addressed by a specific security view. The *security views* for microservices that are established in this thesis are the following: an authentication view (cf. Section 5.3.1), an authorization view (cf. Section 5.3.2), a secure communication view (cf. Section 5.3.3), a secure storage view (cf. Section 5.3.4), a secure build & deployment view (cf. Section 5.3.5), and a monitoring view (cf. Section 5.3.6). Each security view thus describes one aspect regarding the security of the software application and what it encompasses. We claim that the separation of these security views is necessary in order to analyze the impact that certain security design concepts have on these aspects and the

security goals. Furthermore, it follows the fundamental design principle of separation of concerns. Lastly, separating these views decreases the complexity of the architecture description as it establishes abstractions to reason about the architecture in a defined scope [LL13]. Otherwise, the increased complexity would hinder the understandability and maintainability of the model. Further, it enables us to view certain security concepts from different perspectives and within different scopes. For instance, the concept of an authenticated user may serve different purposes in an authentication view and an authorization view, respectively. Security views are governed by corresponding security viewpoints. A *security viewpoint* defines which modeling languages and techniques are used within its related security view, and how those models are created and to be interpreted. The modeling languages and techniques are specified by the *security model kind*. Concrete instances of the security model kinds are *security models* which comprise the specific security view. Therefore, one security view consists of several security models. We distinguish between offensive models, i.e., *threat models*, and *defensive models*. While both types of models consider the security of a software application, they differ in their perspective and their purpose.

*Threat models*, as discussed in Section 3.2.3, view the security of a system from the perspective of an attacker, and its goal is to identify which parts of the application are vulnerable and can be exploited to cause harm or gain some sort of advantage. Thus, the purpose of threat models is to identify threats. *Threats* (cf. Definition 2.10) represent potential violations of security in a software application. Each threat can be further classified according to its impact following the STRIDE methodology, as discussed in Section 3.2.3. Each such *threat class* poses a negative impact on one security goal. For instance, spoofing threats, i.e., whenever an attacker successfully claims the identity of another person, impact the security goal authenticity, as the attacker can successfully bypass the authentication mechanisms of the system. The threat term needs to be differentiated from other offensive-related terms such as attack, weakness, and vulnerability. While a threat is only a potential violation of security, this violation is realized by an *attack* that exploits an existing *vulnerability* in one of the system's *assets*. A *weakness* is a common design or implementation flaw that can appear in a software system. Concrete instances of those weaknesses are vulnerabilities. As discussed in Section 3.2.3, weaknesses and vulnerabilities are collected in the CWE and CVE databases. In total, we can leverage several threat models to identify those threats, e.g., data flow diagrams, process flow diagrams, attack trees, attack libraries [Sho14], as well as threat modeling methodologies like STRIDE [KG99] or PASTA [MU15].

On the other hand, *defensive models* shall be used to communicate which security controls and countermeasures are established. Although not standardized like UML [OMG17], defensive models exist for various purposes, e.g., to model security controls in network or cloud infrastructures [Chi+01; Vys12]. However, to the best of our knowledge, there exist no established or standardized modeling techniques or languages to create defensive models which capture the software engineering perspective. From a software engineering perspective, defensive models shall illustrate concrete principles and practices, i.e., security design concepts, employed by the software engineer in or-

der to enhance the security goals of the application. Within this thesis, we propose a first draft of this methodology as we provide exemplary security views and a catalog of security design concepts. *Security design concepts* (cf. Definition 2.12) are the design building blocks that can be integrated into the architecture model. They can be categorized according to several aspects, the most important ones are to which security view they can be applied, i.e., their use case, and which type of design concept they represent, i.e., how they can be applied. The types of design concepts that we consider in this thesis were defined in Section 4.2. Further investigations and studies could explore which existing modeling languages can be used for defensive modeling, or create new languages to be used in the corresponding security views.

Each security model also sets an *architectural context* which defines which parts of the architecture in terms of assets are considered in this model. An *asset* (cf. Definition 2.9) is any resource of a system, either a software unit, data or hardware resources. *Software units* can be further divided into product unit and development units. A *product unit* is any software unit that is directly employed in the final software product, e.g., microservices, communication paths between microservices, databases, containers, and clusters, among others. *Development units* are those software units that contribute to the software architecture, but are used in the development process, e.g., to build and deploy microservices or to version control the source code. We further distinguish *data* into business data and application data. *Business data* is the set of information primarily used to achieve business goals or to provide customers with business services. For instance, an online banking application might store business data such as customer information, banking accounts, as well as transaction data. On the other hand, *application data* is the set of data that is required by software units to function properly. Examples of application data are secrets, application configuration files and the source code itself. Typical *hardware resources* in a software application are *computing power*, *bandwidth*, and *memory*. For threat modeling and defensive modeling, it is crucial to identify the assets the corresponding models consider to determine which threats might occur and which security design concepts can be applied.

### 5.3. Security views and concerns for microservices

In the following section, we cover each of the aforementioned security views. Each view is described in terms of its functionality, what it encompasses, and what needs to be considered when modeling this view for MSAs. We provide an exemplary list of security concerns per view that need to be addressed in such models and explain why models need to address these sets of concerns.

#### 5.3.1. Authentication view

Authentication is the process of verifying whether the requesting entity is the one it claims to be, thus determining the identity of the requesting entity [Shi07]. The authentication process, therefore, consists of two steps: (1) an identification step in which

the requesting entity presents the verifier with an identifier, and (2) a verification step in which the verifier checks whether the identity claim is legitimate [SB15]. In MSAs, the challenge is that authenticated user details need to be shared securely between microservices over network or communication protocols [NC19b], e.g., HTTP [Fie+99], AMQP [Vin06], gRPC<sup>1</sup>. This is, in contrast to other architectural styles such as monoliths, more complex and error-prone. Thus, the architect needs to decide, among others, whether the authentication process becomes centralized or decentralized, using an internal or external identity provider, and employing single- or multi-factor authentication [Che+19]. Furthermore, due to its distributed nature, it may be necessary that microservices authenticate each other [MCF21b]. Especially in environments where the attack surface of the microservices is more "open", e.g., in IoT or fog-computing environments, proper authentication between those services becomes crucial. This distinction needs to be addressed in an authentication view, and different design concepts are required. Many publications reviewed in the SLR proposed to use concrete protocols and standards like OAuth 2.0 [Har12], OpenID Connect [FKS17], SAML [CMJ15], JSON Web Tokens [JBS15], and mTLS [RD08], among others [NC19b; MCF21b; Cha+22; Mel21].

### Authentication concerns

An authentication view needs to address, among others, the following authentication concerns:

**SC-AN1** Which services/clients can be trusted?

**SC-AN2** Which communication paths exist that need to be authenticated?

**SC-AN3** Which authentication method is used?

**SC-AN4** Which assets (secrets) are required to perform the authentication method?

**SC-AN5** Does the authentication method need to be secured?

**SC-AN6** Who validates the identity claim?

**SC-AN7** Who manages the identity information of services/clients?

An authentication view needs to model which entities can trust each other, and which entities need to authenticate to whom (**SC-AN1**). The trust relationship between services and clients reveals where authentication is needed and where it can be omitted. In order to determine which entities are required to authenticate each other, the model must clarify which entities provide and/or consume services, thus identifying the communication paths between entities that need to be authenticated (**SC-AN2**). Furthermore, the view must expose how services and clients authenticate, i.e., which authentication method is used (**SC-AN3**). Since some authentication methods require additional assets, e.g., mTLS requires client and server certificates, the view needs to illustrate which of those assets are required by which entity (**SC-AN4**). On the other hand, some

---

<sup>1</sup><https://grpc.io/>

authentication methods, e.g., password-based authentication, require that the communication is already secured. Such premises must be modeled by the view (**SC-AN5**). Ultimately, the view needs to address which entity in the authentication process acts as authenticator, thus verifying the identity claim (**SC-AN6**). Especially in more complex authentication scenarios that are introduced with authentication frameworks like OpenID Connect/OAuth 2.0, this becomes crucial. Although the authenticator usually stores identity information, the model needs to illustrate the responsible entity, if it deviates from this (**SC-AN7**).

### 5.3.2. Authorization view

Authorization is the process of granting approval to a system entity, i.e., microservice or client, to access a system resource. [Shi07]. Microservices and clients are assigned certain permissions, i.e., a service/client can only access the resources or functionality of another service if it has the appropriate rights to do so. Authorization is often used in conjunction with authentication: to determine the permissions someone has, that entity must first be identified. There exist several access control models that can be used to model the assignment of permissions, the best known being *Role-based Access Control (RBAC)* [FK92] and *Attribute-based Access Control (ABAC)* [Hu+19]. In RBAC, each service/client is assigned a set of roles and each role is assigned a set of permissions. It is then determined whether the entity that wants to access a resource has the appropriate role to do so. In ABAC, attributes, entities, and environments are the central building blocks for defining access rights. In contrast to RBAC, ABAC allows more fine-grained access models to be developed, but this also increases the complexity. In the context of MSAs, authorization forms a central aspect. It may be required to restrict access on certain APIs a microservice provides. Authorization protocols like OAuth 2.0, SAML, and XACML [SW13] are primarily used to implement authorization policies [Cha+22; PJ19; AC22]. Furthermore, each microservice needs restricted access to the infrastructure it runs on, e.g., in a container.

#### Authorization concerns

An authorization view needs to consider, among others, the following authorization concerns:

**SC-AZ1** Which authorization model is used?

**SC-AZ2** Which roles/attributes need to be considered?

**SC-AZ3** How are roles/attributes assigned to services/clients?

**SC-AZ4** Which permissions are assigned to each role? / Which access rules exist and on which attributes do they depend?

**SC-AZ5** Who determines the permissions of the entity?

**SC-AZ6** Who validates whether the requested action is allowed?

An authorization view must first represent which authorization model is used, e.g., RBAC or ABAC **SC-AZ1**. After an authorization model has been selected, corresponding roles or attributes must be determined that are used to decide on access **SC-AZ2**. Then the view must show which roles or attributes are assigned to each client or service **SC-AZ3**. It must be determined which access rights are assigned to each role or which rules should exist and on which attributes these are based **SC-AZ4**. Furthermore, the view must show by whom the access rights of a microservice are defined and where they are stored **SC-AZ5**. Finally, it must be modeled which component validates the access rights for a new request and determines whether the request may be executed or is rejected **SC-AZ6**.

### 5.3.3. Secure communication view

In MSAs, secure communication deals with the protection of data transmitted over communication channels between clients and microservices, but also between microservices themselves, and is an important aspect that needs to be considered [Zdu+22; YB18b]. Communication paths between clients and microservices need to be secured because these communication paths usually appear over the internet, and thus are more prone to be targeted by attackers. On the other hand, communication between microservices might need to be secured as well [AC22], depending on the environment. For instance, more distributed MSAs like IoT or fog-computing environments left the communication paths between microservices more open to attacks, but also cloud-native applications need to consider hardening their communication paths due to possible compromises in the MSA [Mel21]. Thus, it is required to identify the communication paths in a microservice-based application and which of those communication paths need to be secured. This includes all communication paths between microservices and microservice consumers, as well as those communication paths that are introduced by consuming third-party services. Furthermore, the extent to which the communication paths need to be secured must be taken into account as well. To do this, it is first necessary to analyze which information is transported via which communication paths and what level of confidentiality this data has. In addition, the extent to which the data is processed at the corresponding recipient must be analyzed. Consequently, it must be decided whether the communication path must be encrypted, whether it must be protected against modification, and whether message replay should be prevented.

#### Secure communication concerns

A secure communication view needs to consider, among others, the following secure communication concerns:

**SC-SC1** Which communication paths exist in the microservice architecture?

**SC-SC2** What type of information is transported on these communication paths?

**SC-SC3** Which communication paths need to be secured?



**SC-SC4** How do communication paths need to be secured?

**SC-SC5** Which assets are required to secure the communication path?

A secure communication view must first identify all communication paths that exist in the MSA (**SC-SC1**). Then it is required to identify what type of information is transmitted through those communication paths (**SC-SC2**). In particular, one needs to determine the level of confidentiality of the transmitted data. For instance, personal user information might require more security considerations than other types of data. The next step is to identify the set of communication paths that are generally required to be secured (**SC-SC3**). Not all communication paths might require to use encrypted protocols to be secured. Then, for each of those communication paths that need to be secured, one needs to determine how the respective communication path should be secured (**SC-SC4**). In particular, it must be decided whether the transmitted data must be encrypted to protect it from eavesdropping, whether the data must be protected against modification and whether message replay is a concern. Lastly, further assets might be required to employ respective security design concepts to establish a secure communication path, and those assets are required to be identified (**SC-SC5**). For instance, the mTLS protocol, i.e., TLS where both the server and the client also mutually authenticate themselves, is a possible security design concept to secure communication paths between microservice for confidentiality and integrity but requires that those microservices own respective certificates.

#### 5.3.4. Secure storage view

Secure storage deals with the custody of data in a MSA. Through architecture patterns like shared database or database per service [FL14; Ric19] there are different scenarios in MSAs where data can be stored. In principle, it is important to analyze which data is stored at which locations. A distinction must be made between business and application data. Business data is all the data that is processed in the microservice to execute the business functions. Application data, on the other hand, is information that a microservice requires in order to fulfill its technical functionalities. This includes configuration data in the microservice or the cluster, e.g., CPU and memory limits, but also secrets, keys, and certificates.

##### Secure storage concerns

A secure storage view needs to consider, among others, the following secure storage concerns:

**SC-SS1** Which storage assets exist?

**SC-SS2** Which storage assets need to be secured?

**SC-SS3** How do storage assets need to be secured?

**SC-SS4** Which assets are required to secure the storage asset?

A secure storage view must be able to show which data is stored in which locations (**SC-SS1**). Similar to the secure communication view, the secure storage view needs to determine for each asset whether it needs to be secured (**SC-SS2**) and to what extent the storage asset needs protection (**SC-SS3**). Furthermore, securing the storage asset might introduce further assets (**SC-SS4**), e.g., encryption keys that are required to protect a storage database.

### 5.3.5. Secure build & deployment view

The development of microservices was accompanied by the trend towards DevOps and DevSecOps methodologies and the automation of the build and deployment processes [Bas17]. Enhanced automation enables the build and deployment process to be triggered with every code change, i.e. commit, and the application is thus updated with every change. In the build process, the microservice application is integrated, i.e., necessary dependencies are drawn, the code is compiled if necessary, and then unit and integration tests are performed. Once the build process has been completed, the deployment of the application into the development or production environment begins. If the application is to be packaged in a container image, the image is also published to an image registry, e.g., Docker Hub<sup>2</sup>. Then, the published image is used to create a new container of the application, which is deployed to the target system, such as a Kubernetes<sup>3</sup> cluster. Unlike most other views, the secure build & deployment view represents a view in the development context, rather than seeing the microservice application in the product context. Nevertheless, this view forms an essential aspect of security in MSAs, since elementary processes take place in this view which has a significant impact on the security [KG20]. Besides unit and integration tests, which check the functionality of the application, SAST and DAST can detect security implementation bugs [TSM17; Tor+19], such as XSS or SQL injection threats. SASTs scan the code of the microservice application for potential errors, while DASTs test the application at runtime. Furthermore, dependency scanning or *Software Component Analysis (SCA)* can be used to find existing vulnerabilities in dependencies. It should be noted that in addition to the application dependencies, there are often dependencies in other software parts, for example in the containers in which the microservice application is executed [Tor+18]. Furthermore, the runtime of the microservice application must be configured securely. This includes, among other things, restricting the ingress and egress, i.e., incoming and outgoing traffic, of the microservices based on network policies and restricting the permissions of applications that are executed in the containers [Thr+21]. Lastly, the build pipeline itself is another potential vulnerability that needs to be secured accordingly. However, securing the build pipeline is outside the scope of this view and will not be listed in detail in the following course of the work but should be addressed in future work (cf. Section 8.2).

---

<sup>2</sup><https://hub.docker.com/>

<sup>3</sup><https://kubernetes.io/>

**Secure build & deployment concerns**

A secure build & deployment view needs to consider, among others, the following secure build & deployment concerns:

- SC-SB1** How is the build & deployment process set up, i.e., which phases are executed during this process?
- SC-SB2** Which application and container dependencies exist in each microservice?
- SC-SB3** Which vulnerabilities exist in those dependencies and how severe are they?
- SC-SB4** Which threats or weaknesses are tested automatically against which microservices?
- SC-SB5** On which machines or deployment units are microservices deployed?
- SC-SB6** How are microservices configured?
- SC-SB7** Which capabilities and actions do microservices have permissions for?

A secure build & deployment view must first establish an overview of the build process and how microservices are deployed (**SC-SB1**). This overview must identify all activities and tasks that are executed during this process, e.g., compile, testing, and deployment tasks. The view must be able to identify all dependencies that are included in all software components of the microservice, i.e., any application and container dependencies that are fetched during the build process (**SC-SB2**). Based on this SBOM, the view must list all existing vulnerabilities in those dependencies (**SC-SB3**). Furthermore, this view needs to illustrate which security tests are executed, which threats and weaknesses they cover, and against which microservices they are executed (**SC-SB4**). Lastly, the view should establish an overview of the infrastructure (**SC-SB5**), the configuration of each microservice in detail (**SC-SB6**), and which permissions each microservice ultimately has (**SC-SB7**). Optimally, the view identifies any configuration smells [Rah+21], e.g., microservices that run with root permissions in their container.

**5.3.6. Monitoring view**

Monitoring is the process of observing actions and resources in a MSA, evaluating them, and reacting accordingly if necessary [Shi07]. Observable actions can be, among others, requests from clients or microservices, access and exchange of data, and, on a very fine-granular level, system calls on a host. Resources whose demand is typically observed are CPU, memory, and bandwidth. In particular, in a MSA, due to the high degree of distribution, the difficulty is to adequately collect and, in a centralized or decentralized manner, evaluate all relevant information. Typical monitoring practices include logging, exception tracking, health checks APIs, and distributed tracing [Was+21b]. In the context of security, monitoring is essential to detect and possibly prevent attacks towards the MSA. For instance, IDS/IPSs can be integrated into the MSA to detect denial of service attacks [JQL21; Jac+22; Alm+22] on the network layer or observe when an

intruder performs irregular actions and lateral movement in the microservice network [OY17; YO18; Tor+19; Osm+19]. Thus, security design concepts for monitoring can be placed at different layers of the MSA. On the one hand, IDS/IPS systems can be placed at the border leading into the MSA [Bay+21], e.g., at API gateways, but also monitoring solutions that observe the entire internal microservice network and distributed traces may be applicable [CHC19; JQL21; Jac+22]. Therefore, a distinction must be made between external and internal monitoring. Furthermore, monitoring can be performed at each microservice at different layers. For instance, each request can be checked for harmful content, e.g., code injection attacks, using appropriate input validation [Che+19; NC19b], but monitoring of microservice resources may also be necessary in order to detect potential attacks.

### Monitoring concerns

A monitoring view needs to consider, among others, the following monitoring concerns:

**SC-MT1** Which entities need to be monitored?

**SC-MT2** Which actions need to be traced?

**SC-MT3** How and by whom are traced actions evaluated?

**SC-MT4** How are irregular actions reported?

**SC-MT5** Which prevention measures are executed (if any) on an irregular action and by whom?

A monitoring view must first identify the set of entities, i.e., microservices, clients, communication paths, among others, that need to be continuously monitored (**SC-MT1**). The view must be able to represent which actions and resources exist in the architecture and which of these are being observed (**SC-MT2**). It must be able to represent which components perform the monitoring and which components evaluate the observed actions and resources (**SC-MT3**). Furthermore, this view must illustrate what actions are taken when a conspicuous action or overused resource is detected (**SC-MT4** & **SC-MT5**).

## 5.4. Definition of security

In the previous chapters, it was shown that security, from a software engineering perspective, is usually regarded as a QA. The security QA is composed of several sub-characteristics or security goals (cf. Definition 2.11). Each of these security goals describes the ability of a system to protect data, services and availability from unauthorized access. As described in Section 5.2, security goals are essentially influenced by two forces: threats harm security goals, while security design concepts enhance them. The task of the software engineer is therefore to mitigate potential threats and to increase the security of the software architecture by employing suitable security design

concepts. Achieving these goals ultimately leads to a positive impact on security QA. To describe this, we claim, it is necessary to apply appropriate modeling techniques. In Section 3.2, we presented some modeling techniques available to a software engineer, including quality models, maturity models, and threat models. However, these models lack the software engineering perspective. From a software engineer’s perspective, an appropriate view and associated models must communicate which design concepts are embodied in the architecture. This includes which design patterns and tactics are used, which design principles the architecture follows, and, on a technical level, which protocols, frameworks, and other technologies such as IDS/IPS are used. Furthermore, it is important to assign one or more views to these design concepts. Only then can one effectively analyze whether the established security design concepts achieve the desired effect.

We argue that existing definitions of security, such as those found in ISO/IEC 25010 [ISO11a] do not adequately reflect this software engineering perspective. Although the definition of security architecture provided in the NIST special publication 800-160 [RMO18] employs security-relevant views that convey information about security-relevant elements, it does not specify concrete views and elements that need to be tackled in such an architecture and the impact on the security goals. For this reason, we propose the following definition of security in the context of software engineering, which anchors the aforementioned essential building blocks.

**Definition 5.1: Security (in Software Engineering)**

*Security* in the context of software engineering is a QA that is improved by achieving security goals. It defines the ability of a software application to adequately protect its assets. Each security goal is formulated by a set of security concerns that embody the interest and requirements stated by the architecture’s stakeholders. The effects of security design concepts and threats are described and evaluated by modeling security views.



## 6. Evaluation

Security is a process, not a product.

---

BRUCE SCHNEIER

### Contents

6.1. Evaluation method . . . . .	65
6.1.1. Case study . . . . .	68
6.2. Evaluation results . . . . .	72
6.2.1. General questions . . . . .	72
6.2.2. Authentication view case study . . . . .	73
6.2.3. Modeling security views . . . . .	76

In order to validate the contributions of this thesis, we decided to conduct an evaluation. Specifically, we wanted to find out whether the modeling approach presented in this thesis helps software engineers and architects to better communicate the security of their architectures. For this purpose, we evaluated whether the metamodel and the principle of security views, as well as the collection of security concerns per view, are useful. To conduct the evaluation, we performed semi-structured interviews with research and industry experts. In this evaluation, we specifically discussed the notation and modeling of one security view, namely the authentication view, with the participants using an exemplary case study. In doing so, we wanted to find out how effective this approach is, find its weaknesses, and what research directions can be addressed in future work.

In the first part of this chapter, we present the method of our evaluation, as well as the questionnaire and case study that was prepared before conducting the interviews. Then, in the second part, we present the results of the evaluation. In doing so, we elaborate on, interpret, and analyze the responses of the participants.

### 6.1. Evaluation method

Semi-structured interviews are characterized by a flexible interview style that is nevertheless oriented toward specific questions or topics [RM16]. In contrast to structured interviews, the interviewer has the opportunity to ask follow-up questions and dig into details. In contrast to unstructured interviews, an interview structure provides a framework to which the interview partners can orient themselves. This ensures that the focus of the interview is not lost.

Table 6.1.: Evaluation questionnaire.

<b>General questions</b>
What is your current role?
Do you have experience in modeling security in a software architecture?
How much experience do you have with microservice architectures?
What comprises security in the context of software architecture in your opinion?
<b>Authentication view case study</b>
What does authentication entail in your opinion?
Do you agree with this set of authentication concerns?
Are there any missing authentication concerns that should be added?
Does this model address the aforementioned authentication concerns appropriately?
Does this model effectively communicate the established security design concepts regarding authentication?
Which modeling elements should be added or removed to enhance this model?
Does such a model enhance common understanding and communication about authentication in a software architecture?
<b>Modeling security views</b>
How would you assess the distinction of security modeling into the set of security views established in this thesis, i.e., authentication, authorization, secure communication, secure storage, monitoring, secure build & deployment?
Which views of security need to be considered in a software architecture?
Which concerns would need to be addressed in these security views?
Which modeling notations and languages would you suggest for these views?
To what extent would special-purpose modeling notations and languages for the respective views, as presented in the case study, be helpful in answering the concerns of this view?
Can we study the influence of established security design concepts using security views and corresponding models?
Do such views and models describe security from a software engineering perspective appropriately?
Overall, how would you assess the worth of our approach for modeling security in software architectures and microservice architectures specifically?



In preparation for the interviews, an introductory slide set was prepared which briefly presented the motivation and background as well as the notion of security views and security concerns. In this introductory slide set, the participant was also informed about the structure and procedure of the interviews, as described in this chapter. Each participant received an invitation email including the introductory slide set to subsequently schedule a time to conduct the interviews.

The interviews were structured as follows. Each interview was conducted online via Zoom<sup>1</sup> and was divided into four sections, namely an introduction, general questions, the authentication view case study, and modeling security views. An accompanying set of slides was also created for the interview, which included the key organizational notes, as well as the questions from Table 6.1 and the case study materials.

First, the participant was informed about the motivation and purpose of this thesis and the evaluation in the introduction. The participants were allowed to ask questions about the organization of the evaluation or questions about the content of the thesis, which were then clarified. Each participant was guaranteed the confidentiality of their personal information, that this data would not be processed, published, or passed on to third parties. Subsequently, permission to record the interview for transcription was asked for and, again, the confidentiality of the audio material was guaranteed that it would only be used for this purpose and subsequently deleted.

After that, the second part of the interview began. First, we asked some general questions regarding the specific role the participant is currently performing in their job as well as previous experience with modeling security in software architectures and microservices in general. These questions served as a warm-up for the participant to feel comfortable and slowly introduce the topic of conversation. Furthermore, we wanted to know whether the participant already has experience in security modeling, which modeling approach was followed and whether specific modeling languages and tools were used.

In the third part of the interview, the case study was presented and discussed. First, the participant was asked about his or her understanding of authentication. In this way, we wanted to find out whether there is a general understanding of this security aspect and where the differences lie. Then the participant was introduced to the modeling approach based on the case study. To do this, we first presented the set of authentication concerns that were developed in the context of this thesis (cf. **SC-AN1** to **SC-AN7**) and asked the participant for feedback. Afterward, the models of the case study were presented to the participant. The general architecture model of the Spring Petclinic project was shown and explained as presented in the following section. Consequently, the models of the authentication view were shown, as well as the justification of the individual modeling elements given and which concerns are concretely addressed by it. Details of the case study architecture and authentication models are given in Section 6.1.1. In case of ambiguities, the participant could ask questions subsequently. Finally, the remaining questions about the authentication view case study were asked and discussed. In doing so, we wanted to get feedback from the participant, specifically whether this

---

<sup>1</sup><https://zoom.us/>

modeling was helpful and assisted in communicating and understanding design concepts for authentication. The purpose of the case study was to demonstrate the modeling approach concretely to the participant using an example. In this way, we wanted to bring the concepts that we had captured in the metamodel closer to the participant in order to evaluate them in detail in the fourth part of the interview.

In the fourth and final part of the interview, the modeling approach, i.e., modeling with security views, was evaluated overall. This included the concepts and relationships established in the metamodel as well as the terminology used for the various security views. We first had the participant evaluate whether he or she generally considered the division of security into different perspectives to be useful. Subsequently, the participant was asked to name further views and corresponding concerns that we may not have considered in this thesis. Furthermore, we wanted to find out whether we can model these views more effectively using specific modeling languages and tools and to what extent we can thereby evaluate the influence of the security design concepts involved. Finally, we asked the participant if we had achieved our goal of modeling security from a software engineering perspective through this modeling approach and how he or she would evaluate the approach. By doing so, we wanted to learn specific directions for further research and improvements to our model.

### 6.1.1. Case study

Since the metamodel and the concept of security views are rather abstract artifacts, we decided to evaluate these results utilizing an exemplary case study. This means that each participant in the evaluation was presented with this case study in order to discuss it afterward. The case study includes a fictitious microservice architecture, more specifically an adapted version of the Spring Petclinic microservices project<sup>2</sup>, as well as a set of models intended to represent an authentication view of this architecture. The models were created solely for the purpose of the evaluation in this thesis.

#### Spring Petclinic microservices

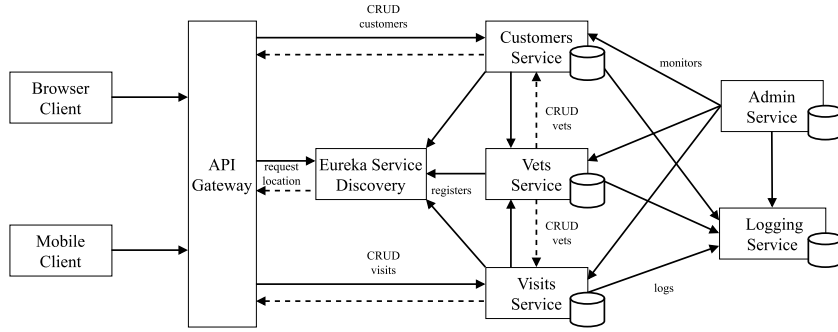
The Spring Petclinic project is an open-source project that is intended to exemplify various technologies for microservices development based on the Spring Framework<sup>3</sup>. The application is used to manage customer data and information about veterinarians and to create visits to them. For the evaluation, we used a slightly adapted version of this architecture as the basis for the case study.

The architecture diagram for the project is shown in Figure 6.1a. The MSA consists of several microservices that implement either a business function or a utility function. The business microservices **Customers Service**, **Vets Service**, and **Visits Service** provide APIs to manage the corresponding data, i.e., create, read, update, and delete (CRUD operations). Each of these microservices is equipped with its own database for

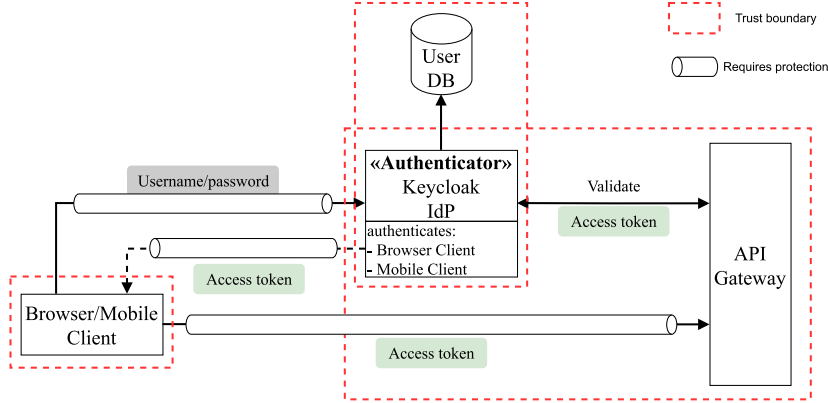
---

<sup>2</sup><https://github.com/spring-petclinic/spring-petclinic-microservices>

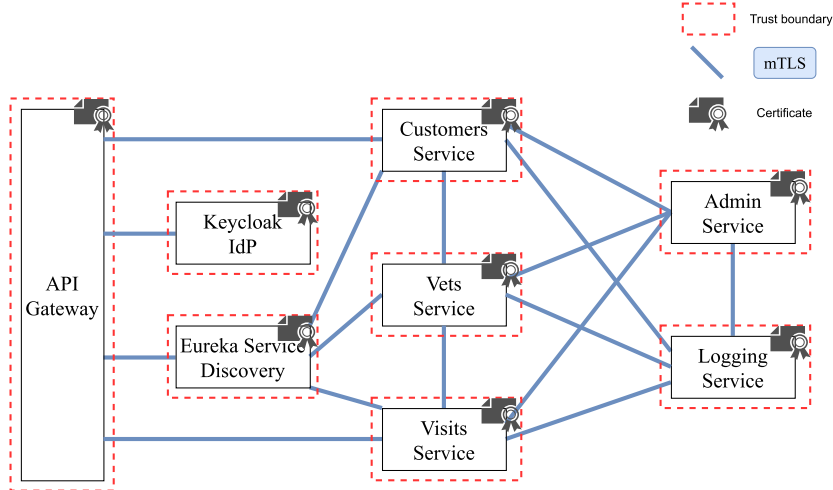
<sup>3</sup><https://spring.io/projects/spring-framework>



(a) The initial Spring Petclinic architecture model.



(b) Authentication model for external clients. Clients authenticate themselves towards the API Gateway using the Keycloak Identity Provider instance as authenticator.



(c) Authentication model for service-to-service communication. Microservices use the mTLS protocol with client certificates to authenticate each other.

Figure 6.1.: The authentication view case study, including the initial Spring Petclinic architecture (a), frontend (b) and backend (c) authentication models.

managing its data. The microservices can be scaled as needed and thus have a dynamic location, i.e., their IP addresses are not static. The frontend consists of a **Mobile Client** and a **Browser Client**. In order for these clients to access the corresponding microservices, requests are routed through an **API Gateway**. The **API Gateway** serves as a separation between the external clients and the internal microservice network. To ensure that the **API Gateway** is informed about the dynamic addresses of the microservices, a **Eureka Service Discovery**<sup>4</sup> microservice is embedded in the architecture. As soon as a microservice is newly instantiated, it registers with the service discovery microservice. The **API Gateway** can then request the address of the requested microservices from the service discovery. Furthermore, there are two other utility microservices, namely the **Admin Service** and the **Logging Service**. The **Admin Service** is used to configure and monitor the individual business microservices. Business microservices send their log information to the **Logging Service**, thus acting as a centralized logging system. The **Admin Service** can query the **Logging Service** for corresponding log information from the business microservices.

### Authentication view

In this case study, we created an authentication view for the Spring Petclinic MSA. This security view is intended to address the authentication concerns we presented in Section 5.3.1 through appropriate models and specific modeling elements, respectively. To create these models, we proceeded as follows. First, for a closer look and clarity, we have chosen two architectural contexts, respectively for the authentication of external clients, depicted in Figure 6.1b, and the authentication between the microservices internally, depicted in Figure 6.1c. In the following, we now explain these models, which authentication concerns they address in the process, and the modeling elements which illustrate the established security design concepts. Note that Figure 6.1 only depicts the final authentication models. The authentication models shown during the interviews were explained and evolved incrementally, cf. Appendix C for a complete overview of all presented models.

In general, the models must be able to represent the extent to which the entities, i.e., clients and microservices, trust each other (cf. **SC-AN1**). For this purpose, we use the modeling element of *trust boundaries*, represented in each case by the red dashed boundary lines in both models. The concept and notion of trust boundaries are often used in DFDs [Sho14]. All elements that are together in one of those boundaries trust each other and do not require authentication. The overlap between trust boundaries and communication paths also addresses the concern of which communication paths need to be authenticated (cf. **SC-AN2**). For the Spring Petclinic architecture, we decided to establish the security design principle *Zero Trust Networking* [Mel21], i.e., no other client or service is trusted overall and thus authentication is required in all cases.

For the authentication of external **Clients**, a combination of **Username/Password** should be used. To ensure a lightweight session for **Clients**, an **Access Token** should

---

<sup>4</sup><https://github.com/spring-cloud/spring-cloud-netflix>

be used that has to be sent and validated with every request of the **Client**, thus employing the architectural security pattern *Access Token* [Sto+18; JLE18; Was+21b]. Furthermore, we employ the security tactics *Password-based Authentication* [Zdu+22] and *Token-based Authentication* [JLE18; PS19; Zdu+22], respectively (cf. **SC-AN3** and **SC-AN4**).

However, the management of user information and access tokens is outside the responsibility of the **API Gateway**. For this reason, an additional microservice is to be integrated for the creation and verification of the token, namely the **Keycloak Identity Provider (IdP)**<sup>5</sup>, which acts as an authenticator. The **Keycloak IdP** manages the relevant identity information (cf. **SC-AN7**) and is responsible for authenticating external clients (cf. **SC-AN6**).

The **Client** first authenticates themselves by sending the **Username/Password** combination for identity verification to the **Keycloak IdP**. If authentication is successful, the **Keycloak IdP** generates a new **Access Token** and sends it to the **Client** as a response. The **Client** uses the **Access Token** to authenticate itself to the **API Gateway**. The **API Gateway**, together with the **Keycloak IdP**, verifies the validity of the **Access Token**.

The communication between **Clients** and the **Keycloak IdP** or **API Gateway** is an open communication path over the internet. However, the employed security tactics depend on the security tactic *Encrypted Communication* [NC19b; Zdu+22]. For this reason, additional protection of these communication paths in the form of encryption is necessary (cf. **SC-AN5**), e.g., by using the security protocol *TLS* [RD08; YB18b; AC22]. From the **Client's** perspective, the communication between **Keycloak IdP** and the **API Gateway** does not require any additional protection, since this communication path takes place in the internal network. Thus, the inclusion of the **Keycloak IdP** in the architecture implied that the trust boundaries were required to be redrawn so that now the **Keycloak IdP** and the **API Gateway** each trust each other. However, this leads to a violation of the security design principle of *Zero Trust Networking* that we follow. Since the **Keycloak IdP** is another microservice in the internal microservice network, this problem is now solved in the second architectural context, the backend.

Although we assume in this case study that the microservices of the architecture have been deployed in an internal network, we will still establish an authentication method between microservices. This is done for two reasons. First, attackers might be able to inject a malicious microservice into the network. Microservice authentication should prevent unknown microservices from being accepted and used in the network. Other forms of compromises may apply as well. Second, the *Zero Trust Networking* [Mel21] security design principle requires that no other service is trusted without first authenticating itself. For this reason, we have also established the modeling element of trust boundaries in the architectural context of the backend (cf. **SC-AN1**). We now correct the violation that the **API Gateway** and the **Keycloak IdP** co-exist in a common trust boundary.

We recognize that authentication is required between each communicating pair of microservices (cf. **SC-AN2**). For authentication between microservices, we now want

<sup>5</sup><https://github.com/keycloak/keycloak>

to employ the security tactic *Protocol-based Authentication* (cf. **SC-AN3**). For the concrete implementation, we use the security protocol *mTLS*, which stands for mutual TLS [RD08; YB18b; AC22]. This protocol requires that each microservice has a digital **Certificate** (cf. **SC-AN4**) to identify itself. Furthermore, this authentication method does not require any further protection in the form of encryption or similar (cf. **SC-AN5**), and each microservice manages its identity information independently (cf. **SC-AN7**) and authenticates itself through the protocol (cf. **SC-AN6**).

## 6.2. Evaluation results

The following section presents the results of the evaluation. We provide a summary and an interpretation of the given answers by the participants of the interviews. The interviews were conducted between October 24 and November 14, 2022. The evaluation transcriptions can be found in Appendix D.

### 6.2.1. General questions

The first part of the interview focuses on the participants. They were asked about their current role and how much experience they have in security modeling and MSAs.

#### What is your current role?

In total, 7 research and industry experts participated in the evaluation from different companies and research facilities. Participants P1 and P7 are researchers for IT security and secure software development. Participant P2 is a data scientist with background knowledge in software development. Participants P3, P4, and P6 are software engineers and architects each employed by an IT consulting company. Participant P5 is an IT security consultant with a focus on secure software development and secure coding.

#### Do you have experience in modeling security in a software architecture?

Due to the different backgrounds, each participant has a different level of experience regarding security modeling in software architectures. Participant P1 mentioned working on several projects in which a model-based development approach for security was followed using standards like IEC 62443 [IEC21] and ISO/IEC 27001 [ISO13] in conjunction with modeling tools and languages like SysML [OMG19] and Enterprise Architect<sup>6</sup>. Participant P5 has experience in threat modeling specifically with STRIDE [KG99] and PASTA [MU15]. The other participants either have some experience with attack libraries like OWASP Top Ten [www] or knowledge gained by university courses and self-studying.

---

<sup>6</sup><https://www.sparxsystems.de/newedition/enterprise-architect-16>

**How much experience do you have with microservice architectures?**

All participants have at least some experience with MSAs either by hands-on experience or knowing the concepts and ideas behind this architectural style. MSA experience or knowledge of distributed architectures similar to MSAs of the participants ranges from 1 to 5 years.

**What comprises security in the context of software architecture in your opinion?**

This question focuses on the concepts and relations we formed in the security metamodel (cf. Section 5.2). Without introducing the metamodel to the participants, we wanted the participants to identify potential gaps and missing concepts in the metamodel, that should be incorporated in an augmented version. According to the participants, the primary focus of security comprises the secure management of assets, i.e., securing data at rest and in transit as well as hardening infrastructure, monitoring and tracing security events, and considering security in development processes. Participant P5 highlighted the importance of threat modeling in the secure software development process. Participant P1 provided a thorough answer depicting how security must be considered in each development phase. This ranges from gathering security requirements in the initial phases, to secure design and implementation best practices, and finally reaching the deployment, monitoring, and verification phase in which security requirements and their fulfillment need to be verified. The information gathered during these phases must also serve as a basis for further iterations in the feedback loop, as certain insights become clearer later in the development process.

**6.2.2. Authentication view case study**

In this part of the interview, the authentication view case study was presented and discussed. In particular, the participants were asked whether the authentication models answer the presented authentication concerns and whether these models clarify aspects of authentication in the architecture.

**What does authentication entail in your opinion?**

This question focuses on whether there is a common understanding of authentication and what it entails. 6 out of 7 participants explained that authentication deals with the identification of individual users and other services and that different authentication methods exist to perform the identification and verification phases. 3 participants recognized that authentication needs to be differentiated from authorization, although both concepts appear in conjunction in most cases. Participant P5 additionally differentiated authentication, i.e., presenting the identity claim to some system, from authentication, i.e., the verification of the claimed identity.

### **Do you agree with this set of authentication concerns?**

The authentication concerns **SC-AN1** to **SC-AN7** were presented to the participants and the reasoning behind these concerns. The participants were asked whether they agree with these concerns or rather see some concerns as redundant or would reformulate them. 4 participants generally agreed with the set of authentication concerns. Participant P1 mentioned that due to the trend towards Zero trust architectures nowadays, concern **SC-AN1** becomes rather redundant, but it makes sense to model trust explicitly. On the same concern, participant P3 mentioned that the necessity of authentication and trust depends on the value of the assets to be protected which should be considered additionally in this concern. Participant P7 noted that this concern also needs to reflect on how to deal with external clients unknown beforehand to prevent spoofing. Participant P7 also suggested rephrasing **SC-AN2** and **SC-AN3** to only include *secure* communication protocols and authentication methods. Deprecated cryptographic algorithms should be excluded by this concern. Lastly, participant P4 noted that **SC-AN7** might indicate that there always exists a central service that manages the identity information about all services and clients, thus leading to a potential single point of failure.

### **Are there any missing authentication concerns that should be added?**

This question focuses on the completeness of authentication concerns identified in this thesis. Participants P6 and P7 indicated that in addition to **SC-AN4** they would add a concern about the secure storage of these introduced assets, specifically where they are stored and how they are secured. Participant P2 would add one concern about the secure introduction of secrets, e.g., API keys and passwords needed by the microservices, into the system. An additional concern about the concrete procedure after successful and failed authentication was suggested. Finally, participant P1 missed a concern about authentication trail logs, i.e., how successful and failed authentication requests are logged by the system, who can access this information, and how are they protected.

### **Does this model address the aforementioned authentication concerns appropriately?**

Before this question was asked, we presented and explained the Spring Petclinic architecture model (cf. Figure 6.1a) and subsequently the authentication models of this architecture. More specifically, we explained which sets of concerns each authentication model addresses and provided reasoning on how these concerns were addressed by the corresponding security design concepts. All authentication models presented to the participants are depicted in Appendix C. In general, all participants agreed that the authentication concerns were addressed by these models. However, 2 participants were confused by the Zero trust violation in the frontend authentication model (cf. Figure 6.1b). After clarification that this model takes the perspective of the client and does not concern with backend authentication, participant P2 suggested to rather anticipating these details in the frontend authentication model. Otherwise, without a supporting explanation, such



violations rather lead to confusion than clarification. In the same model, participant P7 missed modeling elements regarding the verification of the validity of a session, i.e., how long can a client use an access token for identification until it expires. Participants P2 and P4 also missed an authentication model that provides an overview of the employed authentication techniques in the whole architecture. Participant P1 also noted that understanding these authentication models would probably require security knowledge to some extent. However, it is useful for reviews performed by security experts. For completeness, participant P3 would additionally sift through security documents and catalogs like the OWASP Application Security Verification Standard [OWASP19].

**Does this model effectively communicate the established security design concepts regarding authentication?**

Effective security models require that they present which security design concepts are employed in the software architecture. Thus, we asked the participants whether they gained a clear picture of the employed security design concepts from these models. For this question, we received mixed answers. While 6 out of 7 participants agreed that the security design concepts Zero trust, OAuth 2.0/OpenID Connect, mTLS, password-based, token-based, and protocol-based authentication are represented in the models, it was noted that either further security knowledge is required to understand these techniques or additional models should detail implementation aspects of certain security design concepts, e.g., protocols like OAuth 2.0 or mTLS. Participant P2 disagreed that the employed security design concepts were clarified by the authentication models due to the lack of knowledge about which security design concepts exist in the first place.

**Which modeling elements should be added or removed to enhance this model?**

With this question, we wanted to discover whether other modeling elements would increase the understanding of how certain security design concepts are tackled. Participant P2 mentioned that violating the Zero trust design principle in Figure 6.1b should be avoided, thus anticipating the trust boundaries as in Figure 6.1c instead. Another solution suggested was to indicate such information in additional notes in the diagrams. Participants P3 and P7 both indicated that network segments and details about the employed communication protocols form essential information about the security in the architecture, and thus should be included. Participant P5 would also not remove the arrows from the original architecture model. These arrows are little information that supports the reader in understanding the model, without adding too much complexity to the model.

**Does such a model enhance common understanding and communication about authentication in a software architecture?**

All participants agreed that these models enhance the understanding and communication in a team about this security aspect. The importance of visual representations and their

superiority over textual models was noted by several participants. Participant P4 noted that a premise for understanding these models is foundational knowledge of certain security concepts. Participant P6 mentioned that another requirement for the effective use of these models is that the team agrees to a common set of modeling notations and conventions. Participant P2 highlighted the importance of the models' property of being self-explanatory. When additional security experts are required to further describe and explain these models, they fail in their purpose.

### 6.2.3. Modeling security views

In the final part of the interview, the participants were asked to provide feedback about the general modeling approach with security views. They were asked to identify weaknesses and further directions for research.

#### **How would you assess the distinction of security modeling into the set of security views established in this thesis, i.e., authentication, authorization, secure communication, secure storage, monitoring, secure build & deployment?**

This question targets the distinction of the established security views in this thesis. Concretely, the participants were asked whether this set of security views represents a complete overview of required security aspects needed for software architecture and whether some of these views are too dependent to distinguish them. Participants P3, P4, and P5 said that authentication and authorization share too many concerns and responsibilities to treat them separately. In particular, authentication does not make much sense without proper restrictions to certain data and services, and authorization requires preceding authentication in most cases. Participant P7 further argued that authentication and secure communication should be merged since in order to perform certain authentication methods, e.g., password-based authentication, a secure communication channel needs to be established beforehand. Finally, participant P1 suggested splitting the secure build & deployment view into more specific views, e.g. with an additional security testing view. Otherwise, this view has too many responsibilities since it covers a large set of concerns.

#### **Which views of security need to be considered in a software architecture?**

We asked the participants to provide additional security views that they think should be considered as well, targeting the completeness of our established set of security views. Participant P2 suggested including an additional data security view focusing on data processing and further hardening measures outside of the architecture. Participant P4 would add a developer view focusing on access rights and capabilities of developers and development environments, as the security views of this thesis solely focus on the software product. Participant P3 suggested several security views. First, an input validation view should be considered additionally and separately from the monitoring view. A secret management view needs to depict how secrets are introduced and managed through their

whole lifecycle in the application. A backup strategy view should indicate how backups are created how as well as measures and processes on data losses. Lastly, company policies and how they are enforced in the software architecture should be illustrated in a policy view, which was also suggested by participant P7. Participant P7 also identified an incident-response view, i.e., how certain security incidents like attacks are responded to. Participant P1 did not mention general views but suggested maintaining more specific views for certain areas of the application, e.g., different views for frontend, backend, and database security. The reasoning for this was that each of these areas has different security needs and corresponding security measures to implement. Additional abstraction layers should be incorporated in different views.

#### **Which concerns would need to be addressed in these security views?**

We asked the participants to suggest concrete security concerns that would need to be addressed in their provided security views from the preceding question. The data security view should address how and where data is processed, whether the implemented measures conform to data protection and privacy laws, who has accessed which information, and which data confidentiality levels apply, as noted by participant P2. The secret management view needs to illustrate how secrets are protected and how applications can securely access them. The input validation view needs to show how the system alerts on events and how anomalies are detected. The policy view should indicate which policies exist and which are relevant to the architecture. Lastly, the developer view needs to address access rights and capabilities of developers to certain development infrastructure elements, e.g., a Jenkins<sup>7</sup> pipeline.

#### **Which modeling notations and languages would you suggest for these views?**

With this question, we want to identify whether certain modeling languages already exist that can be used for modeling these security views. Apart from UML [OMG17], no further modeling languages or notations were suggested. The advantage of UML is that it is a widely known and commonly accepted modeling standard used by many software developers. Complex languages with completely new syntax would prevent the adoption of these notations, as noted by participants P1, P4, and P6.

#### **To what extent would special-purpose modeling notations and languages for the respective views, as presented in the case study, be helpful in answering the concerns of this view?**

Nevertheless, all participants agreed that special-purpose modeling languages and notations would be helpful for these views to better address their concerns. The primary requirements that these languages need to fulfill are simplicity, unique semantics of modeling elements to avoid ambiguities, and similarity to UML syntax for better recognition

---

<sup>7</sup><https://www.jenkins.io/>

and adoption. Development teams also need to cultivate security modeling in their processes and use a commonly agreed modeling notation. Participants P1, P2, and P5 also indicated that instead of having separate modeling languages for each view, a single notation for all views with sharable modeling elements is preferred as it would lower the learning curve of understanding and effectively employing such a modeling language.

### **Can we study the influence of established security design concepts using security views and corresponding models?**

Assessing or even quantifying the influence of security design concepts, e.g., on certain security goals, seems to be difficult. Participant P1 mentioned that it might be possible to track and compare different projects which employ different security design concepts and models by the number of found security bugs and flaws. Participant P2 argued that the influence of such security design concepts comes down to the extent they address and fulfill their security requirements. Since such an approach is heavily process-oriented and needs to be adopted and lived by the team, it is still a difficult endeavor to quantify security with our methods.

### **Do such views and models describe security from a software engineering perspective appropriately?**

We aim to create a new perspective on security in contrast to the existing offensive perspective. All participants agreed that these security views and corresponding models capture the software engineering aspect of security appropriately. Participant P3 mentioned that further models that detail certain information are still necessary though. Participant P7 further argued that security views like secure build & deployment can not be modeled with existing threat modeling approaches and tools, but our approach enables this.

### **Overall, how would you assess the worth of our approach for modeling security in software architectures and microservice architectures specifically?**

With our final question, we want to explore whether our modeling approach is fruitful and gather issues that need to be tackled in future research. All participants recognized and assessed our approach as valuable, but also mentioned premises that need to hold to be successful. Participants P2 and P4 highlighted the importance of a team adopting standard and formalized modeling notations and languages. This approach enforces the team to think more about security problems and explicitly model those and their solutions. Also, treating security concerns as a checklist that the architecture model needs to address serves as a discussion foundation, as noted by participant P7. Furthermore, participants P5 and P6 highlighted the advantage of visual representations over textual models. However, the true value of this approach is revealed when applied to a concrete project, as indicated by participants P3 and P4.

## 7. Discussion

Yesterday is history, tomorrow is  
a mystery, but today is a gift.  
That's why it is called the  
present.

---

MASTER OOGWAY

### Contents

---

7.1. Research question findings . . . . .	79
7.2. Evaluation discussion . . . . .	81
7.3. Threats to validity . . . . .	82
7.3.1. Internal validity . . . . .	83
7.3.2. External validity . . . . .	84

---

In the following, we discuss the methods and results of our work. First, we elaborate on the impact of the results on the research questions posed at the beginning, and to what extent our results provide answers to them. Through the evaluation, we received valuable feedback about the presented modeling approach. We assess the results of the evaluation and present the weaknesses and proposed improvements of the security model and how to address them. Finally, we discuss the threats to the validity of our research methods to clarify the limitations of our work.

### 7.1. Research question findings

Security is an essential aspect of software systems and must therefore be carefully taken into account in the modeling and creation of the software architecture. However, modeling approaches and methods lack that capture the software engineering perspective. Our central research question addresses this gap.

**(RQ0)** *How can we model the security of software architectures from a software engineering perspective?*

To answer this question, we pose two additional, more specific research questions. First, we aim to clarify *what* exists in terms of security design concepts to be modeled in software architectures. We restrict ourselves to the MSA style to limit the possible solution space. Thus, a description and classification of the security design concepts for MSAs are necessary. Only through a concrete and systematic, comparable description

of security design concepts we claim that it is possible to assess the security of such an architecture.

**(RQ1)** *How can we describe and classify security-related design concepts for software applications in microservice architectures?*

To answer **(RQ1)**, we conduct a Systematic Literature Review following the approach of Kitchenham et al. [KC07] (cf. Chapter 4). Based on our search and selection strategy, we select 52 publications from the 538 publications initially received, from which we extract a total of 77 security design concepts. We classify this set of security design concepts based on their category and created specific catalogs. Furthermore, we create a mapping that maps the security design concepts to the specific security views to which they can be applied. With these catalogs and the mapping to security views, we propose a systematic approach to describe and classify future security design concepts. However, we do not claim completeness or correctness of the catalogs, their schemes and each catalog entry. Instead, our proposal consists of a first draft of such catalogs and shows the effectiveness of our method. The catalog schemes are primarily based on the author's interpretations of the reviewed publications and our claims and experiences. An in-depth evaluation of the catalogs and their schemes is pending and planned for future studies.

The application of these security design concepts further requires a methodology for modeling security in software architectures. Therefore, our second concrete research question addresses *how* the modeling of security is enabled, focusing on the modeling building blocks that are required to capture the important security aspects in a software architecture modeling.

**(RQ2)** *Which modeling building blocks can be used to describe and define the security of software architectures?*

To answer **(RQ2)**, we propose a security metamodel on basis of the ISO/IEC/IEEE 42010 standard [ISO11b] (cf. Chapter 5). This metamodel partitions the modeling of security into different views, where each view models a separate aspect of the security of an architecture. Furthermore, the metamodel enables us to differentiate between *defensive models* depicting security from a software engineering perspective and *threat models* depicting the attacker's perspective. We define the concrete modeling building blocks around a security view and put them in relation to each other. Furthermore, we identify and describe a total of six different security views and their security concerns through literature research and based on our claims and experiences. Lastly, we present a new definition of security in the context of software engineering to capture the essential concepts concisely, and to set the boundaries for what security in this context entails. The security metamodel, the six concrete security views, and the security definition enable software engineers and architects to concretely model the cataloged security design concepts and thus to more precisely examine the effect of their design decisions. Again, we do not claim completeness or correctness of the metamodel, the set of security views and their concerns. Our metamodel proposal sets a foundation of terms and definitions to be refined and evaluated in-depth in further studies.

In the following, we discuss to what extent these results impact our central research question (**RQ0**). With the security design concept catalogs, we create a basic building block for modeling security. The concepts and utility of these catalogs are similar to the attack libraries we presented in Section 3.2.3. They provide detailed information and a schema for classifying and comparing different techniques. These catalogs can thus be used as basic tools for software architects and engineers.

Through the security metamodel, we set the foundation for differentiating security from both the attacker’s perspective and the software engineering perspective. The metamodel captures the key impacts that are relevant for influencing security QA. The distinction between these two perspectives is based on their objectives and the results they achieve. While *threat modeling* identifies the threats and their severity and risk of a software application, *defensive modeling* and the corresponding security views describe the software engineering constructs employed in an architecture, i.e., the design patterns, protocols, components, and design principles. The subdivision into the different security views enables us to differentiate security from different perspectives and focus on certain aspects at a time. The established security concerns of this thesis bridge the gap between the security QA and the techniques that improve it.

Nevertheless, we recognize a gap in this connection. Specifically, there is a lack of modeling languages and notations that we can use to represent the security design concepts and how they address security concerns. In the case study of our evaluation, we made an immature attempt at this, primarily to explain the concepts of the security metamodel to the participants. Although we have shown how to model security from a software engineering perspective, there is still a lack of tools to implement this methodology in practice.

## 7.2. Evaluation discussion

We now discuss the feedback we received in the evaluation interviews. In doing so, we address and analyze the answers from each of the three parts of the interview.

In the last question of the first part of the interview, we roughly analyzed to what extent our security metamodel is complete and which concepts are not yet included in it. Overall, the metamodel covers the most important concepts of security from a software engineering perspective. As also mentioned by one participant, we see threat modeling as an essential part of modeling security in software architectures. Nevertheless, we also see some points of improvement. Specifically, our model does not yet cover the different development phases, as explained by participant P1. The focus of the security views in this thesis is on product security, yet development views and the associated processes are a fundamental part of the architecture. These aspects need to be considered in future extensions of the model.

In the second part of the interview, the authentication view was presented and discussed as a case study. We received positive feedback regarding the authentication concerns and the associated models. The set of authentication concerns was overall accepted by the participants. Some additional concerns were mentioned as additions to our

list. The models also addressed the established authentication concerns in a meaningful way. Weaknesses of the models were partial ambiguity and the resulting confusion, especially regarding trust boundaries in Figure 6.1b. Other models that went deeper into details about protocols and infrastructure and network landscapes were also missing. We agree with these criticisms, but for the scope of the evaluation, more detailed models were not possible due to the limited time available during the interview. However, communication and understanding of such models and this approach is encouraged and provides clarity on security measures in an architecture.

In the last part of the interview, we evaluated the subdivision of security views and the value of this modeling approach. All participants considered the set of security views to be useful. Dependencies between different security views were identified and it was suggested that these views should not be separated. We contend that this separation is nonetheless essential to preserve the separation of concerns. However, we also recognize that a mechanism is needed in the metamodel to account for these dependencies between security views and resolve them in a meaningful way. 5 out of 7 participants also provided further guidance on additional security views that can be explored in more detail in future work. So far, however, concrete security concerns for these new views are missing. Therefore, these would also need to be explored in future work as well.

An essential ingredient for the success of this modeling approach is the establishment of appropriate modeling languages and notations. Through the evaluation, we did not gain insight into existing modeling languages that we can use to model these security views. Here we see a gap that needs to be addressed in future work. Requirements for these modeling languages are simplicity and uniqueness of the resulting models, and preservation of similarity to UML [OMG17]. New modeling languages must be easy to learn and understand by developers and architects. Finally, it is desired that a single modeling language is developed with the capability to model all security views. Modeling elements and dependencies must be representable consistently across views and the complexity of the language must be minimized.

Finally, all participants rated the modeling approach with security views as very useful. It was confirmed that no comparable techniques exist to date to describe security from a constructive software engineering perspective. The adoption of such an approach would, according to the participants, lead to a deeper involvement with security in the team and the project and a better understanding of the security of the architecture. Due to the broad population of our interview participants, i.e., from research and industry, with IT security and software development backgrounds, and different levels of experience, we have thus received confirmation of the value of our work from various disciplines.

### 7.3. Threats to validity

In the following, we discuss the validity threats of the methods and results of this thesis. We rely on the notion of *internal validity* and *external validity* based on definitions proposed by Wohlin et al. [Woh+12] and Feldt et al. [FM10]. Internal validity questions the trustworthiness of the study conducted and whether the observations can establish a



causal relationship between the treatment and the effect within the context of the study. External validity ensures that the results of the conducted study are generalizable, i.e., whether they can be transferred to other contexts.

### 7.3.1. Internal validity

In a study that collects existing knowledge from academic literature to observe a phenomenon and transfer it into a model, there is a risk of selection bias, i.e., selecting only those studies that support the hypothesis [KC07]. To minimize this bias, we adopted the SLR approach for collecting and selecting research. By doing so, we evaluated and selected all possible research we received within our search strategy according to our selection strategy. Nevertheless, our research design and results have some limitations. Due to time and capacity constraints, the generated search strategy obtains only a limited set of publications. The search strategy explicitly targets specific categories of security design concepts. This leads to a fixed set of possible design concepts that we could consider in the context of this SLR. Other categories may therefore have been excluded. This limitation further leads to the fact that the security model may not fully represent reality, e.g., since other security views or concepts and relations could not be considered in the model. This is a clear limitation of the internal validity of our conducted study. However, the completeness of our catalog and the security model is not the goal of this work, but to explore and demonstrate whether it is possible to gather and structure security design knowledge using this methodology.

Another limitation of our qualitative research is that the selection of studies and data extracted are biased to a certain extent based on the author's interpretations. A neutral and objective evaluation of the studies is difficult to achieve. Therefore, we cannot completely exclude bias based on the researchers' interpretations. To address this issue, weekly discussions were held between the author and the supervisor of this thesis. These discussions aimed to clarify any ambiguities and minimize interpretive bias.

Another problem of our research design is the exclusive selection of peer-reviewed publications, i.e. academic articles published in proceedings or journals. The solutions presented in these articles are often experimental and not bulletproof concepts. However, the industry often relies on established standards or de facto standards as well as best practices. These are often presented in gray literature, i.e. white papers, blog articles, videos, etc. Considering gray literature can thus better reflect reality, i.e., which concrete design concepts are used in real-world software architectures. To address this issue, we conducted an evaluation of the security model with research and industry experts in secure software development. An in-depth evaluation of the SLR results, i.e., the catalogs of security design concepts, with research and industry experts was not possible due to time constraints. However, we are aware of and plan to conduct an extensive assessment and evaluation that will further consider the catalogs and model.

Regarding the evaluation, specifically the authentication view case study, we see a limitation that potentially biases the evaluation results. In particular, the authentication models were each presented and explained to every participant. We discussed the reasoning behind each security design concept and how they address certain authentication

concerns. Since these models were explained in addition, we can not provide a definite statement about the influence on the understanding of authentication by these models alone due to a potential framing bias, i.e., the decisions and answers of the participants may be influenced by the way the information is presented. Nevertheless, we argue that due to the topic's abstractness and the limited time we had, this method is the preferred way to present and illustrate the modeling approach and its concepts. Further in-depth studies need to consider that these sorts of biases are prevented by the chosen evaluation method.

### 7.3.2. External validity

External validity ensures that the results of our SLRs and the security model are transferable, i.e., that they can be reused in other contexts [FM10]. We claim that the security design concepts we identify reveal that these design concepts are not exclusively applicable to MSAs. We argue that many of these design concepts, if not all, are also applicable to other architectural styles. Furthermore, the created schemas do not indicate that MSAs benefit exclusively from certain categories of design concepts. Thus, we argue that MSAs do not significantly restrict or withhold security design concepts exclusively that can be applied in such an architecture. The results of our SLR are, therefore, generalizable. However, further investigations, e.g., by exploring and comparing the security design concepts of other architectural styles, are necessary to validate these claims.

Likewise, we claim that the security views we capture in the security metamodel can be found in other architectural styles in the same way and are therefore not exclusive for MSAs. The identified security concerns of each security view address issues that we can also see in other architectural styles. The key difference in the MSA style is that because they are partitioned into many small services, they require a more fine-grained view of each microservice respectively and especially on the communication infrastructure. Although this fine-granular perspective is represented in the security concerns we propose, we argue that these concerns mostly apply to other architectural styles as well. Thus, we claim that the established security concerns are likewise generally valid, however, they are to be weighted in individual cases depending upon software architecture differently in their importance. Again, in-depth investigations tackling these claims are required.

Our methodology can also be utilized to identify security design concepts for other architectural styles. Even though our design concepts are applicable to a large set of architectures, more specific and exclusive design concepts may exist for other architectural styles. In addition, it is possible to conduct research using our methodology for other QAs, e.g., to determine and study design concepts for performance in software architectures.

## 8. Conclusion and future work

Life is very simple, but we insist  
on making it complicated.

---

CONFUCIUS

### Contents

---

8.1. Summary . . . . .	85
8.2. Future work . . . . .	86
8.2.1. Security design concept catalogs . . . . .	86
8.2.2. Security model . . . . .	87

---

Taking a step towards modeling a security view of software architectures, and microservices, in particular, means creating foundational knowledge and understanding as well as a precise notion of what security in this context encompasses that is recognized and accepted by researchers and practitioners alike. With this exploratory work, we aim to establish such a building block that can guide future work. In the following chapter, we briefly summarize the research method and contributions of this thesis. Through the knowledge gained during our work, we have discovered many potential avenues for further development and research. We provide a brief outlook on what directions future work can tackle to extend the contributions and knowledge established in this thesis.

### 8.1. Summary

The primary goal of this thesis is to shed light on the security quality attribute of software architectures. We want to find out which aspects of security have to be considered when modeling software architectures and especially for microservices by establishing a scientific basis for more formal modeling, description of security mechanisms and their assessment. We aim to close the perceived gap of modeling security from a software engineering perspective.

Focusing on the security design concepts that can be employed in a software architecture, we conduct a Systematic Literature Review based on the methodology of Kitchenham et al. [KC07] (cf. Chapter 4). In doing so, we create a search and selection strategy to find certain security design concepts from academia. With this strategy, we selected 52 peer-reviewed publications out of 538 publications initially received from 5 scientific databases. The result of this SLR is a set of 6 different catalogs based on different schemes depending on the category of security design concept, containing 77 security design concepts in total.

Based on these findings and previous notations and definitions (cf. Chapters 2 and 3), we create a security metamodel in Chapter 5 that defines the most important modeling building blocks regarding a *security view* and highlights the differences in modeling between the attacker and software engineering perspectives. To provide concrete direction for modeling software engineering security views, we further introduced six such security views, namely an authentication view, an authorization view, a secure communication view, a secure storage view, a secure build & deployment view, and a monitoring view. For each of these views, we highlighted what these views need to model and what specific security concerns, i.e., questions and requirements, they need to address.

We conduct an evaluation to assess the security metamodel and the modeling approach with security views (cf. Chapter 6). As the evaluation method, we perform semi-structured interviews with research and industry experts. From a set of questions and a case study, we demonstrate the presented modeling approach to the participants and determine to what extent this approach is suitable for modeling security and which weaknesses need to be addressed in future work. Finally, in Chapter 7, we discuss the extent to which we answered the research questions of this thesis and the impact of the results on those questions as well as the limitations of our work.

## 8.2. Future work

The applied research method and the results of this thesis provide a starting point for future work toward systematic modeling of security in software architectures. In the following sections, we propose several research directions that can be tackled in future studies, regarding the security design catalogs and the security model, respectively.

### 8.2.1. Security design concept catalogs

Regarding the results of our conducted SLR, i.e., the security design concept catalogs, we propose the following further research directions. Due to time and capacity constraints, we were not able to perform an in-depth systematic review of all available research on this topic. Further refinement of these catalogs can be achieved by performing additional SLRs similar to our proposed method. In particular, there are two different approaches to how this can be tackled in further research. First, SLRs that follow a more general search and selection strategy may work towards completing the set of possible categories of design concepts. Our search strategy limits us to a defined set of design concepts, thus leading to a possible incomplete set. Second, more fine-granular SLRs can investigate specific design concepts, either by category or by security view. For instance, a future SLR might explore specific security tactics and align its search strategy according to this goal. Another possibility is to search exclusively for security design concepts applicable to a certain security view, e.g., specific monitoring design concepts. Our work provides the basis for these fine-granular approaches by providing a general method that new studies can refer to.

Another consideration regarding future SLRs is to either include or exclusively search

in gray literature. Academic articles often propose methods and techniques that are often not mature and "battle-tested", and therefore not established in standards or de facto standards in the industry. A gray literature analysis of security design concepts may lead to more mature results. Furthermore, a catalog of industry-wide accepted techniques and solutions is more prone to be adopted by practitioners.

Besides gray literature, another interesting method conducted by Márquez and As-tudillo [MA19] was to investigate open-source projects for applied availability tactics in microservice-based systems. Further analyses like these may reveal interesting but rather undocumented security design concepts. However, deep knowledge and understanding of the reviewed projects are prerequisites for this approach.

Another research direction we strongly advise is to perform an extensive evaluation of the proposed catalogs. We were unable to conduct such an evaluation due to the time and capacity limitations imposed by the thesis' regulations. Discussions and brainstorming sessions with industry and research experts may provide the insights to complete and refine the catalogs and increase their overall quality.

Finally, an important step toward the adoption of these catalogs by practitioners and fellow researchers is the transformation into a digital and navigable format. It is necessary and important that a community and culture is formed that addresses and maintains these catalogs, especially in such a fast-paced area like application security. For this, we refer to similar existing catalogs like the presented attack libraries in Section 3.2.3.

### 8.2.2. Security model

Regarding the security model we proposed, we provide the following directions for further research. Although we conducted an evaluation to assess the value and weaknesses of our modeling approach, we argue that an in-depth evaluation is required to assess the completeness and the correctness of the security metamodel more formally. In particular, we do not claim completeness of our model, and further discussions and research are necessary to identify and fill potential gaps. Also, more investigations toward other security views and corresponding concerns can be performed by additional, special-purpose SLRs. The results of our evaluation depict potential security views that require deeper exploration, e.g., data security views, developer views and compliance and policy views. Other development-centric views, e.g., focusing on securing the CI/CD pipeline, are also conceivable. An in-depth case study where the security model is applied and assessed on a real-world project may provide beneficial insights that can be incorporated into the model.

Another fundamental research direction is the exploration of special-purpose modeling notations, languages, and tools to create models that describe the corresponding security views accordingly. Our case study presented in Section 6.1.1 offers initial approaches to how this can be achieved. In particular, an extensive study shall investigate the needs of the security view under consideration and which modeling elements can effectively depict these aspects. Respective tools to create such models efficiently and effectively must also be developed.

Finally, when such modeling languages and tools are established, we can develop and

perform interesting analysis techniques based on these models. For instance, we can investigate potential security smells in a software architecture based on certain properties, e.g., trust boundaries that include too many microservices. Such established analysis methods provide a first step toward the development of a recommendation system that reviews and assesses employed security design concepts in software architecture.

## A. Results of SLR trial searches

This chapter presents the results of our SLR trial searches, which were performed to generate a search strategy that fits our requirements. In Table A.1, the corresponding results of each search strategy are presented. In the following, we provide an explanation and reasoning for the chosen strategies, and how the results are to be interpreted.

Table A.1.: The results of SLR trial searches. Each row depicts the number of publications retrieved from the selected scientific databases.

Search string suffix	Databases					$\Sigma$
	DB1	DB2	DB3	DB4	DB7	
<i>none</i>	248	209	48	29	419	953
<i>technique</i>	24	16	10	4	40	94
<i>pattern</i>	27	10	8	5	37	87
<i>"best practice"</i>	0	0	0	0	3	3
<i>tool</i>	40	32	10	6	55	143
<i>technology</i>	87	83	10	8	145	333
<i>"design principle"</i>	1	0	0	1	3	5
<i>activity</i>	12	11	2	2	20	47
<i>process</i>	78	70	9	8	80	245
<i>mechanism</i>	48	39	10	5	72	174
<i>service</i>	163	154	31	12	276	636
<i>design</i>	104	78	20	15	141	358
<i>development</i>	62	49	20	10	99	240
<i>tactic</i>	2	0	0	0	3	5
<i>protocol</i>	31	18	3	4	42	98
<i>all</i>	239	202	48	80	394	963
<i>all w/o service</i>	223	177	45	68	345	858
<i>all w/o service, process</i>	211	167	45	60	334	817
<i>all w/o service, process, technology, design</i>	161	126	38	37	254	616
<i>all w/o service, process, technology, design, tool</i>	148	111	37	31	211	538

Each row from Table A.1 represents one applied trial search on the chosen scientific databases, i.e., *Web of Science* (DB1), *ACM Digital Library* (DB2), *IEEEExplore* (DB3), *ScienceDirect* (DB4), and *Scopus* (DB7). For each applied trial search, Table A.1 depicts the number of publications retrieved for the specific search strategy from each scientific database separately, and how many publications were retrieved in total. The specific search strategy of each trial search was determined as follows. For each search strategy, we applied the same search filters that were applied in the final search (cf. Table 4.2). The search string we used for each search strategy was formed by a conjunction of the base search string (cf. Section 4.1.4) with the *search string suffix* from Table A.1. For instance, the applied search string of the **technique** row was the following:

(security OR secure) AND (microservice OR microservices OR  
"micro-service" OR "micro-services" OR "micro service" OR "micro  
services") AND technique

The *none* row indicates that no additional search string suffix was provided, i.e., only the base search string was used. As this resulted in too many publications, we refined our search string by additional design concept categories.

The *all* row indicates that every design concept category was involved in that particular search string:

```
(security OR secure) AND
(microservice OR microservices OR "micro-service" OR "micro-services" OR
"micro service" OR "micro services") AND
(technique OR pattern OR "best practice" OR tool OR technology OR
"design principle" OR activity OR process OR mechanism OR service OR
protocol OR tactic OR development OR design)
```

This has led to more retrieved publications than the *none* search strategy provided. This is because we were required to perform a separate search on the *ScienceDirect* (**DB4**) database, as the search engine only allows up to 8 operators (AND/OR). Therefore, the results in the **DB4** column may contain duplicate publications.

Nonetheless, we continued to reduce the corpus of publications to a manageable amount. In particular, we tried to generate a search strategy that generates  $\sim 500$  publications. So we continued as follows. We discussed and chose particular terms of design concept categories, that either yielded too many results or were not useful for further consideration. Thus, from the *all* search strategy, we removed these terms iteratively. This is indicated by the *all w/o* <LIST OF TERMS> rows, i.e., all terms without the particular terms contained in that list. The chosen final search strategy is indicated in the last row, i.e., *all w/o service, process, technology, design, tool*.



## B. Catalog of security design concepts

### B.1. Catalog of security design principles

<b>Name</b>	Principle of least privilege
<b>Security view</b>	Authorization
<b>Summary</b>	Every entity should have the minimum required privileges to complete its tasks.
<b>Intent</b>	Prevent unprivileged entities to may have access data or services.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Secure-by-default, Deny access by default, Zero Trust Networking
<b>Publications</b>	[NC19a]

<b>Name</b>	Deny access by default
<b>Security view</b>	Authorization
<b>Summary</b>	All access to data and services should be denied by default.
<b>Intent</b>	Prevent unprivileged entities to may have access data or services.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Secure-by-default, Zero Trust Networking, Principle of least privilege
<b>Publications</b>	[NC19a]

<b>Name</b>	Secure-by-communication
<b>Security view</b>	
<b>Summary</b>	Developers should respond to security incidents and vulnerabilities, and communicate information about security updates.
<b>Intent</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	
<b>Publications</b>	[NC19a]

<b>Name</b>	Secure-by-design
<b>Security view</b>	
<b>Summary</b>	Consider and integrate security throughout the whole software development life-cycle.
<b>Intent</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Single Responsibility Principle, Maximize API Security
<b>Publications</b>	[NC19a]

## B. Catalog of security design concepts

---

<b>Name</b>	Secure-by-default
<b>Security view</b>	Secure Build & Deployment
<b>Summary</b>	Default configurations for microservices and their composition should be the most secure setting possible.
<b>Intent</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Defense in depth, Deny access by default, Principle of least privilege
<b>Publications</b>	[NC19a]

<b>Name</b>	Secure-by-deployment
<b>Security view</b>	Secure Build & Deployment
<b>Summary</b>	Protect the complete deployment process of microservices and their composition.
<b>Intent</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	
<b>Publications</b>	[NC19a]

<b>Name</b>	Maximize API Security
<b>Security view</b>	Monitoring
<b>Summary</b>	Exposed network interfaces must be minimal and must have strong input validation.
<b>Intent</b>	Entry points to the microservice system are more likely to be attacked and thus need stricter security controls.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Asymmetric node strength, Secure-by-design
<b>Publications</b>	[OY17]

<b>Name</b>	Single Responsibility Principle
<b>Security view</b>	
<b>Summary</b>	Each microservice should only implement one business functionality.
<b>Intent</b>	Reduce LOC per microservice, and thus reduce the number of exploitable bugs.
<b>QA enhanced</b>	complexity, maintainability
<b>QA worsened</b>	
<b>Related Principle</b>	Secure-by-design
<b>Publications</b>	[YB18b]

<b>Name</b>	Avoid unnecessary node relationships
<b>Security view</b>	
<b>Summary</b>	Minimize the number of communication edges in a microservice network.
<b>Intent</b>	If some microservice can reach another microservice through an intermediate microservice, there should be no edge between these in order to reduce the attack surface.
<b>QA enhanced</b>	
<b>QA worsened</b>	complexity, performance
<b>Related Principle</b>	
<b>Publications</b>	[OY17]

## B.1. Catalog of security design principles

<b>Name</b>	Asymmetric node strength
<b>Security view</b>	Monitoring, Secure Communication
<b>Summary</b>	Place more secure nodes at critical network segments, e.g. entry points, to protect nodes guarding the more valuable assets.
<b>Intent</b>	Optimize robustness against low-level exploitation.
<b>QA enhanced</b>	robustness
<b>QA worsened</b>	
<b>Related Principle</b>	Maximize API Security, Defense in depth
<b>Publications</b>	[OY17]

<b>Name</b>	Zero Trust Networking
<b>Security view</b>	Authentication, Authorization, Secure Communication
<b>Summary</b>	All actions must be verified and all data transfers should be encrypted. There is no implicit trust between services, and trust must be evaluated continuously.
<b>Intent</b>	Always assume that the microservice network is already compromised.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Deny access by default, Principle of least privilege
<b>Publications</b>	[Mel21]

<b>Name</b>	Fail fast
<b>Security view</b>	Monitoring
<b>Summary</b>	Microservice systems should tolerate partial failures and limit their propagation.
<b>Intent</b>	Limit the effect of Denial of Service attacks and other exploitable failures.
<b>QA enhanced</b>	availability, resiliency
<b>QA worsened</b>	
<b>Related Principle</b>	
<b>Publications</b>	[YB18b]

<b>Name</b>	Defense in depth
<b>Security view</b>	Authentication, Authorization, Monitoring, Secure Communication
<b>Summary</b>	Apply distinct, perhaps redundant security controls at multiple layers throughout the microservice system.
<b>Intent</b>	In case of an attack, if one security control fails due to an exploited vulnerability, make use of other security controls to weaken or prevent the overall attack.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	Secure-by-default, Asymmetric node strength
<b>Publications</b>	[MCF21b; NC19a]

<b>Name</b>	Security Through Diversity
<b>Security view</b>	
<b>Summary</b>	Add diversity to the microservice system by employing a polyglot architecture, thus making use of different technologies.
<b>Intent</b>	Make (multi-step) attacks less likely to succeed and less effective due to the different exploits required.
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Related Principle</b>	
<b>Publications</b>	[OY17; YB18b]

## B.2. Catalog of security activities

<b>Name</b>	Policy conflict resolution
<b>Security view</b>	Authorization
<b>Goal</b>	Security admins construct access control policies for each microservice, respectively. The composition of microservices, the lack of analysis methods and tools for policy violations, can lead to inconsistencies and incompatibilities. Different (automatic) resolution methods exist, e.g., autonomous, random, traversal sub-base methods, etc.
<b>Stakeholders</b>	Operator
<b>SDLC Phase</b>	Design
<b>Automaticity</b>	Automated
<b>Assets</b>	Security policies
<b>Steps</b>	
<b>Publications</b>	[Liu+20]

<b>Name</b>	API Key Distribution
<b>Security view</b>	Secure Storage
<b>Goal</b>	Each microservice requires API keys and additional information like roles, which it gets from a permission database. The key represents a service or user who is authorized to use it. Each microservice should be able to fulfill a request without connecting to another service.
<b>Stakeholders</b>	
<b>SDLC Phase</b>	Operations
<b>Automaticity</b>	Automated
<b>Assets</b>	API Key, Microservices, Permission database
<b>Steps</b>	1. Initialization phase: the API key is distributed to the instances of a microservice on creation (registration phase) 2. Production phase: the microservice is available to receive requests and authenticate them using the API key. The incoming request contains the authentication data (API key) of the user/service from which the request comes. The service is able to authenticate the request without any management services (cf. auth service or API gateway)
<b>Publications</b>	[Müs+17]

<b>Name</b>	Secret Management
<b>Security view</b>	Secure Storage
<b>Goal</b>	Microservices: Microservices need secrets or certificates when they are required to authenticate to other microservices or to third-party applications. CI/CD pipeline: Pipeline steps might need to access external services, e.g., to deploy Docker images. These external services often require credentials to authenticate, and these credentials must be provided in the corresponding pipeline step securely.
<b>Stakeholders</b>	Operator, Software engineer
<b>SDLC Phase</b>	Operations
<b>Automaticity</b>	Manual
<b>Assets</b>	CI/CD pipeline, Certificates, Keys, Secrets
<b>Steps</b>	
<b>Publications</b>	[NC19b; Thr+21]

<b>Name</b>	Static Application Security Testing (SAST)
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Static security testing that checks the source code and design documents to find errors, code flaws, and potentially malicious code when the code is not being executed
<b>Stakeholders</b>	CI/CD pipeline
<b>SDLC Phase</b>	Testing (CI)
<b>Automaticity</b>	Automated
<b>Assets</b>	Source code
<b>Steps</b>	
<b>Publications</b>	[NC19a; NL21]

<b>Name</b>	Dynamic Application Security Testing (DAST)
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Dynamic security testing validates the runtime behavior of security mechanisms in an application when source code is being executed or the application is running
<b>Stakeholders</b>	CI/CD pipeline
<b>SDLC Phase</b>	Testing (CI)
<b>Automaticity</b>	Automated
<b>Assets</b>	Microservices
<b>Steps</b>	
<b>Publications</b>	[NC19a]

<b>Name</b>	Security policy decomposition and verification
<b>Security view</b>	Authorization
<b>Goal</b>	Information flow security is a difficult property to achieve in a decentralized system (MSAs), as global security policies cannot be decomposed in local security policies without the potential introduction of ill-formed rules leading to global information leaks. This approach presents an activity to specify and reason about decomposed security policies (per component/microservices), and verify the composition of these local policies.
<b>Stakeholders</b>	Software architect
<b>SDLC Phase</b>	Design
<b>Automaticity</b>	Manual
<b>Assets</b>	Component architecture, Security policies
<b>Steps</b>	<ol style="list-style-type: none"> <li>1. Specify security policies</li> <li>2. Derive component architecture</li> <li>3. Refine security policies</li> <li>4. Determine coordination protocols</li> <li>5. Determine component behavior</li> <li>6. Verify security policy</li> </ol>
<b>Publications</b>	[GS19]

<b>Name</b>	Dependency scanning
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Due to the high mass of packages and cumulative dependencies that occur when using package managers, dependencies should be scanned for vulnerabilities in an automated way.
<b>Stakeholders</b>	CI/CD pipeline, Operator, Software engineer
<b>SDLC Phase</b>	Testing (CI), Deployment (CD)
<b>Automaticity</b>	Automated
<b>Assets</b>	Container, Source code
<b>Steps</b>	Upon code changes, the CI pipeline pulls the repository changes and applies the usual integration and deployment steps. In addition, a new pipeline step, the Vulnerability Scan, scans dependencies in container images and application BOMs. If the vulnerability scanner identifies any active vulnerabilities, a report is created in a human-readable format and sent to the operator or developers.
<b>Publications</b>	[Thr+21; NL21]

## B. Catalog of security design concepts

---

<b>Name</b>	Continuous Security Assessments
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Continuous security assessments deal with detecting vulnerabilities in production-running microservices and enforcing security policies. This tactic is employed by using the Security Gateway and Security Health Endpoints patterns, which deal with the problem of discoverability in microservices and serve as Security Enforcement Points (SEPs).
<b>Stakeholders</b>	CI/CD pipeline, Operator
<b>SDLC Phase</b>	Testing (CI), Deployment (CD)
<b>Automaticity</b>	Automated
<b>Assets</b>	Container, Security Gateway, Security Health Endpoints, Security policies
<b>Steps</b>	<ol style="list-style-type: none"><li>1. On code changes, the CI server pulls the code repository and container images from a private image registry.</li><li>2. After building the application, it is deployed to a staging environment, where the CAVAS engine applies pre-deployment security tests.</li><li>3. On success, the image is pushed to the production image registry and the application is re-deployed in the production environment.</li><li>4. Further security tests and security policy checks are performed.</li><li>5. Vulnerabilities found in the application image are reported to the administrator.</li></ol>
<b>Publications</b>	[Tor+19; TSM17; Tor+18]

<b>Name</b>	RBAC assessments
<b>Security view</b>	Authorization
<b>Goal</b>	Identify access control violations or inconsistencies in a microservice-based application. In such a system, such inconsistencies can be introduced more easily due to its decentralized nature. Often, RBAC policies are defined per microservice, and not as a whole. Possible inconsistencies or policy violations are: <ol style="list-style-type: none"><li>1. Missing role</li><li>2. Unknown access</li><li>3. Entity access violation</li><li>4. Conflicting hierarchy</li><li>5. Unrelated access violation</li></ol>
<b>Stakeholders</b>	CI/CD pipeline, Software engineer
<b>SDLC Phase</b>	Implementation, Testing (CI)
<b>Automaticity</b>	Automated
<b>Assets</b>	RBAC model, Source code
<b>Steps</b>	<ol style="list-style-type: none"><li>1. Generate a complete view of microservice communication paths by scraping security metadata (REST calls, REST endpoints)</li><li>2. An analysis module takes descriptions of the method-call graphs and a role-hierarchy model and searches for possible policy inconsistencies and violations.</li></ol>
<b>Publications</b>	[Das+21]

<b>Name</b>	Container Security Audits
<b>Security view</b>	Secure Build & Deployment
<b>Goal</b>	Containers and virtual machines should only use verified operating system platforms or container-specific operating systems. The outbound network traffic sent by the container should be monitored and controlled. The configuration of containers and virtual machines should comply with the configuration standards.
<b>Stakeholders</b>	Operator, Software architect, Software engineer
<b>SDLC Phase</b>	Implementation, Testing (CI)
<b>Automaticity</b>	Manual
<b>Assets</b>	Container
<b>Steps</b>	<p>Phase 1: Docker Base Image Inspection</p> <ul style="list-style-type: none"><li>1.1 Check whether the image is official</li><li>1.2 Check trust of the base image</li><li>1.3 Check the content of the image</li><li>1.4 Uninstall unnecessary packages</li><li>1.5 Disable build cache</li></ul> <p>Phase 2: Dockerfile Configuration</p> <ul style="list-style-type: none"><li>2.1 Base image version</li><li>2.2 Run apt-get install and apt-get update as one command</li><li>2.3 Use COPY instead of ADD</li><li>2.4 Healthcheck</li><li>2.5 Secrets not stored in Dockerfile</li></ul> <p>Phase 3: Image Authentication</p> <ul style="list-style-type: none"><li>3.1 Image signing with DCT</li><li>3.2 Registry authentication</li></ul> <p>Phase 4: Image Authorization</p> <ul style="list-style-type: none"><li>4.1 Create a user per container</li><li>4.2 Permissions for user</li></ul>
<b>Publications</b>	[SZS21; NC19b]

### B.3. Catalog of security tactics

<b>Name</b>	Unencrypted Communication
<b>Security view</b>	Secure Communication
<b>Security context</b>	prevent
<b>Stimulus</b>	A communication path between a microservice and a client / another microservice is established.
<b>Threat (Security Stimulus)</b>	An attacker is able to eavesdrop or manipulate messages on this communication path.
<b>Environment</b>	Network communication
<b>Response</b>	Unencrypted protocols such as HTTP are used for communication between microservices and clients.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	complexity
<b>QA worsened</b>	confidentiality, integrity
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

<b>Name</b>	Microservice Sandboxing
<b>Security view</b>	Monitoring
<b>Security context</b>	react to
<b>Stimulus</b>	An IDS detects an intrusion and notifies the SDN infrastructure controller.
<b>Threat (Security Stimulus)</b>	A suspicious microservice instance / container is identified.
<b>Environment</b>	Monitoring
<b>Response</b>	When an intrusion or suspicious container is detected, a clone from a clean state is created, and all benign connections are redirected to that clone. The suspicious container is moved to a sandbox network.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Osm+19]



<b>Name</b>	Decentralized Certification Authorities
<b>Security view</b>	Secure Build & Deployment
<b>Security context</b>	prevent
<b>Stimulus</b>	Deploy new versions of a microservice executable.
<b>Threat (Security Stimulus)</b>	An attacker might exploit the fact that microservice executables are not integrity protected. Thus, if he is able to intervene in the deployment process, malicious versions of a microservice executable may be deployed.
<b>Environment</b>	Deployment
<b>Response</b>	This approach uses x.509v3 certificates to protect the integrity and authenticity of service executables running in IoT devices. In total, three layers of certificates are used: developers sign their executables and metadata and create a certificate. This is then copied by the store (which distributes the service executables), and also signed. The last layer is performed by the Site-local Certification Authority, which again, copies and signs the certificate from the store (after verifying it). It then distributes the verified service to the corresponding IoT nodes.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	authenticity, integrity
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[PD19; PD18]

<b>Name</b>	Distributed tracing
<b>Security view</b>	Monitoring
<b>Security context</b>	detect
<b>Stimulus</b>	Microservice communication patterns have been established in a network, i.e., a set of communicating microservices and what they communicate is known.
<b>Threat (Security Stimulus)</b>	Unusual communication paths, e.g., when an attacker or intruder performs requests in a microservice network, should be detected.
<b>Environment</b>	Microservice communication
<b>Response</b>	A request in a MSA typically involves multiple microservices. This sequence of operations shall be recorded as the distributed trace. A traceId and spanId identify the corresponding incoming request and all microservices that were conducted during this request.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Jac+22; JQL21]

<b>Name</b>	No authentication / Authentication not required
<b>Security view</b>	Authentication
<b>Security context</b>	
<b>Stimulus</b>	A service or client performs a request to an unrestricted microservice. The microservice does not need to authenticate the requestor.
<b>Threat (Security Stimulus)</b>	
<b>Environment</b>	Authentication
<b>Response</b>	
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

## B. Catalog of security design concepts

---

<b>Name</b>	Plaintext-based authentication
<b>Security view</b>	Authentication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service or client performs a request to a restricted microservice. In order to process the request, the microservice needs to determine the identity and permissions of the requestor.
<b>Threat (Security Stimulus)</b>	A (malicious) client wants to have access to a service/data, which it is not allowed to have.
<b>Environment</b>	Authentication
<b>Response</b>	The client authenticates itself to the microservice with its credentials, usually a pair of username and password. The microservice is responsible to validate the credentials.
<b>Depends on</b>	Encrypted Communication
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

<b>Name</b>	API Keys
<b>Security view</b>	Authentication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service or client performs a request to a restricted microservice. In order to process the request, the microservice needs to determine the identity and permissions of the requestor.
<b>Threat (Security Stimulus)</b>	A (malicious) client wants to have access to a service/data, which it is not allowed to have.
<b>Environment</b>	Authentication
<b>Response</b>	The client is equipped with a unique token that it presents to the requested microservice. Based on this token, the microservice determines the identity of the client.
<b>Depends on</b>	Encrypted Communication
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

<b>Name</b>	Protocol-based authentication
<b>Security view</b>	Authentication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service or client performs a request to a restricted microservice. In order to process the request, the microservice needs to determine the identity and permissions of the requestor.
<b>Threat (Security Stimulus)</b>	A (malicious) client wants to have access to a service/data, which it is not allowed to have.
<b>Environment</b>	Authentication
<b>Response</b>	The client and microservice determine the identity (either of the client or both) by performing a communication protocol. The protocol may also be used to establish an encrypted communication tunnel between these parties.
<b>Depends on</b>	
<b>Implies usage</b>	Encrypted Communication
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

<b>Name</b>	Self-preservation
<b>Security view</b>	Monitoring
<b>Security context</b>	react to
<b>Stimulus</b>	An unexpected number of registered service clients fail in their connections and are pending eviction at the same time.
<b>Threat (Security Stimulus)</b>	Multiple services are affected by a (distributed) denial of service attacks.
<b>Environment</b>	Catastrophic network events
<b>Response</b>	Execute an explicit unregister action when service clients are permanently going away. Any service client that fails 3 consecutive heartbeat renewals is considered to have an unclean termination.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Service Registry
<b>Publications</b>	[MA19]

<b>Name</b>	Rate limiting
<b>Security view</b>	Monitoring
<b>Security context</b>	prevent
<b>Stimulus</b>	A service receives an abnormal amount of requests.
<b>Threat (Security Stimulus)</b>	Multiple services are affected by a (distributed) denial of service attacks.
<b>Environment</b>	Network communication
<b>Response</b>	Apply a rate-limiting policy, i.e., each client is only permitted to perform a limited number of requests. This limit can be based on the subscription model of the application.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Che+19]

<b>Name</b>	Input Validation
<b>Security view</b>	Monitoring
<b>Security context</b>	prevent
<b>Stimulus</b>	A service receives a message from another service or client.
<b>Threat (Security Stimulus)</b>	An attacker introduces malicious messages, e.g., messages containing code injection (XSS, ...) content.
<b>Environment</b>	Monitoring
<b>Response</b>	Validate all messages received, in particular messages received from clients. Apply filters to validate the message in multiple steps.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Che+19; NC19b]

## B. Catalog of security design concepts

---

<b>Name</b>	Limited Egress
<b>Security view</b>	Secure Build & Deployment
<b>Security context</b>	prevent
<b>Stimulus</b>	A microservice opens a connection to another service outside the microservice cluster (egress).
<b>Threat (Security Stimulus)</b>	An attacker intends to open a connection to a server or self-hosted infrastructure, e.g., a reverse shell to gain access to the microservice system.
<b>Environment</b>	Network communication
<b>Response</b>	Minimize the number of connections each microservice can create outside the microservice system.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Mel21]

<b>Name</b>	Encrypted Communication
<b>Security view</b>	Secure Communication
<b>Security context</b>	prevent
<b>Stimulus</b>	A communication path between a microservice and a client / another microservice is established.
<b>Threat (Security Stimulus)</b>	An attacker is able to eavesdrop or manipulate messages on this communication path.
<b>Environment</b>	Network communication
<b>Response</b>	Ensure that the communication path is encrypted and integrity protected. Either protect the messages exchanged by encrypting them on the application layer, or use secure communication protocols like SSL/TLS.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22; NC19b]

<b>Name</b>	Set timeouts
<b>Security view</b>	Monitoring
<b>Security context</b>	react to
<b>Stimulus</b>	Timing-out calls that crash services.
<b>Threat (Security Stimulus)</b>	An attacker actively stalls service calls so they become unavailable to other clients or services, e.g., by a slow-loris attack.
<b>Environment</b>	Microservice communication
<b>Response</b>	Set timing-out calls that take longer than thresholds. If a call reaches the threshold, the service call will be canceled.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Circuit Breaker
<b>Publications</b>	[MA19]

<b>Name</b>	Preventing single dependency
<b>Security view</b>	Secure Communication
<b>Security context</b>	prevent
<b>Stimulus</b>	Latency in transmitting or processing data in a network.
<b>Threat (Security Stimulus)</b>	An attacker actively stalls service calls so they become unavailable to other clients or services, e.g., by a slow-loris attack.
<b>Environment</b>	Network communication
<b>Response</b>	Prevent single dependencies by wrapping all calls to external systems (or dependencies) in an object which typically executes within a separate thread.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Circuit Breaker
<b>Publications</b>	[MA19]

<b>Name</b>	Providing fallbacks
<b>Security view</b>	Secure Communication
<b>Security context</b>	react to
<b>Stimulus</b>	A client service repeatedly suffers from dependency faults.
<b>Threat (Security Stimulus)</b>	An attacker actively stalls service calls so they become unavailable to other clients or services, e.g., by a slow-loris attack.
<b>Environment</b>	Microservice communication
<b>Response</b>	Maintain a small thread pool for each dependency. If it becomes full, requests destined for that dependency will be immediately rejected instead of queued up.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Circuit Breaker
<b>Publications</b>	[MA19]

<b>Name</b>	Observing system services
<b>Security view</b>	Monitoring
<b>Security context</b>	detect
<b>Stimulus</b>	An incoming request is handled by a single or set of microservices.
<b>Threat (Security Stimulus)</b>	The request contains malicious content or is an attempt of an attacker to intrude on the microservice system.
<b>Environment</b>	Monitoring
<b>Response</b>	Observe the behavior of a single or a collection of microservices. In particular, observe which actions are performed per microservice, messages exchanged, as well as the communication paths within the microservice system.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	
<b>Publications</b>	[Zdu+22]

## B. Catalog of security design concepts

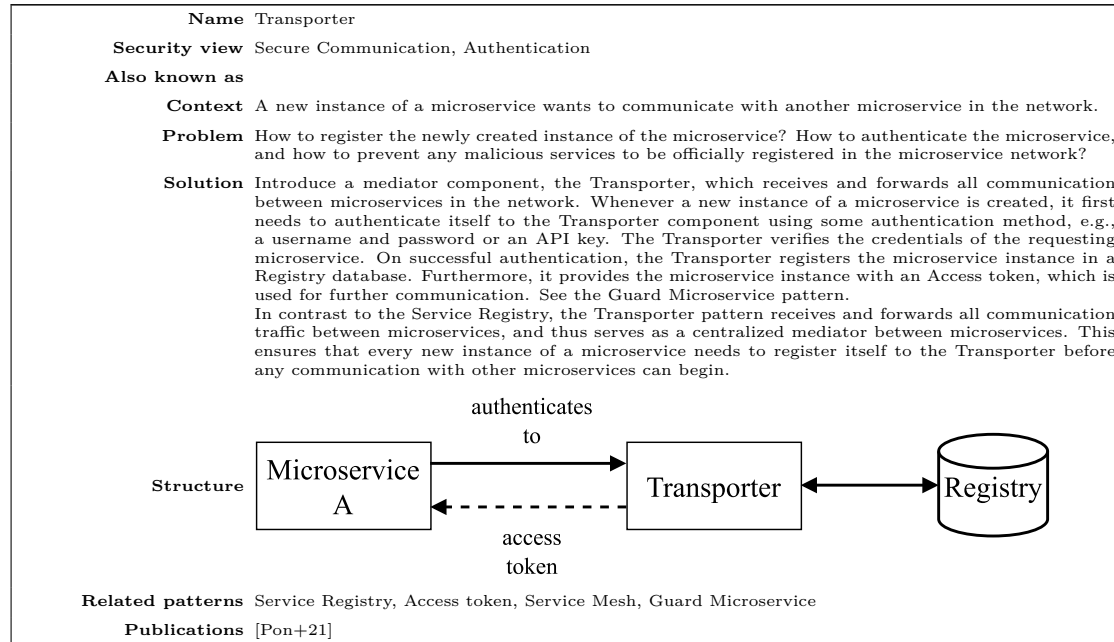
---

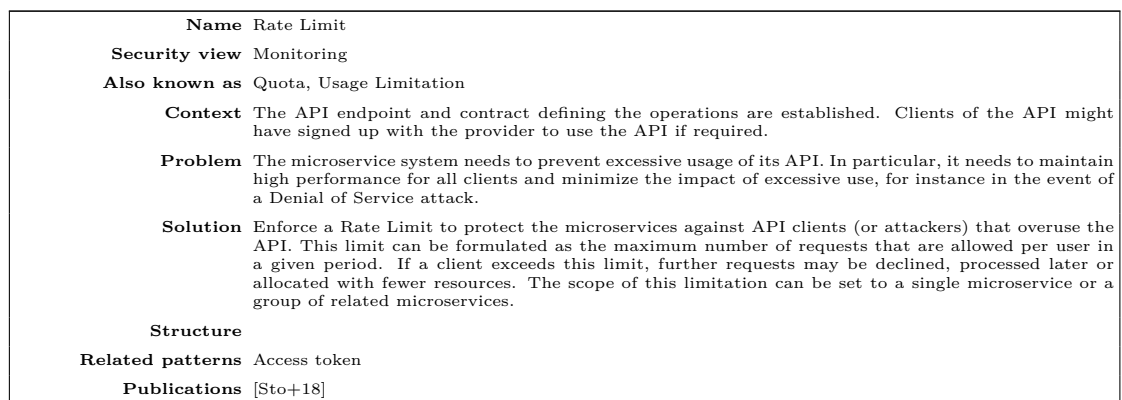
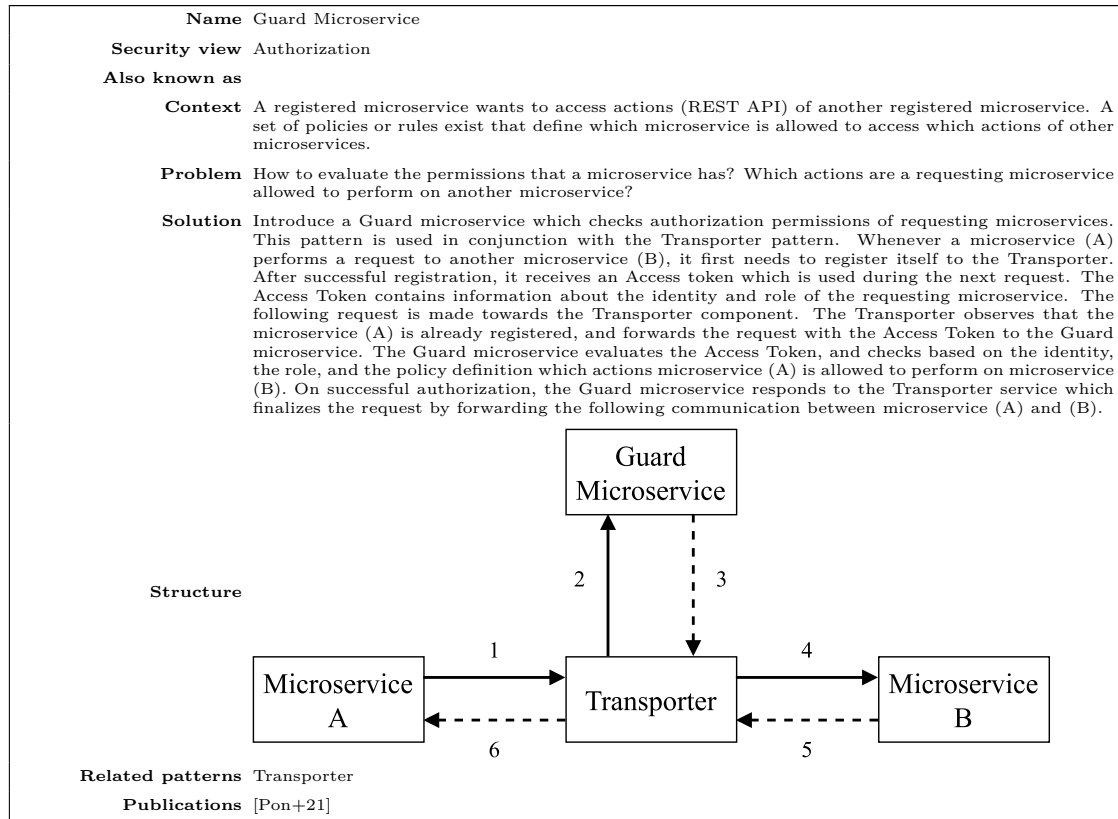
<b>Name</b>	Token-based authentication
<b>Security view</b>	Authentication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service or client performs a request to a restricted microservice. In order to process the request, the microservice needs to determine the identity and permissions of the requestor.
<b>Threat (Security Stimulus)</b>	A (malicious) client wants to have access to a service/data, which it is not allowed to have.
<b>Environment</b>	Authentication
<b>Response</b>	Cryptographic identity tokens are used to authenticate the requestor. The access token contains all relevant authentication and authorization information, as well as a cryptographic signature to verify its integrity.
<b>Depends on</b>	Encrypted Communication
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Access token
<b>Publications</b>	[Zdu+22; PS19; JLE18]

<b>Name</b>	Automated, immutable deployment
<b>Security view</b>	Secure Build & Deployment
<b>Security context</b>	prevent
<b>Stimulus</b>	New changes of a service need to be deployed in the microservice system.
<b>Threat (Security Stimulus)</b>	An attacker introduces malicious changes to the deployed service, e.g., a backdoor, by a supply chain attack.
<b>Environment</b>	Deployment
<b>Response</b>	Make deployments automated and immutable, i.e., instead of applying the changes to the existing service, replace it with a new version of the service. Automate this process in the CI/CD pipeline.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	
<b>QA worsened</b>	
<b>Patterns</b>	Container Manager, Container
<b>Publications</b>	[YB18b]

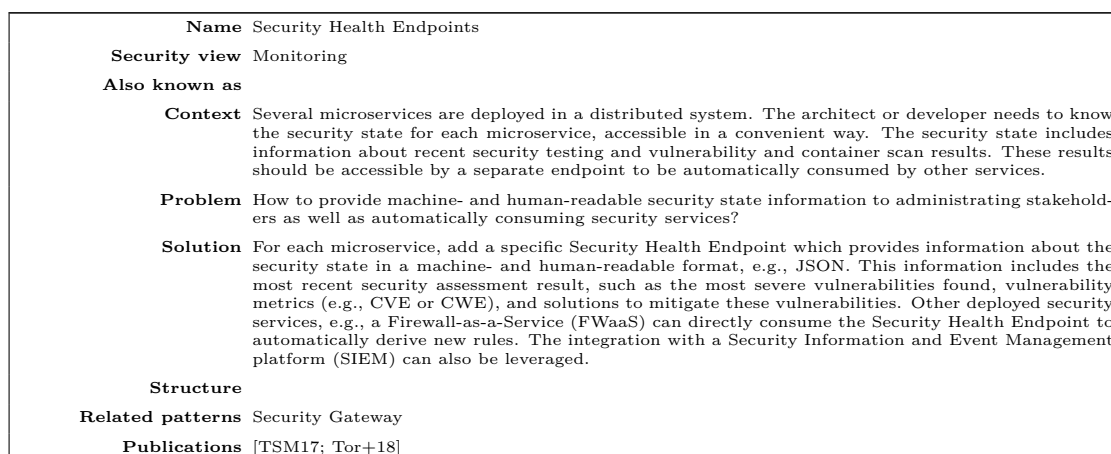
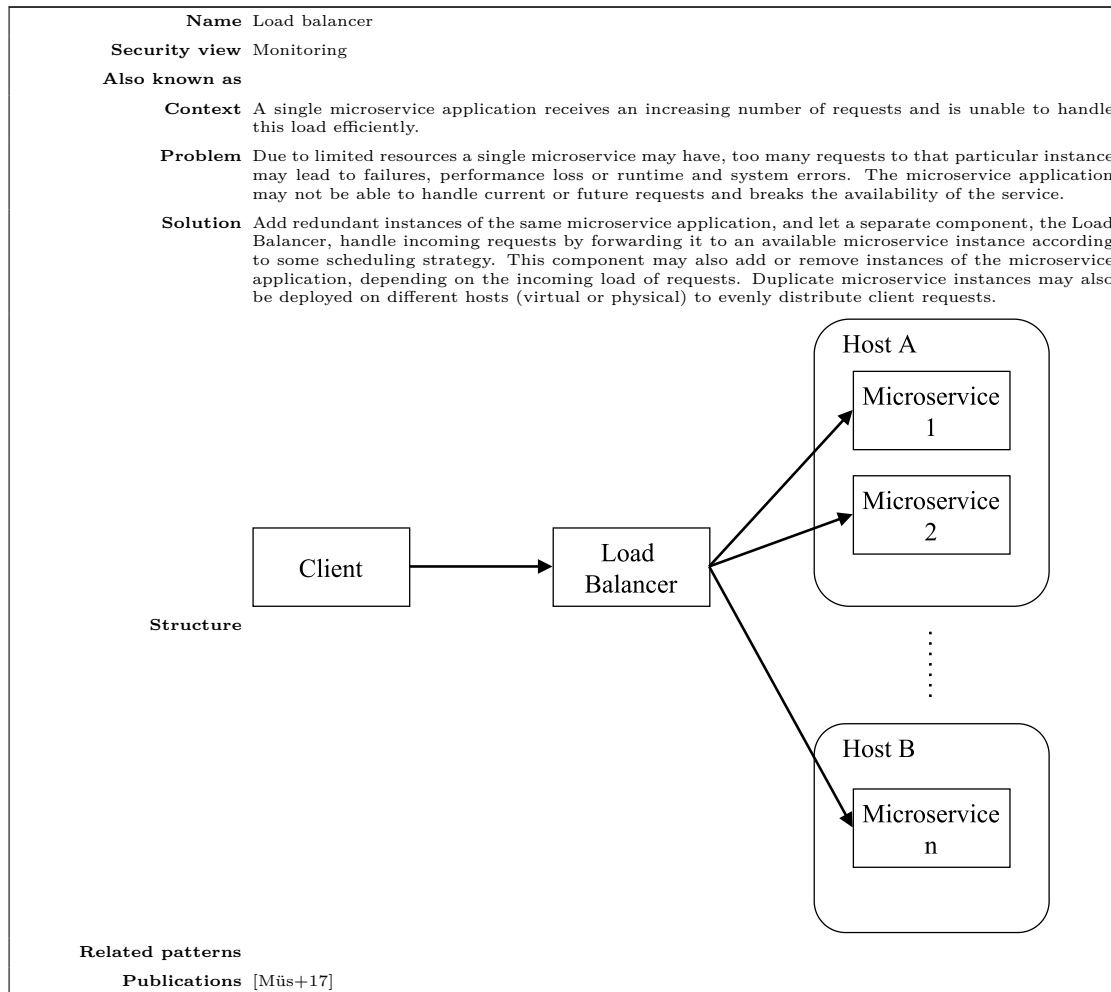
<b>Name</b>	Use asynchronous messaging
<b>Security view</b>	Secure Communication
<b>Security context</b>	prevent
<b>Stimulus</b>	A service fails to respond to a client or service request.
<b>Threat (Security Stimulus)</b>	The requested service is unavailable, e.g., due to a denial of service attack.
<b>Environment</b>	Microservice communication
<b>Response</b>	Apply asynchronous messaging protocols. The client or message sender does not have to wait for a response.
<b>Depends on</b>	
<b>Implies usage</b>	
<b>QA enhanced</b>	availability, resiliency
<b>QA worsened</b>	
<b>Patterns</b>	Asynchronous Messaging
<b>Publications</b>	[MA19]

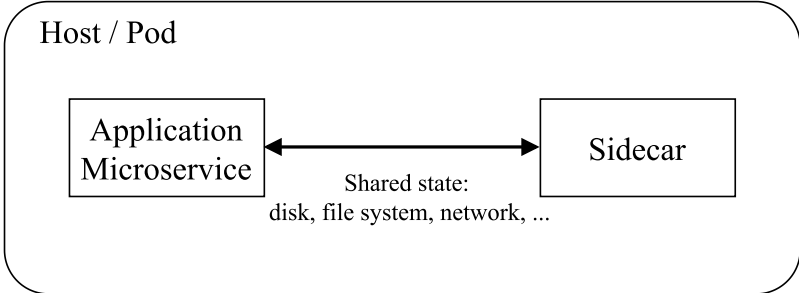
## B.4. Catalog of architectural security patterns

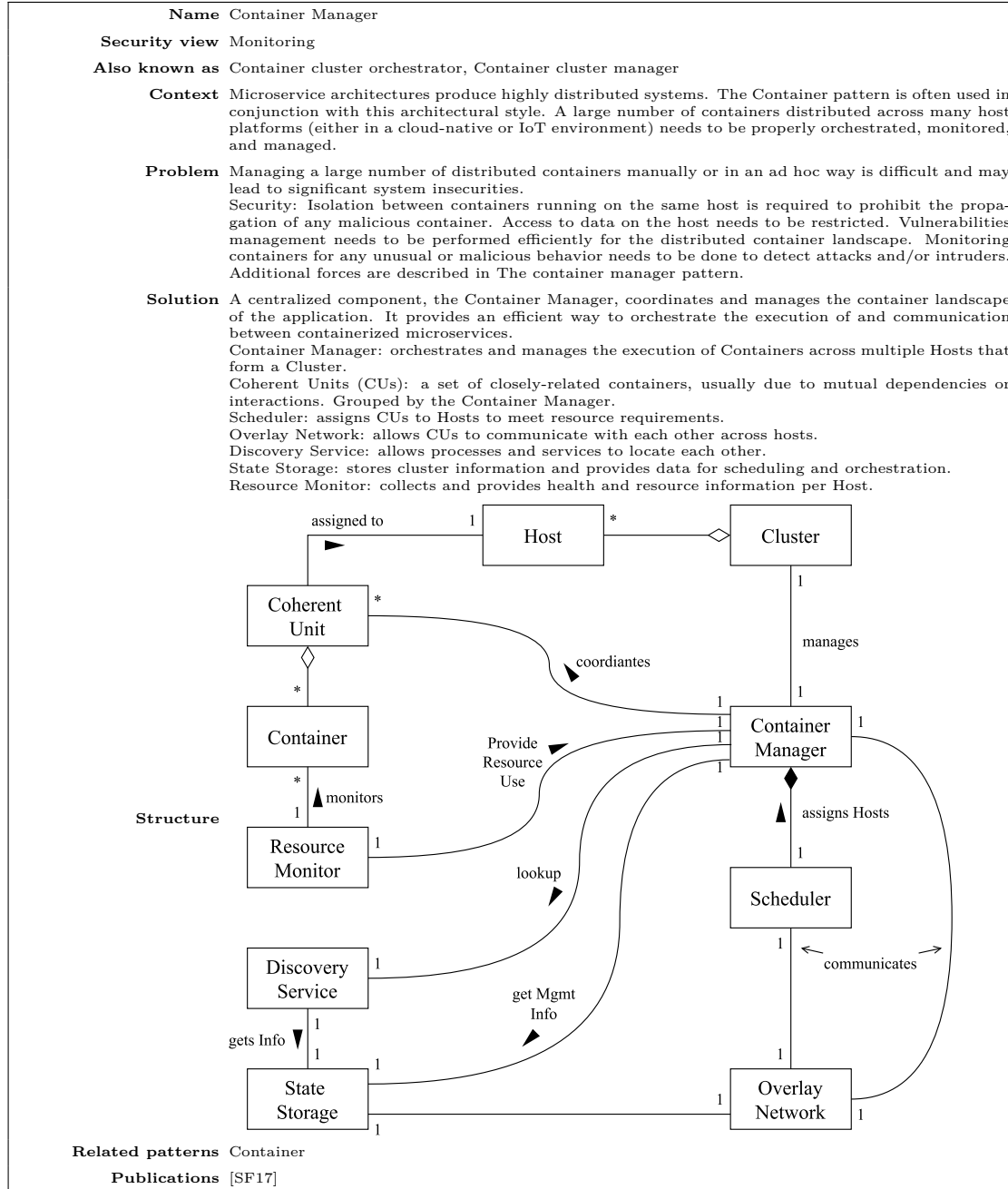


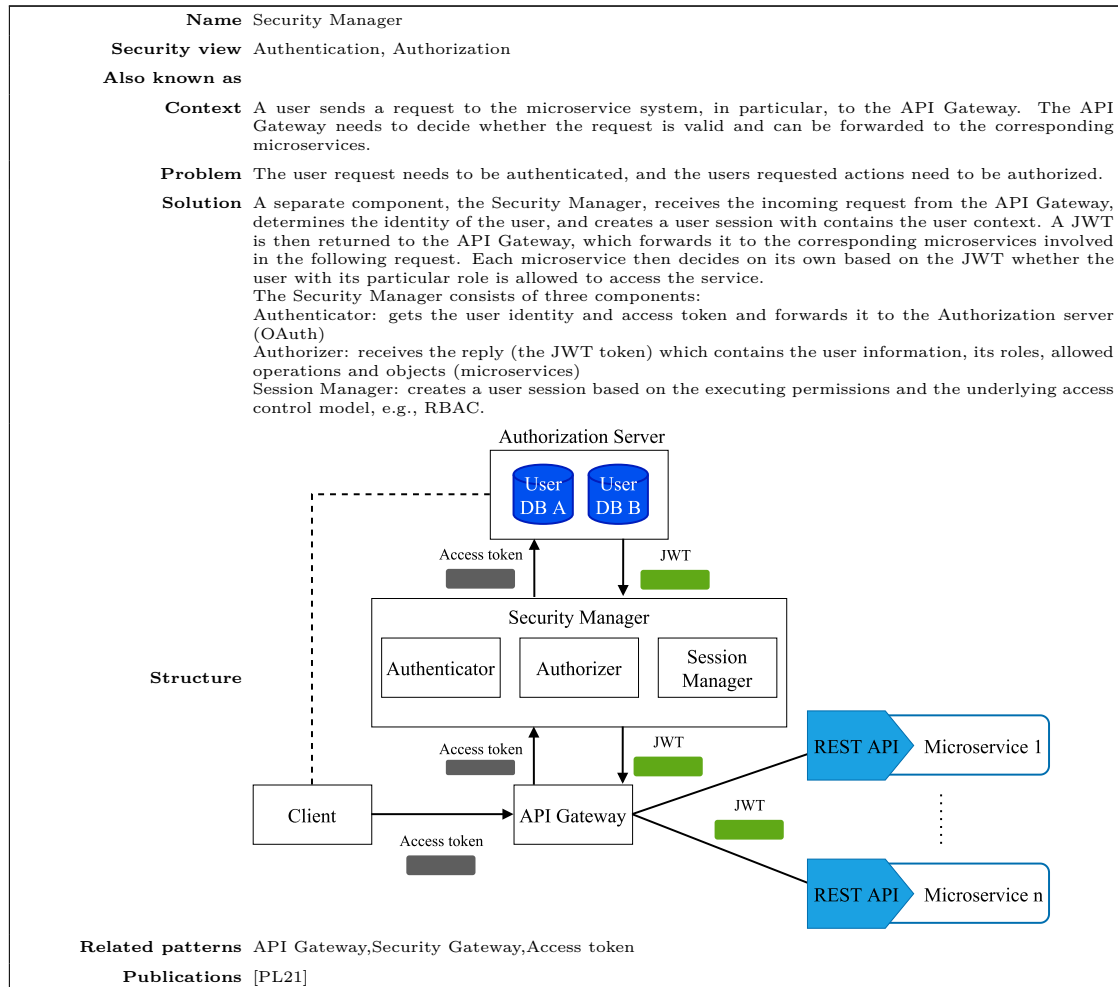


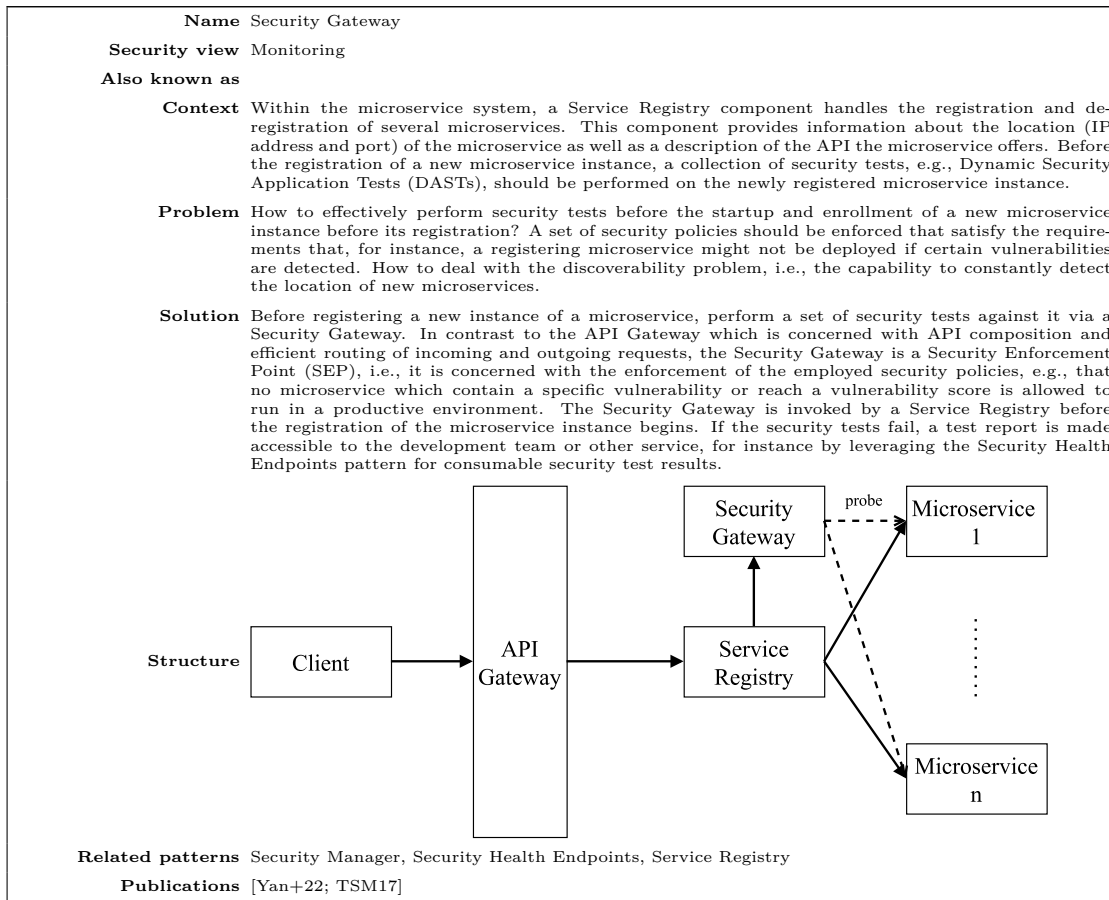




	<b>Name</b> Sidecar
	<b>Security view</b> Authentication, Monitoring, Authorization, Secure Communication
	<b>Also known as</b>
	<b>Context</b> Microservices need to implement functionalities that are not concerned with the business concerns of the application, such as logging or authentication.
	<b>Problem</b> Adding these cross-cutting concerns to the application microservice adds complexity, and couples functionalities within the same component. Although this allows efficient resource sharing, an outage of one such cross-cutting functionality might lead to an outage of the whole microservice.
	<b>Solution</b> Introduce a new component, running alongside the application microservice on the same container/host. The Sidecar processes cross-cutting concerns such as logging, authentication, routing or other tasks without interfering with the application microservice. This component is connected, but not part of the application microservice. Whenever it moves to a different host or pod, the Sidecar component moves along. Since the Sidecar runs in its own process, it can be developed in a different programming language that is suited for its purpose. Stacking or leveraging multiple Sidecars with one application microservice is possible.
<b>Structure</b>	 <pre>graph LR     subgraph Host_Pod [Host / Pod]         direction LR         AM[Application Microservice] &lt;--&gt; S[Sidecar]         subgraph Shared_State [Shared state]             disk             file_system             network         end     end</pre>
	<b>Related patterns</b> Ambassador, Container
	<b>Publications</b> [MR22; SKI19]







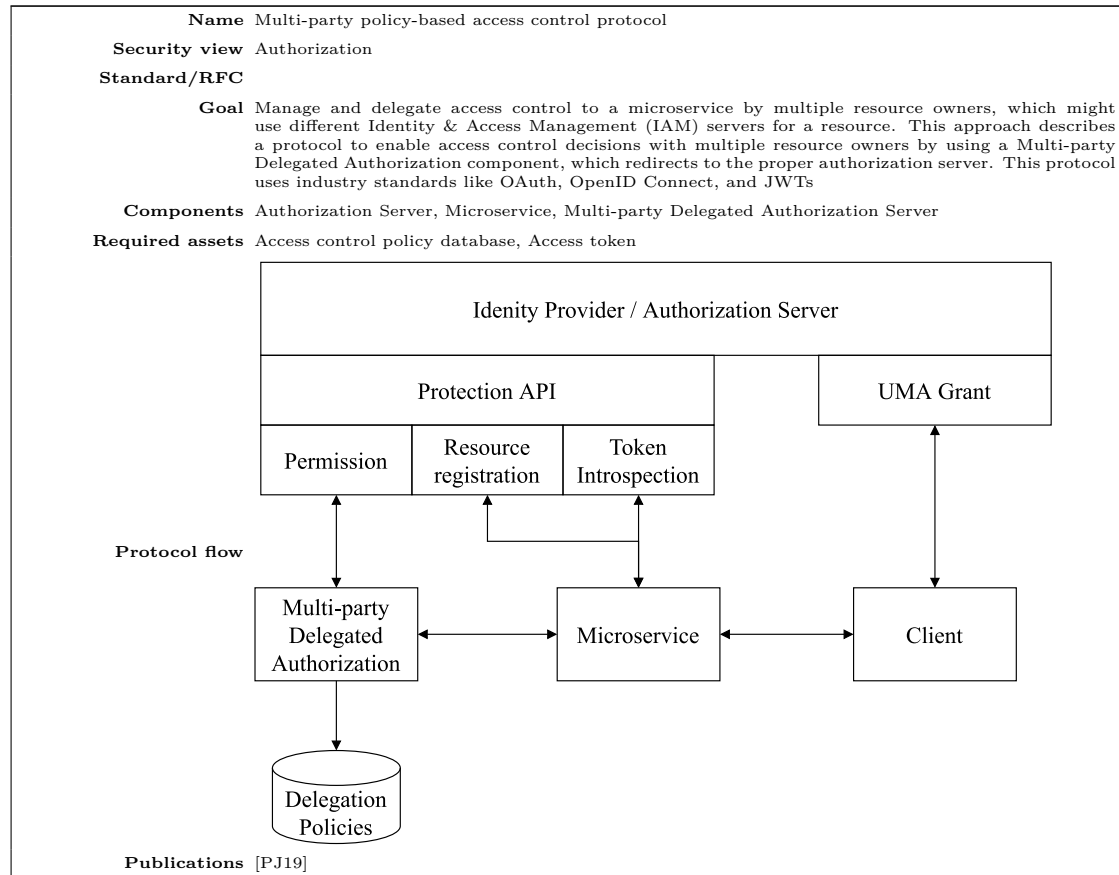
## B. Catalog of security design concepts

Structure	<b>Name</b>	API Gateway
	<b>Security view</b>	Authentication, Authorization
	<b>Also known as</b>	
	<b>Context</b>	A microservice system exposes many different API endpoints. Clients need to access data from multiple services, which can also be dynamically located. Depending on the client's use case, some services may or may not be required. There is a need to request and access multiple microservices (in a specific sequence perhaps) from different clients.
	<b>Problem</b>	How do clients get access to multiple, individual microservice applications? Microservice APIs are often very fine-grained. These details need to be abstracted from the client which is only interested in a subset of APIs that are needed for its specific use case. Microservices might also be implemented using different communication protocols, some of which are not web-friendly. Regarding security, clients need to be authenticated and authorized before they can perform any actions. Implementing these functionalities adds unnecessary complexity to each microservice, and increases the risk of implementation errors.
	<b>Solution</b>	Add an intermediate component between the microservice system and the client, the API Gateway, which is the single entry point to the microservice system. Instead of communicating with each microservice directly, clients perform their requests to the API Gateway which in turn evaluates the request and invokes the corresponding microservices in order. The API Gateway is able to translate the request to different communication protocols and has knowledge about the location and fine-grained API of each microservice. The API Gateway also authenticates each incoming request, for instance in conjunction with an Authorization Server to identify the user. For this purpose, protocols like OAuth 2.0 and OpenID Connect might be used. Identifying the user can be done by leveraging the Access token pattern. Furthermore, the API Gateway may implement other request-validating functionalities, like input validation, intrusion detection and prevention tasks as well as logging and monitoring the incoming requests.
	<pre> graph LR     Client[Client] --&gt; APIGateway[API Gateway]     APIGateway --&gt; MS1[Microservice 1]     APIGateway --&gt; MSn[Microservice n]     </pre>	
	<b>Related patterns</b>	Security Manager, Access token
	<b>Publications</b>	[MR22; Lu+17; Mös+17; Neh+19; Pon+21; AC22]

	<b>Name</b>	Access token
	<b>Security view</b>	Authentication, Authorization
	<b>Also known as</b>	API Key
	<b>Context</b>	A microservice system provides several APIs that subscribed clients may access. These clients need to be identified in order to check the privileges of the client and which services they may have access to.
	<b>Problem</b>	How can a microservice system identify and authenticate different clients making requests? The authentication of the client needs to be established without making the API inaccessible or user-unfriendly, while also minimizing performance and storage impacts.
	<b>Solution</b>	Assign a unique token to each client that the client can present to the API endpoint for identification purposes, either as part of the URL query or in the request body. Generated API keys are required to be unique and should be difficult to guess. UUIDs or other randomly assigned serial numbers are suited for this purpose. This solution provides a lightweight authentication alternative to a full authentication scheme or protocol. It balances basic security requirements with minimal management and communication overhead.
	<b>Related patterns</b>	Rate Limit, Security Manager, API Gateway, Transporter
	<b>Publications</b>	[Was+21b; Sto+18; JLE18]

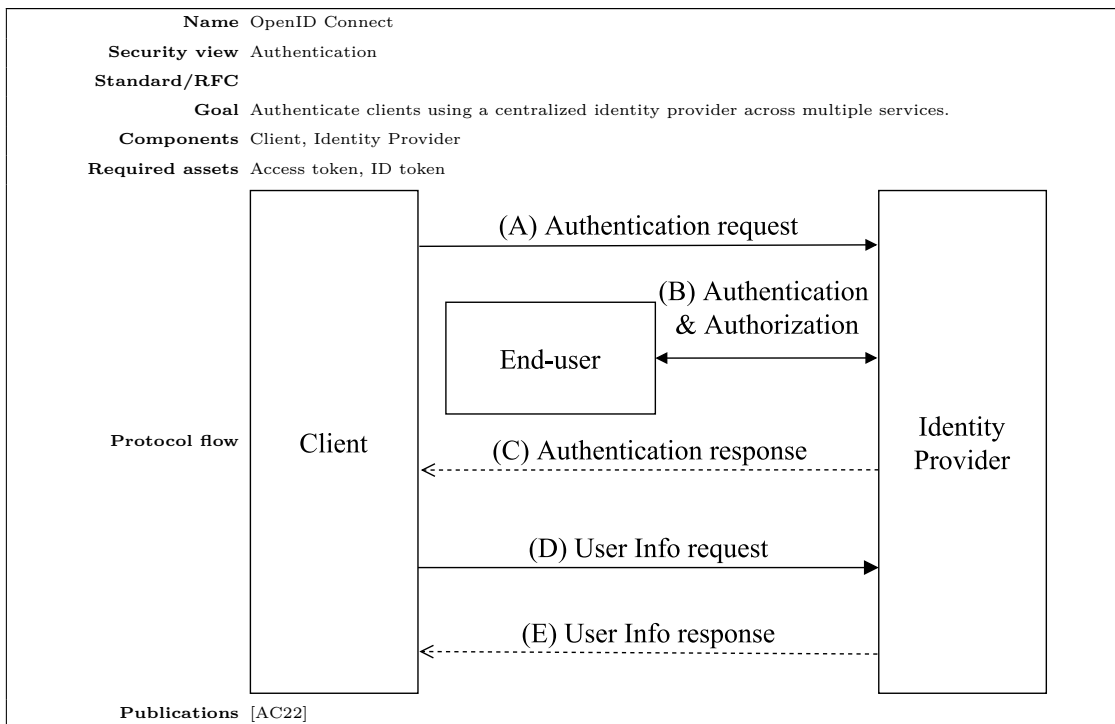
## B.5. Catalog of security protocols

<b>Name</b>	Policy-driven authorisation management & Capability-based access control
<b>Security view</b>	Authorization
<b>Standard/RFC</b>	
<b>Goal</b>	The client sends an access request to the Resource Server, which creates a capability token. Authorization services are used to validate the token and determine whether to accept or deny the request. Upon further requests, the capability token is used in combination with the relevant policy methods to determine the access capabilities.
<b>Components</b>	Authorization Server, Client, Microservice, Policy administration point (PAP), Policy decision point (PDP), Policy enforcement point (PEP), Policy information point (PIP)
<b>Required assets</b>	Access control policy database, Access token
<b>Publications</b>	[Kal+20]

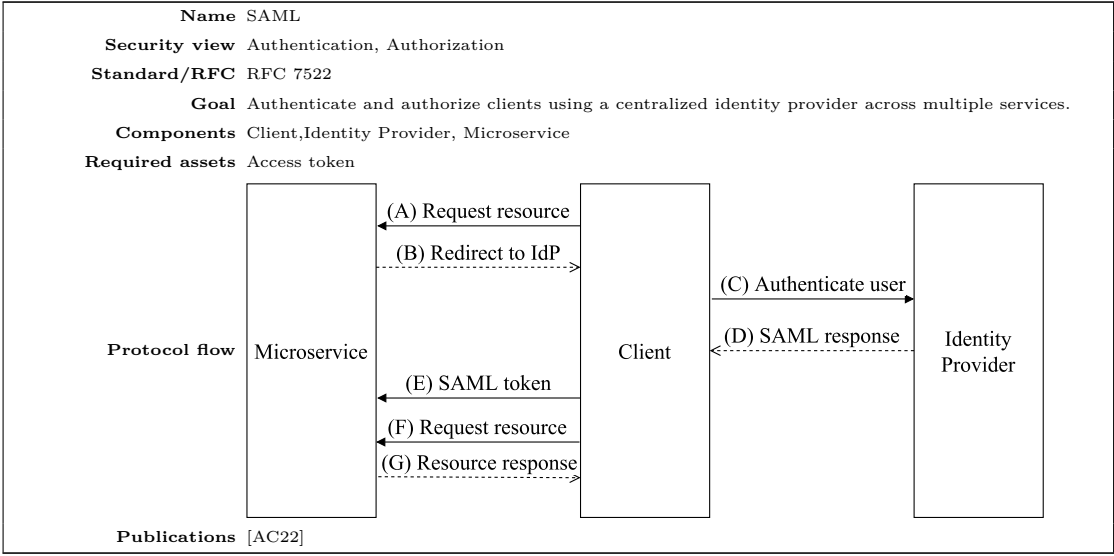


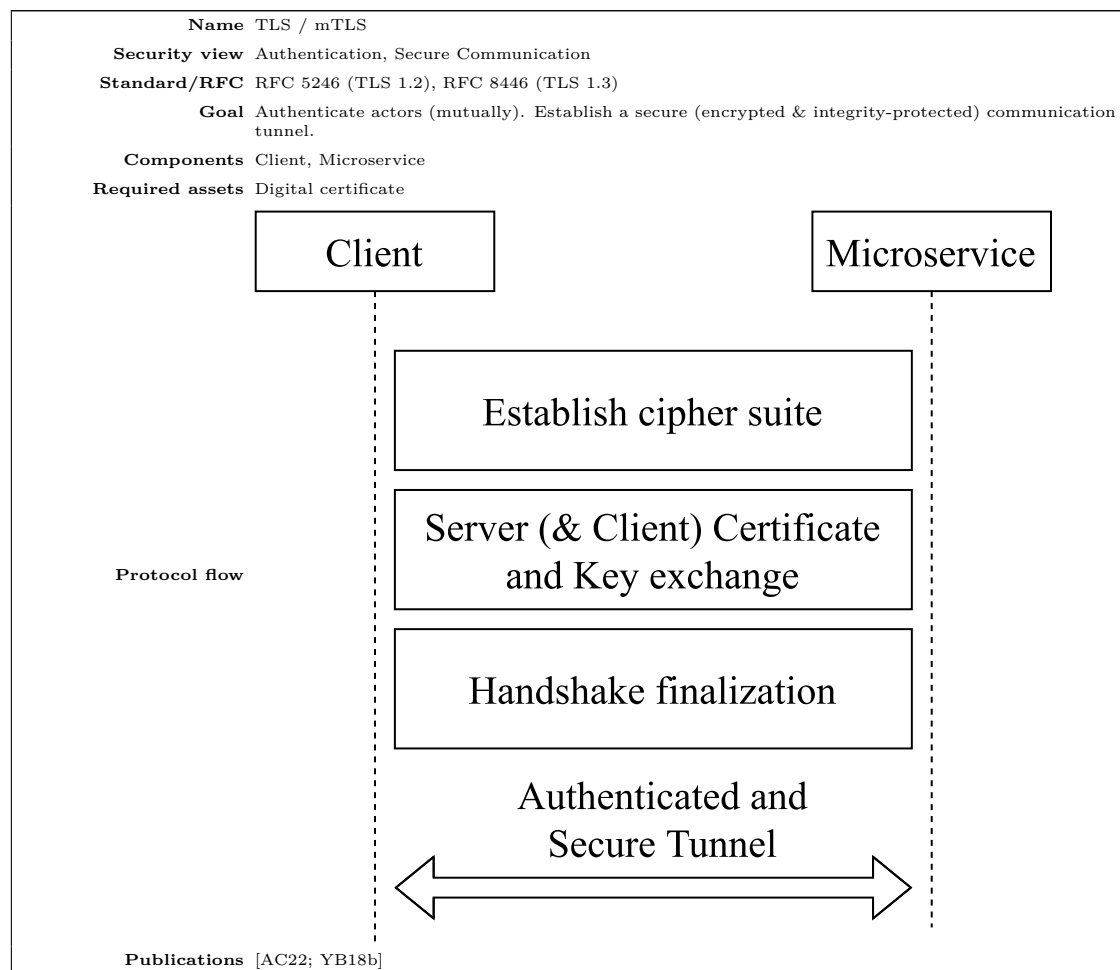
## B. Catalog of security design concepts

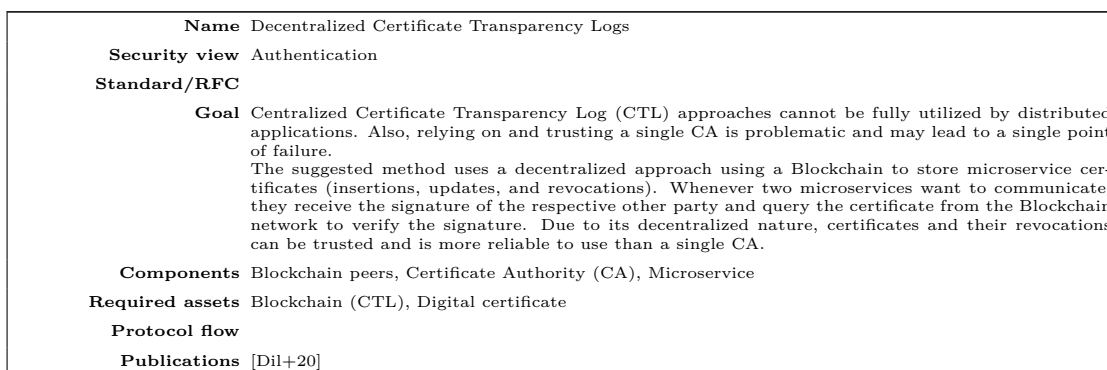
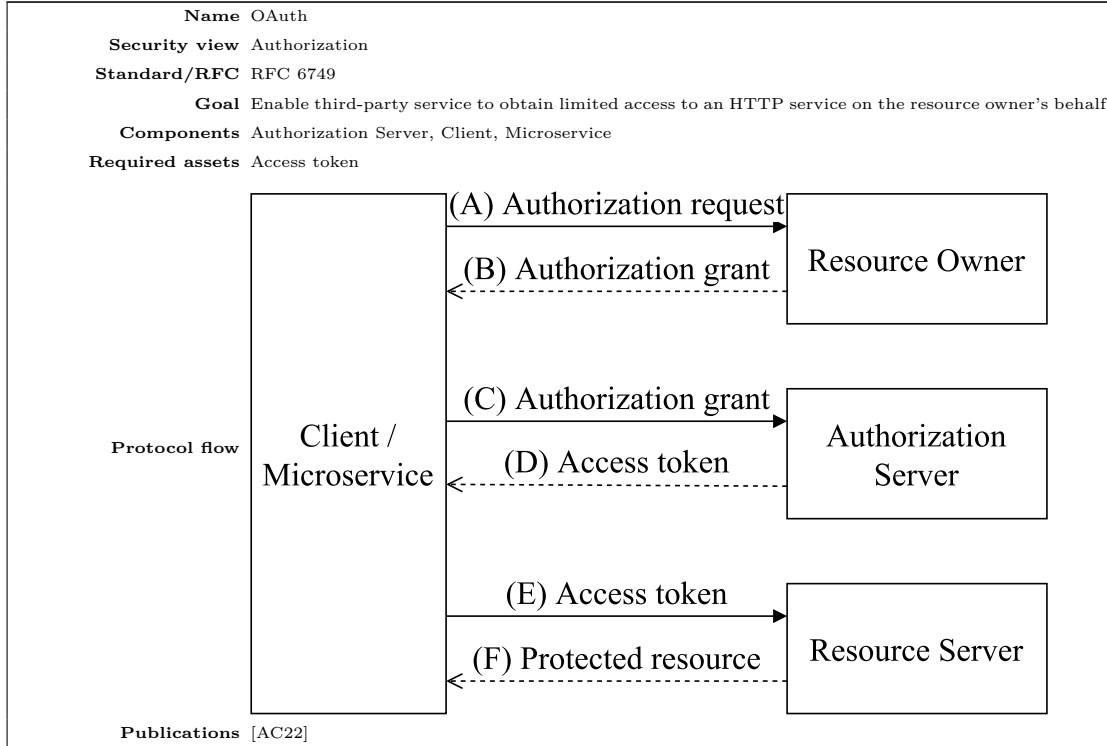
<b>Name</b>	Inter-service authentication, authorization, and communication encryption
<b>Security view</b>	Authentication, Authorization, Secure Communication
<b>Standard/RFC</b>	
<b>Goal</b>	Communication between microservices must be secured with the same level of detail as communication with external clients and services. Otherwise, an attacker is able, once a single service is compromised, to eavesdrop or manipulate the communication between microservices. This protocol implements a key exchange and a role authentication procedure. Authentication is based on either password, shared secrets or X.509 certificates. The provided protocol is similar to TLS or mTLS. Identified challenges for a solution like this are role support, autonomy, ease of use
<b>Components</b>	Microservice
<b>Required assets</b>	Passwords
<b>Protocol flow</b>	
<b>Publications</b>	[JBP18]











## B.6. Catalog of IDS/IPS approaches

<b>Name</b>	XML Injection Detection Tool
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	Penetration Testing Generator (PTG), Test Objective Generation (TOG)
<b>Evaluation asset</b>	Network traffic, XML
<b>Scope</b>	Host-based
<b>Placement</b>	API Gateway, Per microservice
<b>Threat</b>	Code injection
<b>Prevention</b>	
<b>Publications</b>	[SP19]

<b>Name</b>	Unsupervised Service Fissioning
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect, Respond
<b>Algorithm approach</b>	
<b>Evaluation asset</b>	CPU usage, Host resources, Memory usage
<b>Scope</b>	Host-based
<b>Placement</b>	Per microservice
<b>Threat</b>	Application-layer DoS
<b>Prevention</b>	Isolation/Shutdown
<b>Publications</b>	[Baa+20]

<b>Name</b>	Privacy-preserving monitoring & anomaly detection
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect, Respond
<b>Algorithm approach</b>	K-means clustering, SVM, machine learning
<b>Evaluation asset</b>	CPU usage, Disk usage, Host resources, Memory usage, Network traffic, Number of authentication failures, Number of requests/s, Service response time
<b>Scope</b>	Host-based
<b>Placement</b>	Central monitor, Per microservice, Sidecar
<b>Threat</b>	
<b>Prevention</b>	
<b>Publications</b>	[BAR21]

<b>Name</b>	Irregular Traffic Detection for RPC
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	Graph Convolution Network (GCN), machine learning
<b>Evaluation asset</b>	Network traffic, RPC
<b>Scope</b>	Network-based
<b>Placement</b>	
<b>Threat</b>	Account cracking, Batch registration attack
<b>Prevention</b>	
<b>Publications</b>	[CHC19]

<b>Name</b>	Automated Intrusion Response based on Game-theoretic approach
<b>Security view</b>	Monitoring
<b>Capability</b>	Respond
<b>Algorithm approach</b>	minimax algorithm
<b>Evaluation asset</b>	
<b>Scope</b>	Internal, Network-based
<b>Placement</b>	Per microservice
<b>Threat</b>	
<b>Prevention</b>	Isolation/Shutdown, Rollback/Restart service, Scale-up and Scale-down, Split or merge services
<b>Publications</b>	[YO18]

<b>Name</b>	Intrusion Detection System using Resilient Backpropagation Neural Network
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	Resilient Backpropagation Neural Network, machine learning
<b>Evaluation asset</b>	Network traffic
<b>Scope</b>	External, Network-based
<b>Placement</b>	API Gateway, Firewall
<b>Threat</b>	DDoS
<b>Prevention</b>	
<b>Publications</b>	[Alm+22]

<b>Name</b>	API Intrusion Detection System
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	SVM, machine learning
<b>Evaluation asset</b>	API, Bandwidth consumption, Client IP address, HTTP method, HTTP response code, Network traffic
<b>Scope</b>	External, Network-based
<b>Placement</b>	API Gateway
<b>Threat</b>	
<b>Prevention</b>	
<b>Publications</b>	[Bay+21]

<b>Name</b>	Anomaly detection of distributed traffic
<b>Security view</b>	Monitoring
<b>Capability</b>	Detect
<b>Algorithm approach</b>	Diffusion Convolutional Recurrent Neural Network (DCRNN), machine learning
<b>Evaluation asset</b>	Distributed trace, Network traffic, RPC
<b>Scope</b>	External, Network-based
<b>Placement</b>	Per microservice
<b>Threat</b>	Batch registration attack, DDoS, Password Brute Forcing
<b>Prevention</b>	
<b>Publications</b>	[JQL21; Jac+22]



## C. Case study materials

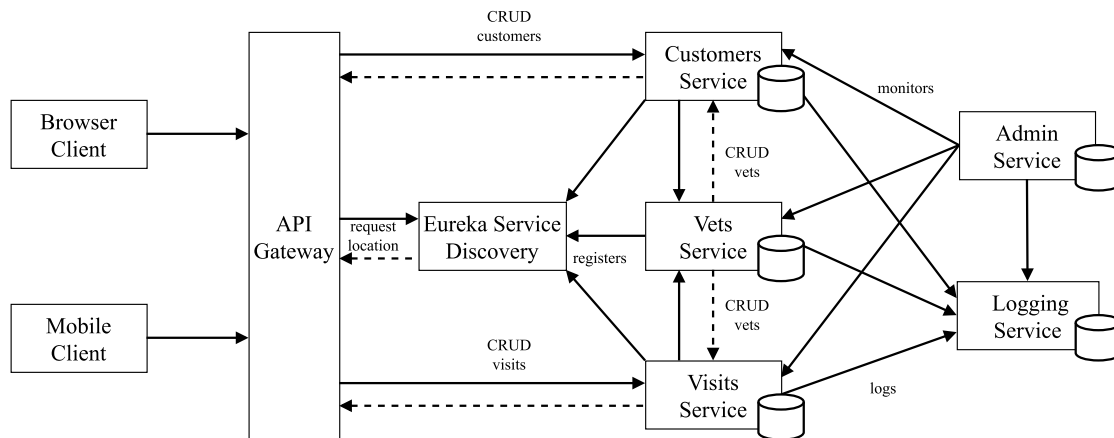


Figure C.1.: The initial Spring Petclinic architecture model.

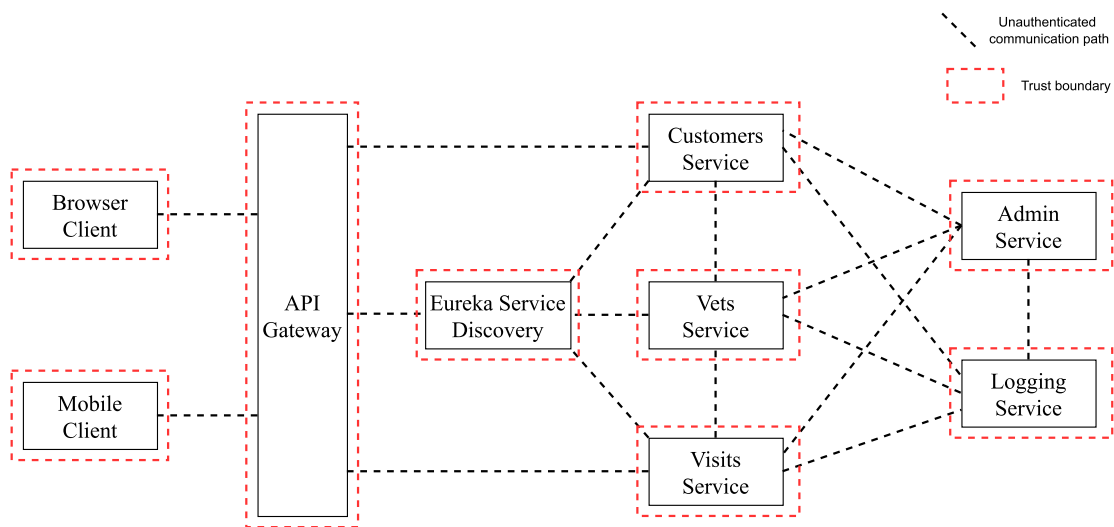


Figure C.2.: Addressing **SC-AN1** and **SC-AN2** by illustrating trust boundaries and which communication paths are still unauthenticated. This illustrates that the Zero trust design principle is followed.

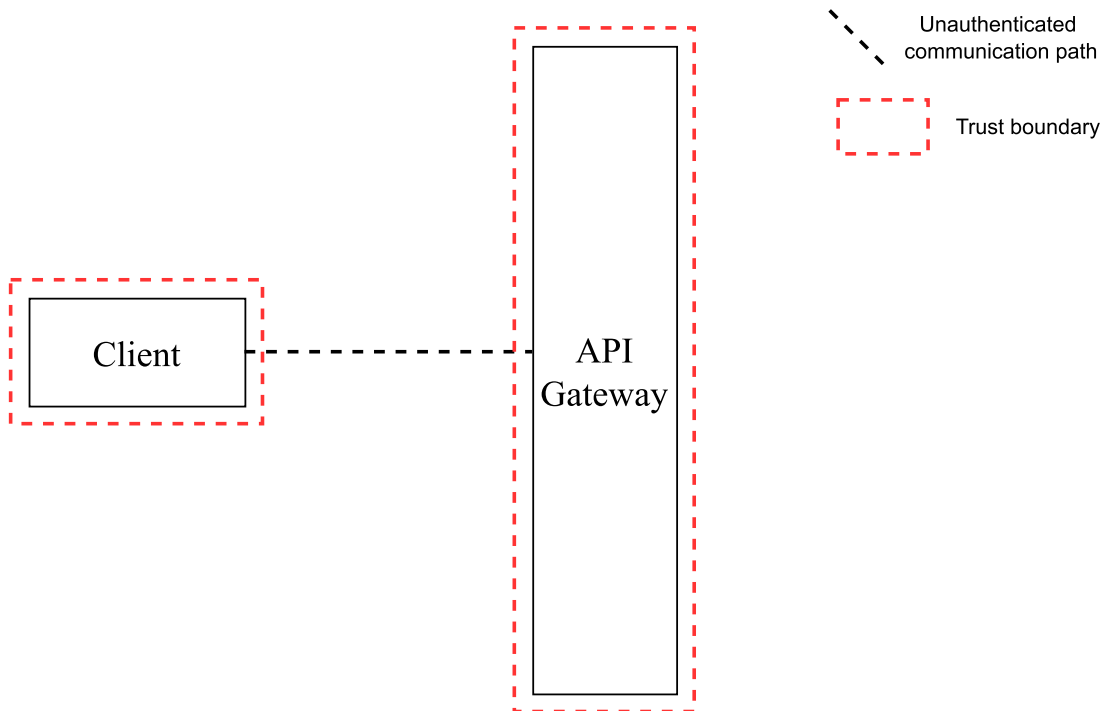


Figure C.3.: The first architectural context that was considered in the case study, namely the frontend part. This includes the communication and authentication of clients towards the API Gateway.

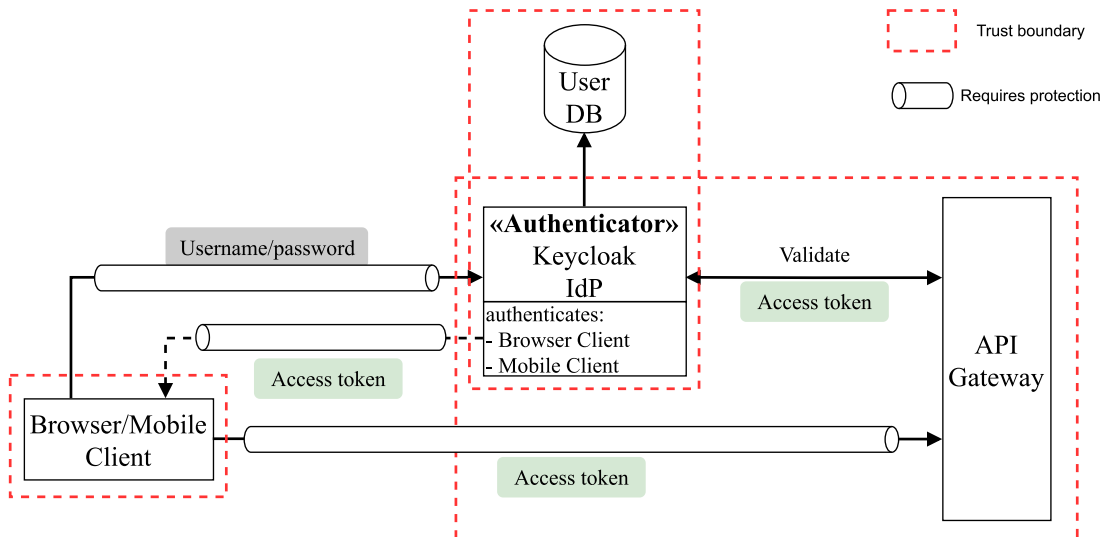


Figure C.4.: Authentication model for external clients. Clients authenticate themselves towards the API Gateway using the Keycloak Identity Provider instance as authenticator.



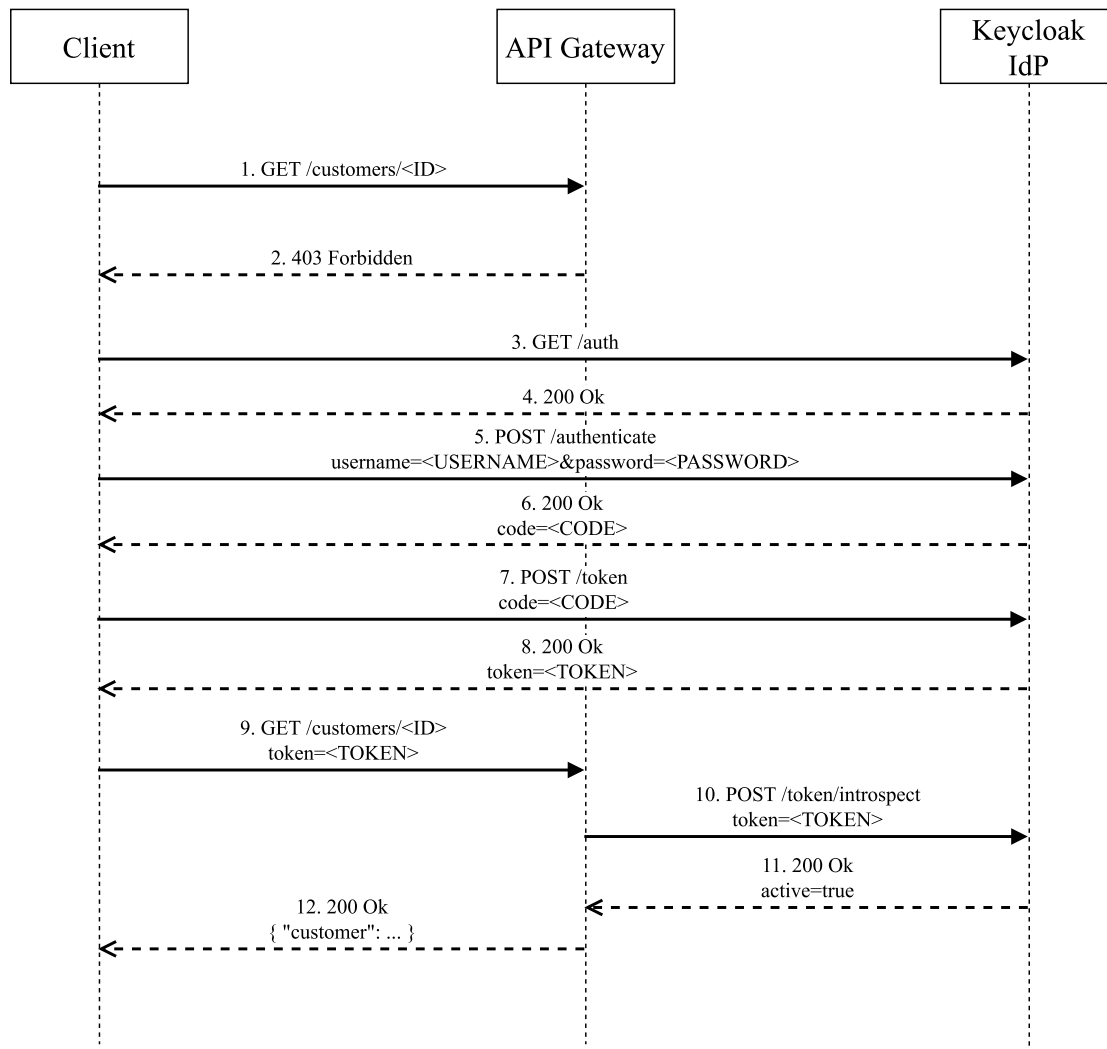


Figure C.5.: A sequence diagram of the communication between client, API Gateway and Keycloak IdP to perform client authentication. The client receives a `CODE` after a successful login by username and password. This `CODE` is exchanged by a `TOKEN` (cf. OAuth Authorization Code Flow grant [Har12]) which is verified by the API Gateway and Keycloak IdP afterward.

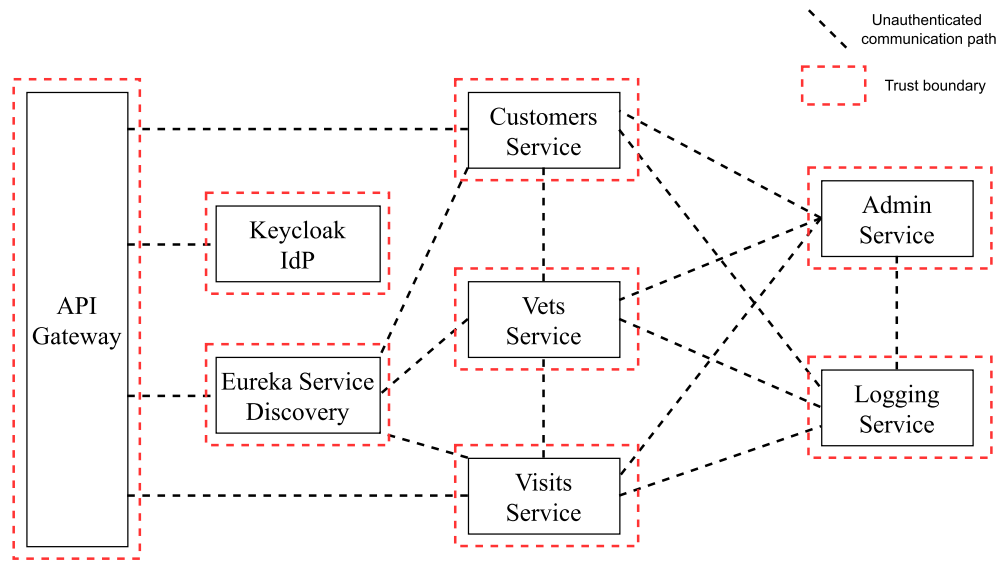


Figure C.6.: The second architectural context that was considered in the case study, namely the backend part. This includes the communication and authentication between microservices.

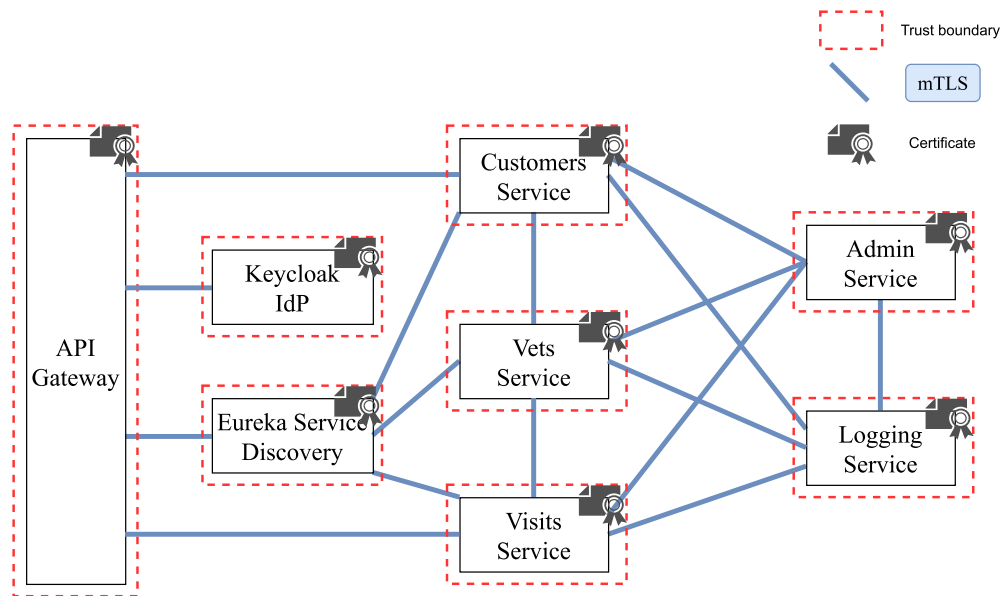


Figure C.7.: Authentication between microservices is achieved by employing the mTLS [RD08] security protocol. Each microservice is assigned a digital certificate, which is exchanged during the mTLS handshake protocol for authentication. These certificates hold the identity information of the respective microservice and a digital signature to verify.

## **D. Evaluation transcription**

The following pages present the shortened transcriptions of the performed interviews of the evaluation.

Table D.1.: Interview transcriptions, participants 1-4.

	Participant P1	Participant P2	Participant P3	Participant P4
Position	Professor for IT Security	Data Scientist	Software Architect	Lead Developer
Security modeling experience	<ul style="list-style-type: none"> <li>Model-based development</li> <li>Defining and modeling security requirements in SysML and Enterprise Architect.</li> <li>No specific security modeling tools were used, but rather standard tools were used in which security requirements were added to certain system interfaces.</li> <li>Rather risk-based analysis methods in conjunction with standards like IEC 62443 or ISO/IEC 27001. Regarding modeling diagrams, no systematic method was followed.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, had contact with the aforementioned concepts of security views, i.e., authentication, authorization, etc., but not explicitly modeling those as such. Rather approaching the question "which aspects exist to harden the application's security?", without separating it into several views.</li> <li>No particular modeling approaches were used.</li> <li>Primarily modeled security with use case diagrams, but without including specific security concerns.</li> </ul>	<ul style="list-style-type: none"> <li>A bit, security is rather considered incidentally than systematically with concrete security catalogs and formal models.</li> <li>Sift through catalogs like OWASP Top Ten or similar cheat sheets, but no application of concrete modeling techniques like threat modeling.</li> </ul>	<ul style="list-style-type: none"> <li>Security basics that are learned during studies</li> <li>Self-studying, i.e., reading books and articles, being up-to-date</li> <li>No security modeling during architecture design in particular</li> </ul>
Microservice experience	<ul style="list-style-type: none"> <li>Experience as Head of IT Security in a company that employed a cloud-based MSA based on Docker.</li> <li>Not actively developed on concrete microservices, but I was responsible for security design and reviews of the architecture.</li> <li>Project experience: 4 years</li> </ul>	<ul style="list-style-type: none"> <li>Currently working solely with MSAs, implementing 7 microservices and communicating with additional 10 microservices.</li> <li>Project experience: 1.5 years</li> </ul>	<ul style="list-style-type: none"> <li>Primarily working on hybrid architectures, i.e., that have aspects of multiple architectural styles (microservices and monoliths)</li> <li>Project experience: 4-5 years</li> </ul>	<ul style="list-style-type: none"> <li>Experience on MSA concepts, and differences to monoliths, but not in-depth</li> <li>Project experience: 3-4 years</li> </ul>

<p><b>What comprises security in the context of software architectures?</b></p>	<ul style="list-style-type: none"> <li>• Security must be considered in the different development phases depending on the software development model.</li> <li>• Requirements engineering: security requirements need to be collected and assessed by the project's stakeholders.</li> <li>• Design: consider security and develop a secure architecture which is depicted in diagrams. Security concerns must be addressed in these models. With threat modeling, additional requirements can be derived.</li> <li>• Deployment: how is the implementation of the designed architecture ensured, e.g., using techniques like IaC. Essentially, how to bring secure design into the production environment? Also, having development and staging environments to test security implementations and whether they fulfill security requirements.</li> <li>• Development process: defining certain controls during development phases, e.g., modeling architectures as an essential step.</li> <li>• Monitoring and verification: observe the system, and verify that the requirements are fulfilled. Also performing security tests in form of penetration tests and fuzzing for further inspections.</li> <li>• Lastly, having a feedback loop of the gained information from the verification phase and using this data for the next iterations to continuously adapt the architecture.</li> </ul>	<ul style="list-style-type: none"> <li>• Communication with other services, i.e., data arrives where it should and can only be accessed by those who are authorized.</li> <li>• Data is stored securely, and limitations of access rights need to be preserved.</li> <li>• Availability of data is ensured, i.e., they can be retrieved at any time.</li> <li>• Also that users are transparently informed about the security of the application they use, i.e., they can see that and how data and communication are secured, which access rights they have and how they retrieve them. This can be achieved by proper documentation, but these guarantees must be verifiable.</li> </ul>	<ul style="list-style-type: none"> <li>• There are many things to consider, e.g., from a developer perspective</li> <li>• Hardening the infrastructure: access to systems</li> <li>• Implementation (backend &amp; frontend)</li> <li>• Social components: security awareness of customers and developers</li> <li>• A problem is that security needs to be considered in many areas</li> </ul>	<ul style="list-style-type: none"> <li>• Knowledge about security best practices regarding the development of software architectures</li> <li>• OWASP Top 10</li> <li>• Depends on the application domain, e.g., when sensible user data is involved</li> </ul>
---	--	--	---	---

What does authentication entail?	<ul style="list-style-type: none"> <li>Authentication identifies a person based on some properties, e.g., credentials, and biometrics.</li> <li>The proof that the person or system holds indeed the claimed identity.</li> <li>Authentication does not concern the access rights of a person or system but is rather included in authorization, e.g., based on a specific role someone has.</li> </ul>	<ul style="list-style-type: none"> <li>I need to prove credibly that I have certain access rights.</li> <li>It includes verification of claimed access rights.</li> <li>Also includes a right and role system, but that also goes more in the direction of authorization.</li> <li>Authentication and authorization are often difficult to differentiate because they are so closely related.</li> </ul>	<ul style="list-style-type: none"> <li>That users and/or systems login and identify themselves using certain authentication methods, e.g. passwords, MFA, Face ID</li> </ul>	<ul style="list-style-type: none"> <li>Identification and verification of the user during an authentication process, e.g., password-based</li> <li>Needs to be differentiated from authorization</li> </ul>
Do you agree with these authentication concerns?	<ul style="list-style-type: none"> <li>Regarding <b>SC-AN1</b>: nowadays, we are strongly moving towards Zero trust, i.e., authentication is required anywhere.</li> <li>But it makes sense to model is explicit, e.g., when access to a deep database server in the network is trusted and does not require further authentication.</li> </ul>	<ul style="list-style-type: none"> <li>Yes</li> </ul>	<ul style="list-style-type: none"> <li>Regarding <b>SC-AN1</b>: let's assume that we have a Kubernetes cluster in an internal network, i.e., all services are trusted.</li> <li>But some services provide more sensitive data. Thus, even within a trusted environment, authentication might be necessary. Therefore, trust never exists.</li> <li>There might be services that are not worth protecting, e.g., a web server that provides public content.</li> <li>The necessity of authentication depends on the value of the provided data of these services.</li> </ul>	<ul style="list-style-type: none"> <li>In principle yes</li> <li><b>SC-AN7</b> might indicate that there is a central component or service that manages and controls all secret information about other services. This is a potential single point of failure.</li> </ul>
Are there any missing authentication concerns?	<ul style="list-style-type: none"> <li>I would add concern about audit trail logs.</li> <li>How are successful and failed authentication requests logged and who can access them? How are they managed and protected from manipulation?</li> <li>Otherwise it seems good. These concerns are on a very abstract level, but I guess this is intended.</li> </ul>	<ul style="list-style-type: none"> <li>How to securely introduce secrets into the application? (Secret Management)</li> <li>How to proceed after successful/failed authentication?</li> </ul>	<ul style="list-style-type: none"> <li>Ad hoc, I can't think of any other concerns, unless we were to discuss them in detail.</li> </ul>	<ul style="list-style-type: none"> <li>No</li> </ul>
Does this model address these authentication concerns?	<ul style="list-style-type: none"> <li>Definitely, but security know-how is required to understand these models.</li> <li>I see there is a shift to bring these topics into the language of software engineering.</li> <li>For security experts that review these kinds of architectures, it is helpful.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, but there is no overview of how it works in completeness because no model addresses all concerns simultaneously.</li> <li>In particular, some models violate rules that were made before on purpose (cf. Figure 6.1b). A complete overview of the architecture could solve such ambiguities.</li> </ul>	<ul style="list-style-type: none"> <li>It makes sense to follow this approach, but I can not make a statement about its completeness.</li> <li>For this purpose I would sift through existing security documents like OWASP ASVS to highlight the specific areas.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, but not each model addresses the full set of security concerns.</li> <li>After a clarification discussion, the participant agreed that it is not required that each model addresses all security concerns at once, but each model might address a subset.</li> </ul>

<p><b>Does this model effectively communicate established security design concepts?</b></p>	<ul style="list-style-type: none"> <li>• Yes and no.</li> <li>• There is a need for required security knowledge.</li> <li>• There might be an additional step for the developers. From my experience, they would be confused about some implementation details, e.g., how certificates are deployed to the microservices or how to employ mTLS.</li> <li>• On the other hand, a supporting security champion can advise developers on these types of questions.</li> <li>• When syntax and semantics are clear and consistent, it is helpful to understand, e.g., when a resource must be protected and how.</li> </ul>	<ul style="list-style-type: none"> <li>• No, not really because the applied design concepts were not known as such beforehand.</li> </ul>	<ul style="list-style-type: none"> <li>• Yes, but especially regarding the implementation, some models need to be more specific and include more details.</li> <li>• For instance, there exist several OAuth authorization flows, some of them are deprecated and shouldn't be used. The architecture must reflect details about the authentication method in use and how specifically they are employed.</li> </ul>	<ul style="list-style-type: none"> <li>• Yes, but these models might suffer from being outdated since protocols like TLS might become deprecated at some point. This can lead to architectural erosion.</li> <li>• Suggestion to rather use implementation-agnostic notations, e.g., arbitrary current secure protocol.</li> <li>• Principles like Zero Trust on the other hand are timeless.</li> </ul>
<p><b>Which modeling elements should be added/removed?</b></p>	<ul style="list-style-type: none"> <li>• Regarding the authentication view, I don't have any suggestions.</li> </ul>	<ul style="list-style-type: none"> <li>• The client-side authentication model (cf. Figure 6.1b) should not purposefully violate the Zero Trust principle, although it is tackled in another model. Instead, it should anticipate the trust boundaries and mTLS protocol between Keycloak IdP and API Gateway as in the server-side authentication model (cf. Figure 6.1c).</li> <li>• Otherwise such a model causes confusion and is very error-prone.</li> <li>• Another solution would be to explicitly note when certain rules or principles are violated on purpose and resolved in other models.</li> <li>• An explicit list or marks of applied design concepts was missing.</li> <li>• Also a model for a complete overview should be added.</li> </ul>	<ul style="list-style-type: none"> <li>• If shown to a customer, trust boundaries could be removed for simplification, since the Zero trust design principle specifies this anyway. Showing it on one slide can be useful.</li> <li>• It's difficult to assess whether modeling elements should be added or removed because we treat security rather implicitly, but it is good to model security separately to identify weaknesses precisely.</li> <li>• Otherwise, you don't have an overview of what is happening in the architecture.</li> <li>• I think network segments should be included as well because it becomes important for implementation, especially when you deal with multiple cloud providers. For instance, where exactly is the Keycloak IdP located and how do components communicate with each other, i.e., which protocols are used?</li> </ul>	<ul style="list-style-type: none"> <li>• In the beginning, the notion of trust boundaries was confusing. Speaking of trust boundaries in conjunction with zero trust clarified that.</li> <li>• Adding modeling elements does not make sense, since it becomes too complicated.</li> <li>• Removing modeling elements also means removing relevant information.</li> <li>• Simplifying modeling elements, e.g., that each microservice has a certificate, instead of having a specific notation for each microservice, would help.</li> <li>• Also having a more general notation instead of modeling specific protocols like TLS and certificates would be useful.</li> </ul>

<p><b>Does such a model enhance common understanding/communication regarding authentication?</b></p>	<ul style="list-style-type: none"> <li>• Generally yes. When you have a visual representation and a mapping of security requirements, it is helpful.</li> <li>• One can think about adding more concrete models to these views explaining implementation details to become more independent from a security expert.</li> </ul>	<ul style="list-style-type: none"> <li>• Definitely. Such models help to think about security concerns in-depth. Also, the specific designation of concerns, views, and design concepts helps to understand and formalize how security is implemented in the architecture.</li> <li>• It is required to have such documentation, especially when new colleagues have to deal with the architecture and need to understand which security controls are applied.</li> <li>• It is important that the documentation is self-explanatory, i.e., one can understand it without additional explanations from another person. Otherwise, it is useless. For instance, the purpose and idea behind Figure 6.1b were only comprehensible because it was explained additionally.</li> <li>• Furthermore, models need to be diagrams, and not fully textual. Highlighting certain modeling elements, e.g., trust boundaries, by using specific colors helps to grasp the important aspects of the model.</li> </ul>	<ul style="list-style-type: none"> <li>• Certainly. Modeling security explicitly helps to highlight these things better than making it implicit.</li> </ul>	<ul style="list-style-type: none"> <li>• It is very useful when a common foundational understanding of security is established beforehand.</li> <li>• When building a new architecture and you have a reference architecture that can be trusted to be secure and defines to what extent certain components need to be secured in a specific way.</li> <li>• Such a model would be easily extensible for new projects.</li> <li>• Concept of concerns helps to understand and address where authentication is actually necessary and required.</li> <li>• These models enhance communication within a team, e.g., when new services are added by addressing and re-evaluating the authentication concerns.</li> <li>• Again, a premise is that a common understanding of security in general exists, e.g., the difference between authentication and authorization.</li> </ul>
<p><b>How would you assess the distinction into security views?</b></p>	<ul style="list-style-type: none"> <li>• Generally I would say that it is a good distinction.</li> <li>• Secure Build &amp; Deployment is a rather large area, splitting this view up might be useful. For instance, having a separate view for security testing.</li> </ul>	<ul style="list-style-type: none"> <li>• In principle yes. It covers the set of security concepts that I am aware of in my experience.</li> </ul>	<ul style="list-style-type: none"> <li>• Authentication and authorization could be merged, because they often considered together, at least implementation-wise they are.</li> </ul>	<ul style="list-style-type: none"> <li>• Cross-cutting concerns between security views can be problematic. For instance, authentication and authorization share a lot of concerns and responsibilities. This would lead to several models with duplicate information.</li> </ul>



Which security views need to be considered additionally?	<ul style="list-style-type: none"> <li>I can think of more specific views for certain areas of a microservice architecture, e.g., different views for frontend and backend, or databases because there are more specific requirements and mechanisms in these different contexts.</li> <li>Thus, additional views that lower the abstraction level might be useful when moving toward implementation.</li> <li>In contrast, these general models apply to a wide variety of architectures.</li> </ul>	<ul style="list-style-type: none"> <li>A data security view should be added, although it might be out of scope for the architecture.</li> <li>This view should address how and where data of the application is processed by its users and whether further hardening of the data security is necessary.</li> <li>This needs to be compliant with certain laws, e.g., GDPR. For instance, when cloud services are involved, such concerns need to be addressed.</li> </ul>	<ul style="list-style-type: none"> <li>Perhaps a separate view for input validation (not part of monitoring), including IDS/IPS and Web Application Firewalls (WAFs).</li> <li>Secret management view, i.e., how are Secrets managed and perhaps rotated, e.g., using secret management tools like Azure Keyvault.</li> <li>Backup strategy view, i.e., how are backups created, what is the backup strategy if data is lost.</li> <li>Security Policy view, i.e., modeling policies that are enforced by the company. The problem is that these policies affect possibly all other views.</li> </ul>	<ul style="list-style-type: none"> <li>Developer view. The aforementioned views focus on product security. But developers often have wide access to infrastructure services that impose security risks.</li> </ul>
Which concerns need to be addressed in this view?	<ul style="list-style-type: none"> <li>More specific concerns for more specific areas, e.g., frontend/backend distinction.</li> </ul>	<ul style="list-style-type: none"> <li>How and where is data processed?</li> <li>Does it conform to data protection and privacy laws, e.g., GDPR?</li> <li>Who has accessed which information at which time?</li> <li>Which data confidentiality levels apply, e.g., where is sensitive personal information located and processed?</li> </ul>	<ul style="list-style-type: none"> <li>Secret management: how are secrets protected? How can applications access those secrets securely?</li> <li>Input validation: how the system alerts any events? Which anomalies can be detected?</li> <li>Policy view: which policies exist? Which are relevant for the architecture?</li> </ul>	<ul style="list-style-type: none"> <li>A developer view needs to address the access rights of a developer, e.g., who can have access to a Jenkins pipeline.</li> <li>A significant role that needs to be considered is the trade-off between security and comfortability.</li> </ul>
Which modeling notations/languages would you suggest?	<ul style="list-style-type: none"> <li>My only suggestion is to be consistent with the team's or company's modeling languages and standards.</li> <li>For instance, when UML is heavily employed, it is necessary to adapt these UML models rather than creating completely different modeling languages for security. Otherwise, this would increase the gap when integrating security modeling into the architecture.</li> </ul>	<ul style="list-style-type: none"> <li>Not any in particular, due to missing experience in that area.</li> </ul>	<ul style="list-style-type: none"> <li>No, I can not judge that.</li> </ul>	<ul style="list-style-type: none"> <li>UML, because it is an established and well-known modeling language.</li> </ul>

<p><b>Are special-purpose modeling notations/languages helpful?</b></p>	<ul style="list-style-type: none"> <li>I think it is useful to have a consistent language to model these views, perhaps with the ability to use modeling elements in multiple views to prevent that developers need to incorporate them into different languages.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, it does make sense. However, having separate languages for each view is too much. Especially when certain modeling elements from one security view are needed in another.</li> <li>Having a unified, standardized language would also increase security awareness.</li> <li>It can also be an obstacle requiring learning a new modeling language.</li> </ul>	<ul style="list-style-type: none"> <li>I think so, but they need to be intuitive and not too complex.</li> <li>They need to be understood without reading exhaustive documentation.</li> <li>Special-purpose languages help to model these aspects uniquely and to avoid ambiguities.</li> </ul>	<ul style="list-style-type: none"> <li>Only if these modeling languages extend the UML family.</li> <li>The advantage when using a notation that is similar to UML is that it is easily recognizable and comprehensible.</li> </ul>
<p><b>Can we study the influence of applied security design concepts?</b></p>	<ul style="list-style-type: none"> <li>That is very difficult. I also think that is heavily process-oriented.</li> <li>I think this modeling technique, especially with special-purpose modeling languages, is good, but like any other methodology, it is important to have and follow the right processes.</li> <li>It is much better than doing nothing or only having requirements.</li> <li>It might be possible to map the influence empirically, e.g., tracking how many bugs or security flaws I have found during penetration testing.</li> <li>I think studying these by verification and testing phases, and comparing the results between projects that have used this modeling approach against those that haven't.</li> </ul>	<ul style="list-style-type: none"> <li>Having these concerns and models that address them or a set of these concerns simplifies the quantification of the security of the architecture.</li> <li>Thus it helps to determine the influence of a design concept regarding how it addresses the corresponding concerns.</li> <li>Quantifying security is closely related to what extent the architecture fulfills the security requirements formulated as concerns.</li> </ul>	<ul style="list-style-type: none"> <li>Of course, it depends on how it is lived in the team.</li> <li>One possibility is to enforce that such security views are required during the creation of concepts, i.e., that these views are included in our concept templates.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, if the models that were created are indeed correct and represent the architecture.</li> </ul>
<p><b>Do such views capture the software engineering perspective?</b></p>	<ul style="list-style-type: none"> <li>Definitely yes. When you compare it with the initial architecture model, there is a greater value to it.</li> </ul>	<ul style="list-style-type: none"> <li>Security design concepts are needed by software developers. Having the knowledge and terminology of such concepts helps to understand which constructive possibilities and recommendations are available to solve a certain security problem.</li> <li>These views provide something that can be used hands-on.</li> </ul>	<ul style="list-style-type: none"> <li>It needs to be more detailed, but in general yes.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, up to the point that the developer view is missing.</li> <li>If a developer is compromised, it quickly becomes a serious problem.</li> </ul>

<p><b>How would you assess the worth of this modeling approach?</b></p>	<ul style="list-style-type: none"> <li>• For software architectures, this modeling approach is very good, because it is very generic.</li> <li>• For microservices, I am missing which deployment environments exist and how microservices are deployed to these environments. For instance, whether we have a container or OS environment, we illuminate the differences between certain microservice architectures. Such an infrastructure view might be useful.</li> </ul>	<ul style="list-style-type: none"> <li>• It provides additional value when everyone in the team adopts a standardized and formalized modeling approach for security.</li> <li>• Also, it may prohibit thinking of security as an afterthought.</li> <li>• For MSA it may be of further value because the communication paths and interconnections amplify and therefore more places exist where information can be captured by a malicious party.</li> </ul>	<ul style="list-style-type: none"> <li>• The value of this approach becomes clear when applied to a concrete project.</li> <li>• I can imagine that it is meaningful and there is much to gain.</li> <li>• Especially to get team members to actively deal with security topics, i.e., making these modeling aspects concrete.</li> <li>• Communication of security topics with customers and within a team.</li> </ul>	<ul style="list-style-type: none"> <li>• The main problem with addressing security in a software architecture is that it takes a lot of time to deal with this topic.</li> <li>• The true value of this modeling approach becomes clear when it is established in a project, as it would probably save time when addressing the security of an architecture.</li> <li>• Having standard processes and modeling tools eases understanding and comprehensibility and saves time.</li> <li>• The primary value is to have a reference model which can be trusted.</li> </ul>
---	---	--	---	---

Table D.2.: Interview transcriptions, participants 5-7.

	Participant P5	Participant P6	Participant P7
Position	IT Security Consultant	Frontend Developer	Applied Scientist
Security modeling experience	<ul style="list-style-type: none"> <li>Primarily experiences in threat modeling with data flow diagrams.</li> <li>Modeling approaches: STRIDE (for threat identification &amp; classification) and PASTA (for risk analysis).</li> <li>Using threat databases for quicker identification of relevant threats.</li> <li>Modeling notations: DFDs to depict trust boundaries and threats in systems</li> </ul>	<ul style="list-style-type: none"> <li>No particular experience in security modeling</li> </ul>	<ul style="list-style-type: none"> <li>Experience from secure software development university courses</li> </ul>
Microservice experience	<ul style="list-style-type: none"> <li>Little to no hands-on experience with microservice architectures, but experience with distributed systems.</li> <li>Concepts of MSAs are clear.</li> </ul>	<ul style="list-style-type: none"> <li>Project experience: 1 year</li> </ul>	<ul style="list-style-type: none"> <li>Hands-on experience on MSAs at work</li> <li>Project experience: 1 year</li> </ul>
What comprises security in the context of software architectures?	<ul style="list-style-type: none"> <li>Threat modeling is one of the most important security topics</li> <li>Security aspects differ depending on the context. For instance, authentication is important, but not in every context. Identifying where certain security aspects are required and necessary is difficult.</li> <li>Most things can be reduced to the CIA triad. Where do we need confidentiality, integrity, and availability?</li> <li>Secure coding and a secure development process, e.g., with SAST, manual testing and penetration testing.</li> <li>No matter how much you test, the perfect software does not exist and they all contain some vulnerabilities. Thus, it is important to have a software development lifecycle to react to such events and patch any findings.</li> </ul>	<ul style="list-style-type: none"> <li>Data encryption, to prevent others to read along</li> <li>Prevent malicious actors from gaining access and manipulation</li> <li>Traceability of data and actions</li> </ul>	<ul style="list-style-type: none"> <li>Identification and protection of assets, i.e., not everyone is allowed to access</li> <li>What is the purpose of that particular service, what data is processed, and what needs to be protected?</li> <li>Who is allowed to do what (authentication and authorization)</li> <li>Regarding architecture, we need to know which information is allowed to be exchanged between services. Not every microservice may have permission to access every other service.</li> </ul>
What does authentication entail?	<ul style="list-style-type: none"> <li>It consists of three parts:</li> <li>Authentication: the process of presenting someone or a system my identity with some credentials</li> <li>Authetification: verification of identity claim, whether it is valid or fake</li> <li>Authorization: after I am successfully authenticated, what am I allowed to do? Which files can I access? To which microservices can I perform requests?</li> </ul>	<ul style="list-style-type: none"> <li>Whenever data is fed into a system, the system needs to authenticate the user.</li> <li>The user needs to identify himself.</li> <li>The system needs to determine whether the user is authorized to do that.</li> </ul>	<ul style="list-style-type: none"> <li>Requesting and retrieving credentials</li> <li>Credentials need to be stored securely on server side, i.e., hashing with salt.</li> <li>The communication on which credentials are exchanged must also be protected.</li> <li>Users must be enforced to choose secure passwords.</li> </ul>

Do you agree with these authentication concerns?	<ul style="list-style-type: none"> <li>Generally, I would agree that this set of concerns is correct and complete.</li> <li>If I would think about it for some time, I guess I would find some additional concerns that are missing.</li> </ul>	<ul style="list-style-type: none"> <li>Yes</li> </ul>	<ul style="list-style-type: none"> <li><b>SC-AN1</b> must also reflect how to deal with external unknown clients and spoofing.</li> <li>Rephrase <b>SC-AN2</b> and <b>SC-AN3</b> to include which <i>secure</i> communication paths and authentication method is used. Deprecated, insecure cryptographic algorithms need to be excluded.</li> </ul>
Are there any missing authentication concerns?	<ul style="list-style-type: none"> <li>Regarding authentication, it seems to be complete.</li> <li>Some concerns regarding authentication and authorization might be missing.</li> </ul>	<ul style="list-style-type: none"> <li>How is identity information stored, and how are they possibly secured (if needed)?</li> </ul>	<ul style="list-style-type: none"> <li>In addition to <b>SC-AN4</b>: How to secure the asset? This should be added to as an authentication concern because we introduce additional assets because of the employed authentication method.</li> </ul>
Does this model address these authentication concerns?	<ul style="list-style-type: none"> <li>Yes. The models depict which concerns are addressed by which solutions.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, it does address the authentication concerns.</li> </ul>	<ul style="list-style-type: none"> <li>Figure 6.1b does not explain the validity of the session, this needs to be addressed in the model. This includes how long is an access token valid, and when a user needs to re-authenticate.</li> <li>In the same model, it is confusing that the Keycloak IdP and API Gateway are in the same trust boundary.</li> <li>Communication paths, i.e., which protocols are used, are not clear.</li> </ul>
Does this model effectively communicate established security design concepts?	<ul style="list-style-type: none"> <li>Yes.</li> </ul>	<ul style="list-style-type: none"> <li>From these models, details about the authentication method, e.g., how mTLS works and why it is used, are not clear.</li> <li>The remaining models present which design concepts are used.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, but information about the validity of the session and details about communication protocols in which paths are protected should be included.</li> </ul>
Which modeling elements should be added/removed?	<ul style="list-style-type: none"> <li>Perhaps leaving the arrow notation on the authentication models.</li> <li>Technically it is not important for authentication, but it is small support that doesn't cost much.</li> </ul>	<ul style="list-style-type: none"> <li>No. All in all, the modeling elements capture the design concepts and how they address the concerns.</li> </ul>	<ul style="list-style-type: none"> <li>Modeling elements that indicate how long an access token is valid.</li> <li>Details about communication protocols and network boundaries should be included, e.g., by labeling certain communication paths.</li> </ul>
Does such a model enhance common understanding/communication regarding authentication?	<ul style="list-style-type: none"> <li>Definitely. Sure, you can talk much about authentication and how it is implemented, but such diagrams make things clear quickly.</li> </ul>	<ul style="list-style-type: none"> <li>Yes. It clarifies which security mechanisms are in use and how microservices communicate with each other using encryption protocols.</li> <li>To effectively use these models as communication pinpoint in a team, a premise is that the team agrees to use a specific modeling standard and follows its conventions.</li> </ul>	<ul style="list-style-type: none"> <li>It enhances understanding and communication.</li> <li>It also provides a good basis for discussions and establishes an overview of what is actually employed.</li> </ul>

How would you assess the distinction into security views?	<ul style="list-style-type: none"> <li>In most cases, authentication and authorization must be considered together.</li> <li>On the other hand, it is meaningful to separate those since different concerns exist for these views which are addressed by different mechanisms.</li> <li>For technical reasons, it makes sense to split those up, e.g., for authorization, certificates don't matter since they are used for authentication.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, these views mostly cover the set of security modeling aspects.</li> </ul>	<ul style="list-style-type: none"> <li>Authentication and Secure Communication should be merged because authentication requires a secure communication path.</li> <li>There exist dependencies between views, i.e., certain views require modeling concepts of other views. It is difficult to separate those.</li> </ul>
Which security views need to be considered additionally?	<ul style="list-style-type: none"> <li>No, I can't think of any ad-hoc.</li> </ul>	<ul style="list-style-type: none"> <li>I can't think of any additional views.</li> </ul>	<ul style="list-style-type: none"> <li>An incident-response view, i.e., a plan on how to react to certain security events.</li> <li>Policy view against insider threats, i.e., how to deal with internal threats, e.g., a former employee that can publish confidential company data.</li> </ul>
Which concerns need to be addressed in this view?	-	-	<ul style="list-style-type: none"> <li>Policy view: who can access which internal information, e.g., code? Who has admin rights? What happens when an employee terminates? Employee awareness and training. Enforce that employees work at least together on projects to prevent a single employee includes backdoors.</li> <li>Adding to build and deployment view: how are pull requests and code reviews handled?</li> </ul>
Which modeling notations/languages would you suggest?	<ul style="list-style-type: none"> <li>I think the authentication models of the case study were very approachable.</li> <li>For other views, different notations are probably necessary.</li> </ul>	<ul style="list-style-type: none"> <li>UML because it is a standardized modeling language that every developer understands and knows.</li> <li>Also UML provides a large toolbox of modeling elements to express such things.</li> </ul>	<ul style="list-style-type: none"> <li>Adapted component diagrams should be sufficient.</li> </ul>
Are special-purpose modeling notations/languages helpful?	<ul style="list-style-type: none"> <li>It probably makes sense to have one modeling language for all views, if applicable.</li> <li>You would rarely only work on one view, therefore it helps when I don't need to rethink different notations.</li> <li>Additional models, e.g., the sequence diagram of the case study, help to understand the architecture from different angles.</li> <li>A combination of new special-purpose languages and UML diagrams would probably be successful.</li> </ul>	<ul style="list-style-type: none"> <li>Yes, but the modeling languages cannot become too complex. Otherwise, they are adopted by the development teams.</li> <li>Also, the team needs to agree and actually adopt the modeling language.</li> </ul>	<ul style="list-style-type: none"> <li>Yes. The modeling elements must be unique and clear though.</li> <li>Ambiguities as in Figure 6.1b may lead to more confusion and should be avoided.</li> </ul>

Can we study the influence of applied security design concepts?	<ul style="list-style-type: none"> <li>• Maybe. It is difficult to assess this ad-hoc.</li> </ul>	<ul style="list-style-type: none"> <li>• It helps to better document which security mechanisms are present in the whole system.</li> <li>• Using these design concepts enforces thinking about how the architecture should look and enhances argumentation and discussion on why certain design concepts are employed.</li> </ul>	<ul style="list-style-type: none"> <li>• This needs to be made explicit in the modeling, i.e., which security goals are addressed by the design concepts established.</li> <li>• When it is done implicitly, this is not clear.</li> <li>• An additional checklist of the security goals addressed might be helpful.</li> </ul>
Do such views capture the software engineering perspective?	<ul style="list-style-type: none"> <li>• From a software engineering perspective, I think so.</li> <li>• Software engineers mostly don't have in-depth security knowledge but are proficient with description languages. They see these models and know what is going on.</li> </ul>	<ul style="list-style-type: none"> <li>• Yes, this is the whole purpose of using these kinds of diagrams.</li> </ul>	<ul style="list-style-type: none"> <li>• Yes, for the authentication view it is achieved.</li> <li>• Also, views such as the build and deployment view cannot be modeled effectively using threat modeling for example, which is an offensive view.</li> </ul>
How would you assess the worth of this modeling approach?	<ul style="list-style-type: none"> <li>• It becomes more visible.</li> <li>• The different security goals are mapped to the concrete security views.</li> <li>• This way, it is easily clear how, e.g., authentication is designed in a certain architecture.</li> <li>• Every software engineer sees these models and can immediately understand them.</li> </ul>	<ul style="list-style-type: none"> <li>• When the system becomes larger, and different authentication methods are available, modeling this kind of concept improves the understanding of the architecture and can provide a good overview.</li> <li>• Otherwise, it can become too complex when only a textual representation is employed.</li> <li>• It is important to split up the model and only consider certain aspects per model, otherwise diagrams can be too complex to comprehend.</li> </ul>	<ul style="list-style-type: none"> <li>• It is useful to have these sets of concerns as a checklist of what needs to be considered for each view.</li> <li>• These models help to ensure that each concern is addressed and can serve as a discussion foundation.</li> </ul>





# Bibliography

- [AAE16] N. Alshuqayran, N. Ali, and R. Evans. “A systematic mapping study in microservice architecture.” In: *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016* (Dec. 2016). DOI: 10.1109/SOCA.2016.15 (cit. on p. 18).
- [AC22] M. G. de Almeida and E. D. Canedo. “Authentication and Authorization in Microservices Architecture: A Systematic Literature Review.” In: *Applied Sciences (Switzerland)* 12.6 (2022). ISSN: 20763417. DOI: 10.3390/app12063023. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85126960560&doi=10.3390%2Fapp12063023&partnerID=40&md5=5f95106a7ccbf0b16eb2fa26e68c0140> (cit. on pp. 48, 57, 58, 71, 72, 112, 114–117).
- [Alm+22] M. Almiani et al. “Resilient Back Propagation Neural Network Security Model For Containerized Cloud Computing.” In: *Simulation Modelling Practice and Theory* 118 (2022). ISSN: 1569190X. DOI: 10.1016/j.simpat.2022.102544. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85127476658&doi=10.1016%2Fj.simpat.2022.102544&partnerID=40&md5=28bbfb37ebe9afc5b1a8e4bee447a6dc> (cit. on pp. 61, 119).
- [Baa+20] A. F. Baarzi et al. “Microservices made attack-resilient using unsupervised service fissioning.” English. In: *Proceedings of the 13th European Workshop on Systems Security, EuroSec 2020*. EuroSec '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 31–36. ISBN: 9781450375238. DOI: 10.1145/3380786.3391395. URL: <https://doi.org/10.1145/3380786.3391395> (cit. on p. 118).
- [BAR21] B. Bhargava, P. Angin, and R. Ranchal. “Privacy-preserving data sharing and adaptable service compositions in mission-critical clouds.” In: *CEUR Workshop Proceedings*. Ed. by G. S. Jain S. Vol. 2786. CEUR-WS, 2021, pp. 60–66. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85102514645&partnerID=40&md5=848411fbf7f367dae9e7aec2a055878> (cit. on p. 118).
- [Bas17] L. Bass. “The Software Architect and DevOps.” In: *IEEE Software* 35.1 (Jan. 2017). ISSN: 07407459. DOI: 10.1109/MS.2017.4541051 (cit. on p. 60).

- [Bay+21] G. Baye et al. "API Security in Large Enterprises: Leveraging Machine Learning for Anomaly Detection." In: *2021 International Symposium on Networks, Computers and Communications, ISNCC 2021*. Oct. 2021, pp. 1–6. ISBN: 9780738113166. DOI: 10.1109/ISNCC52172.2021.9615638 (cit. on pp. 62, 119).
- [BBL76] B. W. Boehm, J. R. Brown, and M. Lipow. "Quantitative evaluation of software quality." In: *Proceedings of the 2nd international conference on Software engineering* (Oct. 1976). DOI: 10.5555/800253.807736 (cit. on p. 10).
- [BCK21] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. 4th ed. 2021. ISBN: 0-13-688609-4 (cit. on pp. 1, 43, 44, 46).
- [BDP06] M. Broy, F. Deissenboeck, and M. Pizka. "Demystifying Maintainability." In: *Proceedings of the 2006 International Workshop on Software Quality*. New York, New York, USA: Association for Computing Machinery, May 2006, pp. 21–26. ISBN: 1595933999. DOI: 10.1145/1137702 (cit. on p. 1).
- [Ber+22] D. Berardi et al. "Microservice security: a systematic literature review." In: *PeerJ Computer Science* 7 (Jan. 2022). ISSN: 23765992. DOI: 10.7717/PEERJ-CS.779/SUPP-2. URL: <https://peerj.com/articles/cs-779> (cit. on pp. 20, 32).
- [Bil+22] P. Billawa et al. "Security of Microservice Applications: A Practitioners' Perspective on Challenges and Best Practices." In: (Feb. 2022). DOI: 10.48550/arxiv.2202.01612. URL: <https://arxiv.org/abs/2202.01612v1> (cit. on pp. 2, 14, 19).
- [BM99] J. Bosch and P. Molin. "Software architecture design: Evaluation and transformation." In: *Proceedings - ECBS 1999, IEEE Conference and Workshop on Engineering of Computer-Based Systems* (1999), pp. 4–10. DOI: 10.1109/ECBS.1999.755855 (cit. on pp. 10, 11).
- [Bog+19] J. Bogner et al. "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality." In: *Proceedings - 2019 IEEE International Conference on Software Architecture - Companion, ICSA-C 2019* (May 2019). DOI: 10.1109/ICSA-C.2019.00041 (cit. on p. 11).
- [BWZ17] J. Bogner, S. Wagner, and A. Zimmermann. "Towards a Practical Maintainability Quality Model for Service- and Microservice-based Systems." In: *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings* (2017). DOI: 10.1145/3129790. URL: <https://doi.org/10.1145/3129790.3129816> (cit. on p. 10).
- [CH13] Y. Cherdantseva and J. Hilton. "Information Security and Information Assurance: Discussion about the Meaning, Scope, and Goals." In: *Organizational, Legal, and Technological Dimensions of Information System Administration*. IGI Global, Sept. 2013. Chap. 10, pp. 167–198. ISBN: 9781466645271. DOI: 10.4018/978-1-4666-4526-4.CH010 (cit. on p. 13).

- 
- [Cha+22] A. Chatterjee et al. “SFTSDH: Applying Spring Security Framework with TSD-Based OAuth2 to Protect Microservice Architecture APIs.” English. In: *IEEE Access* 10 (2022), pp. 41900–41920. ISSN: 21693536. DOI: 10.1109/ACCESS.2022.3165548 (cit. on pp. 56, 57).
- [CHC19] J. Chen, H. Huang, and H. Chen. “Informer: Irregular traffic detection for containerized microservices RPC in the real world.” English. In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019*. 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2019, pp. 389–394. ISBN: 9781450367332. DOI: 10.1145/3318216.3363375. URL: <https://www.sciencedirect.com/science/article/pii/S2667295222000022> (cit. on pp. 62, 118).
- [Che+19] W. Chen et al. “API security at a glance.” In: *IMCIC 2019 - 10th International Multi-Conference on Complexity, Informatics and Cybernetics, Proceedings*. Vol. 1. International Institute of Informatics and Systemics, IIIS, 2019, pp. 111–116. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85066017007&partnerID=40&md5=b2b4208ca79d9cda c8754f2ae14acf2e> (cit. on pp. 56, 62, 101).
- [Chi+01] S. D. Chi et al. “Network Security Modeling and Cyber Attack Simulation Methodology.” In: *Information Security and Privacy. ACISP 2001. Lecture Notes in Computer Science* 2119 (July 2001), pp. 320–333. ISSN: 16113349. DOI: 10.1007/3-540-47719-5\_26/COVER. URL: [https://link.springer.com/chapter/10.1007/3-540-47719-5\\_26](https://link.springer.com/chapter/10.1007/3-540-47719-5_26) (cit. on p. 54).
- [Chr77] A. Christopher. *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure)*. New York: Oxford University Press, 1977. Chap. Ch. 59. ISBN: 0195019199 (cit. on p. 45).
- [Cle+10] P. C. Clements et al. *Documenting Software Architectures: Views and Beyond*. 2nd ed. 6. Oct. 2010. ISBN: 0321552687 (cit. on pp. 7, 11).
- [CM78] J. P. Cavano and J. A. McCall. “A framework for the measurement of software quality.” In: *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues* (Jan. 1978). DOI: 10.1145/800283.811113 (cit. on pp. 10, 21).
- [CMJ15] B. Campbell, C. Mortimore, and M. Jones. *Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants*. RFC 7522. May 2015. DOI: 10.17487/RFC7522. URL: <https://www.rfc-editor.org/info/rfc7522> (cit. on p. 56).
- [Das+21] D. Das et al. “On automated RBAC assessment by constructing a centralized perspective for microservice mesh.” English. In: *PeerJ Computer Science* 7 (Feb. 2021), pp. 1–24. ISSN: 23765992. DOI: 10.7717/peerj-cs.376 (cit. on p. 96).

- [DD09] T. Dyba and T. Dingsoyr. “What do we know about agile software development?” In: *IEEE Software* 26.5 (Aug. 2009), pp. 6–9. ISSN: 07407459. DOI: 10.1109/MS.2009.145 (cit. on p. 1).
- [Dil+20] D. Dilshan et al. “MSChain: Blockchain based Decentralized Certificate Transparency for Microservices.” In: *MERCon 2020 - 6th International Multidisciplinary Moratuwa Engineering Research Conference, Proceedings*. July 2020, pp. 1–6. ISBN: 9781728190594. DOI: 10.1109/MERCon50084.2020.9185320 (cit. on p. 117).
- [DJS19] D. Deogun, D. B. Johnsson, and D. Sawano. *Secure By Design*. 2019. ISBN: 9781617294358 (cit. on p. 13).
- [Dra+16] N. Dragoni et al. “Microservices: yesterday, today, and tomorrow.” In: *Present and Ulterior Software Engineering* (June 2016). DOI: 10.48550/arxiv.1606.04036. arXiv: 1606.04036. URL: <https://arxiv.org/abs/1606.04036v4> (cit. on pp. 2, 11).
- [Fie+99] R. Fielding et al. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. June 1999. DOI: 10.17487/RFC2616. URL: <https://www.rfc-editor.org/info/rfc2616> (cit. on pp. 11, 56).
- [FK92] D. Ferraiolo and R. Kuhn. “Role-Based Access Controls.” In: *Proceedings of the 15th National Computer Security Conference* (Oct. 1992), pp. 554–563. URL: <https://csrc.nist.gov/publications/detail/conference-paper/1992/10/13/role-based-access-controls> (cit. on p. 57).
- [FKS17] D. Fett, R. Kusters, and G. Schmitz. “The Web SSO Standard OpenID Connect: In-depth Formal Security Analysis and Security Guidelines.” In: *Proceedings - IEEE Computer Security Foundations Symposium* (Sept. 2017). ISSN: 19401434. DOI: 10.1109/CSF.2017.20. arXiv: 1704.08539 (cit. on p. 56).
- [FL14] M. Fowler and J. Lewis. *Microservices*. 2014. URL: <https://martinfowler.com/articles/microservices.html> (visited on 10/19/2022) (cit. on pp. 2, 11, 12, 59).
- [FM10] R. Feldt and A. Magazinius. “Validity Threats in Empirical Software Engineering Research-An Initial Survey.” In: *Proceedings of the 22nd International Conference on Software Engineering & Knowledge Engineering (SEKE’2010)*. Redwood City, San Francisco Bay, CA, USA, Jan. 2010, pp. 374–379 (cit. on pp. 82, 84).
- [Fow06] M. Fowler. *Writing Software Patterns*. Aug. 2006. URL: <https://www.martinfowler.com/articles/writingPatterns.html> (visited on 08/04/2022) (cit. on p. 45).
- [Gam+94] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, Nov. 1994. ISBN: 0201633612 (cit. on pp. 24, 45).

- [GS19] C. Gerking and D. Schubert. “Component-based refinement and verification of information-flow security policies for cyber-physical microservice architectures.” English. In: *Proceedings - 2019 IEEE International Conference on Software Architecture, ICSA 2019*. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2019, pp. 61–70. ISBN: 9781728105284. DOI: 10.1109/ICSA.2019.00015 (cit. on p. 95).
- [GS94] D. Garlan and M. Shaw. “An Introduction to Software Architecture.” In: (1994) (cit. on pp. 7, 11).
- [Ham18] M. H. Hamilton. *Keynote: The Language as a Software Engineer*. May 2018. URL: <https://www.youtube.com/watch?v=ZbV0F0Uk5lU> (visited on 10/11/2022) (cit. on p. 1).
- [Ham19] M. Hamzehloui. “A Study on the Most Prominent Areas of Research in Microservices.” In: *International Journal of Machine Learning and Computing* 9.2 (Apr. 2019), pp. 242–247. DOI: 10.18178/ijmlc.2019.9.2.793. URL: <https://www.researchgate.net/publication/333033011> (cit. on p. 2).
- [Har12] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Oct. 2012. DOI: 10.17487/RFC6749. URL: <https://www.rfc-editor.org/info/rfc6749> (cit. on pp. 56, 123).
- [Hof15] T. Hoff. *Deep Lessons from Google and eBay on Building Ecosystems of Microservices*. Dec. 2015. URL: <http://highscalability.com/blog/2015/12/1/deep-lessons-from-google-and-ebay-on-building-ecosystems-of.html> (visited on 10/11/2022) (cit. on p. 1).
- [Hu+19] V. Hu et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Aug. 2019. DOI: 10.6028/NIST.SP.800-162. URL: <https://csrc.nist.gov/publications/detail/sp/800-162/final> (cit. on p. 57).
- [HY20] A. Hannousse and S. Yahiouche. “Securing Microservices and Microservice Architectures: A Systematic Mapping Study.” In: *Computer Science Review* 41 (Mar. 2020). DOI: 10.1016/j.cosrev.2021.100415. URL: <http://arxiv.org/abs/2003.07262> (cit. on pp. 14, 18).
- [IEC21] *Understanding IEC 62443 | IEC*. Feb. 2021. URL: <https://www.iec.ch/blog/understanding-iec-62443> (visited on 11/17/2022) (cit. on p. 72).
- [ISO11a] *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Tech. rep. International Organization for Standardization, Mar. 2011. URL: <https://www.iso.org/standard/35733.html> (cit. on pp. 2, 10, 13, 20, 21, 63).

- [ISO11b] *ISO - ISO/IEC/IEEE 42010:2011 - Systems and software engineering — Architecture description*. Tech. rep. International Organization for Standardization, Dec. 2011. URL: <https://www.iso.org/standard/50508.html> (cit. on pp. 7–9, 52, 80).
- [ISO13] *ISO/IEC 27001:2013 - Information technology — Security techniques — Information security management systems — Requirements*. Tech. rep. International Organization for Standardization, Oct. 2013. URL: <https://www.iso.org/standard/54534.html> (cit. on p. 72).
- [ISO19] *ISO - ISO/IEC 25020:2019 - Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality measurement framework*. Tech. rep. International Organization for Standardization, July 2019. URL: <https://www.iso.org/standard/72117.html> (cit. on pp. 10, 21).
- [ISO91] *ISO - ISO/IEC 9126:1991 - Software engineering — Product quality*. Tech. rep. International Organization for Standardization, Dec. 1991. URL: <https://www.iso.org/standard/16722.html> (cit. on p. 21).
- [Jac+22] S. Jacob et al. “Anomalous distributed traffic: Detecting cyber security attacks amongst microservices using graph convolutional networks.” In: *Computers and Security* 118 (2022), p. 102728. ISSN: 01674048. DOI: 10.1016/j.cose.2022.102728. URL: <https://www.sciencedirect.com/science/article/pii/S0167404822001237> (cit. on pp. 50, 61, 62, 99, 119).
- [JBP18] K. Jander, L. Braubach, and A. Pokahr. “Defense-in-depth and Role Authentication for Microservice Systems.” English. In: *Procedia Computer Science*. Ed. by E. Shakshuki and A. Yasar. Vol. 130. Procedia Computer Science. SARA BURGERHARTSTRAAT 25, PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS: ELSEVIER SCIENCE BV, 2018, pp. 456–463. DOI: 10.1016/j.procs.2018.04.047 (cit. on p. 114).
- [JBS15] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. RFC 7519. May 2015. DOI: 10.17487/RFC7519. URL: <https://www.rfc-editor.org/info/rfc7519> (cit. on p. 56).
- [JLE18] L. V. Jánoky, J. Levendovszky, and P. Ekler. “An analysis on the revoking mechanisms for JSON Web Tokens.” English. In: *International Journal of Distributed Sensor Networks* 14.9 (Sept. 2018). ISSN: 15501477. DOI: 10.1177/1550147718801535. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85054150323&doi=10.1177%2F1550147718801535&partnerID=40&md5=e1ce1659d7827487cdf00605b466a5ad> (cit. on pp. 45, 71, 104, 112).

- [JQL21] S. Jacob, Y. Qiao, and B. Lee. “Detecting cyber security attacks against a microservices application using distributed tracing.” English. In: *ICISSP 2021 - Proceedings of the 7th International Conference on Information Systems Security and Privacy*. Ed. by P. Mori, G. Lenzini, and S. Furnell. AV D MANUELL, 27A 2 ESQ, SETUBAL, 2910-595, PORTUGAL: SCITEPRESS, 2021, pp. 588–595. ISBN: 9789897584916. DOI: 10.5220/0010308905880595 (cit. on pp. 50, 61, 62, 99, 119).
- [Kal+20] D. Kallergis et al. “CAPODAZ: A containerised authorisation and policy-driven architecture using microservices.” English. In: *Ad Hoc Networks* 104 (July 2020). ISSN: 15708705. DOI: 10.1016/j.adhoc.2020.102153 (cit. on p. 113).
- [Kan03] K. Kandt. “Software design principles and practices.” In: (2003). URL: <https://trs.jpl.nasa.gov/bitstream/handle/2014/10550/02-2593.pdf?sequence=1> (cit. on p. 41).
- [KC07] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Software Engineering Group, School of Computer Science, Mathematics, Keele University, and Department of Computer Science, University of Durham, July 2007. DOI: 10.6028/NIST.SP.800-204 (cit. on pp. 4, 29–31, 37, 39, 40, 51, 80, 83, 85).
- [KG20] R. Kumar and R. Goyal. “Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC).” In: *Computers & Security* 97 (Oct. 2020). ISSN: 0167-4048. DOI: 10.1016/J.COSE.2020.101967 (cit. on p. 60).
- [KG99] L. Kohnfelder and P. Garg. *The Threats to our Products*. Tech. rep. Microsoft, 1999 (cit. on pp. 13, 21, 23, 54, 72).
- [Kim+21] G. Kim et al. *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. 2nd ed. IT Revolution Press, Dec. 2021. ISBN: 1950508404 (cit. on p. 2).
- [Kit+97] B. Kitchenham et al. “The SQUID approach to defining a quality model.” In: *Software Quality Journal* 6 (3 1997). ISSN: 15731367. DOI: 10.1023/A:1018516103435 (cit. on pp. 20, 21).
- [KK21] A. Kotov and J. Klein. *SEI Software Architecture Principles and Practices Overview Training*. Tech. rep. Pittsburgh, PA 15213: Software Engineering Institute, Carnegie Mellon University, Mar. 2021. URL: <https://apps.dtic.mil/sti/citations/AD1145705> (cit. on pp. 14, 15, 43).
- [KMM18] M. Kalske, N. Mäkitalo, and T. Mikkonen. “Challenges When Moving from Monolith to Microservice Architecture.” In: *Current Trends in Web Engineering*. Springer International Publishing, 2018. ISBN: 9783319744322. DOI: 10.1007/978-3-319-74433-9\_3/FIGURES/6. URL: [https://link.springer.com/chapter/10.1007/978-3-319-74433-9\\_3](https://link.springer.com/chapter/10.1007/978-3-319-74433-9_3) (cit. on p. 2).

- [Kru95] P. Kruchten. “Architectural Blueprints-The ”4+1” View Model of Software Architecture.” In: *IEEE Software* 12.6 (1995). DOI: 10.1109/52.469759 (cit. on p. 1).
- [Li+21] S. Li et al. “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review.” In: *Information and Software Technology* 131 (Mar. 2021). ISSN: 0950-5849. DOI: 10.1016/J.INFSOF.2020.106449 (cit. on pp. 2, 19).
- [Liu+20] A. Liu et al. “A MRP-based policy conflict resolution mechanism for micro-service composition.” In: *Proceedings - 2020 7th International Conference on Information Science and Control Engineering, ICISCE 2020*. Dec. 2020, pp. 1556–1564. ISBN: 9781728164069. DOI: 10.1109/ICISCE50968.2020.00309 (cit. on p. 94).
- [LL13] J. Ludewig and H. Lichter. *Software Engineering : Grundlagen, Menschen, Prozesse, Techniken*. 3rd ed. Dpunkt.verlag GmbH, Apr. 2013. ISBN: 3864900921 (cit. on p. 54).
- [Lu+17] D. Lu et al. “A Secure Microservice Framework for IoT.” English. In: *Proceedings - 11th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2017*. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2017, pp. 9–18. ISBN: 9781509063208. DOI: 10.1109/SOSE.2017.27 (cit. on p. 112).
- [MA19] G. Márquez and H. Astudillo. “Identifying availability tactics to support security architectural design of microservice-based systems.” English. In: *ACM International Conference Proceeding Series*. Ed. by L. Duchien et al. Vol. 2. ECSA ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 123–131. ISBN: 9781450371421. DOI: 10.1145/3344948.3344996. URL: <https://doi.org/10.1145/3344948.3344996> (cit. on pp. 87, 101–104).
- [Mar08] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st ed. 6. Prentice Hall, Aug. 2008. ISBN: 9780132350884 (cit. on p. 41).
- [MCF21a] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira. “Security in Microservices Architectures.” In: *Procedia Computer Science* 181 (Jan. 2021). ISSN: 1877-0509. DOI: 10.1016/J.PROCS.2021.01.320 (cit. on p. 14).
- [MCF21b] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira. “Security in microservices architectures.” English. In: *Procedia Computer Science*. Ed. by M. M. CruzCunha et al. Vol. 181. Procedia Computer Science. SARA BURGERHARTSTRAAT 25, PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS: ELSEVIER SCIENCE BV, 2021, pp. 1225–1236. DOI: 10.1016/j.procs.2021.01.320 (cit. on pp. 56, 93).
- [McG06] G. McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, Jan. 2006. ISBN: 9780321356703 (cit. on p. 12).



- [Mel21] R. Melton. “Securing a Cloud-Native C2 Architecture Using SSO and JWT.” In: *IEEE Aerospace Conference Proceedings*. Vol. 2021-March. Mar. 2021, pp. 1–8. ISBN: 9781728174365. DOI: 10.1109/AERO50100.2021.9438218 (cit. on pp. 42, 56, 58, 70, 71, 93, 102).
- [Mie+10] A. Miede et al. “A generic metamodel for IT security - Attack modeling for distributed systems.” In: *ARES 2010 - 5th International Conference on Availability, Reliability, and Security* (2010). DOI: 10.1109/ARES.2010.17 (cit. on pp. 12, 52).
- [MMR14] J. P. Miguel, D. Mauricio, and G. Rodríguez. “A Review of Software Quality Models for the Evaluation of Software Products.” In: *International Journal of Software Engineering & Applications (IJSEA)* 5 (6 2014). DOI: 10.5121/ijsea.2014.5603 (cit. on pp. 10, 20).
- [MR22] N. Mohammadi and A. Rasoolzadegan. “A Pattern-aware Design and Implementation Guideline for Microservice-based Systems.” In: *2022 27th International Computer Conference, Computer Society of Iran (CSICC)*. Feb. 2022, pp. 1–6. DOI: 10.1109/csicc55295.2022.9780516 (cit. on pp. 108, 112).
- [MU15] M. M. Morana and T. Ucedavelez. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. Wiley, May 2015, p. 696. ISBN: 978-0-470-50096-5. URL: <https://www.wiley.com/en-us/Risk+Centric+Threat+Modeling%3A+Process+for+Attack+Simulation+and+Threat+Analysis-p-9780470500965> (cit. on pp. 13, 54, 72).
- [Müs+17] D. Müssig et al. “Highly scalable microservice-based enterprise architecture for smart ecosystems in hybrid cloud environments.” English. In: *ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems*. Ed. by S. Hammoudi et al. Vol. 3. AV D MANUELL, 27A 2 ESQ, SETUBAL, 2910-595, PORTUGAL: SciTePress, 2017, pp. 454–459. ISBN: 9789897582493. DOI: 10.5220/0006373304540459. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85023201542&doi=10.5220%2F0006373304540459&partnerID=40&md5=1300da5e850a197ac1fbb22d670561dd> (cit. on pp. 94, 107, 112).
- [NC19a] P. Nkomo and M. Coetzee. “Development Activities, Tools and Techniques of Secure Microservices Compositions.” English. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by S. H. Heng and J. Lopez. Vol. 11879 LNCS. Lecture Notes in Computer Science. GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND: SPRINGER INTERNATIONAL PUBLISHING AG, 2019, pp. 423–433. ISBN: 9783030343385. DOI: 10.1007/978-3-030-34339-2\_24 (cit. on pp. 91–93, 95).

- [NC19b] P. Nkomo and M. Coetzee. “Software Development Activities for Secure Microservices.” English. In: *Computational Science and Its Applications – ICCSA 2019*. Ed. by S. Misra et al. Vol. 11623 LNCS. Lecture Notes in Computer Science. GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND: Springer International Publishing, 2019, pp. 573–585. ISBN: 9783030243074. DOI: 10.1007/978-3-030-24308-1\_46 (cit. on pp. 56, 62, 71, 94, 97, 101, 102).
- [Neh+19] A. Nehme et al. “Securing Microservices.” English. In: *IT Professional* 21.1 (Jan. 2019), pp. 42–49. ISSN: 1941045X. DOI: 10.1109/MITP.2018.2876987 (cit. on p. 112).
- [New15] S. Newman. *Building microservices: designing fine-grained systems*. 1st ed. O’Reilly Media, 2015. ISBN: 1491950358 (cit. on pp. 11, 12).
- [Ngu20] C. D. Nguyen. *A Design Analysis of Cloud-based Microservices Architecture at Netflix*. May 2020. URL: <https://medium.com/swlh/a-design-analysis-of-cloud-based-microservices-architecture-at-netflix-98836b2da45f> (visited on 10/11/2022) (cit. on p. 1).
- [NL21] S. Nadgowda and L. Luan. “Tapiseri: Blueprint to modernize DevSecOps for real world.” In: *WoC 2021 - Proceedings of the 2021 7th International Workshop on Container Technologies and Container Clouds*. Association for Computing Machinery, Inc, 2021, pp. 13–18. ISBN: 9781450391719. DOI: 10.1145/3493649.3493655. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85121746017&doi=10.1145%2F3493649.3493655&partnerID=40&md5=cc09617d696d6146c374da9d785be48c> (cit. on pp. 43, 95).
- [OMB07] L. O’Brien, P. Merson, and L. Bass. “Quality attributes for service-oriented architectures.” In: *Proceedings - ICSE 2007 Workshops: International Workshop on Systems Development in SOA Environments, SDSOA’07* (2007), pp. 3–9. DOI: 10.1109/SDSOA.2007.10 (cit. on p. 10).
- [OMG17] *Unified Modeling Language*. Tech. rep. Object Management Group, Dec. 2017 (cit. on pp. 9, 54, 77, 82).
- [OMG19] *OMG Systems Modeling Language (OMG SysML™)*. Tech. rep. Object Management Group, Nov. 2019. URL: <https://www.omg.org/spec/SysML/1.6/> (cit. on p. 72).
- [Osm+19] A. Osman et al. “Sandnet: Towards High Quality of Deception in Container-Based Microservice Architectures.” In: *IEEE International Conference on Communications*. Vol. 2019-May. May 2019, pp. 1–7. ISBN: 9781538680889. DOI: 10.1109/ICC.2019.8761171 (cit. on pp. 62, 98).
- [OWASP19] *Application Security Verification Standard 4.0*. Tech. rep. OWASP The Open Web Application Security Project, Mar. 2019 (cit. on p. 75).

- [OWASP20] OWASP SAMM v2.0. Tech. rep. OWASP The Open Web Application Security Project, 2020. URL: <https://owaspsamm.org/model/> (cit. on pp. 21, 22).
- [OY17] C. Otterstad and T. Yarygina. “Low-level exploitation mitigation by diverse microservices.” English. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by F. DePaoli, S. Schulte, and E. B. Johnsen. Vol. 10465 LNCS. Lecture Notes in Computer Science. GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND: SPRINGER INTERNATIONAL PUBLISHING AG, 2017, pp. 49–56. ISBN: 9783319672618. DOI: 10.1007/978-3-319-67262-5\_4 (cit. on pp. 62, 92, 93).
- [PD18] M. O. Pahl and L. Donini. “Securing IoT microservices with certificates.” English. In: *IEEE/IFIP Network Operations and Management Symposium: Cognitive Management in a Cyber World, NOMS 2018*. IEEE IFIP Network Operations and Management Symposium. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2018, pp. 1–5. ISBN: 9781538634165. DOI: 10.1109/NOMS.2018.8406189 (cit. on p. 99).
- [PD19] M. O. Pahl and L. Donini. “Giving IoT services an identity and changeable attributes.” English. In: *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2019, pp. 455–461. ISBN: 9783903176157 (cit. on p. 99).
- [Per+19] A. Pereira-Vale et al. “Security mechanisms used in microservices-based systems: A systematic mapping.” In: *Proceedings - 2019 45th Latin American Computing Conference, CLEI 2019*. 2019. ISBN: 9781728155746. DOI: 10.1109/CLEI47609.2019.235060 (cit. on pp. 14, 19, 31).
- [Per+21] A. Pereira-Vale et al. “Security in microservice-based systems: A Multivocal literature review.” In: *Computers and Security* 103 (Apr. 2021). ISSN: 01674048. DOI: 10.1016/J.COSE.2021.102200 (cit. on pp. 14, 19).
- [PJ16] C. Pahl and P. Jamshidi. “Microservices: A systematic mapping study.” In: *Proceedings of the 6th International Conference on Cloud Computing and Services Science*. Vol. 1. SciTePress, Apr. 2016. ISBN: 9789897581823. DOI: 10.5220/0005785501370146 (cit. on p. 18).
- [PJ19] D. Preuveneers and W. Joosen. “Towards multi-party policy-based access control in federations of cloud and edge microservices.” English. In: *Proceedings - 4th IEEE European Symposium on Security and Privacy Workshops, EUROS and PW 2019*. 345 E 47TH ST, NEW YORK, NY 10017 USA: IEEE, 2019, pp. 29–38. ISBN: 9781728130262. DOI: 10.1109/EuroSPW.2019.00010 (cit. on pp. 57, 113).

- [PL21] C. Pasomsup and Y. Limpiyakorn. “HT-RBAC: A Design of Role-based Access Control Model for Microservice Security Manager.” In: *Proceedings - 2021 International Conference on Big Data Engineering and Education, BDEE 2021*. Aug. 2021, pp. 177–181. ISBN: 9781665439572. DOI: 10.1109/BDEE52938.2021.00038 (cit. on p. 110).
- [Pon+21] R. P. Pontarolli et al. “Towards security mechanisms for an industrial microservice-oriented architecture.” In: *2021 14th IEEE International Conference on Industry Applications, INDUSCON 2021 - Proceedings*. Aug. 2021, pp. 679–685. ISBN: 9781665441186. DOI: 10.1109/INDUSCON51756.2021.9529415 (cit. on pp. 105, 106, 112).
- [PS19] T. Pradeep Pai and K. L. Shashikala. “Smart City Services - Challenges and Approach.” In: *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Perspectives and Prospects, COMITCon 2019*. Institute of Electrical and Electronics Engineers Inc., Feb. 2019, pp. 553–558. ISBN: 9781728102115. DOI: 10.1109/COMITCon.2019.8862243. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85074132448&doi=10.1109%2FCOMITCon.2019.8862243&partnerID=40&md5=46a2e29e8ce7f3b9b9035f554ddb8e97> (cit. on pp. 45, 71, 104).
- [PW92] D. E. Perry and A. L. Wolf. “Foundations for the study of software architecture.” In: *ACM SIGSOFT Software Engineering Notes* 17.4 (Oct. 1992). ISSN: 0163-5948. DOI: 10.1145/141874.141884. URL: <https://dl.acm.org/doi/abs/10.1145/141874.141884> (cit. on pp. 1, 7).
- [Rah+21] A. Rahman et al. “Security Smells in Ansible and Chef Scripts.” In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30.1 (Jan. 2021), pp. 1–31. ISSN: 15577392. DOI: 10.1145/3408897. arXiv: 1907.07159. URL: <https://dl.acm.org/doi/10.1145/3408897> (cit. on p. 61).
- [RD08] E. Rescorla and T. Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: 10.17487/RFC5246. URL: <https://www.rfc-editor.org/info/rfc5246> (cit. on pp. 56, 71, 72, 124).
- [Ric19] C. Richardson. *Microservices Patterns*. 2019. ISBN: 9781617294549. URL: <https://learning.oreilly.com/library/view/microservices-patterns/9781617294549/> (cit. on p. 59).
- [RM16] C. Robson and K. McCartan. *Real World Research: A Resource for Users of Social Research Methods in Applied Settings*. 4th ed. Wiley, Jan. 2016. ISBN: 978-1118745236 (cit. on p. 65).
- [RMO18] R. Ross, M. McEvilly, and J. C. Oren. *Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Mar. 2018. DOI: 10.6028/NIST.SP.

- 800-160V1. URL: <https://csrc.nist.gov/publications/detail/sp/800-160/vol-1/final> (cit. on p. 63).
- [RS16] C. Richardson and F. Smith. *Microservices: From Design to Deployment*. 2016 (cit. on p. 11).
- [SB15] W. Stallings and L. Brown. *Computer Security Principles and Practice*. 3rd ed. Pearson, 2015. ISBN: 9780133773927 (cit. on pp. 10, 13, 56).
- [SC06] M. Shaw and P. Clements. “The golden age of software architecture.” In: *IEEE Software* 23.2 (Mar. 2006), pp. 31–39. ISSN: 07407459. DOI: 10.1109/MS.2006.58 (cit. on p. 1).
- [Sch06] R. A. Schrenker. “Software engineering for future healthcare and clinical systems.” In: *Computer* 39.4 (Apr. 2006), pp. 26–32. ISSN: 00189162. DOI: 10.1109/MC.2006.139 (cit. on p. 1).
- [Sch99] B. Schneier. *Attack Trees - Schneier on Security*. 1999. URL: [https://www.schneier.com/academic/archives/1999/12/attack\\_trees.html](https://www.schneier.com/academic/archives/1999/12/attack_trees.html) (visited on 08/31/2022) (cit. on pp. 13, 21, 24).
- [SEI10] *What is your definition of software architecture?* Tech. rep. Software Engineering Institute, Carnegie Mellon University, Dec. 2010 (cit. on p. 7).
- [SF17] M. H. Syed and E. B. Fernandez. “The container manager pattern.” English. In: *ACM International Conference Proceeding Series*. Vol. Part F1320. 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2017. ISBN: 9781450348485. DOI: 10.1145/3147704.3147735 (cit. on p. 109).
- [Shi07] R. W. Shirey. *Internet Security Glossary, Version 2*. RFC 4949. Aug. 2007. DOI: 10.17487/RFC4949. URL: <https://www.rfc-editor.org/info/rfc4949> (cit. on pp. 2, 12, 13, 38, 55, 57, 61).
- [Sho14] A. Shostack. *Threat Modeling: Designing for Security*. 1st ed. May. Wiley, Feb. 2014. ISBN: 1118809998 (cit. on pp. 2, 13, 54, 70).
- [Sim05] K. D. Simon. “The value of open standards and open-source software in government environments.” In: *IBM Systems Journal* 44.2 (2005), pp. 227–238. ISSN: 00188670. DOI: 10.1147/SJ.442.0227 (cit. on p. 1).
- [SKI19] S. Suneja, A. Kanso, and C. Isci. “Can container fusion be securely achieved?” In: *WOC 2019 - Proceedings of the 2019 5th International Workshop on Container Technologies and Container Clouds, Part of Middleware 2019*. WOC '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 31–36. ISBN: 9781450370332. DOI: 10.1145/3366615.3368356. URL: <https://doi.org/10.1145/3366615.3368356> (cit. on p. 108).

- [SP19] D. M. Stallenberg and A. Panichella. “JCOMIX: A search-based tool to detect XML injection vulnerabilities in web applications.” English. In: *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Ed. by M. Dumas et al. 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2019, pp. 1090–1094. ISBN: 9781450355728. DOI: 10.1145/3338906.3341178 (cit. on p. 118).
- [Sta10] W. Stallings. *Cryptography and Network Security: Principles and Practice*. 5th ed. Pearson, Jan. 2010. ISBN: 978-0136097044 (cit. on pp. 2, 10).
- [STH18] J. Soldani, D. A. Tamburri, and W. J. V. D. Heuvel. “The pains and gains of microservices: A Systematic grey literature review.” In: *Journal of Systems and Software* 146 (Dec. 2018). ISSN: 0164-1212. DOI: 10.1016/J.JSS.2018.09.082 (cit. on pp. 2, 11, 12, 18).
- [STM17] J. C. Santos, K. Tarrit, and M. Mirakhorli. “A catalog of security architecture weaknesses.” In: *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings* (June 2017). DOI: 10.1109/ICSAW.2017.25 (cit. on p. 24).
- [Sto+18] M. Stocker et al. “Interface quality patterns - Communicating and improving the quality of microservices APIs.” English. In: *ACM International Conference Proceeding Series*. 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2018. ISBN: 9781450363877. DOI: 10.1145/3282308.3282319 (cit. on pp. 71, 106, 112).
- [SW13] R. Sinnema and E. Wilde. *eXtensible Access Control Markup Language (XACML) XML Media Type*. RFC 7061. Nov. 2013. DOI: 10.17487/RFC7061. URL: <https://www.rfc-editor.org/info/rfc7061> (cit. on p. 57).
- [SWC10] D. Samadhiya, S. H. Wang, and D. Chen. “Quality models: Role and value in software engineering.” In: *ICSTE 2010 - 2010 2nd International Conference on Software Technology and Engineering, Proceedings* 1 (2010). DOI: 10.1109/ICSTE.2010.5608852 (cit. on p. 10).
- [Szo+20] D. Szopinski et al. “Software tools for business model innovation: current state and future challenges.” In: *Electronic Markets* 30.3 (Sept. 2020), pp. 469–494. ISSN: 14228890. DOI: 10.1007/S12525-018-0326-1/TABLES/7. URL: <https://link.springer.com/article/10.1007/s12525-018-0326-1> (cit. on p. 1).
- [SZS21] W. S. Shameem Ahamed, P. Zavorsky, and B. Swar. “Security Audit of Docker Container Images in Cloud Architecture.” In: *ICSCCC 2021 - International Conference on Secure Cyber Computing and Communi-*

- ations. May 2021, pp. 202–207. ISBN: 9781665444156. DOI: 10.1109/ICSCC51823.2021.9478100 (cit. on p. 97).
- [Thr+21] S. Throner et al. “An Advanced DevOps Environment for Microservice-based Applications.” In: *Proceedings - 15th IEEE International Conference on Service-Oriented System Engineering, SOSE 2021*. Aug. 2021, pp. 134–143. ISBN: 9781665434775. DOI: 10.1109/SOSE52839.2021.00020 (cit. on pp. 43, 60, 94, 95).
- [Tor+18] K. A. Torkura et al. “CAVAS: Neutralizing application and container security vulnerabilities in the cloud native era.” English. In: *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*. Ed. by R. Beyah et al. Vol. 254. Lecture Notes of the Institute for Computer Sciences Social Informatics and Telecommunications Engineering. GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND: SPRINGER INTERNATIONAL PUBLISHING AG, 2018, pp. 471–490. ISBN: 9783030017002. DOI: 10.1007/978-3-030-01701-9\_26 (cit. on pp. 60, 96, 107).
- [Tor+19] K. A. Torkura et al. “A cyber risk based moving target defense mechanism for microservice architectures.” English. In: *Proceedings - 16th IEEE International Symposium on Parallel and Distributed Processing with Applications, 17th IEEE International Conference on Ubiquitous Computing and Communications, 8th IEEE International Conference on Big Data and Cloud Computing, 11th*. Ed. by J. J. Chen and L. T. Yang. IEEE International Symposium on Parallel and Distributed Processing with Applications. 10662 LOS VAQUEROS CIRCLE, PO BOX 3014, LOS ALAMITOS, CA 90720-1264 USA: Institute of Electrical and Electronics Engineers Inc., 2019, pp. 932–939. ISBN: 9781728111414. DOI: 10.1109/BDCLOUD.2018.00137. URL: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063900937&doi=10.1109%2FBDCLOUD.2018.00137&partnerID=40&md5=afbbe1214a8795ace12d84163534431b> (cit. on pp. 60, 62, 96).
- [TSM17] K. A. Torkura, M. I. Sukmana, and C. Meinel. “Integrating continuous security assessments in microservices and cloud native applications.” English. In: *UCC 2017 - Proceedings of the 10th International Conference on Utility and Cloud Computing*. 1515 BROADWAY, NEW YORK, NY 10036-9998 USA: ASSOC COMPUTING MACHINERY, 2017, pp. 171–180. ISBN: 9781450351492. DOI: 10.1145/3147213.3147229 (cit. on pp. 47, 60, 96, 107, 111).
- [UF14] A. V. Uzunov and E. B. Fernandez. “An extensible pattern-based library and taxonomy of security threats for distributed systems.” In: *Computer Standards & Interfaces* 36.4 (June 2014). ISSN: 0920-5489. DOI: 10.1016/J.CSI.2013.12.008 (cit. on p. 52).

- [Vin06] S. Vinoski. “Advanced Message Queuing Protocol.” In: *IEEE Internet Computing* 10.6 (Nov. 2006), pp. 87–89. ISSN: 10897801. DOI: 10.1109/MIC.2006.116 (cit. on p. 56).
- [Vys12] M. Vysoký. “Diagram of Security.” In: *Information Sciences and Technologies Bulletin of the ACM Slovakia* 4.1 (2012), pp. 39–42 (cit. on p. 54).
- [Wag+12] S. Wagner et al. “The Quamoco product quality modelling and assessment approach.” In: *Proceedings - International Conference on Software Engineering* (2012). ISSN: 02705257. DOI: 10.1109/ICSE.2012.6227106. arXiv: 1611.04433 (cit. on pp. 20, 22).
- [Wag+16] S. Wagner et al. “Operationalised product quality models and assessment: The Quamoco approach.” In: *Information and Software Technology* 62 (1 Nov. 2016). DOI: 10.1016/j.infsof.2015.02.009. URL: <http://arxiv.org/abs/1611.09230><http://dx.doi.org/10.1016/j.infsof.2015.02.009> (cit. on pp. 20, 22).
- [Was+21a] M. Waseem et al. “Design, monitoring, and testing of microservices systems: The practitioners’ perspective.” In: *Journal of Systems and Software* 182 (Dec. 2021). ISSN: 0164-1212. DOI: 10.1016/J.JSS.2021.111061. arXiv: 2108.03384 (cit. on p. 14).
- [Was+21b] M. Waseem et al. “Design, monitoring, and testing of microservices systems: The practitioners’ perspective.” English. In: *Journal of Systems and Software* 182 (Dec. 2021). ISSN: 01641212. DOI: 10.1016/j.jss.2021.111061. arXiv: 2108.03384 (cit. on pp. 61, 71, 112).
- [WM11] M. E. Whitman and H. J. Mattord. *Principles of Information Security*. 4th ed. Jan. 2011. ISBN: 978-1-111-13821-9 (cit. on pp. 10, 12).
- [Woh+12] C. Wohlin et al. *Experimentation in Software Engineering*. 1st ed. Springer Berlin, Heidelberg, June 2012. ISBN: 978-3-642-29043-5. DOI: 10.1007/978-3-642-29044-2/COVER. URL: <https://link.springer.com/book/10.1007/978-3-642-29044-2> (cit. on p. 82).
- [Woh14] C. Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering.” In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, May 2014. ISBN: 9781450324762. DOI: 10.1145/2601248.2601268 (cit. on p. 39).
- [Wuy+18] K. Wuyts et al. “Effective and Efficient Privacy Threat Modeling through Domain Refinements.” In: SAC ’18. Pau, France: Association for Computing Machinery, Apr. 2018, pp. 1175–1178. ISBN: 9781450351911. DOI: 10.1145/3167132.3167414 (cit. on p. 20).
- [wwwa] CAPEC - Common Attack Pattern Enumeration and Classification. URL: <https://capec.mitre.org/index.html> (visited on 08/31/2022) (cit. on pp. 24, 30).



- 
- [wwwb] *CAPEC - New to CAPEC?* URL: [https://capec.mitre.org/about/new\\_to\\_capec.html](https://capec.mitre.org/about/new_to_capec.html) (visited on 08/31/2022) (cit. on p. 24).
- [wwwc] *CVE - Common Vulnerability Enumeration.* URL: <https://cve.mitre.org/index.html> (visited on 08/31/2022) (cit. on pp. 25, 43).
- [wwwd] *CWE - Common Weakness Enumeration.* URL: <https://cwe.mitre.org/index.html> (visited on 08/31/2022) (cit. on pp. 24, 30).
- [www e] *Microsoft Threat Modeling Tool overview - Azure | Microsoft Docs.* URL: <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool> (visited on 08/31/2022) (cit. on p. 24).
- [wwwf] *MITRE ATT&CK®.* URL: <https://attack.mitre.org/> (visited on 08/31/2022) (cit. on p. 25).
- [wwwg] *OWASP Threat Dragon | OWASP Foundation.* URL: <https://owasp.org/www-project-threat-dragon/> (visited on 08/31/2022) (cit. on p. 24).
- [www h] *OWASP Top Ten | OWASP Foundation.* URL: <https://owasp.org/www-project-top-ten/> (visited on 08/31/2022) (cit. on pp. 25, 72).
- [Yan+22] K. Yan et al. “Design and Application of Security Gateway for Transmission Line Panoramic Monitoring Platform based on Microservice Architecture.” In: *IEEE 6th Information Technology and Mechatronics Engineering Conference, ITOEC 2022*. Vol. 6. Mar. 2022, pp. 714–721. ISBN: 9781665431859. DOI: 10.1109/ITOEC53115.2022.9734463 (cit. on pp. 47, 111).
- [YB18a] T. Yarygina and A. H. Bagge. “Overcoming Security Challenges in Microservice Architectures.” In: *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018* (May 2018). DOI: 10.1109/SOSE.2018.00011 (cit. on p. 14).
- [YB18b] T. Yarygina and A. H. Bagge. “Overcoming Security Challenges in Microservice Architectures.” In: *Proceedings - 12th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2018 and 9th International Workshop on Joint Cloud Computing, JCC 2018*. Mar. 2018, pp. 11–20. ISBN: 9781538652060. DOI: 10.1109/SOSE.2018.00011 (cit. on pp. 58, 71, 72, 92, 93, 104, 116).
- [YO18] T. Yarygina and C. Otterstad. “A game of microservices: Automated intrusion response.” English. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Ed. by S. Bonomi and E. Riviere. Vol. 10853 LNCS. Lecture Notes in Computer Science. GEWERBESTRASSE 11, CHAM, CH-6330, SWITZERLAND: SPRINGER INTERNATIONAL PUBLISHING AG, 2018, pp. 169–177. ISBN: 9783319937663. DOI: 10.1007/978-3-319-93767-0\_12 (cit. on pp. 62, 119).

- [YZ10] S. Yu and S. Zhou. “A survey on metric of software complexity.” In: *2nd IEEE International Conference on Information Management and Engineering*. Vol. 2. 2010, pp. 352–356. ISBN: 9781424452644. DOI: 10.1109/ICIME.2010.5477581 (cit. on p. 1).
- [Zdu+22] U. Zdun et al. “Microservice Security Metrics for Secure Communication, Identity Management, and Observability.” In: *ACM Transactions on Software Engineering and Methodology* (Apr. 2022). ISSN: 1049-331X. DOI: 10.1145/3532183. URL: <https://doi.org/10.1145/3532183> (cit. on pp. 45, 58, 71, 98–100, 102–104).

# Acronyms

**ABAC** Attribute-based Access Control

**API** Application Programming Interface

**ATT&CK** Adversarial Tactics, Techniques & Common Knowledge

**CAPEC** Common Attack Pattern Enumeration and Classification

**CI/CD** Continuous Integration/Continuous Delivery

**CVE** Common Vulnerabilities and Exposures

**CWE** Common Weakness Enumeration

**DAST** Dynamic Application Security Testing

**DFD** Data Flow Diagram

**GoF** Gang of Four

**IaaS** Infrastructure-as-a-Service

**IaC** Infrastructure-as-Code

**IDS/IPS** Intrusion Detection System/Intrusion Prevention System

**MSA** Microservice Architecture

**OWASP SAMM** OWASP Software Assurance Maturity Model

**QA** Quality Attribute

**RBAC** Role-based Access Control

**SAML** Security Assertion Markup Language

**SAST** Static Application Security Testing

**SBOM** Software Bill of Materials

**SCA** Software Component Analysis

**SDLC** Software Development Life-cycle

**SLR** Systematic Literature Review

**SMS** Systematic Mapping Study

**XSS** Cross-Site Scripting