St. Pölten University of Applied Sciences

/informatik & security

/fh/// st.pölten

A Systematic Literature Review of Inter-Service Security Threats and Mitigation Strategies in Microservice Architectures

Master thesis

For attainment of the academic degree of

Master of Science in Engineering (MSc)

submitted by

Markus Sveggen

52106213

in the

University Course Cyber Security and Resilience at St. Pölten University of Applied Sciences

Supervision

Advisor: Dr. Philipp Haindl, MSc

St. Pölten, January 31, 2024 _____ _____

(Signature author) (Signature advisor)

St. Pölten University of Applied Sciences, Campus-Platz 1, A-3100 St. Pölten,T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Declaration

I hereby affirm that

- I have written this thesis independently, that I have not used any sources or aids other than those indicated, and that I have not made use of any unauthorised assistance.
- I have not previously submitted this thesis topic to an assessor, either in Austria or abroad, for evaluation or as an examination paper in any form.
- this thesis corresponds to the thesis assessed by the assessor.

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Date *Signature*

Abstract

Prioritizing security is essential in microservice architectures, given its distributed composition involving numerous services and network spanning interactions. This architectural style, which can include hundreds to thousands of services, inherently presents an expanded attack surface compared to traditional monolithic applications. Furthermore, the polyglot nature of microservices, which encompasses services developed and deployed using diverse programming languages and technologies, further complicates the security landscape.

This thesis presents an extensive systematic literature review, analyzing 52 publications specifically in the context of security threats and mitigation strategies within the area of inter-service security in microservice architectures. This focus was deliberately chosen due to the distinct nature of inter-service security issues as compared to those encountered in monolithic applications. The primary contribution of this thesis is a detailed conceptualization of identified security threats and mitigation strategies, organized into categories derived directly from the reviewed publications. In addition, an evaluation of the publications has been carried out based on the level of focus with which various topics have been discussed.

Preface

This thesis was inspired by my growing interest in microservices, which was sparked by the podcast episode "Tech Talk: Domain Driven Design and Microservices" with Adam Gordson Bell and his guest Jan Machacek [1] on the podcast "CoRecursive: Coding Stories". This podcast episode delved into microservice architecture, a topic I previously knew little about. Their discussion, which focused on modularity in microservice architecture, along with the ability to facilitate extensions and promote autonomy among development teams, deeply resonated with my passion for agile software development, software architecture, and design patterns.

The attractive concept of microservice architecture led me to contemplate its security implementation. The newly emerging security concerns, particularly regarding inter-service interactions, raised questions about potential complexities in managing security. Therefore, this thesis is an exploration of the security challenges inherent in microservice architecture and an examination of the efforts made to mitigate these potential security threats.

I would like to express my sincere gratitude to my thesis supervisor, Dr. Philipp Haindl, MSc for discussing ideas with me, guiding me through the field of research, and being a great help throughout the work with the thesis. His expertise has been instrumental in my research journey. I would also like to thank my partner, family, and friends for supporting me during this intensive period of academic endeavor.

Contents

1. Introduction

In this section, we introduce the topic and problems addressed in the thesis, detail our research questions, and explain the motivation behind conducting this study. In addition, we provide an outline of the thesis structure.

1.1. Topic

This thesis aims to investigate the security threats prevalent in microservices systems, as well as the mitigation strategies that can be employed to enhance the security posture of the system. The scope has been defined to focus on *inter-service security* in microservice architectures. We define inter-service security as the ensemble of strategies, excluding edge-level security mechanisms, dedicated to safeguarding the connections and interactions between microservices. We describe inter-service security threats as those specifically targeting the interconnections and interactions within the microservices architecture. We have decided to emphasize inter-service security in microservices, recognizing its distinctive aspects compared to traditional monolithic environments. The aim of this work is to heighten awareness among those transitioning to microservices about the unique challenges and issues related to inter-service security.

1.2. Problem

Transitioning from a monolithic architecture to microservices architecture can bring numerous advantages for a software system, including improved scalability, a more conducive environment for autonomous teams, and a system that is better suited to the ever-evolving software landscape. Despite these benefits, the microservice architecture also presents some potential challenges that need to be addressed, particularly in the domain of security. The communication between various microservices is one aspect not present in a

monolithic architecture, and it can be challenging to understand all the security risks related to inter-service communication.

To better examine the security risks related to inter-service communication in a microservices architecture, three research questions were formulated to adequately gather the adequate information. This study seeks to answer three research questions. The first investigates the research facets, contributions facets, publication trends, and venues of the publications using bibliometric data. The second question identifies the security issues discussed in the publications and classifies them according to a predefined set of categories. The third and last research question examines the mitigation strategies proposed in the publications and categorizes them based on a predefined set of categories.

1.3. Research Questions

The research questions formulated for this thesis are designed to guide the investigation and provide comprehensive answers covering the meta-analysis of the publications and the categorization and synthesis of data relevant to inter-service security threats and mitigation strategies.

RQ1: *What types of research methods, contributions, and publication venues are typical in this field?*

- ***RQ1.1:*** *What are typical research facets of studies in this field?*
- ***RQ1.2:*** *What are typical contribution facets of studies in this field?*
- ***RQ1.3:*** *What are the typical publication channels in this field?*
- ***RQ1.4:*** *What are typical publication venues in this field?*
- ***RQ1.5:*** *How has the distribution of publications evolved over the years in this field?*

RQ2: *What are the security threats related to the inter-service domain in microservices?*

- ***RQ2.1:*** *What are the security threats related to the inter-service domain in microservices?*
- ***RQ2.2:*** *How can these security challenges be categorized?*

RQ3: *What are effective mitigation measures for microservice security threats?*

- ***RQ3.1:*** *What are practical approaches and strategies to mitigate microservice security threats?*
- ***RQ3.2:*** *How can these mitigation strategies be categorized?*

1.4. Motivation

As the microservice architecture is a relatively new way to structure a large complex software system, it introduces unique challenges and pitfalls that software architects and developers must navigate. One of the most radical changes from a monolithic architecture is that many of the previous system calls now have to be performed as network calls. These network calls may cross both security zones and networks. The shift from system calls to network calls is mentioned in Sam Newman's section on "Microservice Pain Points", where these network calls are tied to both latency and security issues [2]. Migrating from a monolithic system to a microservice system without understanding the security implications related to inter-service communication can lead to poorly secured systems.

The motivation behind this thesis is to gain a deeper insight into the security challenges posed by inter-service communication in a microservices architecture and to identify their suitable mitigation strategies. This is aimed at helping those transitioning from a monolithic architecture to a microservices architecture while preserving the high level of inter-service security they were accustomed to in the monolithic environment.

1.5. Thesis Outline

Chapter 1 introduces the topic, problems, challenges, and motivation of this systematic literature review. Subsequently, Chapter 2 offers essential background on inter-service security in microservice architecture and reviews related studies, particularly those employing research methodologies similar to this thesis. Chapter 3 describes the research methodology used in this thesis. Furthermore, Chapter 4 looks into the bibliometrics and facets of the publications gathered through a systematic literature review. Chapter 5 categorize and describes the inter-service security threats reported in the included publications, followed by Chapter 6 describing and categorizing the mitigation strategies presented in the included publications. Subsequently, Chapter 7 provides a summary of the findings of this work and compares the results, offers the authors' interpretation of the findings, and additionally presents a systematic map that visually illustrates the main security focus of the reviewed publications. Furthermore, Chapter 8 presents and discusses the validity of this work. Finally, Chapter 9 concludes this thesis and sketches possible directions for future work.

2. Background and Related Work

2.1. Background

The microservice architecture is an architectural design pattern that has become vividly adopted during the past decade to build large, loosely coupled systems. This background chapter lays the foundation for understanding the domains of software architecture and design patterns, and the intricacies of the microservices architecture pattern and how it differs from its counterpart, the monolithic architecture. This chapter will also present some of the design patterns and security challenges and measures that are common to the microservices architecture, as well as those specific to inter-service communication.

2.1.1. Software Architecture and Design Patterns

The design and development of large, scalable and complex systems is perceived by many as a challenging task. Martin [3, p. xvi] states that "Software has structure, many structures, and many kinds of structure, but its variety eclipses the range of physical structures found in buildings". Software architecture is the mere process of making fundamental decisions about the high-level structure of computer systems, and, just as with buildings, decisions made during software architecture carry large implications, which can be difficult to change later in the Software Development Lifecycle (SDLC).

One of the ways in which structure can be achieved in computer systems is to utilize proven reusable design patterns to solve problems before they become unmanageable. Design patterns can be of varying granularity, ranging from very abstract, like the *Microservices Architecture* pattern, to very granular patterns, like the *Money* pattern (a way to represent a monetary value in a software system).

The content and details of design patterns can differ, largely shaped by their author's approach and perspective. Martin Fowler, a reputable author in the field of design patterns, describes them as "a chunk of

advice" (about a topic) and emphasizes that the patterns need to be tweaked a little before being applied to a commonly recurring problem [4, p.10].

Design patterns are usually categorized according to their use case, with similar or alternative patterns often grouped together. Understanding the relationships between design patterns is crucial in order to use them effectively. Some design patterns complement each other well, while others may not be suitable for use in conjunction. For instance, the *Saga* pattern and the *Messaging* pattern commonly used in an MSA are compatible and can be effectively combined in use.

A variety of design patterns are commonly or exclusively applicable to the microservices architecture, as discussed in depth in Chris Richardson's book "Microservices Design Patterns" [5]. These design patterns encompass a wide array of categories, particularly focusing on inter-service communication and security within microservices architecture. A notable example is the *Token* security pattern, often employed in microservice environments where in-memory session management is impractical due to the nature of the distributed system. The *token* pattern emphasizes secure storage and transfer of session data using tokens. Patterns like this address typical challenges in microservice architecture, thus forming the essential foundation for its implementation. The significance of microservices patterns as key building blocks in the microservice architecture is emphasized, especially in research publications on the matter. Moreover, the deployment and design of these patterns within a microservices system is crucial to the discussion of microservices security, as the implications of these design patterns can impact the security of the system as a whole, both positively and negatively.

2.1.2. The Microservice Architecture

Sam Newman defines microservices as "an approach to distributed systems that promote the use of finely grained services that can be changed, deployed, and released independently" [2, p. xvii]. It can also be added that the microservice architecture is an architectural design pattern and that the loosely coupled services that make up the architecture run in different processes, for example, Kubernetes nodes, and communicate through network calls. Kubernetes [6] is a container orchestration framework widely used to deploy microservices. The nodes in a Kubernetes cluster are the worker machines that run the applications, and can be a virtual or physical machine [7]. In the context of microservices, a service can be, for example, a Java JAR file deployed that interacts with other microservices, third-party applications, and end users. Figure 2.1 illustrates a simple microservices application where the front-end and microservices are imple-

mented independently and the traffic between the microservices and the end user is mediated through an API gateway.

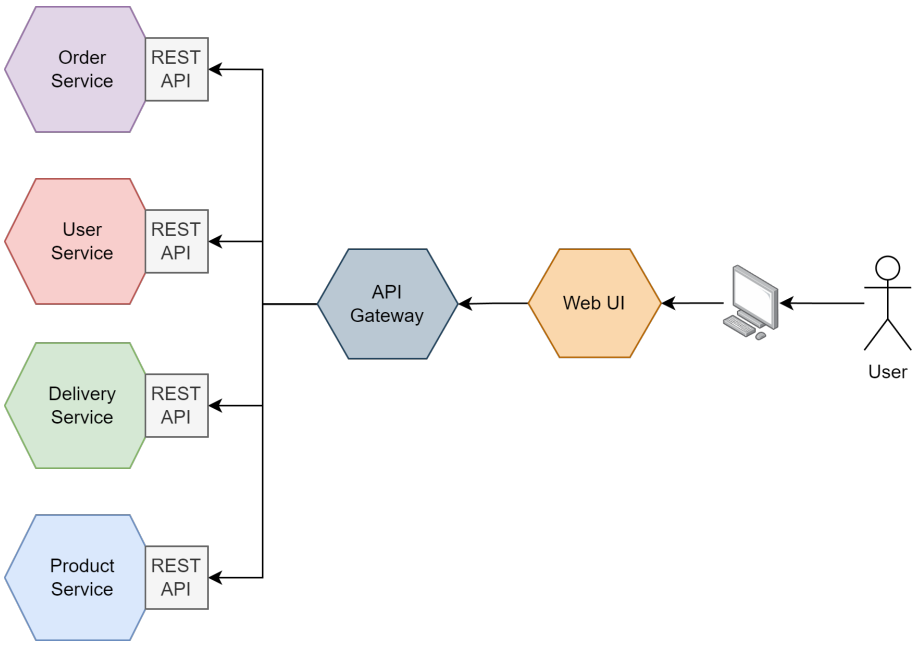

Figure 2.1.: Microservices Architecture

The microservices architecture is often used in conjunction with DevOps practices and deployed as individual containers or virtual machines (or as lambda functions in a serverless deployment [5, pp.416-427]) in the cloud, where scaling opportunities are well suited for dynamic microservices [8]. This makes microservices a highly flexible and scalable solution, enabling applications to rapidly adapt to varying loads and demands.

Monolithic vs. Microservice Architecture

The monolithic architecture is often seen as the traditional way to architect a business application and, compared to microservice architecture, can be deployed as a single service in a single process.

Monolithic architectures come with a set of significant challenges, especially when systems grow in size and complexity. Dragoni *et al.* [9] describes six key issues related to the monolithic architectural pattern:

1. Maintenance becomes increasingly difficult as the system evolves.
2. Dependency issues can complicate development and updates.
3. Minor changes to a single module may require rebuilding and restarting of the entire application.

4. Conflicting resource requirements, which can lead to a sub-optimal one-size-fits all configuration.
5. Opportunities for scaling are often limited, constraining performance.
6. Technology lock-in is a risk, as the entire application is usually based on a single programming language and technology stack.

Due to these and other challenges, numerous companies and organizations have chosen the microservice architecture as a viable alternative, as it can help mitigate or minimize many of these problems. Microservices are independently developed, built and deployed, resource-effective, horizontally, and vertically scalable, and each service can be built with a different programming language and technology stack. This makes microservices a good alternative for development teams that are dealing with some or all of these issues. Chris Richardson specifically points to the "monolithic hell" situation as a major drive for organizations to migrate to the microservices architecture [5]. This situation arises when monolithic systems expand and become increasingly challenging to develop and maintain, increasing the time required to release new features and patches. Figure 2.2 illustrates a simple monolithic application where all business domains are encapsulated within a single service, including the front-end UI with which the end user interacts.

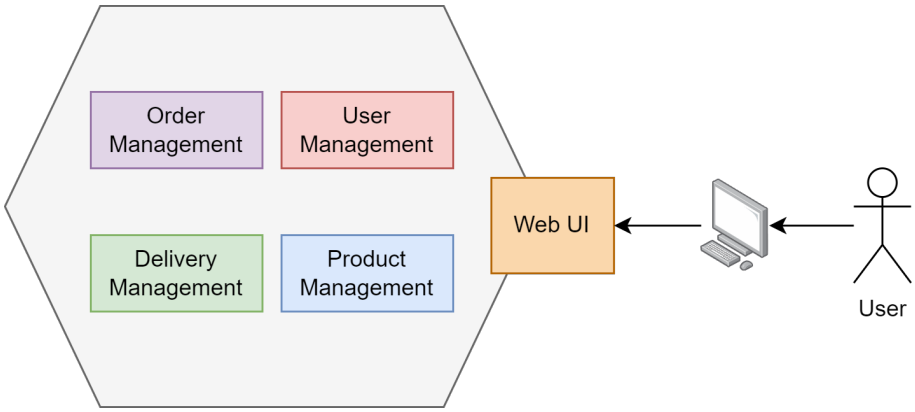

Figure 2.2.: Monolithic Architecture

Migrating from a large monolith to many microservices

The migration process from a monolithic architecture to a microservices architecture is a vital part of the way microservices are adopted. Taibi *et al.* [10] investigated the motivations for the transition from monolithic to microservices and concluded that microservices are often the last resort for practitioners when their monolith becomes too large and unmanageable. This migration process can take a long time, and there are various ways in which this migration can take form, one of which is to apply the *Strangler application pattern* to

the existing monolithic architecture, a concept in which the monolithic application is gradually turned into a microservice application by developing a new application around the legacy application [5, p. 432]. As this process can span years, it is not unusual that systems are in-between a full monolithic system and a full microservices system. This can result in the use of a variety of different technologies, programming languages, and networking protocols at the same time.

The *Anti-Corruption Layer* design pattern, as described by Richardson [5, pp.447] is one of the patterns specifically suited for translating between different domain models, for example, one domain model in the old monolith and one domain model in a newly built microservice. The transition to a microservices architecture brings with it certain considerations that require more attention than others, such as authentication. In a monolithic application, authentication can be managed using in-memory sessions, but in a microservices architecture, it must be persistent, as knowledge about active sessions must be shared among several microservices. Hence, it may be that in the migration stage to a microservices system, that two different ways of handling authentication are used simultaneously. The migration process is also a possible root for security issues.

Ponce *et al.* [11] determined that database migration is one of the most difficult tasks when transitioning to a microservices architecture. In some cases, the databases were not moved to the recommended architecture of one database per service, and thereby left out of the migration completely. Other challenges such as those related to setting up transactions and managing a complex environment with automated tooling were also mentioned. Some of the potential root causes of security concerns that are attributed to the microservices architecture could therefore be due to the difficulties associated with transitioning to this type of architecture.

Inter-service Communication

The flow of data between services in a microservices architecture is defined as inter-service communication. This is also the aspect that differentiates the microservices architecture the most from a classic monolithic architecture and can result in large amounts of information being exchanged within one network or across several network boundaries. The customer's journey from finding a product, adding it to the shopping cart, and purchasing the product can produce a lot of network traffic between services. This flow of data is one of the complexities of microservices that needs to be mastered in order to successfully apply the microservices architecture. Too many unnecessary network calls between microservices can be expensive and unnecessarily slow down the system.

Microservices Design Patterns

Similar to the monolithic architecture, the microservices architecture is also a structural blueprint that comes with a wide range of additional design patterns and tactics. The collection of design patterns used to develop and deploy microservices is what we call microservice patterns, and they make up the building blocks for applying the microservices architecture. Design patterns are frequently cited in conjunction with microservices because they play a critical role in shaping how these systems are developed and operated. The assortment of design patterns applicable to the Microservices Architecture differs depending upon the entity responsible for their formulation. Richardson [5], describes the *Microservices Pattern* as the first pattern addressed in his work. He also elaborates on various additional design patterns intended to complement this foundational pattern. Collectively, these constitute what he refers to as *Microservices Design Patterns*, covering topics such as communication, service discovery, deployment, business logic, security, refactoring, and more. It is important to consider these design patterns in the context of improving security, particularly when building and maintaining robust microservices architectures.

Inter-service Communication Patterns

In microservices architectures with large amounts of inter-service communication, synchronous calls can cause network bottlenecks due to waiting blocks, and hence the use of asynchronous communication for inter-service communication has become a popular alternative to improve system efficiency. Since many of the asynchronous communication protocols can also be performed through a mediator, such as a message bus, this communication method is ideal for a distributed system such as the microservices architecture. Newman [2] outlines two primary methods for inter-service communication: *Synchronous Blocking* and *Asynchronous Nonblocking*. In *Synchronous Blocking*, services interact through synchronous calls that wait for responses, potentially causing delays. On the other hand, *Asynchronous Nonblocking* allows services to communicate without waiting for immediate responses, thus avoiding wait-related blocks.

In contrast to a monolithic architecture, where synchronous communication, such as Representational State Transfer (REST) is commonly preferred, the unique communication requirements of a microservices architecture play a crucial role in distinguishing it. Understanding how these communication demands differ in a microservices environment, and their implications on security, is vital for a successful migration and secure operation.

2.1.3. Microservice Security

Implementing robust security measures is crucial in a microservice architecture, especially given its distributed nature. The distributed nature of microservice architecture can potentially increase the system's vulnerability to security threats by expanding the general attack surface [12, pp. 7-8]. Additionally, microservices applications are frequently deployed in the cloud using virtualization technology, automated pipelines, and various third-party software, which further broadens the attack surface.

When it comes to discussing security within the context of microservices, there are several important aspects that need to be handled, such as the authentication & authorization of both users and between services and applications. In their book, Siriwardena *et al.* [12] divide microservices security into four main categories; edge-level security, service-to-service communication, secure deployment and secure development. Using Siriwardena *et al.* [12] way of dividing microservices security into four topics, we can summarize as follows:

Edge Security

Edge security deals with aspects such as end-user user authentication & authorization, rate limiting & throttling us of the API Gateway and Web Application Security. This security domain is usually more similar between monolithic and microservice applications than some of the other security domains, especially inter-service security which is inherent to microservices [12].

Service-to-service communication

Service-to-service communication (referred to in this thesis as inter-service communication) is the security domain that deals with security in the complexity of communication, interaction, and operation between a set of microservices. This security domain deals with the authentication & authorization between services (e.g. using Mutual TLS (mTLS)), secure communication between services (e.g. using encrypted channels), and other security-related topics such as service discovery, communication within the service mesh, use of the sidecar pattern, network segmentation and firewalls, and intrusion detection. User authorization can also be performed on the service level, conforming to a distributed authorization scheme [12].

Deploying Microservices

The deployment, operation and orchestration of microservices is also a domain which is relevant to security. Microservices are, as previously stated, usually deployed in the cloud using orchestration and container technologies such as Kubernetes [6] and Docker [13]. The usage of these technologies can make microservice architecture more complex to manage, and broaden the microservices attack range e.g. to third-party vulnerabilities etc. [12].

Development of Microservices

In terms of microservice development, there are also some security tactics and design patterns particularly useful for enforcing security measures in microservices. One of these is the *Token pattern* discussed in Section 2.1.1. There are also several other patterns used for developing microservices that deal with security, such as the *Service Mesh* pattern, the *Sidecar* pattern, the *API Gateway* pattern, various service discovery patterns, and more [2, 5].

2.2. Related Work

In this chapter, we examine the available research and other literature on the topic of microservices security.

2.2.1. Security in Microservice Architectures

There are various works in both literature and research that focus on the security of microservices and its numerous subtopics. Two prominent authors and spokesmen in the field of microservices, Chris Richardson and Sam Newman, have both written books on the topic of microservices, where security has been granted different degrees of focus. Sam Newman's book "Building Microservices" [2] has a chapter dedicated to Security, where topics such as security trust boundaries, applications security, securing data, and authentication and authorization are addressed. Richardson's book "Microservices Patterns" [5] devotes little attention to microservices security, but presents one security pattern, the "Pattern: Access token". Richardson's material on the microservices security topic does almost exclusively focus on edge-level security topics such as authentication & authorization of end-users, and how tokens are used to pass information from API

gateways to services. Siriwardena *et al.* [12] book on the topic of microservices security, "Microservices Security in Action" addresses topics such as Edge security, Inter-service communications security, Secure deployment, and Secure development, and each of these main topics have been covered in-depth in the book. NIST's organization has published several reports with best practices covering microservices security, such as "Security Strategies for Microservices-based Application Systems" by Chandramouli [14] For scientific publications, there is a broad range of publications discussing microservices security. The earliest publications discussed microservices security as a subtopic of Service-oriented architecture (SOA) security, but recent publications have addressed microservices security as its own specific topic.

2.2.2. Literature reviews on Microservice Security

Several literature reviews have been conducted in the area of microservice security, although no identified study has solely focused on the topic of inter-service security.

Soldani *et al.* [15] performed a grey literature review on the topic of microservices. This study examined the "pains & gains" of microservice architecture, identifying the difficulties with the architecture, as well as the benefits of applying this architectural pattern. Additionally, this study identified and examined several security-related pains & gains. Pereira-Vale *et al.* [16] conducted a systematic literature review on security in microservice-based systems. This review derived 15 security mechanisms from a selection of academic literature and grey literature. Berardi *et al.* [17] conducted a comprehensive systematic literature review on 290 publications and collected metadata on the publications, looked at areas lacking research, and the approaches taken by microservice practitioners to tackle security issues. Ponce *et al.* [18] conducted a multivocal literature review on the topic of microservices security. The authors analyzed 58 publications and formulated ten security smells, as well as a set of methods to mitigate these security smells.

3. Methods

The purpose of this chapter is to explain the research process, the selection and inclusion criteria, and the guidelines for conducting literature reviews. It also describes the sources used for the study and how these sources were collected and filtered. The goal is to make the research process reproducible for other systematic reviewers.

3.1. Study Design and Methodology

During the course of this study, a systematic literature review was performed to answer the research questions posed for this thesis. A systematic review is a meta-analysis of research publications in which publications for a given topic are collected and evaluated. The data collected during this process is then further analyzed, and the quality of the evaluated publications is often determined. This research methodology is preferred due to its reproducibility and objectivity [19].

Selected parts of the updated *PRISMA* 2020 (Preferred Reporting Items for Systematic reviews and Meta-Analyses) guidelines have been used as guidance when performing this systematic literature review. This collection of evidence-based reporting items is designed to help systematic reviewers answer "why the review was done, what the authors did, and what they found" [20]. The PRISMA guideline contributes to improving the transparency and reproducibility of research, by providing a set of concrete means to do that. We have adhered to some of the relevant reporting items from this framework. [20], but we have not used PRISMA to its full extent. Our approach to fostering transparency and reproducibility is reflected in the following aspects of our methodology, adapted from the PRISMA 2020 guidelines [20]: The databases utilized for the literature review are explicitly stated in Section 3.6. This section also includes a flow chart detailing the research process. Details about the automated tool used to streamline the review process are provided in Section 3.4.1. The selection process for the publications is thoroughly described in Section 3.4

and Section 3.7. A comprehensive list of all included publications can be found in Table C.1, along with their respective research and contribution aspects, and the security focus assigned to each. Information pertaining to the categorization of publications, particularly concerning security threats and mitigation strategies, is detailed in Chapter 5 and Chapter 6 under their respective topics.

3.2. Piloting

Initial search queries were conducted from March to August 2023, with an iterative process to refine them. This refinement aimed to ensure two key outcomes: 1) comprehensive inclusion of relevant publications and 2) minimization of irrelevant publication retrieval. During these initial searches, it was noted that the queries also retrieved publications related to topics such as "5G infrastructure" and "fog meshes". Although microservices are used in these areas and technologies, they were considered outside the scope of this study, which focuses specifically on the security of microservices within IT business applications.

Given the relevance of some publications on non-IT business applications to microservices in the context of IT business applications, a manual approach was chosen to include or exclude publication from these seemingly irrelevant topic domains. This decision was made instead of relying solely on automated methods, such as employing exclusion keywords in search queries, which could inadvertently exclude relevant studies.

The primary database searches for the study were carried out between August 25 and 26, 2023. These searches utilized a variety of prominent databases, i.e. IEEE, SpringerLink, ScienceDirect, Wiley, and ACM Digital Library. These databases were chosen to ensure a comprehensive coverage of relevant literature in the field and because of their extensive collections of scientific and technical literature, relevant to the study's focus.

3.3. Search Strategy

The search string used to perform database searches can be linked to the *Population and Intervention* parts of the PICO scheme [21]. Table 3.1 contains the components that synthesize the search string, how it is linked to the PICO scheme, and the reason for including the search string in the study.

The synonyms chosen for the PICO *Intervention* step were chosen based on the terms used in the publica-

PICO	Search Term	Reasoning
Population	"microservice*" AND ("secur*" OR "threat*" OR "vulnerab*")	Studies mentioning microservices and (cyber) security or common cyber security keywords, e.g. "vulnerability" in the same context.
Intervention	AND ("inter-service" OR "interservice" OR "service-to-service" OR "inter-process" OR "interprocess")	Studies mentioning inter-service interactions and synonyms of this term in the context of microservices.

Table 3.1.: Database search query and its correlation to PICO

tions discovered from the preliminary searches, in addition to the terms used by prominent scholars on the topic. The search query was formulated to retrieve a large number of publications that discuss the topics of microservices and inter-service security within the same context. For each database, a set of search query filters was used to limit the retrieved publications to those most relevant for the study. Each database has different filtering options, and the list of search query filters used for each database is listed in Table 3.2.

Database	Filter
IEEE	Content type: Conference paper and Journal article
Springer	Content types: Conference paper and Article
ACM Digital Library	Content type: Research Article
Wiley	Publication type: Journals
Science Direct	Article type: Review Articles and Research articles. Subject areas: Computer Science, Engineering and Decision Sciences

Table 3.2.: Database filters used

Additionally, we observed that some studies do not employ the term "inter-service" communication or any of the common synonyms for this term and instead use alternative expressions such as "communication between services" or discuss the concept without explicitly stating the communication to be exclusively between different services. Due to these reasons and to increase the overall quality of this literature review, we decided to perform one iteration of backward snowballing on the set of publications included from the database searches. This part is described in more detail in Section 3.5.

3.4. Eligibility Criteria

The resulting studies recovered from database searches were evaluated for a set of defined inclusion and exclusion criteria to consistently omit irrelevant studies. The complete list of the inclusion and exclusion criteria that were defined is described below.

Inclusion criteria:

I1: Publications elaborating on security issues related to inter-service security issues in a microservices architecture

I2: Publications elaborating on mitigating actions for security issues related to inter-service security issues in a microservices architecture

I3: Publications addressing limitations to mitigating actions for security issues related to inter-service security issues in a microservices architecture

Exclusion criteria:

E1: Duplicates

E2: Publications in a language other than English

E3: Bachelor, Master or PhD thesis

E4: Non peer-reviewed work

E5: Insufficient Microservices Focus

E6: Limited Security Discussion

E7: Off-topic domains

3.4.1. Use of Review Tools

The systematic review platform—Rayyan[1]—was utilized for screening publications retrieved from the database searches. This choice was made based on Rayyan's capability to execute multiple functions, thereby expediting the review process and enhancing the review quality. The actions carried out with Rayyan included the following:

- Tagging publications according to exclusion criteria or marking them as included, aiding in tracking

[1] https://www.rayyan.ai/

the review process, and compiling statistics relevant for the meta-analysis of the research process.

- Identifying duplicates, publications in languages other than English, and non-peer-reviewed works, thereby streamlining the selection of relevant literature.
- Associate a set of provided keywords with either the inclusion or exclusion criteria and using these keywords to search in the titles and abstracts of publications added to Rayyan, providing assistance in the evaluation of the studies. For example, "security", "microservices", and "inter-service" were designated as keywords for inclusion, whereas "blockchain", "5G", and "6G" were marked as keywords for exclusion. These keywords were not the sole basis for making decisions but served as helpful indicators during the review, particularly when reading abstracts. Notably, while "blockchain" was an exclusion keyword, certain publications that mentioned it were still relevant for inclusion. The primary reason for its exclusion was the large volume of irrelevant publications that discussed this technology, especially in conjunction with other non-relevant technologies like "IoT", "manufacturing", and "Industry 5.0".

3.5. Snowballing

To cover publications not retrieved from the initial database search (due to the narrow scope of the topic), we performed the *backward snowballing* procedure on the publications included in the primary searches. The backward snowballing procedure is a methodical approach for citation searching and is defined by Wohlin [22] as "using the reference list to identify new papers to include." Wohlin [22, p. 3] also supplements this process by looking at where references are actually referenced in the text; therefore, the context of references matters when determining whether a referenced publication should be included.

The methodology for conducting the backward snowballing process starts by choosing a start set of publications and going through the reference list of each of these publications and either including or excluding the referenced publications. Several factors are used to determine whether a paper should be included and these should be performed sequentially.

Backward snowballing is a systematic process that should be replicable, and there are several indicators that should be used when this process is carried out. First, the title of the referenced publication should be consistent with the study inclusion criteria. Second, the location and context of the references in the text should be analyzed. This includes ensuring that the referenced paper is referenced for a topic that matches

one or more of the inclusion criteria. Lastly, the text should be filtered based on the abstract, introduction, and/or conclusion to see if the publication is suitable for inclusion [22].

3.6. Resulting Study Population

During the inclusion and exclusion process, publications retrieved from primary database searches were evaluated according to the defined inclusion and exclusion criteria. To coherently label all publications, exclusion criteria EQ5 ("Insufficient Microservices Focus"), EQ6 ("Limited Security Discussion") and EQ7 ("Off-topic domains") were defined to label the studies that did not meet any of the inclusion criteria.

The publications were evaluated on the basis of their abstract, introduction, and/or conclusion. If a research paper met any or all of the inclusion criteria, the publications were thoroughly examined. The following table displays the breakdown of the research process:

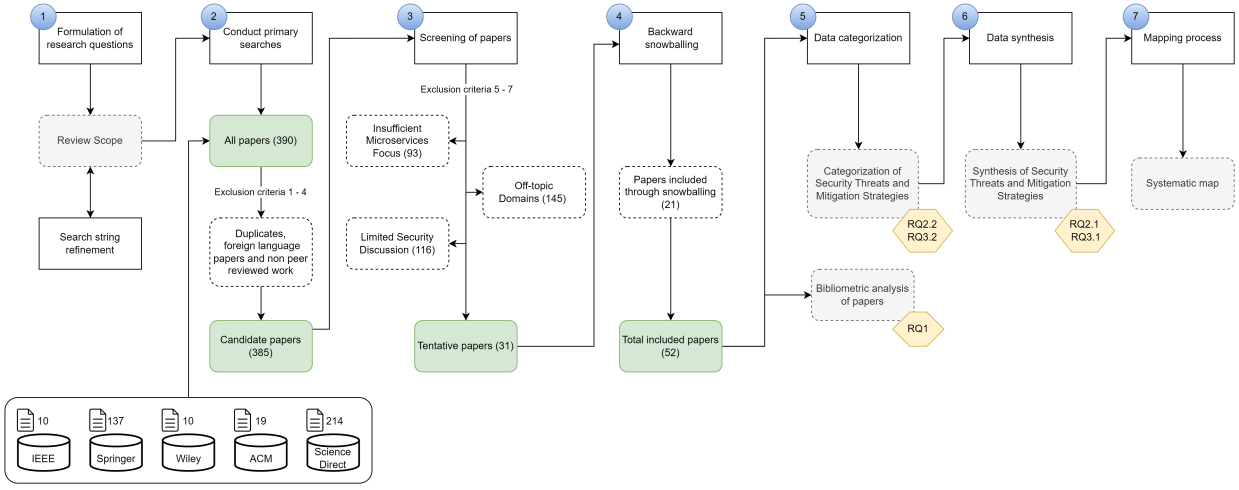

Figure 3.1.: Research process and inclusion & exclusion statistics, modified from Haindl et al. [23]

The efficiency rates (pertinent papers retrieved out of the number of retrieved) for each of the databases are listed in Table 3.3.

Following the screening process of publications, as detailed in step 3 of Figure 3.1, a total of 31 publications were selected for inclusion in the study.

One iteration of backward snowballing was carried out on the start set of these 31 publications. During this

Database	Calculation	Percentage
IEEE	2/10	20%
Springer	8/137	5.8%
Science Direct	14/214	6.5%
Wiley	1/10	10%
ACM Digital Library	6/19	31.5%

Table 3.3.: Primary search efficiency rates

process, 63 publications were selected as candidates. Of these candidates, 21 publications were considered suitable for inclusion, leading to an efficiency rate of 67.7% and an increase in the number of publications included from 31 to 52.

3.7. Paper Categorization

The publications were categorized according to two factors, corresponding to RQ2 5 and RQ3 6. For this categorization, an ordinal scale consisting of the values; *None*, *Low*, *Medium* and *High* were used to evaluate the degree of focus the different publications dedicated to the sub-topics defined for RQ2 and RQ3 (security threats for RQ2 and mitigation strategies for RQ3). This was done to differentiate between studies briefly discussing one or more topics (e.g. systematic reviews) and studies with a more granular and in-depth focus on one or a more topics.

We determined the *level of focus* for each publication's discussion of a topic based on the specific predefined criteria in Table 3.4.

3.7.1. Scope of Topics

As discussed in Section 2.1, our use and definition of *inter-service security* has intentionally been kept broad. This choice reflects our observation that the realm of inter-service security often overlaps with and goes beyond the initially anticipated boundaries. Our approach to include topics that are relevant to microservices security, yet not entirely focused on inter-service security, is a strategic choice intended to shed light on broader and pertinent aspects revealed in diverse scholarly works.

Level of Focus	Description
None	No parts of the text are related to the given topic.
Low	A small amount of content is related to the given topic, but does not go beyond superficial details. For example, the text includes a sentence or a few sentences related to the given topic.
Medium	Substantial content is related to the given topic. The content goes beyond the surface and provides a richer understanding of the topic.
High	In-depth focus on the given topic, which goes well beyond elementary details on the topic. The target audience is usually well-versed on the given topic.

Table 3.4.: Levels of Focus

Furthermore, this inclusive approach allows us to present a more holistic view of microservices security, capturing nuances and intersecting areas that might otherwise be overlooked. By doing so, we aim to provide a comprehensive understanding that encompasses the multifaceted nature of security challenges in microservice architectures, thus offering valuable insights for both practitioners and researchers in the field.

3.7.2. Definition of Categories

In order to define the categories to which each publication was assigned, we followed an iterative process. First, we defined a set of pilot categories (*Inter-Service Authentication and Authorization*, *Containerization and Orchestration* and *Secure Communication*). For each publication, we conducted a detailed analysis to determine if its content aligned with any of the pre-defined categories. In cases where a publication's content did not correspond to any existing category, we recorded this discrepancy. When multiple publications addressed a topic that was not already categorized, we established a new category specifically for that topic. This approach ensured that our categories accurately reflected the topics discussed in the publications and were specifically tailored to the microservice inter-service security domain. The categories used throughout Chapter 5 and Chapter 6 represent the outcome of this iterative process and are the final version of the defined categories. The tables listed under each topic in both Chapter 5 and Chapter 6 contains the categorization decisions made, with the respective publications, levels of focus and category name.

4. Bibliometric Analysis and Facets of Publications

After having applied the inclusion and exclusion criteria (see Chapter 3) to the publications retrieved from both the primary searches and the reverse snowballing, we ended up with a total of 52 publications considered relevant for this systematic literature review. In this chapter, we present the categorization of these publications based on their respective research facets (see Section 4.1), contribution facets (see Section 4.2), publication channels (see Section 4.3), publication venues (see Section 4.4) and publication years (see section 4.5).

The complete table containing information about how each publication was categorized in accordance with the research facets and contribution facets can be found in Table C.1.

4.1. Research Facets

To categorize the publications according to their research facets, which refer to the methodologies and approaches used in the research, we employed the categories and their respective descriptions as outlined in Table A.1, in the Appendix.

In Figure 4.1, we see that most of the publications were categorized as validation research. If we group this together with evaluation research, we see that more than 50% of the analyzed studies fit either one of these two categories, indicating a significant focus in the field on research that validates existing theories, models, or frameworks or evaluates their effectiveness in practical scenarios. There is also a substantial amount of publications that proposes solutions to the problems addressed (23%), without performing extensive validation or evaluation of the solution.

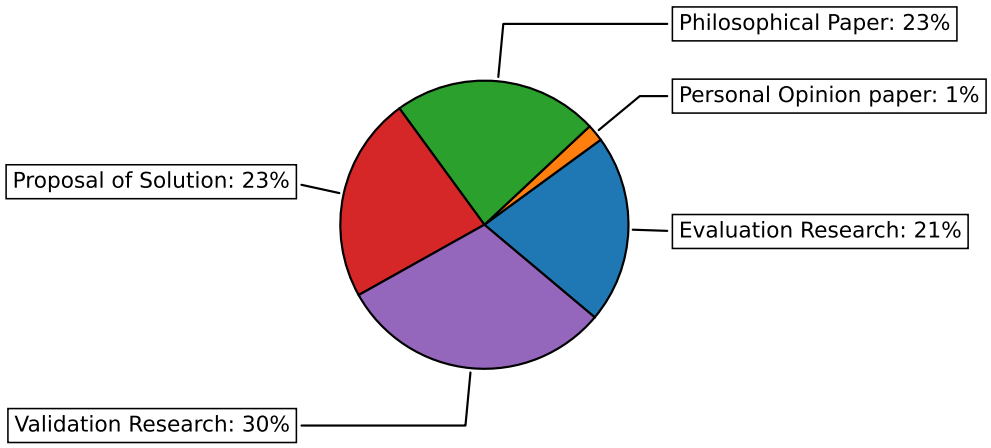

Figure 4.1.: Distribution of research facets

There are relatively few publications, accounting for just 1%, that mainly express personal opinions. Most of the assessed publications base their content and viewpoints on existing research or introduce new solutions, which are then subjected to various degrees of validation and evaluation. This indicates a strong emphasis on existing research or thorough testing of new concepts or ideas, rather than relying on subjective perspectives.

4.2. Contribution Facets

To categorize the publications according to their contribution facets, which refer to the specific contributions made by their research, we employed the categories and their respective descriptions as outlined in Table B.1, in the Appendix.

In Figure 4.2 we see that the majority of the examined publications primarily focus on methods for addressing security threats in microservice architectures, accounting for 34% of the total of evaluated publications. Additionally, a significant number of these works contribute models (15%) to the field, e.g., systematic literature reviews and systematic mapping studies. Many of the assessed publications contribute conceptual frameworks for managing microservices security and categorize their content into microservices mitigation strategies, design patterns, security tactics, and similar concepts, as exemplified in works like [24–26].

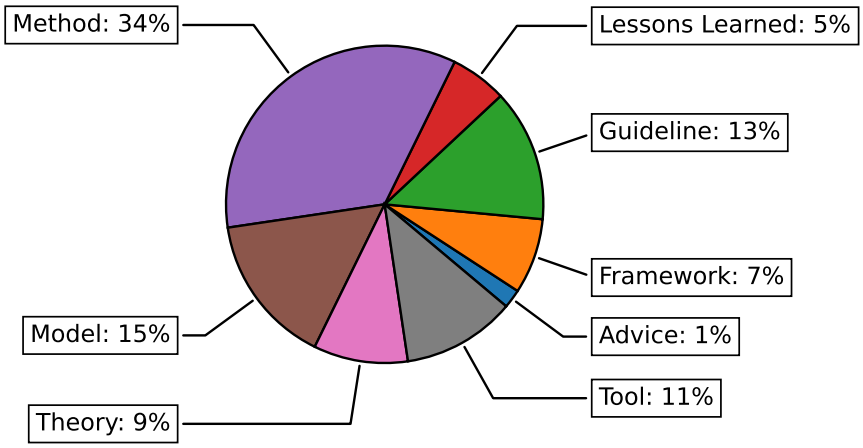

Figure 4.2.: Distribution of contribution facets

However, a smaller segment, about 5%, of these publications specifically contributes Lessons Learned derived from practical experiences or empirical studies, and only 1% of the publications present advice based on personal opinions.

4.3. Publication channel

The studies were published through different publication mediums and a review of these channels was carried out. Most studies and $\approx \frac{2}{3}$ of all studies were published as Conference Papers. The remaining $\approx \frac{1}{3}$ were published as Journal Papers. The prevalence of Conference Papers may suggest that the research field is in its infancy and evolving stages, indicating the need for more comprehensive and detailed research. However, a detailed analysis of these publications is essential to substantiate this hypothesis and fully understand the implications of this trend.

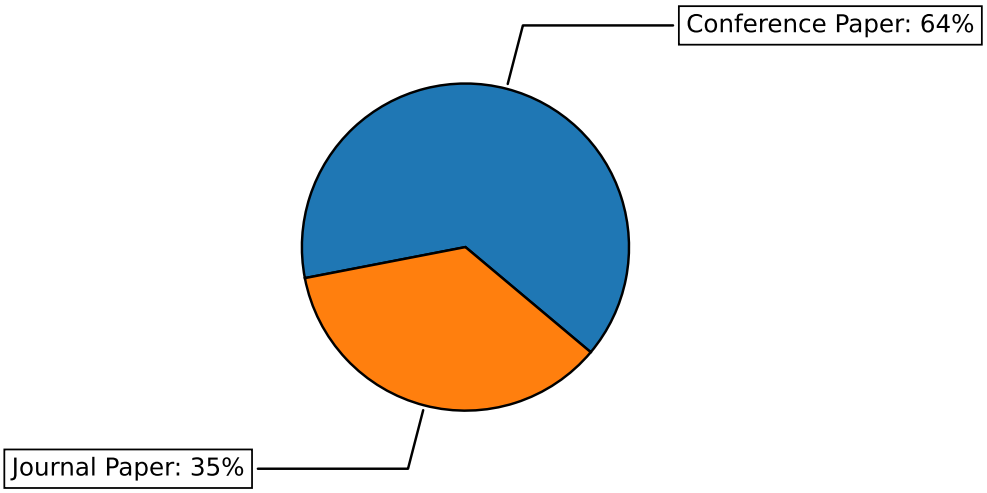

Figure 4.3.: Publication channel

4.4. Publication venue

As can be seen in Section 4.4 a wide variety of different publication venues have been used for the dissemination of microservice security publications. The venues used most frequently are the "Journal of Systems and Software" (6 studies) and the "Computers & Security" (4) studies. These are both reputable journals. However, most venues can be linked to one or two microservice security publications. As mentioned in Section 4.3, most venues are conferences. Additionally, a significant portion of conferences are less prominent or not as widely known in the broader academic field, which could be an indicator that the area lacks higher-quality research.

Table 4.1: Publication venues

Venue	Studies
Journal of Systems and Software	6
Computers & Security	4
ARES	2
International Conference on Utility and Cloud Computing	2
Security and Privacy in Communication Networks	2

Continued on next page

Table 4.1: Publication venues (Continued)

Venue	Studies
Software Architecture	2
ACM SIGCOMM Conference	1
ACM Transactions on Software Engineering and Methodology	1
ACM/SIGAPP Symposium On Applied Computing	1
Advances in Service-Oriented and Cloud Computing	1
Computational Science and Its Applications - ICCSA	1
Computer Science Review	1
Conference on Service-Oriented Computing and Applications (SOCA)	1
Distributed Applications and Interoperable Systems	1
ECSA	1
Empirical Software Engineering	1
EuroPLoP: European Conference on Pattern Languages of Programs	1
Foundations and Practice of Security	1
ICSE	1
ISA Transactions	1
IT Professional	1
Information and Software Technology	1
International Conference on Advancements in Computing (ICAC)	1
International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)	1
International Conference on Cloud Computing Technology and Science (CloudCom)	1
International Conference on ENTERprise Information Systems	1
International Conference on High Performance Switching and Routing (HPSR)	1

Continued on next page

Table 4.1: Publication venues (Continued)

Venue	Studies
International Conference on Service-Oriented System Engineering (SOSE)	1
International Conference on Software Architecture Companion (ICSA-C)	1
International Requirements Engineering Conference Workshops (REW)	1
International Symposium on Parallel and Distributed Processing with Applications (ISPA)	1
Latin American Computing Conference (CLEI)	1
MIDDLEWARE	1
PeerJ Computer Science	1
Science of Computer Programming	1
Service-Oriented Computing	1
Service-Oriented and Cloud Computing	1
Software Architecture ECSA	1
Software Technologies: Applications and Foundations	1
Symposium on Service-Oriented System Engineering (SOSE)	1
USENIX Security Symposium	1

4.5. Publication year

The year of publication of the examined studies was also collected and combined to observe the trends between them, as can be observed in Figure 4.4. All studies examined were published between 2015 and 2023, covering a gap of only eight years. This underscores the fact that microservices represent a relatively new concept. The pivotal article *Microservices: a definition of this new architectural term* by Fowler [27] was published in 2014 and marked a significant moment in the field of microservices development. Consequently, the subsequent emergence of publications concentrating on microservices security, particularly the security between inter-service communications, in the following years appears to be a natural progression. The year with the most studies analyzed in this study was 2021, with 11 studies (21%) published. The year

with the second highest number of publications was 2019. In this year, 9 studies (17%) were published. Coincidentally, the renowned books in the microservices community, *Building Microservices* by Newman [2] and *Microservices Patterns* by Richardson [5], were published in 2021 and 2019. The trend line also predicts an increase in publications related to microservices in the upcoming period.

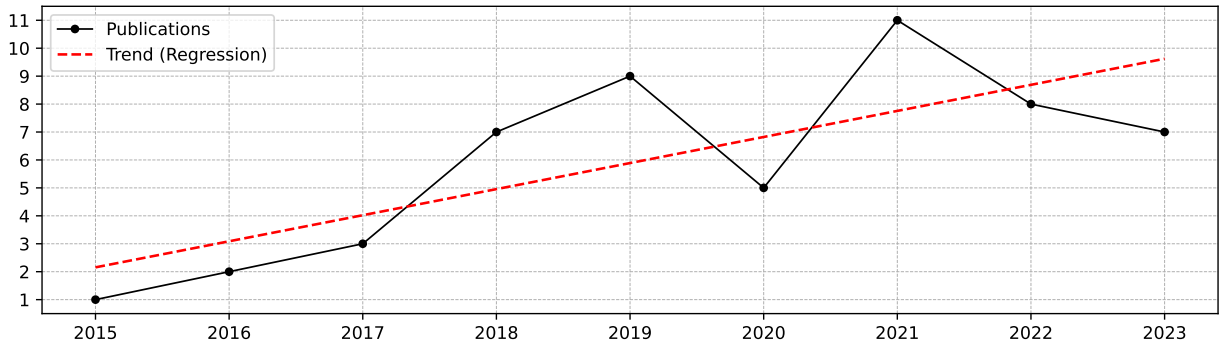

Figure 4.4.: Trend of publications the last decade

5. Inter-Service Security Threats in Microservice Architectures

This chapter provides a comprehensive analysis of the content obtained from the evaluated publications. Additionally, each publication has been associated with relevant inter-service security threat categories based on the topics they address. We have also evaluated the *level of focus* that each publication dedicates to a specific category. The categorization, encompassing both the categories of inter-service security threat and the level of focus is illustrated through diagrams and tables, supplemented by explanatory text. More details on how the categorization process was concluded are described in Section 3.7.

In Figure 5.1 we can see that most of the topics are discussed with a *Low* level of focus. For example, all publications discussing *Network attacks* do this at a *High* level of focus and do not go into detail about specific threats. In several publications, we see that security threats and issues are listed as a risk to consider, but the reasons why these security threats occur are often less discussed.

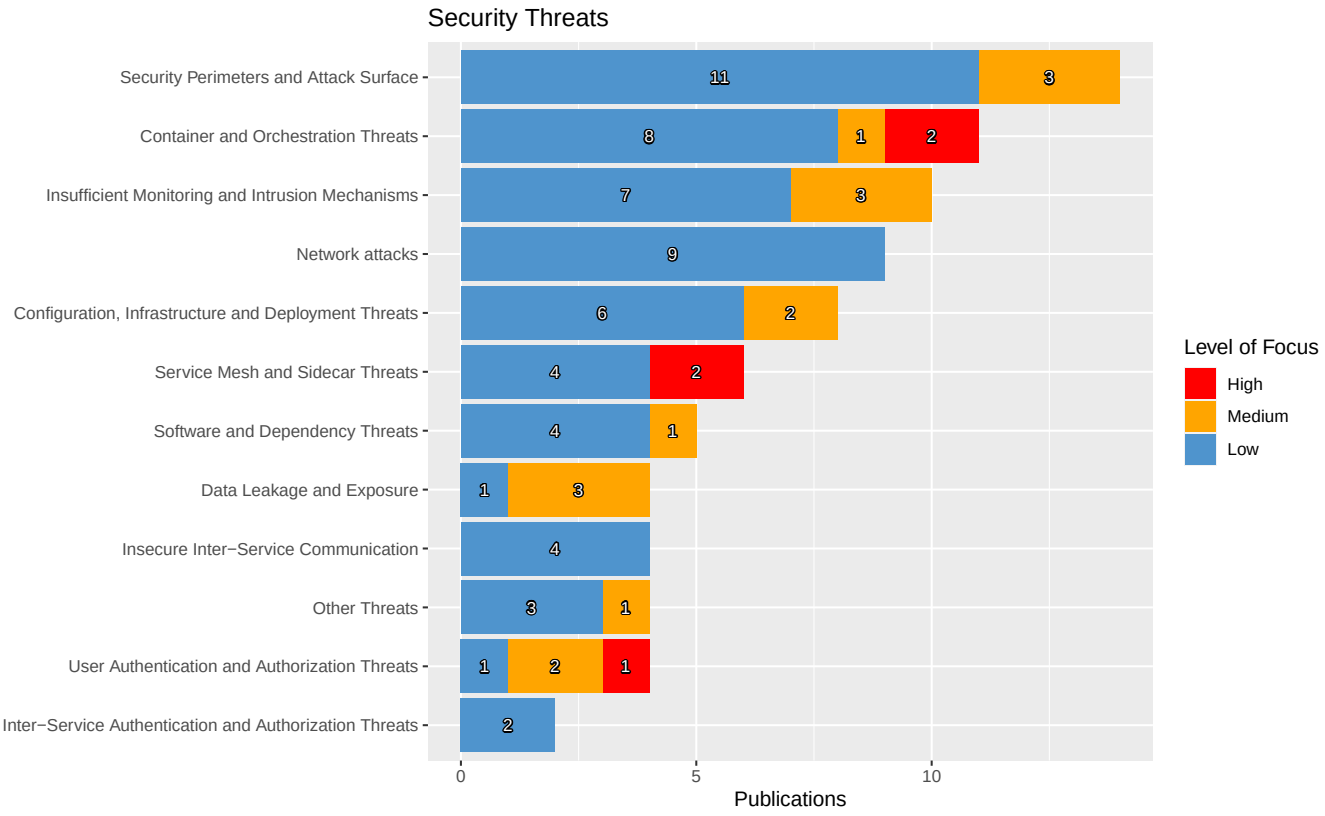

Figure 5.1.: Distribution of security threats addressed

5.1. User Authentication and Authorization Threats

Although the topic of user authentication and authorization is not exclusively related to inter-service security, this subject does intersect with the domain of inter-service security. As an example, user tokens are passed on to and between services or when authorization is carried out on a service-specific basis. Due to these and other factors, we have decided to incorporate the relevant content of the included publications that discuss user authentication and authorization.

As observed in Figure 5.2, there is only a limited set of publications that address security threats related to *User Authentication and Authorization Threats*, but with a distribution spanning all levels of focus. The aggregate proportion of studies discussing this subject totals $(1.9\% + 3.8\% + 1.9\%) = 7.7\%$ of all publications included in the study discuss this topic. The majority, consisting of two publications, treat the topic with a *Medium* level of focus.

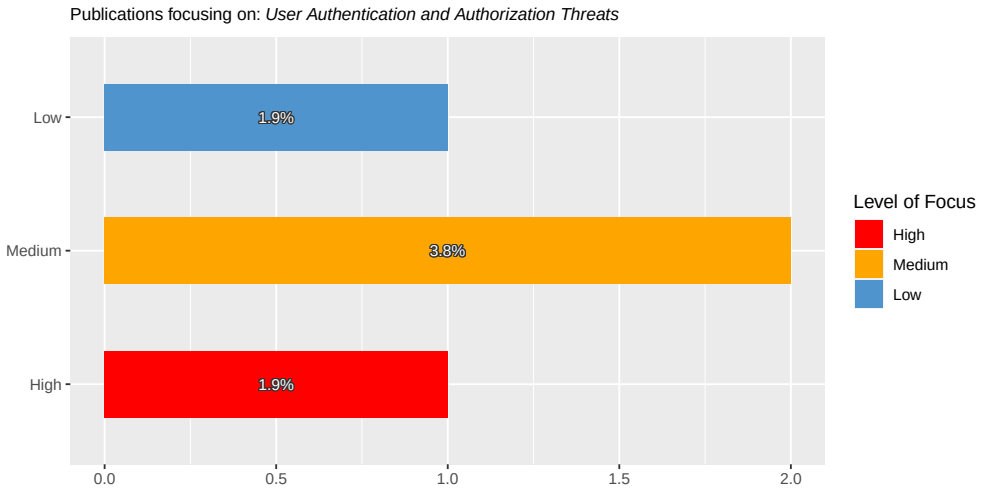

Figure 5.2.: Distribution of User Authentication and Authorization Threats

Publications	**Level of Focus**
[18]	High
[15, 28]	Medium
[29]	Low

Table 5.1.: User Authentication and Authorization Threats

5.1.1. Non-Uniform Access Control

Ponce *et al.* [18] addresses the *Insufficient Access Control* security smell, an issue which arises when a microservice architecture fails to enforce access control uniformly across all its microservices. The study highlights that this oversight can lead to vulnerabilities, notably creating potential attack vectors for the Confused Deputy Problem. The authors also emphasize that in a microservices architecture, access control and identity management require a different approach compared to a monolithic application. Specifically, identity verification must be automated when transferred between different services.

5.1.2. Issues with Centralized Authorization

The security smell entitled *Centralized Authorization*, as defined by Ponce *et al.* [18], occurs in a microservice architecture when authorization responsibilities are limited to a single microservice, typically located at the edge of the system. This configuration overlooks the benefits of the distributed nature inherent in

microservices by failing to implement fine-grained access control at the level of each microservice. Instead, it opts for centralized control, which can also introduce latency issues. This security smell is, as with the *Insufficient Access Control* security smell - an enabler for the *Confused Deputy Problem* as the services are forced to trust the central authorization service. Nehme *et al.* [29] similarly states that the Confused Deputy Problem could occur when the access token is only verified at the API Gateway.

Soldani *et al.* [15] report that the pains associated with access control in microservice architectures could be mitigated if appropriate tools or technologies were available to perform decentralized access control.

5.1.3. Using Multiple Points for Authentication

Ponce *et al.* [18] identify the *Multiple User Authentication* security smell, which occurs in microservice architectures where user authentication is handled at multiple points. The study underlines that this practice increases the attack surface and escalates the development efforts required to maintain and secure these multiple authentication points. Similarly, Soldani *et al.* [15] report that distributed authentication and access control increases the overall attack surface of the systems, due to the many open ports and APIs.

5.1.4. Security Mechanism Weaknesses

Nehme *et al.* [28] explore the security challenges present in the mechanisms used for microservice architectures. This study highlights that OAuth2 [30] access scopes are limited to static and coarse-grained levels, resulting in challenges in managing flexible policies. Additionally, it points out the complexity involved in auditing these scopes. The publication also identifies vulnerabilities in microservices linked to OpenID Connect [31], which is built on OAuth2. It is noted that these mechanisms often depend on a singular token to access all microservices, thus creating the *Powerful Token Theft* issue. Furthermore, the study brings to light the *Confused Deputy Problem*, a significant security risk in this context.

5.2. Insecure Inter-Service Communication

Transmitting and receiving data through insecure means could allow for data to be intercepted, manipulated and degrade the microservices system. The topic of *Insecure Inter-Service Communication* covers different

means of network communication performed between services in a microservice architecture.

The security issue of transmitting and receiving data using insecure methods is as seen in Figure 5.3 discussed in 7.7% of the included studies. All publications that discuss this issue do so with a *Low* level of focus.

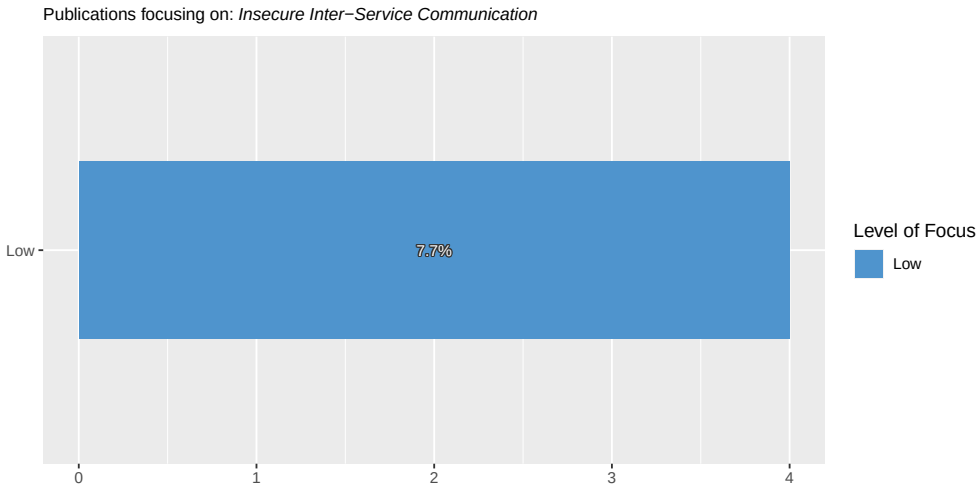

Figure 5.3.: Distribution of Insecure Inter-Service Communication

Publications	Level of Focus
[16, 18, 32, 33]	Low

Table 5.2.: Insecure Inter-Service Communication

Ponce *et al.* [18] define the *Unsecured Service-to-Service Communications* security smell, which occurs when no measures have been taken to secure inter-service communication. The authors report that the absence of secure communication methods can leave the system open to MITM attacks, eavesdropping, and data manipulation that could affect the confidentiality, integrity, and availability of the system. Waseem *et al.* [33] emphasize that one of the complexities in securing microservice architectures arises from dealing with insecure inter-service communication. This aspect is a significant contributor to the challenge of addressing security concerns in microservice design.

5.2.1. Increased Attack Surface due to Inter-Service Communication

Pereira-Vale *et al.* [16] and Wang *et al.* [32] highlight that the attack surface in a microservice architecture is expanded due to inter-service communication. This increase in vulnerability arises because communication

between services occurs over a network rather than being confined to local interactions.

5.3. Data Leakage and Exposure

In the context of *Data Leakage and Exposure*, the selected publications discuss threats associated with excessive data exposure or unintended data breaches within microservice ecosystems. Given that microservice architecture tends to accommodate expansive and complex systems, efforts to minimize data exposure present considerable challenges, as evidenced in the analyzed publications.

The associated security issues with data leakage and exposure are discussed in about $(5.8\% + 1.9\%) = 7.7\%$ of the publications studied. Furthermore, 5.8% of the publications assessed have a *Medium* level of focus on the topic.

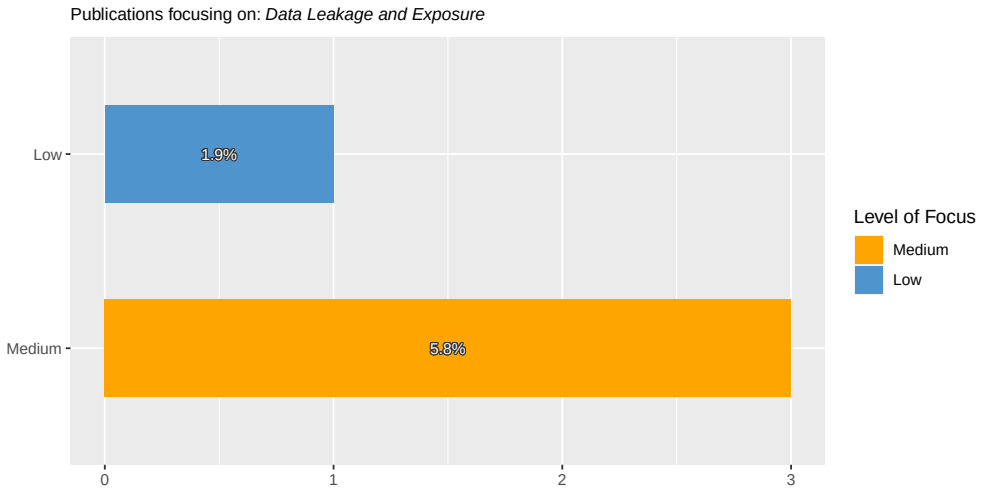

Figure 5.4.: Distribution of Data Leakage and Exposure

Publications	Level of Focus
[18, 34, 35]	Medium
[36]	Low

Table 5.3.: Data Leakage and Exposure

5.3.1. Leakage of Diagnostic Information

Rezaei Nasab *et al.* [36] discuss the vulnerability of diagnostic endpoints to attacks. It emphasizes the risk of sensitive information leakage if these endpoints are not adequately secured.

5.3.2. Exposure of Unencrypted data

The security smell *Exposure of Unencrypted data* defined by Ponce *et al.* [18] is a security smell occuring when microservices expose sensitive data due to insufficient security controls and storage mechanisms, for example, when data are stored in plaintext without being encrypted. The study reports that leakage or exposure of data to an adversary could affect the confidentiality and integrity of the microservice architecture.

5.3.3. Data Exposure through APIs

As described in Genfer *et al.* [34], information exchange in a microservice architecture through APIs is high, and data transfers usually include sensitive and private data. Therefore, the study highlights that data exposure is a high security risk. The study points to excessive data exposure through APIs as a security risk that should be avoided to prevent sensitive data from being exposed. The authors define excessive data exposure as data that are neither consumed by the receiving API nor routed to another API. Zdun *et al.* [35] report the associated security risks with revealing more than necessary information about system internals in API error messages, as this increases the attack surface. The study also reports that excessive usage of APIs by a user can degrade the availability of a microservices system.

5.4. Inter-Service Authentication and Authorization

Security threats pertaining to inter-service access control mechanisms are the subject of discussion in 3.8% of the publications included in our study. Figure 5.5 suggests that the topic has received relatively limited attention in the selected literature.

In the publication by Ponce *et al.* [18], the term *Unauthenticated traffic* is defined as a security smell. This refers to the scenario where communication between services occurs without proper authentication. Walsh

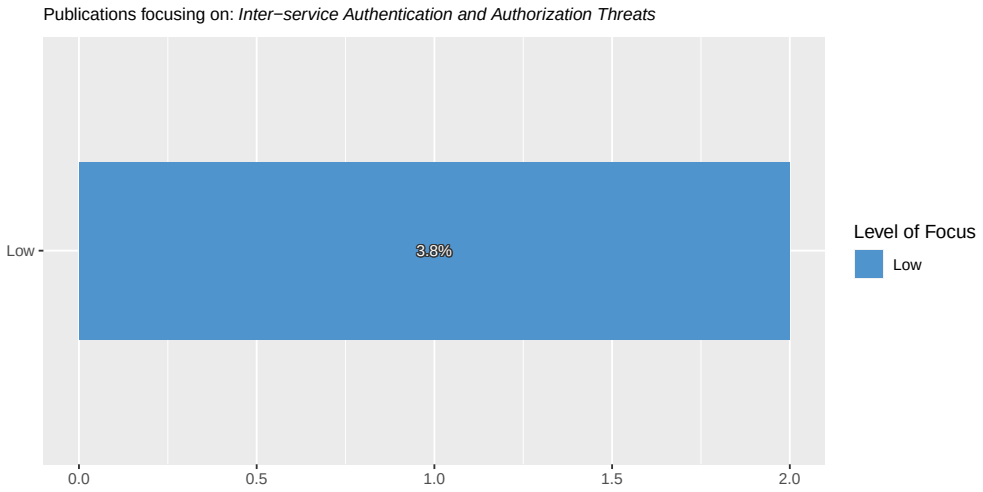

Figure 5.5.: Distribution of Inter service Authentication and Authorization Threats

Publications	**Level of Focus**
[18, 37]	Low

Table 5.4.: Inter-Service Authentication and Authorization Threats

et al. [37] observed a notable gap in existing research on authentication mechanisms between services. This highlights the need for further exploration and development in this specific area of study.

5.5. Network Attacks

During the course of this categorization process we noted that several publications discuss different network level attacks, such as *Man-in-the-middle (MITM)* and *Denial-of-Service (DoS)* attacks. Hence, publications discussing relevant topics were assigned to the topic of *network attacks*.

17.3% of the analyzed publications discuss one or more network attacks that could pose a threat to microservice systems. This is a substantial amount, although all publications discuss these threats on a *Low* level of focus, indicating that while recognized as relevant, these threats have not been the primary focus of detailed examination by the publication authors.

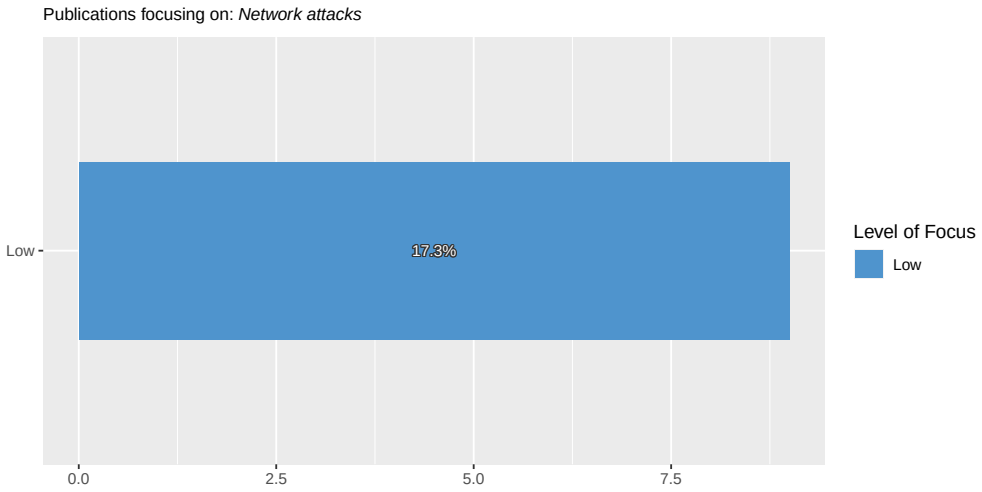

Figure 5.6.: Distribution of Network attacks

Publications	Level of Focus
[17, 18, 24, 38–43]	Low

Table 5.5.: Network attacks

5.5.1. Denial of Service

The threat of DoS attacks for a microservices architecture is reported in several studies [17, 18, 24, 39, 40, 42]. Zdun *et al.* [40] report that DoS attacks could lead to service-level degradation or availability issues in a microservice architecture. The large attack surface usually associated with a microservice system is by Márquez *et al.* [24] attributed to making a microservice system more susceptible to DoS attacks. The use of utility microservices, such as sidecars, is reported to be associated with an increased risk of DoS attacks by Suneja *et al.* [41], as these utility microservices could contain unpatched DoS related vulnerabilities. The flexibility of microservices can also be exploited in DoS attacks, where attackers might scale up less secure microservices to overwhelm more secure central components.

5.5.2. Man in the Middle (MITM)

The threat of MITM attacks is recognized in several studies [39, 40, 43]. However, these sources primarily highlight the vulnerability of microservices to such attacks, without providing extensive details or information beyond this statement.

5.5.3. Session Hijacking

Torkura *et al.* [43] addresses the issue of session/token hijacking, but does so with a relatively low level of detail. The discussion of this particular threat is not extensively elaborated upon in the paper.

5.5.4. Service Discovery Attacks

Compromise of service discovery mechanisms and the potential registration of a malicious node in a microservices system is a security issue highlighted by Yarygina *et al.* [42]. This kind of attack can cause adversaries to redirect communication to themselves.

5.6. Service Mesh and Sidecar Threats

Service meshes and sidecars are extensively employed in the realm of microservices reflecting their importance for this architecture. This section includes publications reporting security threats and issues for service mesh and/or sidecar technologies in a microservice architecture.

Of the analyzed publications we see that $(7.7\% + 3.8\%) = 11.5\%$ address security issues related to service meshes and sidecars and the use of these architectural layers as seen in Figure 5.9. The majority of publications (7.7%) address this topic with a *Low* level of focus, while the remaining 3.8% of the publications address this issue with a *High* level of focus, indicating that there are some in-depth studies available for these threats.

Publications	**Level of Focus**
[38, 44]	High
[41, 45–47]	Low

Table 5.6.: Service Mesh and Sidecar Threats

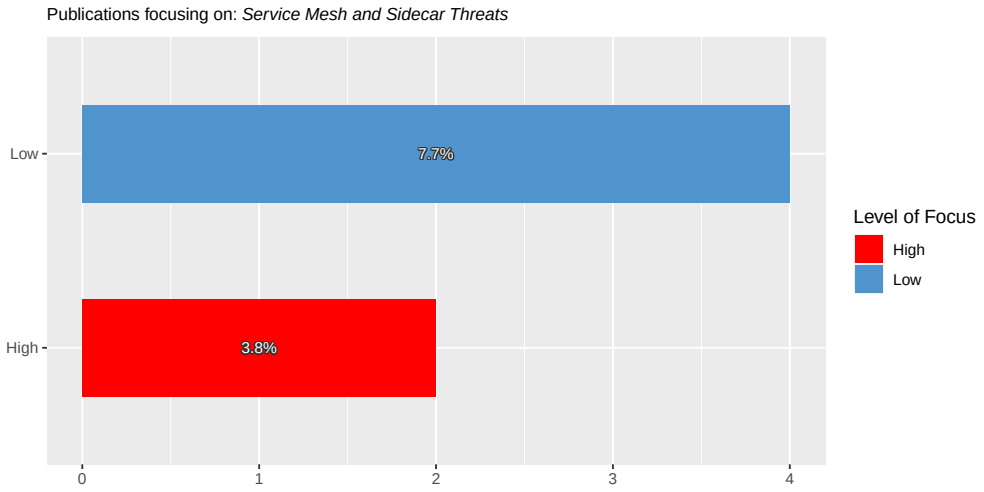

Figure 5.7.: Distribution of Service Mesh and Sidecar Threats

5.6.1. Service Mesh Issues

A service mesh is an infrastructure layer that provides interservice communication, network traffic management, such as load balancing, traffic routing, service discovery, and observability-related tasks such as tracing requests. In addition, service meshes provide several security enhancement capabilities, such as access control, communication encryption, and rate limitation [5, 48]. Consequently, when using a service mesh, much of the functionality that previously had to be made within the microservices codebase is shifted to the service mesh, which simplifies the implementation of commonly used functionality in a microservice architecture independent of the technologies used to develop microservices [2, pp. 162-169]. Richardson [5, pp. 380-382] references Service Meshes as a deployment design pattern (the *Service Mesh* pattern), although other sources do not necessarily label it as a design pattern, but rather as an infrastructure layer or platform.

A service mesh consists of a few building blocks that together make up this infrastructure layer; the data plane (also called mesh proxy) and the control plane Newman [2, pp. 162-169].

5.6.2. Registry Connectivity

Aktypi *et al.* [45] highlight fundamental design issues in contemporary service mesh technologies, particularly the challenges of consistently establishing a connection to a central registry.

5.6.3. Manipulations and Malicious traffic - missing keys

According to El Malki *et al.* [46], the lack of use of encrypted generated keys and certificates, and the lack of use of a certificate management service outside the service mesh, may make the service mesh susceptible to manipulations and malicious traffic.

5.6.4. Lack of Self Protection

The lack of policy autonomy within the service meshes is addressed by R. Alboqmi *et al.* [47]. The issue is that manual intervention is required to make changes within the service mesh configuration at run-time to protect against potential threats.

5.6.5. Threats from Service Mesh Tools

The study by Hahn *et al.* [44] explored security vulnerabilities in widely used service mesh tools, which were mainly attributed to misconfigurations and the lack or inadequacy of security mechanisms. The focus of this study was primarily on the Consul [49] service mesh, which was used as a model, and also included evaluations of Istio [48] and Linkerd (version 2) [50]. The study found that, unlike Consul, both Istio and Linkerdv2 required a Kubernetes cluster to implement their security features, leading to greater complexity in the deployment of these service mesh technologies.

5.6.6. Vulnerabilities in the Istio Service Mesh

In a study that focused on vulnerabilities associated with Istio [48], several significant security concerns were identified [38] and a majority were the result of misconfigurations of the data and control plane or publicly known vulnerabilities.

5.6.7. Sidecar Issues

Suneja *et al.* [41] discuss the act of injecting utility microservices (such as sidecars) into the container of a core (regular) microservice to allow close collaboration between services. The publication labels this as a

security concern, as the lack of isolation between the services would mean that vulnerabilities affecting the utility microservice would also put the core microservice at risk, possibly resulting in the full compromise of the core microservice.

5.7. Software and Dependency Threats

Security issues caused by insecure code, poor architectural choices and vulnerable dependencies are discussed in $(7.7\% + 1.9\%) = 9.6\%$ (as seen in Figure 5.8) of all the publications, with a majority of these publications discussing these security threats on a *Low* level of focus.

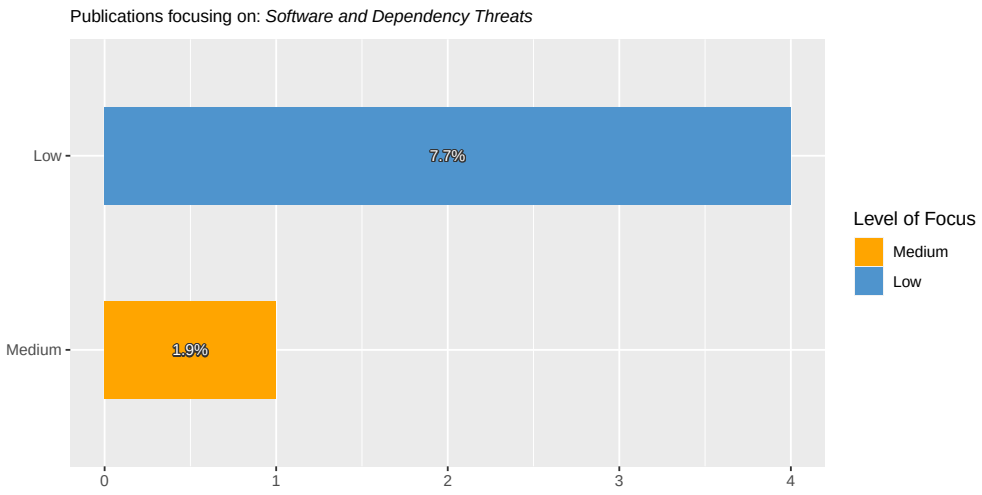

Figure 5.8.: Distribution of Software and Dependency Threats

Publications	Level of Focus
[51]	Medium
[16, 33, 52, 53]	Low

Table 5.7.: Software and Dependency Threats

5.7.1. Lack of Security Patterns

The lack of security patterns tailored for the microservice architecture is reported in Pereira-Vale *et al.* [16], where the study discusses the lack of defined microservices security patterns in the academic literature.

5.7.2. Difficulty in Developing Microservices

Waseem *et al.* [33] reported that three out of six interviewees in their study stated that "poor design and introducing insecure code for microservice systems" [33, p. 16] are reasons why addressing security coordinates in a microservice system is a complex and challenging task.

Ahmadvand *et al.* [53] argued that the decomposition of a software system into a microservices system is usually done with the interest of a few stakeholders and therefore does not account for the requirements of the entire system. The research presented by Waseem *et al.* [26] suggests that applications designed with a microservice architecture may be more susceptible to vulnerabilities than those built using a monolithic architecture. This increased risk is in part attributed to the use of third-party components, which is more prevalent in microservice-based applications.

5.7.3. Homogeneous Environments

In another study, Torkura *et al.* [51] identified a significant risk associated with the adoption of homogeneous microservices in the architecture of the system. This practice, commonly used to avoid the complexity of maintaining multiple technology stacks typical in polyglot architectures, can lead to the proliferation of shared vulnerabilities among services. The risk of these shared vulnerabilities is particularly pronounced in such homogeneous microservice environments, where the uniformity of the technology stack can amplify security risks.

5.8. Container and Orchestration Threats

Containerization and orchestration are both essential building blocks for most microservice systems, as they allow for the rapid and flexible scaling opportunities which are one of the advantages of applying the microservice architecture.

A large number $(15.4\%+1.9\%+3.8\%) = 21.1\%$ of the evaluated publications report security issues directly related to the use of containerization and orchestration tools and platforms in a microservice environment, as seen in Figure 5.9. This considerable percentage signifies that this collection of security threats is relatively popular among the analyzed publications.

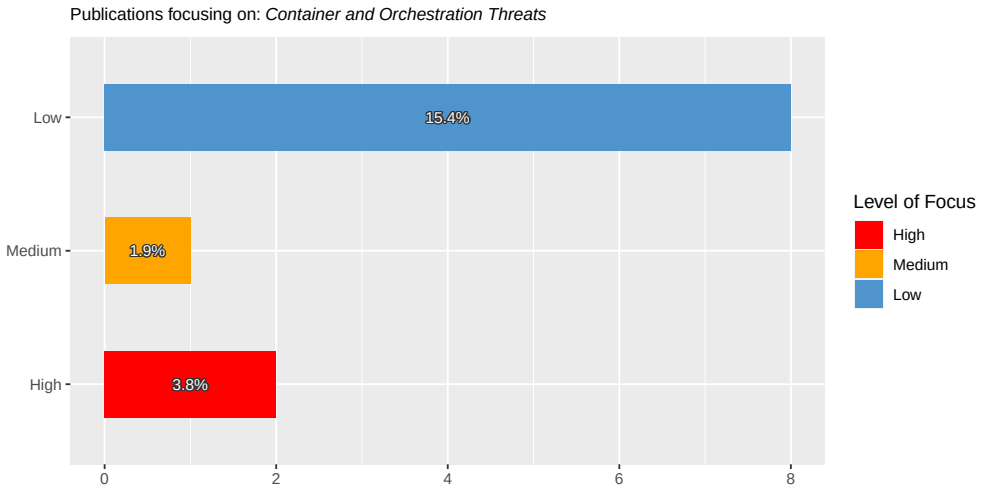

Figure 5.9.: Distribution of Container and Orchestration Threats

Publications	Level of Focus
[38, 54]	High
[55]	Medium
[29, 42, 51, 56–60]	Low

Table 5.8.: Container and Orchestration Threats

5.8.1. Vulnerable Container Images

Several works [29, 55, 58, 60] emphasize the large number of easily exploitable vulnerabilities found in public container images, and Yarygina *et al.* [42] state that the use of vulnerable and/or malicious container images poses a great security concern. Li *et al.* [60] express how the inherent mutual trust between services (commonly found in a microservice architecture) makes a container compromise even more dangerous, as it could allow lateral pivoting between services. Billawa *et al.* [58] also state that a compromise of one container could lead to a compromise of other containers and services and that the use of vulnerable container images may be due to the use of unverified images found in public repositories. The use of images from untrusted sources that have not been validated is noted as one of the most critical image vulnerabilities by Torkura *et al.* [54]. Nehme *et al.* [29] however highlight the prevalence of vulnerabilities found in official Docker images; hence, the problem with vulnerabilities found in container images is not exclusively limited to the use of unverified container images.

5.8.2. Containers

Li *et al.* [56] present the *Containerization* tactic, which "achieves virtualization through containers" [56, p. 11]. However, the publication does convey that infrastructure cloud (micro)services which need to access low-level features from the Operating System are difficult to containerize without sacrificing security. The study by Hannousse *et al.* [57] highlights two primary security concerns associated with the use of containers in the deployment of microservices: first, the potential for containers to be compromised through unauthorized access and second, the inherent vulnerabilities that may exist within the container images themselves.

5.8.3. Lack of Automatic Updates for Containers

Torkura *et al.* [54] state that containers derived from other container images (parent images) are not automatically updated and patched in a way similar to traditional software, which could make them vulnerable even though no vulnerabilities were known at the time of container creation. This is consistent with the findings of Torkura *et al.* [51] in which the risks of using homogeneous microservices are communicated. It is reported that if several microservices are created from the same base image, then a vulnerability in that base image would potentially affect all the homogeneous microservices derived from it.

5.8.4. Misconfigurations

Torkura *et al.* [54] do additionally highlight the risk of misconfiguration of container images that could result in the creation of vulnerabilities. Similar risks are also present for container run-time configurations and container engine configurations, as addressed by Minna *et al.* [55], which observed a high number of container host devices using default configurations.

5.8.5. Applications Built into Container Images

The security concern related to the practice of building applications in images is highlighted by Torkura *et al.* [54] as this approach can potentially create an attack vector for side-channel attacks.

5.8.6. Containerization Platforms

Zeng *et al.* [38] outline security issues within containerization platforms. The study synthesized CVEs that affect different containerization platforms and ranked the Docker platform as the most vulnerable of the components evaluated (Docker [13], containerd [61], runc [62]), 80% of its vulnerabilities being related to the escalation of privileges. The possible issue of container escape possibilities is also discussed, which could potentially allow an attacker to access files or perform actions outside the container.

5.8.7. Orchestration Tools

In their study, Zeng *et al.* [38] also performed a mapping of public CVEs to the various components utilized in the Kubernetes orchestration platform. These components include API Server, Kubectl [63], and Kubelet [64]. The study identified the API Server component as the most vulnerable, with 30% of CVEs related to this component. Moreover, 80% percent of all identified vulnerabilities are caused by privilege escalation attacks, with a large majority coming from tools that work together with the Kubernetes platform, such as plugins. The Kubernetes network and its different layers (Layer 2 and Layer 3) were also evaluated, and it was discovered that 90% of the vulnerabilities were related to increased privileges and 50% of these were caused by MITM attacks resulting from misconfigurations.

5.8.8. Excessive Privileges

A security risk arising from giving too many privileges to Docker containers is expressed by Ibrahim *et al.* [59]. Since some containers need excessive privileges in order to function properly, and thereby are granted access to the Docker daemon, they could potentially access every service in the microservice system, as these are all controlled by the Docker daemon.

5.9. Insufficient Monitoring and Intrusion Mechanisms

This category was defined to map research discussing how current microservice architectures are vulnerable due to inadequate monitoring and intrusion mechanisms. This includes a detailed look at various aspects of intrusion mechanisms, such as detection, response, and prevention.

As seen in Figure 5.10, $(13.5\%+5.8\%) = 19.3\%$ of the included publications discuss the topic of *insufficient monitoring and intrusion mechanisms*.

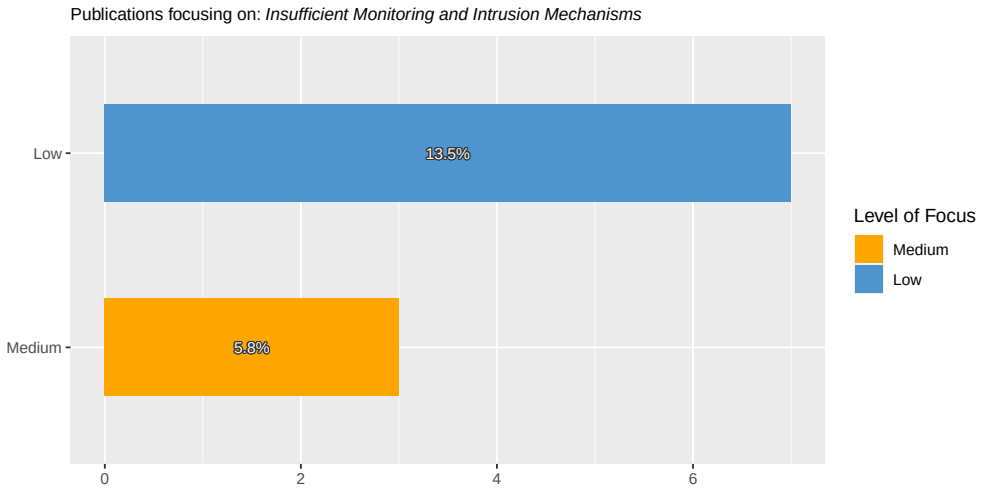

Figure 5.10.: Distribution of Insufficient Monitoring and Intrusion Mechanisms

Publications	**Level of Focus**
[24, 25, 65]	Medium
[16, 29, 52, 58, 66–68]	Low

Table 5.9.: Insufficient Monitoring and Intrusion Mechanisms

5.9.1. Low Traceability

The lack of traceability in microservices systems was reported in two studies [52, 58]. Alshuqayran *et al.* [52] noted that the problem of tracing a request through several hops, i.e., microservices, is a complex task that requires more research.

5.9.2. Lack of Reaction, Detection and Recovery Mechanisms

The observation that microservice systems are more geared toward preventing attacks than detecting, responding to, or recovering from them is detailed in a study by Márquez *et al.* [24]. This study also highlights the lack of discussion and development on recovery mechanisms for microservice systems in the event of a cyber attack, which are crucial to restore the system to its normal operating state. Yarygina *et al.* [65] point

out a shortfall in research on Intrusion Response Systems (IRS) tailored for microservices architectures, particularly in the face of a growing number of potential attacks. The study also notes the limited focus of research on the self-protection capabilities of microservices.

5.9.3. Lack of Intrusion Detection Systems

Pereira-Vale *et al.* [16] reported that there are a limited number of intrusion detection systems specifically designed to detect anomalies in microservice architectures.

5.9.4. Difficulties with Security Monitoring

The challenges associated with security monitoring in microservice architectures are reported in Nkomo *et al.* [67], particularly in systems where containers are used. The complexity arises when Network Address Translation (NAT) is used to allocate public addresses to containers that host microservices. This allocation complicates the identification of network traffic for specific services. The situation becomes even more complex with the use of containers, as it becomes difficult to determine which specific service is operating within each container. Similarly Sun *et al.* [68] report that the complexity of monitoring in clouds that are not owned by product owners can make deployed microservices attractive targets for adversaries. Furthermore, Billawa *et al.* [58] note the issue of limited visibility in microservices architectures. This low visibility, characterized by the challenge of tracking system failures due to the decentralized and non-restricted placement of microservices, can make it difficult to recover from attacks. The study highlights that this inherent characteristic of microservice architecture complicates the process of effectively identifying and addressing security breaches.

5.9.5. Deep-Packet Inspection Issues

Nehme *et al.* [29] discuss the difficulties associated with traffic monitoring and filtering near the application for the purpose of deep packet inspection. The study emphasizes that the rules for deep packet inspection need to be specifically customized for the microservice architecture, underscoring the complexity of this task.

5.10. Security Perimeters and Attack Surface

As observed in Figure 5.11, $(21.2\% + 5.8\%) = 27\%$ of the included publications discuss topics related to *Security Perimeters and Attack Surface*. This is a large number of the included publications, stressing how popular this topic is.

It was also observed that a large number of these publications discuss this security topic with a *Low* level of focus, and many of them discuss the topic of Section 5.10.1, but in little detail.

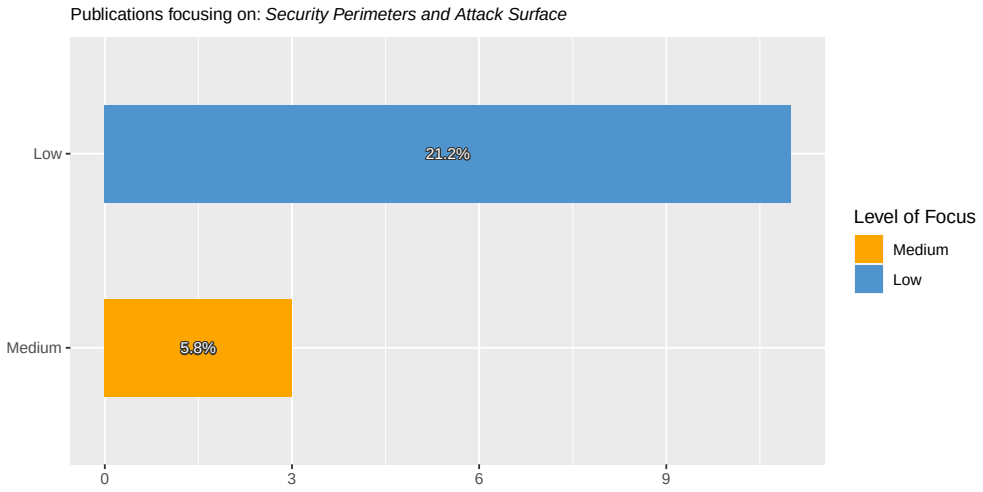

Figure 5.11.: Distribution of Security Perimeters and Attack Surface

Publications	Level of Focus
[18, 66, 67]	Medium
[15, 29, 39, 43, 51, 57, 58, 60, 68–70]	Low

Table 5.10.: Security Perimeters and Attack Surface

5.10.1. Increased Attack Surface

The issue of having the attack surface increase when adhering to a microservice architecture compared to a monolithic architecture is reported in several studies [15, 29, 39, 43, 51, 58, 60, 67, 70], with the main reason being the distributed nature of microservices where data exchanges have been shifted from within a single application to being performed across one or more networks. Soldani *et al.* [15] additionally mention that the increase in the number of open ports, accessible APIs, and distributed access control mechanisms

in a microservices system are the main factors that contribute to the increase in the attack surface. Chondamrongkul *et al.* [39] assert that the horizontal scaling of microservices concurrently expands the attack surface proportionally with the scaling, as new ports are opened for each deployed service.

5.10.2. Pivoting and Trust Issues

In the realm of microservices security, the issue of implicit trust between services has been reported as a potential security risk by two studies [66, 68]. As noted by Sun *et al.* [68], this can lead to scenarios where an adversary compromising one service might result in a takeover of the entire system. Complementary studies, such as Li *et al.* [69], also discuss the risks of pivoting and additional attacks on microservices after an initial compromise. Furthermore, Sun *et al.* [68] report on the limited flexibility in controlling the trust placed on different microservices. The practice of *trusting the network* is studied in Ponce *et al.* [66], where 13 other studies were identified to categorize this practice as negative to system security. Adherence to the practice of *trusting the network* means blindly trusting that all network traffic is legitimate. The study states that insecure inter-service communication mechanisms can be implemented by practitioners, with the justification that the network is secure and has undergone filtering and inspection by the edge-level security controls. Another example in which the bad practice of *trusting the network* has been observed is when microservices trust network components such as API Gateways by their identity, as this could allow the compromise of these network components to also be a risk to the microservices they communicate with. The findings presented in Hannousse *et al.* [57] look at insecure configuration as a factor that explains why the compromise of one service could lead to complete takeover of the entire microservices system.

5.10.3. Indefinable Security Perimeters

Nkomo *et al.* [67] shed light on the case where security perimeters become undefinable when using microservices. The authors argue that the use of containerization (itself allowing port mapping, the use of dynamic addressing, and microservice scaling) complicates the work of implementing security perimeters similar to those used to secure monolithic systems.

5.10.4. Unnecessary Privileges

The security smell of granting *Unnecessary Privileges to Microservices* is described in Ponce *et al.* [18]. It occurs when privileges beyond those required to conclude business functionality are granted to microservices, e.g., when a microservice can interact with a message queue without explicitly needing to do so to fulfill its business functions. The study reported that 12 of the 58 selected studies had reported negative security characteristics related to this security smell. The study also reported that this security smell contributes to an increase in the attack surface for the microservices architecture.

5.11. Configuration, Infrastructure and Deployment Threats

As observed in Figure 5.12, $(11.5\% + 3.8\%) = 15.3\%$ of the included studies discuss security threats related to the *Configuration, Infrastructure and Deployment* of microservice architectures, with majority of these publications discussing these security threats with a *Low* level of focus.

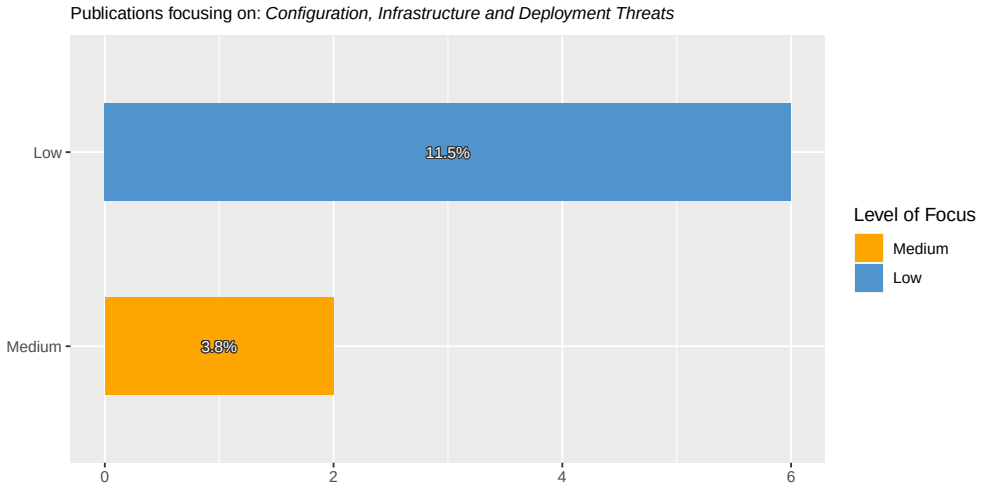

Figure 5.12.: Distribution of Configuration, Infrastructure and Deployment Threats

Publications	Level of Focus
[18, 57]	Medium
[33, 42, 43, 54, 58, 71]	Low

Table 5.11.: Distribution of Configuration, Infrastructure and Deployment Threats

5.11.1. Agnostic Technology and Vulnerability Detection

Microservices can be developed using different programming languages, libraries, etc. and still operate and communicate together in one system. Additionally, microservices systems are often quite complex using a range of different technologies and tools e.g. for monitoring, traffic mediation, data storage etc. Torkura *et al.* [43] attribute the technological diversity as a complicator of vulnerability detection, since each technology in use needs to be identified and tested for vulnerabilities.

5.11.2. Cloud Infrastructure

In the context of addressing security challenges within a microservice architecture, Waseem *et al.* [33] point out that the use of insecure cloud infrastructure to deploy and scale microservices contributes to the security issues faced.

5.11.3. Malicious Cloud Provider

The security threats related to the adoption of cloud technologies in a microservice architecture are explored in Yarygina *et al.* [42]. This publication describes cloud technologies as introducing "a myriad of security concerns" [42, p. 14]. Among the primary security challenges identified is the extensive control that cloud providers have over the infrastructure. This level of control could potentially allow a malicious provider to gain complete control over the microservice system, posing a significant security threat.

5.11.4. Secrets and Data at Rest

Ponce *et al.* [18] highlight a range of security issues associated with the storage of data and secrets in microservices. An identified key concern is the *Hardcoded Secrets* security smell, which occurs when secrets are embedded directly within the application code or the configuration files used for deployment. Furthermore, their research points to the critical problem of not encrypting data, particularly data at rest. This issue is categorized under the security smell *Exposure of non-encrypted data*, which arises in situations such as storing unencrypted data or using encryption mechanisms compromised by publicly known vulnerabilities. Furthermore, the study underscores a potentially greater risk: the use of in-house developed cryptographic

code that has not undergone extensive testing. This approach can lead to a false sense of security, as such code may not be as robust as well-tested, established cryptographic solutions.

5.11.5. Security Tools Challenge of Discovering Microservices

Torkura *et al.* [54] discuss the challenge facing security tools in detecting microservices once they have been scaled in a cloud environment. This highlights the difficulty in maintaining visibility and security oversight in dynamically scaling microservice architectures.

5.11.6. Soft and Hard Infrastructure Attacks

The work of Hannousse *et al.* [57] delves into the categorization of attacks on microservices, distinguishing between *Soft-Infrastructure* and *Hard-Infrastructure*. *Soft-Infrastructure* attacks refer to vulnerabilities in the software components of a microservice architecture, including elements such as message queues, network mediation software, and monitoring tools. On the other hand, *Hard-Infrastructure* attacks target the hardware components of a software system, focusing on vulnerabilities that may have been introduced intentionally or unintentionally during the manufacturing process. These hardware vulnerabilities present potential exploit opportunities for adversaries. Hannousse *et al.* [57] also report that comparatively less attention is paid to securing microservices against *Hard-Infrastructure* attacks. This indicates a potential gap in the security measures being implemented, with a stronger focus typically placed on the software aspects (*Soft-Infrastructure*) rather than the hardware vulnerabilities.

5.11.7. Polyglot and Homogeneous environments

Billawa *et al.* [58] discuss the complexities involved in the deployment of microservices that are written in various programming languages, each with its own versions and life cycles. Furthermore, the study emphasizes the challenges of implementing effective security measures in this diversity of programming languages. This is due to the fact that each language typically utilizes different frameworks, necessitating distinct security considerations and approaches for each. The multiplicity of frameworks and the need to tailor security practices to each one of these, provides layers of complexity to ensure robust security in such heterogeneous microservice architectures.

5.12. Other Security Threats

Some of the analyzed publications discussed topics related to the inter-service security threats, but did not fit into any of the previously listed topic categories.

As observed in Figure 5.13, $(5.8\% + 1.9\%) = 7.7\%$ of the included studies discuss security threats that do not fit into any of the other categories.

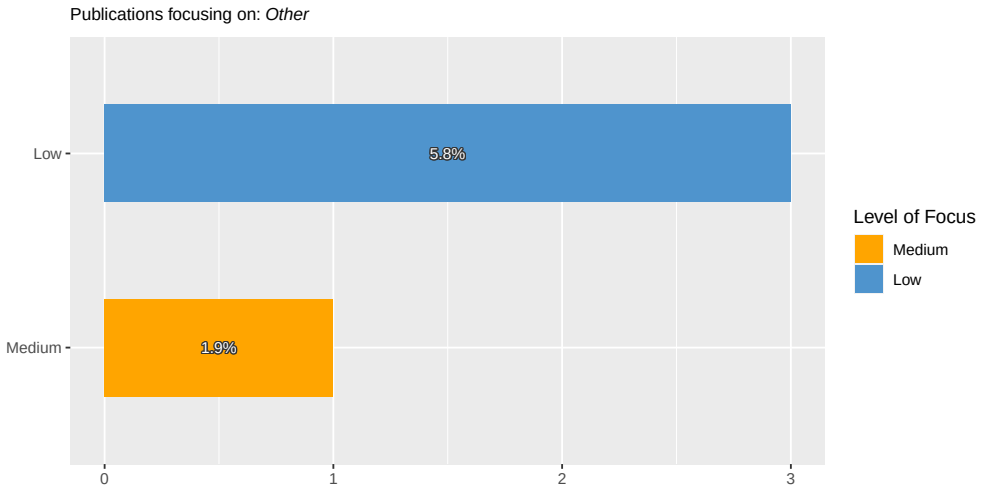

Figure 5.13.: Distribution of Other Security Threats

Publications	Level of Focus
[71]	Medium
[17, 60, 67]	Low

Table 5.12.: Other Security Threats

5.12.1. Insider Threats

Ahmadvand *et al.* [71] discuss the security threats associated with insiders having access to microservices systems. In the study, insiders are referred to as the people that access the tools responsible for managing microservices (e.g. DevOps engineers, sysadmins). The study presented several different attack paths with different security threats targeting both *Data Assets* and *Behavior Assets*. For *data assets*, the study particularly stressed the security threats linked to the manipulation of logs. In the case of *behavior assets*, it underscored the risk of manipulation of services, hardware containers and similar modules. Furthermore, it

noted how such interference could facilitate attacks like *In-memory attacks*.

5.12.2. Threat Modeling and Risk Assessments

Nkomo *et al.* [67] highlight the security challenges inherent in conducting threat modeling and risk assessment within a microservice architecture. The independent structure of development teams in such architectures combined with continuous delivery practices increases the risk of releasing vulnerable code before adequate risk assessments and threat models are developed. Additionally, Berardi *et al.* [17] note the absence of threat models specifically tailored to the microservice architecture, adding complexity to developers' workload.

5.12.3. Missing Inter-Service Security Policies

Li *et al.* [60] state that there is a need for more flexible mechanisms to generate inter-service security policies for the microservice architecture. The study states that current practices are based on manual configuration, which becomes more difficult as the microservice system scales. It is also worth noting that the authors of the publications state that the tool developed during their research is intended to overcome these challenges.

6. Inter-Service Mitigation Strategies in Microservice Architectures

This chapter provides a comprehensive analysis of the content obtained from the evaluated publications. In addition, each publication has been associated with relevant categories for mitigation strategies based on the topics they address. We have also evaluated the level of focus that each publication dedicates to a specific category. The categorization, encompassing both the inter-service mitigation categories, and the level of focus is illustrated through diagrams and tables, supplemented by explanatory text. More details on how the categorization process was concluded are described in Section 3.7.

As depicted in Figure 6.1, the distribution of publications discussing various mitigation strategy topics at either a *Low* or *Medium* level of focus appears to be quite similar. A considerable number of the assessed publications address mitigation strategies at a *Medium* level of focus, as observed in categories like *Security Policies* and *Secure Code, Design Patterns, and Architecture*, among others. In addition, many of these publications provide guidelines for handling security threats, often offering recommendations at a *Medium* level of focus for several topics.

From Figure 6.1 we can see that the mitigation techniques most discussed fall into the category *User Authentication and Authorization*.

6.1. User Authentication and Authorization

As observed in Figure 6.2, $(5.8\% + 13.5\% + 5.8\%) = 25.1\%$ of the assessed publications discuss mitigation strategies related to *User Authentication and Authorization*, and the majority of thee publications do so with a *Medium* level of focus. However, there is also a small segment (5.8%) of the assessed publications, that delves more deeply into this area, discussing it with a *High* level of focus.

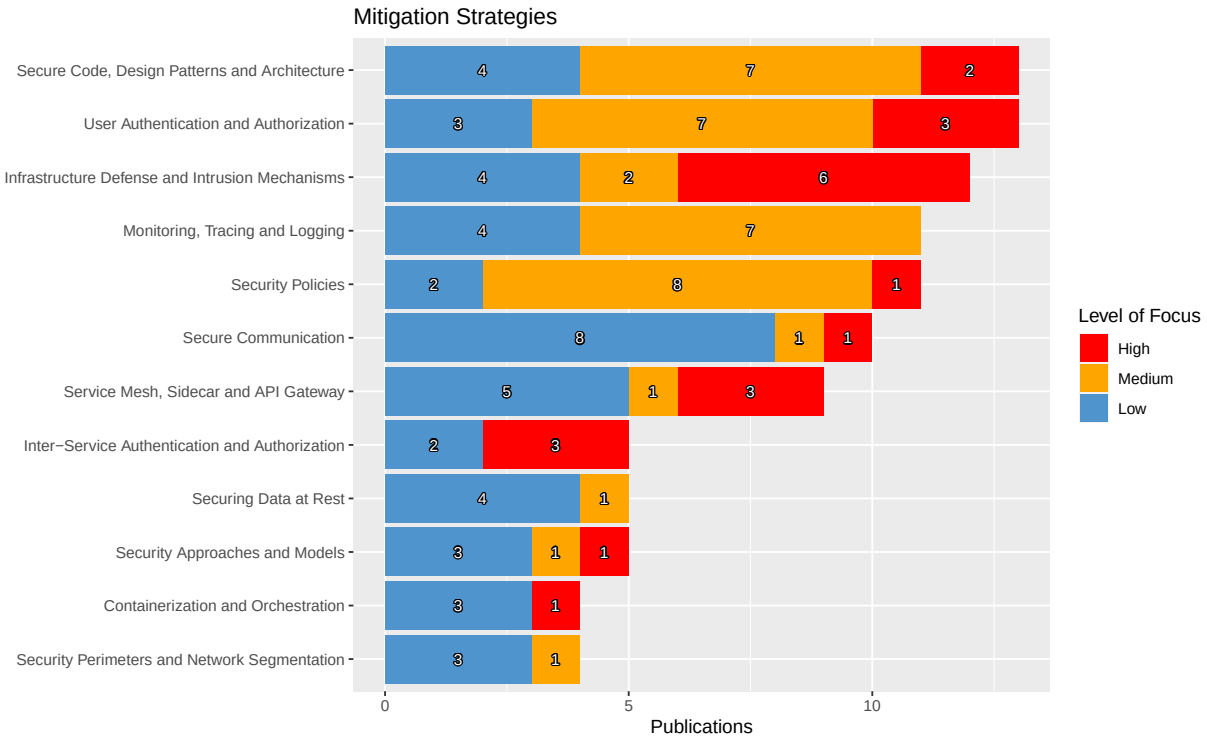

Figure 6.1.: Distribution of mitigation topics discussed

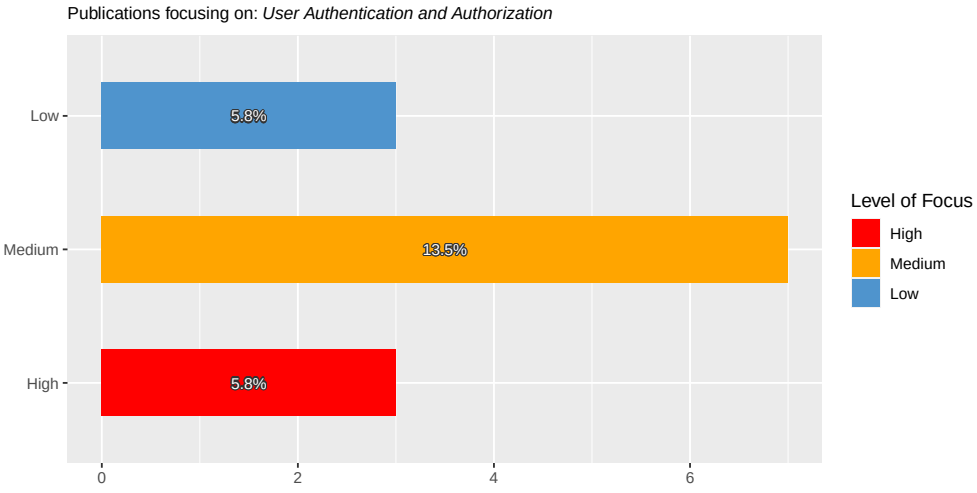

Figure 6.2.: Distribution of User Authentication and Authorization

Li *et al.* [56] mention the use of federated identity as one of the solutions needed to implement the *Authentication and Authorization* tactic in a microservice architecture and use technologies like OpenID Connect & JWT and OAuth2. Rezaei Nasab *et al.* [36] define several security practices related to user authentication, authorization, and credentials. Zdun *et al.* [40] specifically address the identification and authentication of

Publications	Level of Focus
[28, 36, 72]	High
[18, 35, 40, 42, 66, 71, 73]	Medium
[16, 56, 58]	Low

Table 6.1.: User Authentication and Authorization Distribution

API clients. The research outlines two distinct design choices for implementing these processes for API clients. The first option eschews any authentication measures, which is only advisable if the risks of misuse are negligible. The second approach advocates for enforcing authentication, a process that can be securely implemented using protocols such as "OAuth, SAML, Kerberos or LDAP" [35, p. 78].

6.1.1. Tokens

The usage of security tokens, such as JWT is referenced in a large amount of the analyzed publications. This is not a surprise, since the *Token pattern* is a well-established security design pattern used in the microservice architecture [5]. Security tokens have contributed to being a security enhancing factor in regard to identity management in microservices. Billawa *et al.* [58] report that the JWT standard is effective for securely transmitting claims between two parties in a microservice architecture. Waseem *et al.* [73] introduce the *Access and identity tokens* pattern, which utilizes tokens and access-based protocols such as JWT and OAuth 1 & 2. These tokens can carry user data that can be validated, for example, by an edge-service. The authors assert that the implementation of this pattern improves aspects such as "Confidentiality, Integrity, Accountability, Authenticity, and Recoverability" [73, p. 138].

Rezaei Nasab *et al.* [36] discuss various security practices regarding tokens and credentials, evaluating their effectiveness based on interviews with microservice practitioners. The use of JWT for the handling of session revocations and expirations, together with the practice of employing asymmetric encryption for JWT, was considered absolutely useful or useful by almost all the interviewed practitioners. Furthermore, 78.83% of the interviewed practitioners deemed encrypting tokens as either absolutely useful or useful when the tokens are to be exposed outside the microservices boundaries.

6.1.2. User Authorization

Nehme *et al.* [29] advise that to mitigate the *Confused Deputy Problem* in microservices, it is crucial to verify the scope of access tokens in incoming requests, while also recognizing the benefits of having a central authorization service, for example, for audit purposes. Rezaei Nasab *et al.* [36] discovered in their study that the majority of microservice professionals they interviewed considered the implementation of an API gateway in large systems absolutely useful for authorizing requests to microservices.

Waseem *et al.* [73] discuss the *Edge-level authorization* pattern, which enforces authorization at the API gateway and contributes to increase both the security and integrity of the system. The authors also present the *Service-level authorization* pattern, where authorization is enforced at each individual microservice. This is achieved through access control policies, and the authors recommend this pattern in comparison to the *Edge-level authorization* as it increases the granularity of the access control policies, and moreover it improves the availability, security and integrity of the system.

Nehme *et al.* [28] implemented a tool based on OAuth 2 and XACML and two open standards to combat some of the challenges associated with the the implementation and enforcement of access control in microservices. "The architecture involves an Access Control Server (ACS) acting as an OAuth 2 and XACML server, consumer microservices (CMS) containing OAuth 2 client credentials and requiring access to resources, Resource Microservices (RMS) hosting and exposing assets, and a Gateway (GW) to secure each microservice." [28, p. 6] Their implementation mitigates the issue of the *Confused Deputy* problem, and partially mitigates against token theft. Additionally, the system enables the enforcement of fine-grained access policies. The use of the OAuth 2.0 authorization protocol is recommended as an effective mitigation measure for the security smell *Insufficient Access Control* as discussed by Ponce *et al.* [18]. Moreover, Billawa *et al.* [58] mention it as an effective security protocol.

Decentralized Authorization

Ponce *et al.* [18] explain that the *Centralized Authorization* security smell can be addressed by adopting the *Decentralized Authorization* mitigation approach. This method, as they note, depends on a token-based authorization protocol, which allows for authorization to be implemented at the service level. Xi *et al.* [72] present a scheme to enable decentralized access control for microservices and a decision model aimed at achieving efficient access control in a microservice architecture. Their distributed access control scheme

is based on the blockchain, where smart contracts are used to store the policies used for microservices authorization. In relation to the validation of the scheme, the authors stated: "...our scheme not only ensures confidentiality, integrity, and non-repudiation, but also good performance. The results of the experiment also show that our scheme has better efficiency than the classic access control schemes for microservices" [72, p. 8]

6.1.3. User Authentication

Ahmadvand *et al.* [71] state that the authentication of end users must be enforced before any changes can be made to sensitive data, to preserve integrity in a microservice architecture. Additionally, the authors state that configuration changes, e.g. to container images, must be authenticated.

Zdun *et al.* [40] introduce two *Architectural Design Decisions (ADDs)* centered around authentication: *Backend Authentication (BE_AE)*, dealing with interactions between systems like service discovery and microservices, and *Authentication on Paths from Clients or UIs to System Services (CP_AE)*, focusing on interactions between a system and end-users/UIs. Both ADDs incorporate the same Security Tactics, with *Token-based Authentication* (authentication using identity tokens, such as JWT tokens) and *Protocol-based Authentication* (authentication using encrypted protocols, such as SSL) being recommended as the most secure options for both scenarios.

Ponce *et al.* [66] discuss the *Defense-In-Depth* and *Follow The Zero Trust Principle* security practices, in which both recommend using *OpenID Connect* for authentication and OAuth 2.0 for access control.

OpenID Connect

OpenID Connect [31] is described as an effective mitigation strategy to counter the *Unauthenticated Traffic* security smell, as highlighted by Ponce *et al.* [18]. The authors also point out that this authentication protocol is typically used in conjunction with a JSON Web token to transfer OpenID Connect session information. Billawa *et al.* [58] also discuss the recommendations for the use of *OpenID Connect* in their publication, noting that this is supported by several sources in the gray literature.

6.2. Service Mesh, Sidecars and API Gateways

Mitigation strategies related to the *Service Mesh, Sidecars and API Gateways* are discussed in a substantial number of publications (see Figure 6.3), adding up to a total of $(9.6\% + 1.9\% + 5.8\%) = 17.3\%$, with the majority of these publications devoting little focus to the topic (9.6%). However, 5.8% of the publications discuss the topic with a *High* level of focus.

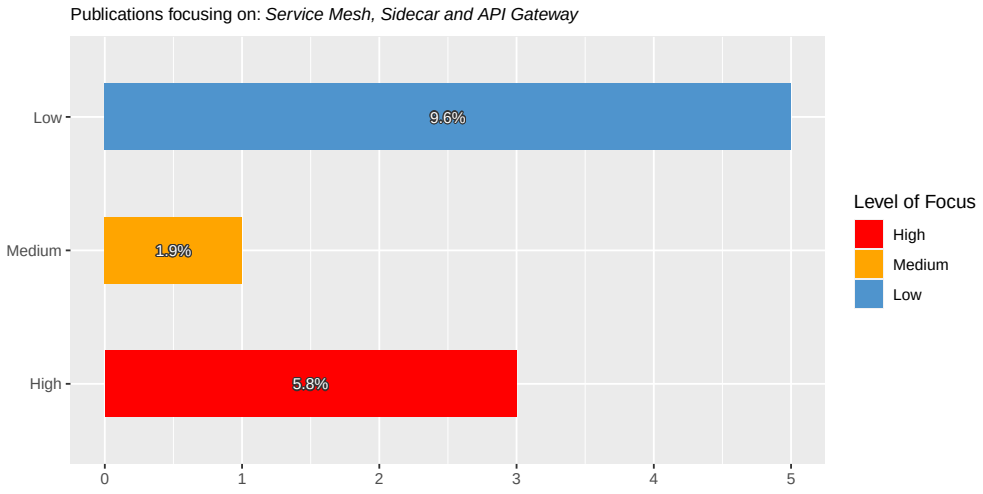

Figure 6.3.: Distribution of Service Mesh, Sidecar and API Gateway

Publications	Level of Focus
[45–47]	High
[41]	Medium
[16, 18, 36, 73, 74]	Low

Table 6.2.: Service Mesh, Sidecar and API Gateway

6.2.1. Service Mesh

A service mesh consists of a few building blocks that together make up this infrastructure layer; the data plane (also called mesh proxy) and the control plane Newman [2, pp. 162-169].

Aktypi *et al.* [45] introduce the *Themis* framework, designed to enable secure peer-to-peer communication and facilitate a secure communication network for a service mesh without the need for a central certificate authority. *Themis* aims to improve security in microservice architectures using a service mesh by addressing

security design flaws in modern service meshes through a dual-layer architecture. The upper layer supports a peer-to-peer network, while the lower layer offers an alternative to the commonly used mTLS protocol for secure inter-service communication and authentication. The framework also includes features for distributed identity management [45]. Developed as a prototype, *Themis* was released as an open source project on GitHub, with its performance overhead evaluated as minimal, averaging only a throughput overhead of 1.24%. However, the GitHub project has remained inactive for the past two years, with limited activity beyond initial input from its creators.

El Malki *et al.* [46] suggest several practices to enhance security in microservice architectures with service meshes. They recommend prioritizing encryption and efficient key management, achievable either via API keys or a central certificate authority in the Control Plane. Additionally, they propose setting up authorization in either the Control or Data Plane. R. Alboqmi *et al.* [47] provides a mean of securing a microservices architecture using self-protection measures in the service mesh. The authors describe self-protection as a means to adapt to security threats autonomously and automatically once they are detected. To enable self-protection in service mesh, the authors overcame the previous limitations of static configuration, by developing a system that dynamically alters traffic routes and controls information flow based on threat assessments and defined preferences. Sedghpour *et al.* [74] emphasize that the use of *extended Berkeley Packet Filter (eBPF)* combined with service meshes is becoming more popular in the microservice architecture. Considered as a lightweight, sandboxed *Virtual Machine (VM)*, *eBPF* is embedded within the Linux kernel, where calls are made through kernel hooks. The authors state that this enables security actions to be performed with low overhead.

6.2.2. Sidecar

Suneja *et al.* [41] explore various methods to securely implement container fusion, such as the *Sidecar* pattern, ensuring that fusion does not increase the overall risk of the system. In their evaluation, they assume the presence of malicious code within the fused container. They also set up a sidecar to securely attach to a microservice but limiting access to critical system components such as disk, memory, network states, and resource statistics. Their demonstration shows that their sidecar implementation does not negatively impact the security of the system, while only introducing negligible performance overhead.

6.2.3. API Gateway and Backend for frontend (BFF)

Several studies [16, 18, 36, 73] point to the positive security attributes that the adoption of the *API gateway* concept has for the microservice architecture. Waseem *et al.* [73] report that implementation of the API gateway pattern improves the Security, Availability, and Portability of a microservice architecture. Similarly to the application of the API Gateway, Waseem *et al.* [73] also reports that employing the Backend for frontend (BFF) pattern enhances the same security properties.

Ponce *et al.* [18] recommend to *Add an API Gateway* to mitigate the risks associated with having publicly accessible microservices, adding that by enforcing security at the API Gateway before the requests reaches the microservices adds an additional layer of security to the architecture, which is not there when microservices are directly exposed to the public environment.

6.3. Secure Code, Design Patterns and Architecture

Secure Code, Design Patterns and Architecture are discussed in several publications (see Figure 6.4) in relation to mitigating security threats for the inter-service security domain. $(7.7\% + 13.5\% + 3.8\%) = 25\%$ of the publications evaluated discuss the use of *Secure Code, Design Patterns and Architecture* to promote security in a microservice architecture, and most of these publications discuss the topic with a *Medium* level of focus.

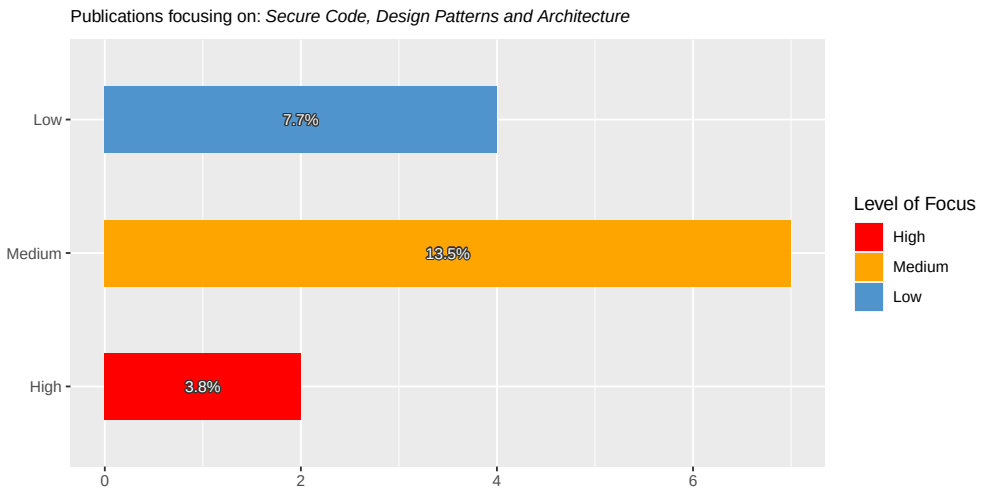

Figure 6.4.: Distribution of Secure Code, Design Patterns and Architecture

Publications	Level of Focus
[53, 75]	High
[24, 35, 40, 58, 66, 67, 73]	Medium
[18, 52, 76, 77]	Low

Table 6.3.: Secure Code, Design Patterns and Architecture

6.3.1. Security by Design and Dev(Sec)Ops

Billawa *et al.* [58] identify *Security by Design* and *DevSecOps* as two of the best practices to improve microservice security. The study also highlights how these two practices are related and how pushing security to the left can be an important tool in order to secure microservices systems. Alshuqayran *et al.* [52] identified a list of 50 tools that are effective in implementing microservice architecture within a DevOps framework, while the most populous category is security tools, where the authors identified 14 tools. Ponce *et al.* [66] recommend adopting the *DevSecOps* practice and implementing *Enact Continuous Security Testing* as strategies to address security issues related to the *Non-Scalable Security Controls* security smell. They explain that *DevSecOps* integrates security considerations early in the development lifecycle, while *Enact Continuous Security Testing* ensures ongoing automated security assessments throughout the lifecycle of the microservice architecture.

6.3.2. Security Design Patterns and Tactics

Bass *et al.* [78] differentiate between design patterns and tactics, explaining that tactics are strategies used to comply with design patterns. As stated by the authors, "patterns package tactics" [78, p. 227], clearly define that patterns are more abstract and harder to adhere to than tactics. Billawa *et al.* [58] state that design patterns are often used to develop secure microservices. The study mentions that patterns such as the *API gateway* pattern, *Command Query Responsibility Segregation (CQRS)* pattern and the *Circuit Breaker* pattern each bring security properties relevant for the microservice architecture. Márquez *et al.* [24] discuss the use of design patterns and tactics in the development of microservices, and particularly those patterns and tactics focused on availability in microservices systems.

Guiding Development of Secure Microservices

Waseem *et al.* [73] discuss how decision models can be used to choose the appropriate design patterns and strategies in a microservice architecture, and also define a decision model to choose the patterns and frameworks directly related to microservice security. The decision model is structured in a way that requires architects to assess which attributes (such as security, confidentiality, integrity, availability, and others) hold the highest priority for the architecture. Each design pattern and strategy are thereby connected to one or more of these attributes, either on a positive or negative level. For example, the *Edge-level authorization* pattern increases the *Security* and *Integrity* of the system when adhered to. Figure 6.5 illustrates the decision model for microservices security, where strategies / patterns are represented within the rectangles. The "+" symbol represents positive attributes, while the "-" symbol represents negative attributes, and the lines between the rectangles symbolize how different patterns / strategies complement each other.

Zdun *et al.* [35] categorize approaches for creating quality-focused microservices APIs through reusable decisions, which encompass design patterns, practices, or specific decisions. Similarly, Zimmermann *et al.* [77] develop a pattern language that is applicable for creating secure microservice APIs, among other uses. Ponce [76] outlines their research approach in developing a taxonomy of security smells and mitigation strategies specific to microservice security. This taxonomy, detailed in Ponce *et al.* [18], provides a set of refactorings that can guide the development of secure microservices. These refactorings are similar to security tactics in their concrete definitions and in the directive manner in which they should be implemented. Nkomo *et al.* [67] provide a series of suggested practices for the secure development of microservices. These recommendations include applying and validating secure software development practices, safeguarding configurations (such as those for containers) during runtime, continuously monitoring assets, employing automated adaptation measures in response to attacks, and thoroughly documenting the security requirements of the microservices architecture.

6.3.3. Validating Adherence to Security Tactics

Zdun *et al.* [40] investigated the utilization of security tactics and assessed the adherence to these in the microservice architecture. The authors highlight the challenges in creating secure microservices, particularly noting the difficulties and potential for errors in manually verifying the system's adherence to various security tactics. They then derived a set of ADDs identified from the reviewed literature. ADDs consist of several

alternative security tactics that can be applied, each with its own benefits and shortcomings. The authors conceptualized their findings into several ADDs each focused on a certain microservice security aspect; observability, secure communication and authentication & identity management. The authors then created metrics to evaluate the different security tactics to automatically determine whether a system conforms to the analyzed ADDs and security tactics.

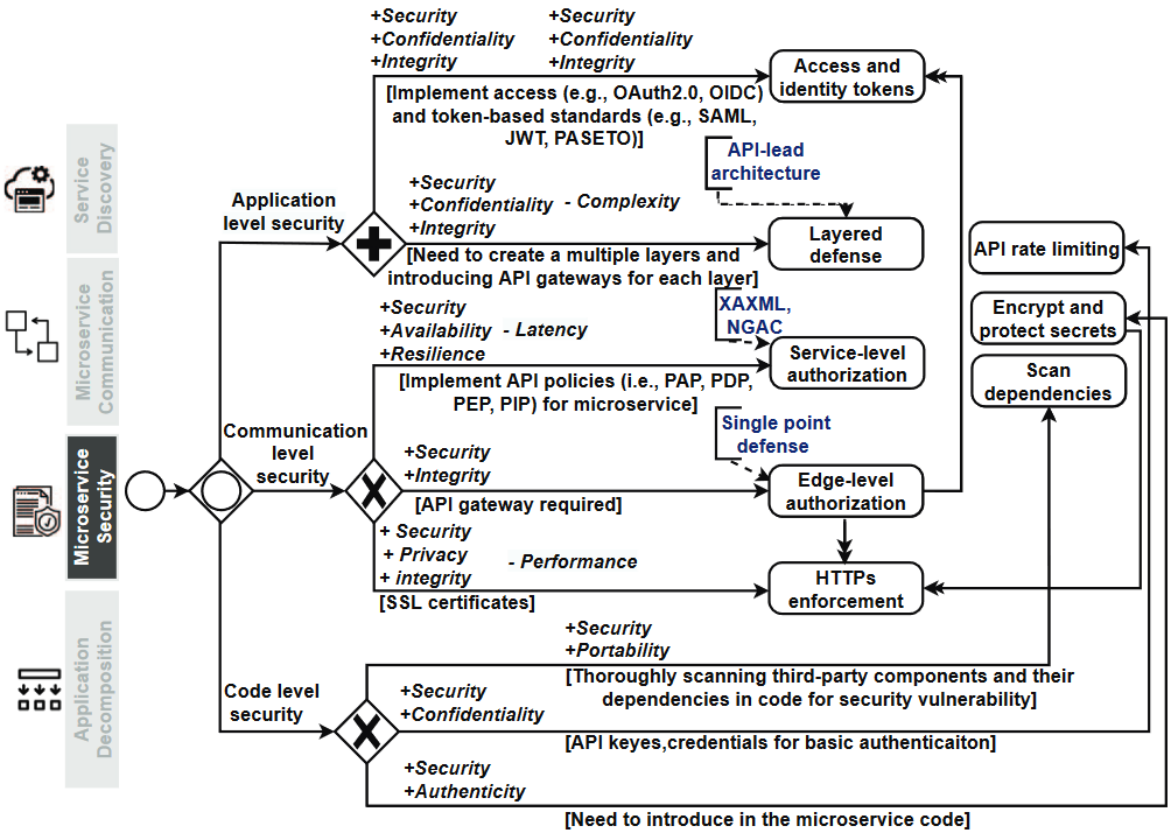

Figure 6.5.: Microservices security decision model, retrieved from [73, p. 139]

6.3.4. Adhering to the Microservice Architecture

Ahmadvand *et al.* [53] introduce a methodology for transitioning a system from a monolithic to a microservice architecture, ensuring both security preservation and optimal scalability. The authors also implemented this methodology in a fictional case study to demonstrate its practical applicability.

6.3.5. Detecting Vulnerable Dependencies

Waseem *et al.* [73] describe how the *Scan dependencies* strategy can be applied in the microservice architecture. This strategy, as described by the authors, is applied to detect vulnerabilities in the development pipeline and codebase.

6.3.6. Security Related Code Identification

Schneider *et al.* [75] introduce a method for generating dataflow diagrams for Java microservices that contain security annotations. When doing this, security-related parts of the source code can be located. In their approach, the authors utilize the snowballing technique, which involves searching for specific keywords within an application's source code and iteratively identifying new keywords through this process. The research validation yielded a precision rate exceeding 90% and a recall rate of 85%.

6.4. Securing Data at Rest

The topic of secure data storage is addressed in a subset of the publications (see Figure 6.6), representing $(7.7\% + 1.9\%) = 9.6\%$ of the total studies analyzed. However, the depth of focus on this topic is predominantly minimal. Within this group, only one study is categorized with a *Medium* level of focus to the topic, as per the categorization criteria. The remaining studies that touch on this topic do so with a relatively lower degree of detail or emphasis.

Publications	**Level of Focus**
[18]	Medium
[58, 71, 73, 79]	Low

Table 6.4.: Securing Data at Rest

6.4.1. Encryption of Data at Rest

The necessity of encrypting data at rest is a key focus of the study by Mateus-Coelho *et al.* [79]. This publication strongly advocates for the use of public and widely recognized cryptographic algorithms. Emphasis

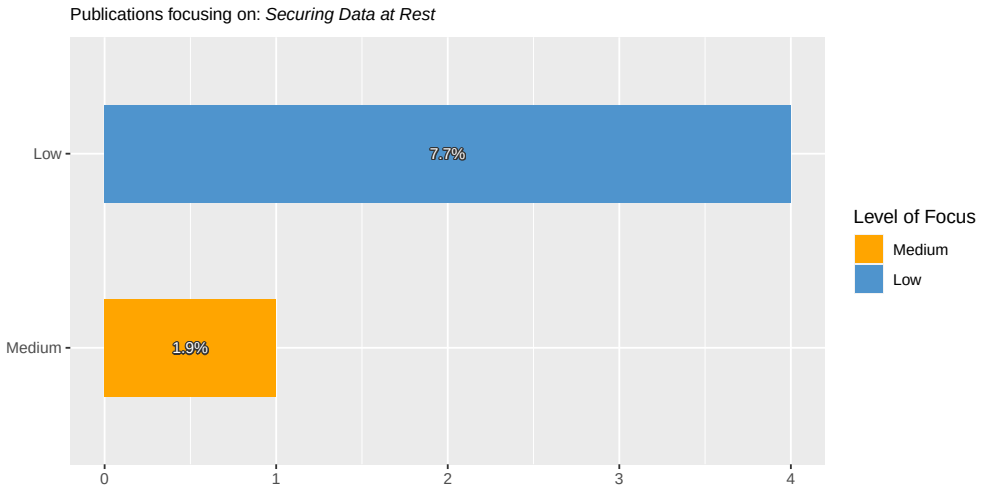

Figure 6.6.: Distribution of Securing data at rest

is placed on the advantages of these algorithms, including regular updates with security patches, comprehensive penetration testing, and ongoing scrutiny by security experts. The study highlights that such rigorous maintenance and validation processes ensure the effectiveness and reliability of these algorithms in protecting data at rest. The study by Ponce *et al.* [18] also endorses the use of established and widely recognized encryption technologies for encrypting data at rest. This recommendation underscores the importance of relying on proven cryptographic methods that have undergone extensive vetting and testing in the security community.

6.4.2. Encrypting and Managing Secrets

The findings presented by Ponce *et al.* [18] delineate the importance of encrypting secrets when stored at rest as a fundamental mitigation strategy. The study further advises against the practice of storing secrets in repositories designated for application code, as well as avoiding their placement in environment variables or in proximity to the application itself. This approach is emphasized as a crucial measure to improve security and prevent unauthorized access to sensitive information. Billawa *et al.* [58] also stress the importance of encrypting the data at rest so that they remain confidential. Waseem *et al.* [73] define the security strategy *Encrypt and protect secrets*, which involves using secret vaults such as HashiVault [80] and Microsoft Azure Key Vault [81] for secret storage. Ahmadvand *et al.* [71] emphasize the importance of secure secret management, particularly in differentiating access between development and production environments. It advocates a system in which developers can access secrets during the development phase, but these secrets

are tightly controlled and restricted in production environments. The research stresses that in production, secrets should be generated and distributed in a confidential manner, ensuring that they are not accessed or exposed to insiders or unauthorized individuals.

6.5. Containerization and Orchestration

Of the analyzed publications (see Figure 6.7), $(5.8\% + 1.9\%) = 7.7\%$ of them discuss mitigative strategies related to *Containerization and Orchestration*, with the majority of these (5.8%) devoting a *low* level of focus to the topic.

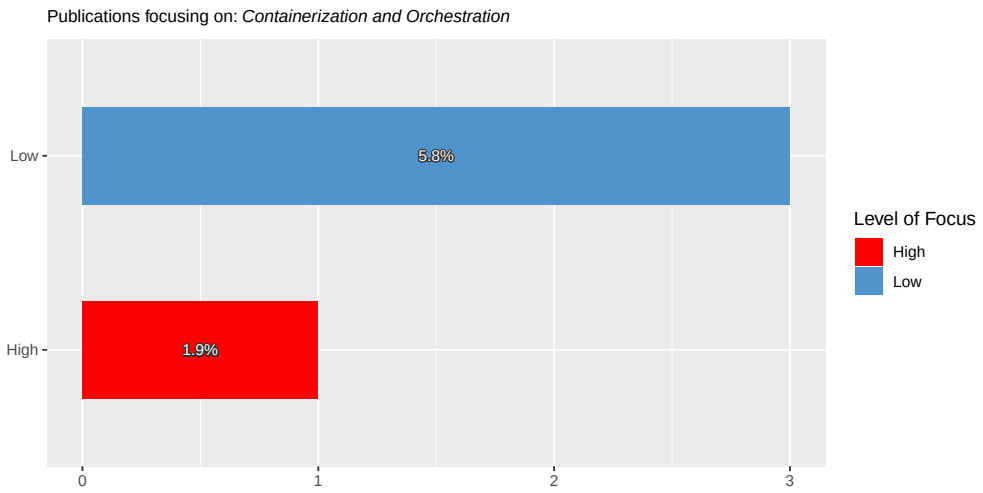

Figure 6.7.: Containerization and Orchestration Distribution

Publications	Level of Focus
[54]	High
[29, 58, 67]	Low

Table 6.5.: Containerization and Orchestration

6.5.1. Container firewalls

Nehme *et al.* [29] suggest that container firewalls should be used to analyze all requests it receives, whether it is originating from the API gateway or other microservices.

6.5.2. Mitigating Vulnerabilities in Containers

Torkura *et al.* [54] address the difficulties in identifying vulnerabilities in container images within microservice architectures and introduce their prototype, *CAVAS (Cloud Aware Vulnerability Assessment System)*, designed to detect and validate these vulnerabilities (including the elimination of false positives). Illustrated in Figure 6.8, the prototype features several components, including a *Policy Store* for storing security policies applied by the *Security Gateway* and modules for vulnerability scanning & validation and a module for analysis of image vulnerabilities *CAVAS* is intended for use in both development and production environments, with the aim of identifying vulnerabilities as early as possible within the Software Development Lifecycle (SDLC).

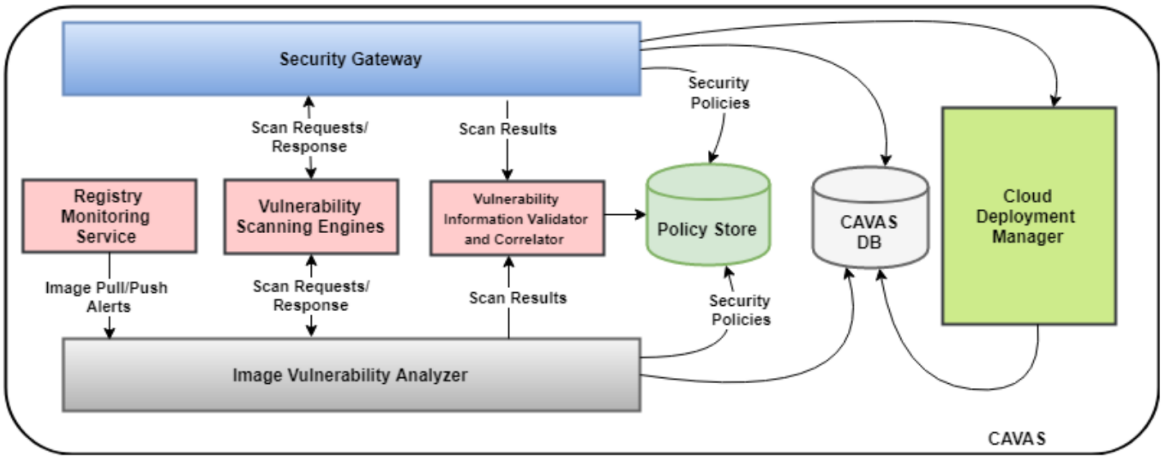

Figure 6.8.: Architecture for the CAVAS prototype, retrieved from [54]

The authors conducted various validation tests of the tool in a lab environment to assess its security qualities. They specifically evaluated the tool's capability in detecting and validating vulnerabilities, as well as its efficiency in doing so. Additionally, they assessed the effectiveness of the tool's security policies. Through these experiments, the authors showed that the tool is significantly more effective than traditional security testing techniques, showing a 31. 4% improvement in the identification of vulnerabilities compared to these conventional methods. The authors also developed and integrated a specialized set of policies into the tool, with the aim of identifying *shared-code* vulnerabilities prior to their deployment in containers.

Nkomo *et al.* [67] present several protection measures for a list of identified security threats. To mitigate the threats of an insecure runtime infrastructure, the authors list several mitigation strategies. This includes vulnerability scanning of containers before deployment to production, creation, and validation of secure

configuration for all infrastructure components (this should be done in an automated manner), and to group containers based on relative sensitivity. Relative sensitivity is used in this study to describe the data and / or behavior sensitivity associated with the containers. Billawa *et al.* [58] identified the security practice of employing *Immutable Containers* as a key best practice in their research. This approach entails restricting updates to containers once they are deployed. Moreover, the authors recommend external data storage separate from the containers to ensure data preservation and easy access if container replacement is necessary.

6.6. Monitoring, Tracing and Logging

Several publications (see Figure 6.9), amounting to $(7.7\% + 13.5\%) = 21.2\%$, discuss the topic of *Monitoring, Tracing and Logging* and the means of using these methods for mitigating security threats in the microservice architecture. However, none of the assessed publications discuss mitigative measures related to this topic with a *High* level of focus.

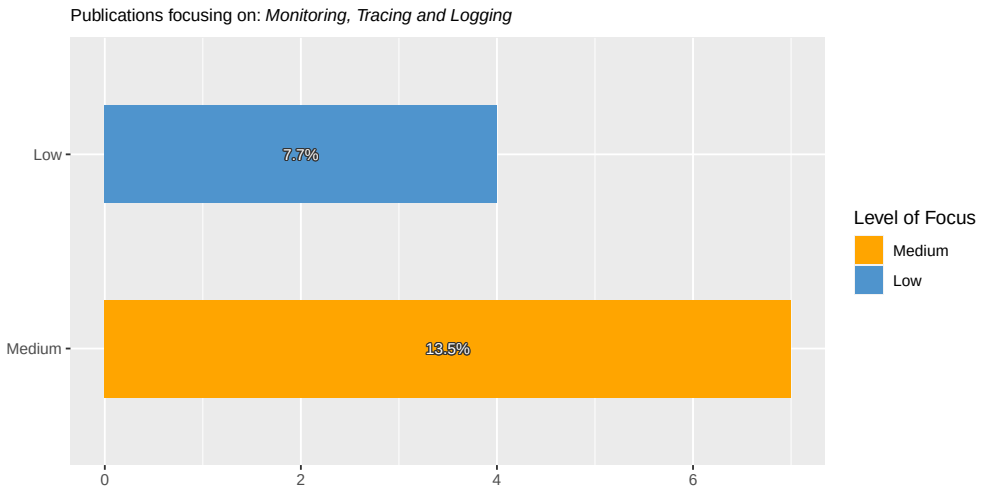

Figure 6.9.: Distribution of Monitoring, Tracing and Logging

Publications	Level of Focus
[38, 40, 43, 54, 56, 68, 82]	Medium
[29, 67, 71, 83]	Low

Table 6.6.: Monitoring, Tracing and Logging

6.6.1. Observability

Zdun *et al.* [40] introduce the *Microservice Observability (OBS)* Architectural Design Decisions (ADD), which encompasses various related security tactics as alternative decision points. In this ADD, the authors recommend observing both internal and external microservices. Observance, in this context, refers to the enforcement of logging, monitoring, and the execution of request tracing.

6.6.2. Security Health Endpoint

The concept of the *Security Health Endpoint* is introduced by Torkura *et al.* [43] as a component of their *Security Gateway* prototype implementation. The authors highlight that this concept improves the monitoring abilities of microservice architectures, ensuring the achievement of environmental awareness. The *Security Health Endpoint* is based on the *Health Endpoint Monitoring* design patter, which describes an API endpoint provided by a microservice. This endpoint allows the retrieval of system information essential to assess the health of the microservice. When following the *Security Health Endpoint* concept, relevant security metrics should be retrievable from endpoint(s) exposed by the microservice(s); for example, this could be implemented as an */security-health* API endpoint. The *Security Health Endpoint* provides various types of information, including, among other things, key insights from the most recent security assessment. This includes details of specific affected components and their related vulnerabilities, along with metrics like CVEs Common Vulnerabilities and Exposures and CWEs Common Weakness Enumeration associated with each vulnerability. The authors point to the benefit of being able to consume this information from other security tools automatically and make an example where firewalls can retrieve updated security information from the individual services to construct automated firewall rules. Torkura *et al.* [54] employ the *Security Health Endpoint* in their *CAVAS* prototype.

6.6.3. Request Tracing

Kalubowila *et al.* [83] introduce an external validation solution aimed at reducing latency within microservice architectures. Their approach involves the use of a trace analyzer, which is designed to detect errors in the architecture. This is achieved through a tracing mechanism that monitors requests as they pass through various services in the microservice architecture, identifying any errors or issues related to latency.

6.6.4. Monitoring

Nehme *et al.* [29] recommend to perform monitoring of every critical part of the microservice architecture. Zeng *et al.* [38] present several security tools that enable microservice monitoring solutions. The authors also recommend to monitor the containers used for deploying microservices. Li *et al.* [56] present the *Security Monitor* security tactic, which is used to monitor the microservice architecture to find anomalies that could cause security issues. Otterstad *et al.* [82] present the *Security Monitor Service*, also referenced in Section 6.9. A key aspect of maintaining a microservice architecture is monitoring the system to detect and respond to anomalies. Ahmadvand *et al.* [71] suggest enabling tracing mechanisms for changes to sensitive data within such architectures. They also emphasize the importance of logging insider activities, such as those of DevOps personnel, to improve security. Nkomo *et al.* [67] advocate for monitoring the behavior of microservices during runtime to ensure system integrity. Furthermore, Sun *et al.* [68] introduce their prototype, *FlowTap*, which is designed for network monitoring in microservice environments. The tool "establishes monitoring relationships between microservices and security monitors, allowing them to enforce policies over the network traffic seen by the microservice" [68, p. 57].

6.7. Inter-Service Authentication and Authorization

In the context of *Inter-Service Authentication and Authorization* (see Section 6.7), a total of $(3.8\%+5.8\%) = 9.6\%$ of the publications address mitigation strategies related to this topic. The majority of these publications discuss the topic with a *High* level of focus, but the overall number of publications addressing the topic is nevertheless relatively low.

Publications	**Level of Focus**
[37, 42, 69]	High
[18, 66]	Low

Table 6.7.: Inter-Service Authentication and Authorization

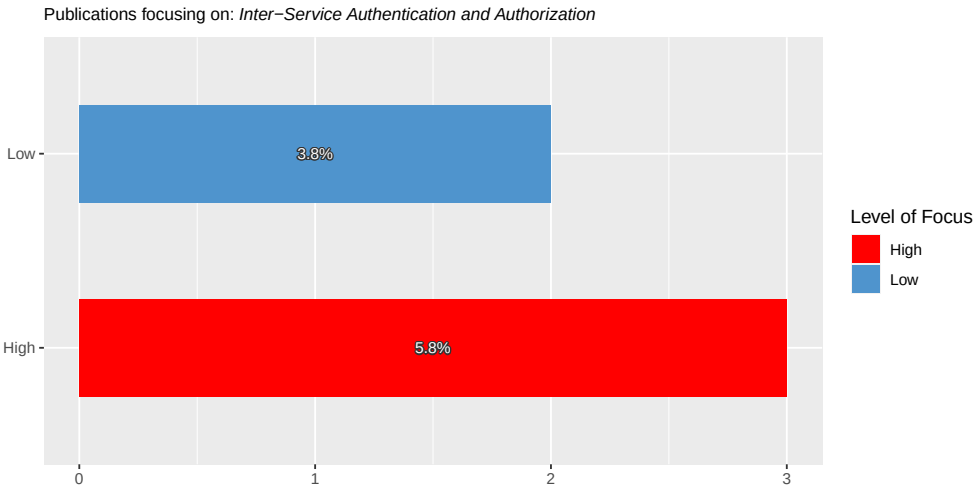

Figure 6.10.: Distribution of Inter-Service Authentication and Authorization

6.7.1. Inter-Service Authentication

Walsh *et al.* [37] examined the current TLS standards used for inter-service authentication and attestation (process of verifying identity & integrity) e.g. Baseline Mutual TLS and Federated Attested TLS-PSK. The authors examined the standards performance using benchmarking tests, but did not assess the relative security level of the standards. In summary, the authors deem all the assessed protocols as capable of providing trustworthiness to a microservice architecture, but with varying degrees of performance overhead.

Mutual TLS (mTLS)

Several studies [18, 42, 66] advocate for the use of mTLS to ensure authentication at the inter-service level. Similarly, Ponce *et al.* [18] advocate the use of mTLS to enable authentication between services to mitigate the security smell *Unauthenticated Traffic*, which is also defined in the study. Yarygina *et al.* [42] also present a prototype for securing and enabling inter-service authentication with mTLS, *Public key infrastructure (PKI)* and tokens. The study discovered that the implications of enabling these security measures provided minimal performance overhead.

6.7.2. Inter-Service Authorization

Li *et al.* [69] introduce the *Jarvis* tool, developed for the automated creation of inter-service authorization policies. This tool examines the architecture of microservices prior to deployment, identifying potential communication patterns among services to form corresponding policies. Furthermore, *Jarvis* actively observes the operational behavior of the microservice architecture, gathering updated information to refine and adapt these policies accordingly. The tool was developed to mitigate the issues associated with having to manually configure inter-service authorization policies, which could be both a tedious and error-prone manual task, especially considering the flexible architecture of microservices constantly in change.

6.8. Secure Communication

Secure Communication mitigation strategies are presented in $(15.4\% + 1.9\% + 1.9\%) = 19.2\%$ of publications studied (see Figure 6.11), although the majority (15.4%) discuss the topic with a *low* level of focus.

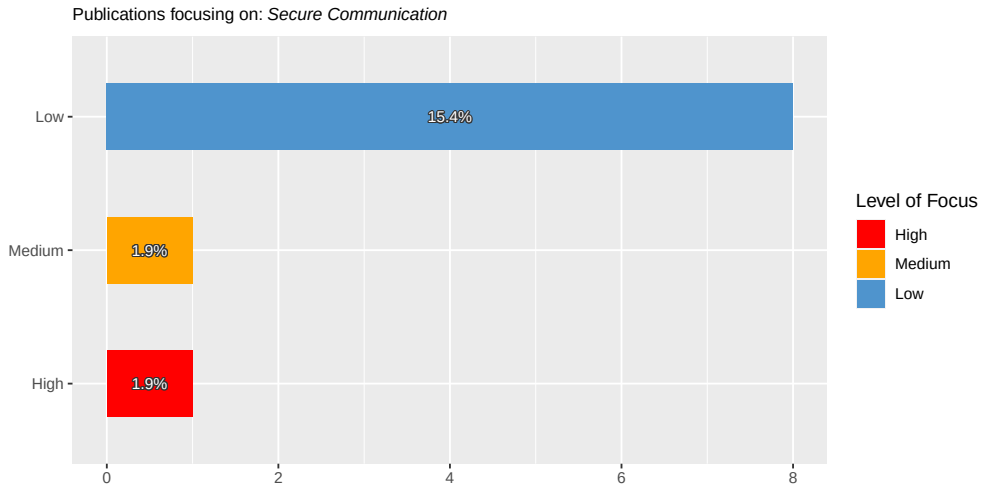

Figure 6.11.: Distribution of Secure Communication

Publications	Level of Focus
[34]	High
[37]	Medium
[18, 36, 39, 66, 67, 70, 71, 73]	Low

Table 6.8.: Secure Communication

6.8.1. Transport Layer Security (TLS)

Several publications discuss the use of TLS for securing inter-service communication [16, 36, 37, 67]. Walsh *et al.* [37] discuss different means of ensuring secure communication between microservices. The publication focuses primarily on the mutual authentication part of the communication, but each of the mechanisms mentioned uses TLS, e.g. Baseline Mutual TLS and Centralized Attested TLS. Nkomo *et al.* [67] recommend the use of TLS as one of the mitigation strategies used to deal with the security threats; *Unauthorized access*, *Insecure application programming interfaces*, *Insecure microservice discovery* and *Insecure message broker*. Rezaei Nasab *et al.* [36] define several security practices to improve security in microservices communication, based on the evaluation of information from the gray literature and interviews with microservices practitioners. One of these security practices advocates the use of TLS protection in service-to-database communication to ensure that the connection is encrypted and cannot be intercepted or tampered with. Ahmadvand *et al.* [71] created a framework aimed at reinforcing integrity protection within microservice architectures. A key requirement of this framework was the ability of services to attest to the integrity of other services, ensuring verification that a service is indeed the entity it claims to be.

6.8.2. Mutual TLS (mTLS)

Ponce *et al.* [18] suggest that the security issue of non-secure inter-service communication can be mitigated using Mutual TLS (mTLS) for inter-service communication. The authors state that the use of mTLS helps ensure that traffic is encrypted bidirectionally and cannot be intercepted or altered. Ponce *et al.* [66] also recommend using mTLS for the same purpose. Miller *et al.* [70] state that the use of mTLS for communication between pods (group of containers on the same Kubernetes host) contributes to achieving the security of data in transport.

6.8.3. HTTPS and FTPS

Waseem *et al.* [73] present the *HTTPS enforcement strategy* that encourages the use of the HTTPS protocol, as opposed to HTTP, for inter-service communication. This strategy should be used to maintain an encrypted connection between the services, which is not possible using only the native HTTP protocol. Also, Chondamrongkul *et al.* [39] recommend using encrypted protocols such as HTTPS, or other encrypted protocols such as FTPS for communication in a microservice architecture, but emphasizes that this should be

done when communication is done on a public network, to prevent data tampering and eavesdropping.

6.8.4. API Security

Waseem *et al.* [73] describe the *API rate limiting strategy*, which should be applied to hinder adversaries from performing effective brute-force attacks, excessive API calls, etc. Genfer *et al.* [34] conducted a study on identifying and preventing data exposure in microservice APIs by determining the minimum amount of data required for successful exchanges. The objective of their research was to identify the optimal data threshold as a means to reduce the risk of data breaches. The study incorporated a method to monitor traffic between APIs using a source code detector, which the authors developed in a previous research project, as cited in [84]. The authors used the definition "All incoming data received by an API that is neither directly consumed nor routed to another API can be considered excessively exposed." [34, p. 4] to serve as a criterion for identifying data overexposure. Based on this definition and their source code analyzer tool, the authors were able to identify several cases of overexposure in two public and private microservices projects (Lakeside Mutual [85] and eShopOnContainers [86]). However, it is important to note that not all aspects of data exposure fell within the scope of their study.

6.9. Infrastructure Defense and Intrusion Mechanisms

Security measures and mitigation strategies related to *Infrastructure Defense and Intrusion Mechanisms* are presented in $(7.7\% + 3.8\% + 11.5\%) = 23\%$ of the assessed publication (see Figure 6.12). The majority of these publications (11.5%) have a *High* level of focus on the topic.

Publications	**Level of Focus**
[51, 59, 65, 71, 82, 87]	High
[36, 42]	Medium
[15, 24, 39, 56]	Low

Table 6.9.: Infrastructure Defense and Intrusion Mechanisms

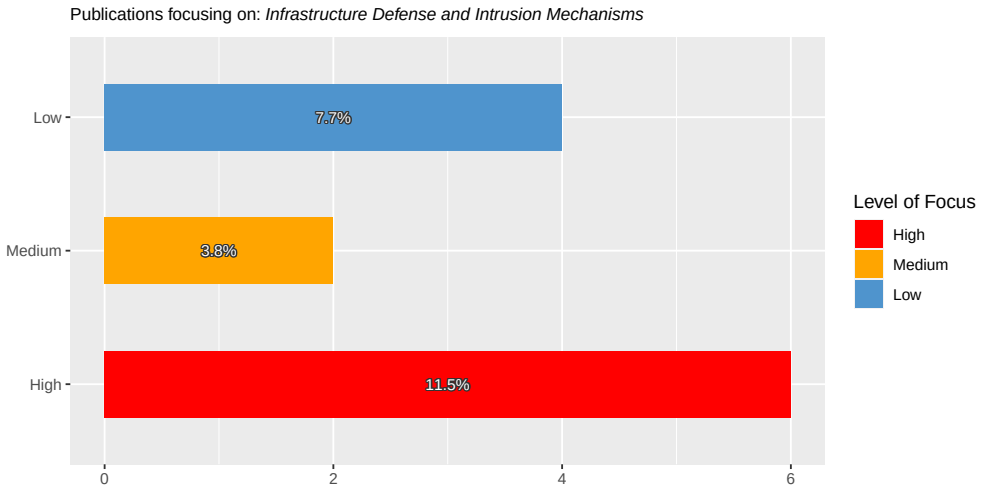

Figure 6.12.: Distribution of Infrastructure Defense and Intrusion Mechanisms

6.9.1. Databases and Environments

Rezaei Nasab *et al.* [36] discuss several security practices regarding databases and environments in microservice architecture. Two of these practices were rated as *Absolutely useful* or *Useful* by the majority of microservice practitioners surveyed. The first practice highlights the importance of implementing more stringent security policies in production environments compared to development environments, while still underscoring the necessity of security in development settings. The second practice emphasizes always enforcing authentication to databases [36, p. 15].

6.9.2. Intrusion Response

Li *et al.* [56] present the *Intrusion Defender* security tactic, which involves identifying the occurrence of insecure states in a microservice(s). The tactic should choose between several means of reacting to insecure states:

- rollback/restart: destroy and spin up a new service instance.
- isolation/shutdown: exclude the insecure service by shutting them down for good
- diversification: change host characteristics by recompiling binary, rewrite binary or by moving service to a host that have different characteristics.
- n-variant service scaling: add additional services using diversification technique. Inconsistencies in services can be compared and infected services can be detected.

Yarygina *et al.* [65] developed a system for intrusion response, capable of responding to network attacks using *game theory* in real-time. Britannica describes game theory as a "branch of applied mathematics that provides tools for analyzing situations in which parties, called players, make decisions that are interdependent." [88]. The system presented by Yarygina *et al.* [65], works by removing, restarting, and relocating microservices at runtime and uses the same defender actions as Li *et al.* [56], including the additional actions in the following list:

- diversification through cloud provider: relocate service to another cloud provider
- split or merge services: split or merge service instances on functional code level using dedicated tools (not yet in existence)
- isolation/shutdown: isolate or shutdown service instances

Chondamrongkul *et al.* [87] propose an approach for conducting automated security analysis within a microservice architecture. They claim that their solution can automatically identify security threats and demonstrate attack scenarios along with the potential impacts of these attacks.

6.9.3. Integrity Protection

Ahmadvand *et al.* [71] defined six requirements for maintaining integrity in a microservice architecture, and additionally developed a framework for microservice integrity protection. Their requirements for integrity protection can be listed as follows:

- "Enable authentication/tracing of sensitive data changes by end user" [71, p. 8]: The need for this requirement is expressed by the lack of observability of data changes that is commonly observed.
- "Protect confidentiality of system secrets in all processes" [71, p. 8]: This includes securing secret management and secret generation.
- "Collect unforgeable evidence of insiders' activities" [71, p. 8]: This involves recording activities and securing them in tamper-proof storage.
- "Detect tampering with static artifacts such as config, script files and binaries" [71, p. 9]: Configuration changes should be authenticated
- "Raise the bar against program tampering attack (intra-service integrity protection)" [71, p. 9]: A large number of observed amounts of threats against service's integrity signifies the importance of this requirement.

- "Enable services to attest to the integrity of their recipients and senders (inter-service integrity protection)" [71, p. 9]: A lack in the current means of verifying request integrity signifies the need for this requirement.

6.9.4. Circuit Breaker

The *Circuit Breaker* pattern is a proxy which rejects requests after numerous consecutive failures [5, p. 78]. Márquez *et al.* [24] state that the *Circuit Breaker* pattern can be used to time out calls to prevent system failures and Yarygina *et al.* [42] recommend using the *Circuit Breaker* pattern to mitigate attacks such as DoS attacks. The *Circuit Breaker* pattern enables partial failures, e.g. to have some microservices fail while others remain in a healthy state, which would prevent the entire microservice architecture from failing.

6.9.5. Firewalls and Packet Inspection

Chondamrongkul *et al.* [39] suggest that a firewall should be used to restrict network traffic flowing from the public to publicly accessible microservices.Soldani *et al.* [15] state that the use of the *API Gateway* pattern simplifies firewalling in a microservice architecture, as requests from the public must be routed through the gateway before they reach the microservices.

6.9.6. Diversification of Microservices

The ability to have a technologically diverse environment when adhering to the microservices architecture is mentioned as one of the pro's of this architectural pattern by Fowler [27], as each microservice can be deployed using different programming languages, frameworks, and other technologies. In a microservice architecture, the only component requiring uniformity, namely the inter-service communication, can also be effectively handled or proxies by a specialized service, like an API gateway.

Yarygina *et al.* [42] advocate for the deployment of heterogeneous microservices to enhance diversity within the microservice architecture. This approach helps mitigate attacks such as low-level exploitation and shared code vulnerabilities, which are more prevalent in homogeneous environments. This recommendation aligns with the insights of Otterstad *et al.* [82], where the authors developed a *Security Monitor Service* to enable software diversity in a microservice architecture at runtime, as well as identifying anomalies in the system

and acting on this, for example, by shutting down infected microservices. Similarly, Torkura *et al.* [51] present the concept of *Moving Target Defense (MTD)* to mitigate shared code vulnerabilities. Their concept, *MTD*, enables diversification at runtime in order to promote uncertainty in the architecture to increase the effort needed to perform successful attacks against the architecture. This is achieved through their techniques for *Automatic Code Generation*, where code is transformed into other programming languages at runtime, using their *Diversification Engine*.

6.9.7. Generation of Attack Graphs

Ibrahim *et al.* [59] introduce a solution for creating automatic attack graphs as a means to identify network threats within the microservice architecture. Their approach utilizes microservices deployed in containers, rather than traditional network nodes, in the construction of attack graphs. Designed for integration into a *Continuous Integration and Continuous Delivery (CI/CD)* pipeline, their method aims at assisting microservice practitioners in discovering potential attack paths in their deployed microservice architectures, to mitigate the risks associated with inter-service pivoting and lateral network movement.

6.10. Security Perimeters and Network Segmentation

The use of *Security Perimeters and Network Segmentation* to mitigate security threats are discussed in $(5.8\% + 1.9\%) = 7.7\%$ of the publications (see Figure 6.13), with the majority of these studies devoting a *Low* level of focus to the topic.

Publications	**Level of Focus**
[82]	Medium
[36, 66, 79]	Low

Table 6.10.: Security Perimeters and Network Segmentation

6.10.1. Private Microservices

Rezaei Nasab *et al.* [36] describe two practices that deal with how *private* microservices should be isolated from other networks. Private microservices are in this context described as internal microservices and should

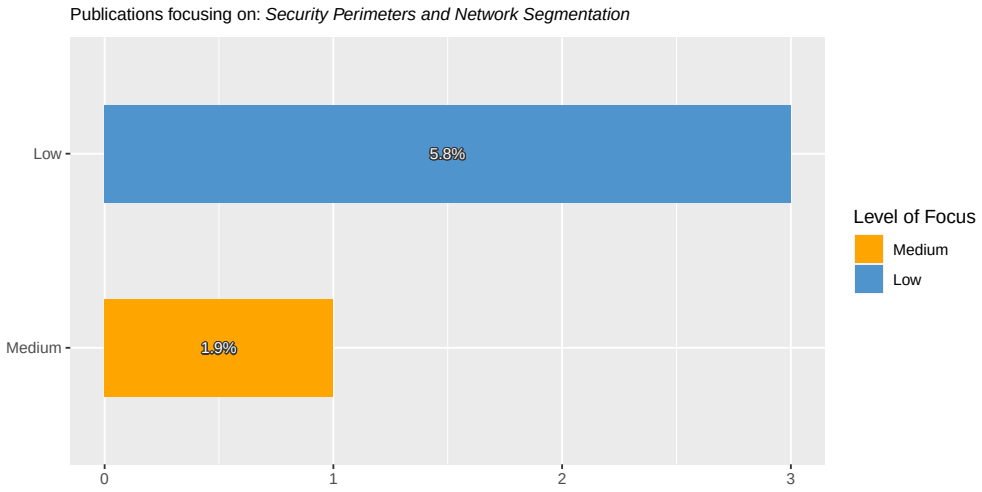

Figure 6.13.: Distribution of Security Perimeters and Network Segmentation

only be accessible by a set of end users and/or applications. The first practice states that it should not be possible to check the existence of a private microservice from an external microservice. The second practice states that service grouping should be applied to microservices to restrict communication and visibility to the member services in the service group. Both of the security practices were deemed as "Absolutely Useful" or "Useful" by most of their study respondents.

6.10.2. Network Segmentation

Mateus-Coelho *et al.* [79] suggest network segregation as a countermeasure against security threats in microservices. According to the authors, distributing microservices across various networks or subnets, managed through firewalls or filtering rules, is an effective strategy to improve security. However, the authors do not describe how microservices should be effectively segregated to increase system security.

Ponce *et al.* [66] additionally state that network segmentation should be applied to enforce the *Zero-Trust Principle*, also called the *Zero-Trust Security Model*. Adherence to this model implies that you assume that the environment has already been breached. Subsequently, the assumption implies the consideration of a potential adversary who attempts to eavesdrop on network traffic, establish unauthorized connections, or engage in other malicious activities [2, pp. 366-367]. Ponce *et al.* [66] recommend to perform network segmentation to adhere to the *Zero-Trust Principle*, and states that some components should be isolated to increase the overall attack surface of the system. However, the authors do not state how the network should be segmented or how to determine which microservices should be isolated.

6.10.3. Microservices Isolation

The concept of isolating microservices is discussed in detail by Otterstad *et al.* [82]. The authors examined how the isolation of microservices, together with the divergence of software, could mitigate the exploitation of system layers, also known as *low-level exploitation* in the publication. The study discusses the use of automatic microservice isolation when anomalies, which could pose potential security risks, have been detected by a *Security Monitor Service*. The publication suggests isolating a single node or multiple nodes reporting similar anomalies to protect the microservice architecture. The authors argue that automated isolation, combined with software diversity, improves defense against low-level exploitation.

6.11. Security Approaches and Models

As seen in Figure 6.14, $(5.8\% + 1.9\% + 1.9\%) = 9.6\%$ of the assessed publications discuss means of mitigation security threats by using *Security Approaches and Models*, and the majority of these (5.8%) discuss the topic with *low* level of focus.

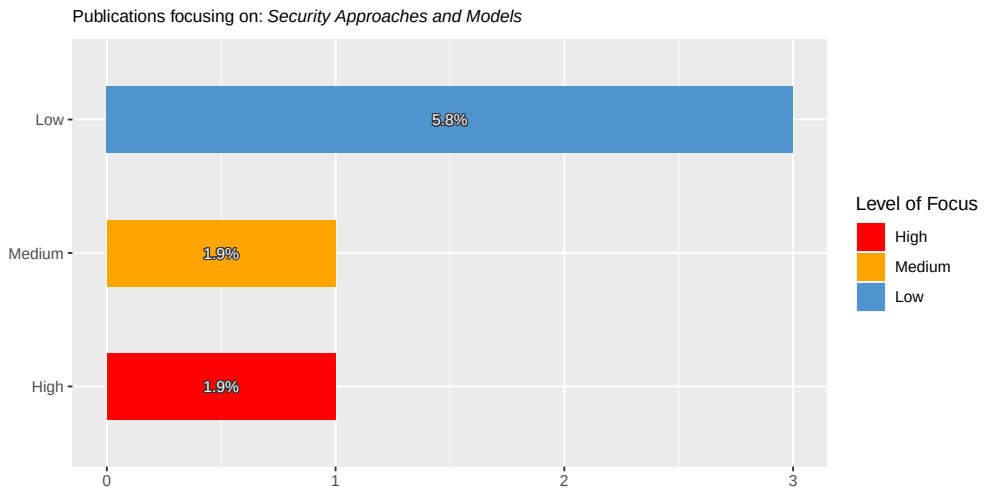

Figure 6.14.: Distribution of Security Approaches and Models

6.11.1. Concept of Least Privileges

Ponce *et al.* [18] recommend to *Follow the Least Privilege Principle* as a security refactoring strategy to mitigate the security smell of granting *Unnecessary Privileges to Microservices*, identified by the same au-

Publications	Level of Focus
[66]	High
[70]	Medium
[18, 58, 73]	Low

Table 6.11.: Security Approaches and Models

thors. Newman [2, p. 347] describe the *Minimum Privilege Principle* as a way to ensure that microservices are given the minimum privileges required to fulfill its tasks. Billawa *et al.* [58] also mention the concept of least privileges as a security best practice that is recommended to adhere to when developing microservices.

6.11.2. Defense-in-Depth

Newman [2, pp. 347-346] describe the principle of *Defense-in-Depth* as a strategy in which defensive measures are positioned at several locations in the microservice architecture. Some of the ways *Defense-in-Depth* can be achieved is by performing network segmentation, limiting the privileges of microservices, and enforcing authorization on a service level. Ponce *et al.* [66] recommend following the *Defense-in-Depth* strategy in order to mitigate the *No Layered Defense* security smell, which as the name describes, occurs when there are not enough layers of security defense measures in place. The authors emphasize the effectiveness of the *Defense-in-Depth* strategy, particularly in facilitating the implementation of adequate defense measures in the outermost layer of a system [66] as cited in [12]. The authors also associate a set of recommendations with the practice of implementing *Defense-in-Depth*, which briefly summarized includes; position every service behind a firewall, use an API gateway to enforce security on incoming requests to a service, use mTLS, OAuth 2.0 and OpenID Connect to enforce encryption and access control between services, follow the *Least Privilege Principle* as discussed in Section 6.11.1, secure all sensitive data through encryption, verify all input received by services for validity, and finally enforce observability measures such as monitoring and enforce logging of information. Furthermore, Billawa *et al.* [58] categorize the practice of *Defense-in-Depth* as one of the best practices for security in a microservice architecture. Waseem *et al.* [73] reference the practice of *Defense-in-Depth* as a design pattern, with the name of *Layered Defense*, stating that it increases both security, confidentiality, and integrity in the microservices architecture. The pattern is implemented through the use of multiple API layers and API gateways, where each layer enforces specific authentication and authorization rules.

6.11.3. Zero-Trust Principle

Ponce *et al.* [66] recommend applying the *Zero-Trust Principle* to mitigate the security smell termed *Trust The Network*. The authors state that all their examined studies discussing this security smell agree that it can be mitigated by applying the *Zero-Trust principle*. The authors also associate a set of recommendations with the practice of implementing the *Zero-Trust Principle*, which briefly summarized includes; use mTLS between services, the use of OpenID Connect to verify identity at the edge, use OAuth 2.0 to enforce authorization at a service level, and enforce network segmentation. In the study by Miller *et al.* [70], the authors follow the principles of zero-trust to define a secure system in which data exchanges between untrusted parties can be performed.

6.12. Security Policies

Among the analyzed publications (see Figure 6.15), $(3.8\% + 15.4\% + 1.9\%) = 21.1\%$ discuss various means of using security policies for improving security in the microservices architecture and implement mitigative measures against inter-service security threats. The vast majority (15.4%) of these publications discuss *Security Policies* with a *Medium* level of focus, as is the case in 8 of the assessed publications.

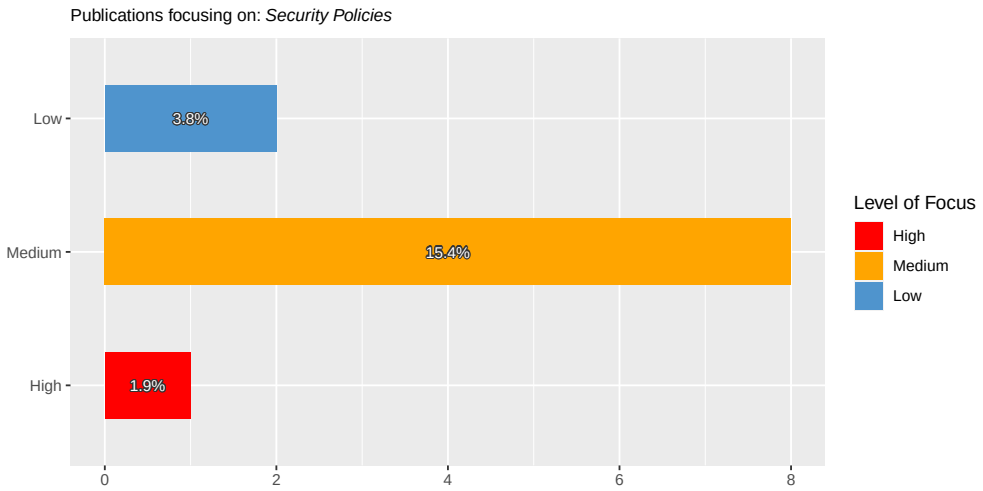

Figure 6.15.: Distribution of Security Policies

Publications	Level of Focus
[54]	High
[28, 43, 60, 68–70, 72, 73]	Medium
[15, 56]	Low

Table 6.12.: Security Policies

6.12.1. Security Policies

Torkura *et al.* [54] describe their tool, *CAVAS*, which features a *Security Gateway* module functioning as a Security Enforcement Point (SEP). This module is capable of enforcing policies for automated vulnerability assessments of container images and facilitating continuous security assessments of microservices, specifically tailored to the microservices' specific technology stack. The *CAVAS* tool is based on foundational concepts previously established by the same authors in a previous research publication [43].

Li *et al.* [56] emphasize the need for security policies for the assessment of certain vulnerabilities and areas of the network. Soldani *et al.* [15] state that one of the gains of microservices is the possibility to enforce fine-grained policies, by using hierarchical groups of subsets of microservices.

6.12.2. Access Control Policies

Xi *et al.* [72] implemented a system for decentralized access control, using policies stored on the blockchain. This approach aims to enforce "distributed and consistent policy management," as stated in their work [72, p. 7].

Nehme *et al.* [28] utilize *eXtensible Access Control Markup Language (XACML)* to create access control policies in their security solution to promote fine-grained access control in microservices. XACML is an open standard markup language for creating policies [12, p. 469]. Waseem *et al.* [73] discusses access control policies in relation to the *Service-level authorization* pattern, which it recommends as this pattern provides more control over the enforcement of access control policies. The authors also state that the pattern includes several API policies, which can be applied in different ways. However, the authors do not note the security implications related to the different ways of applying these policies. OWASP [89] define these policies as follows:

- "Policy Administration Point (PAP): Provides a user interface for creating, managing, testing, and debugging access control rules.
- Policy Decision Point (PDP): Computes access decisions by evaluating the applicable access control policy.
- Policy Enforcement Point (PEP): Enforces policy decisions in response to a request from a subject requesting access to a protected object.
- Policy Information Point (PIP): Serves as the retrieval source of attributes or the data required for policy evaluation to provide the information needed by the PDP to make decisions." [89]

Li *et al.* [69] implemented a feature in their tool *Jarvis* for defining fine-grained access control policies, using inter-service interactions extracted from the source code of microservice architectures. In a different approach, Miller *et al.* [70] developed a workflow employing sidecars to enforce policies, in line with the zero-trust security model. Meanwhile, Sun *et al.* [68] introduced *FlowTap*, a tool for enforcing policies based on inter-service network traffic. Additionally, Li *et al.* [60] developed *AutoArmor*, automating the generation of inter-service authorization policies, proving to be effective with minimal overhead.

7. Discussion

In the categorization process for both Chapter 5 and Chapter 6, a noticeable trend emerged, showing that the majority of the reviewed publications placed greater emphasis on describing mitigation strategies rather than elaborating on specific security threats for the microservice architecture. When security threats were addressed, they were generally given a low level of focus, as established through our categorization methodology (see Section 3.7).

7.1. Security Threats

Regarding security threats, we noticed that a significant number of publications addressed issues related to the *Security Perimeters and Attack Surface* 5.10 in microservice architectures. However, most of these publications only discussed the topic at an abstract level, lacking in-depth analysis and detailed exploration.

Furthermore, many of the assessed publications focused on cloud technologies such as *container and orchestration threats*, as discussed in Section 5.8. Microservices, often used alongside these technologies, present unique security challenges. The publications assessed explored issues related to both the technological diversity in microservice architectures and the use of homogeneous microservices. These two concepts, though seemingly contradictory, suggest that there is no definitive approach to designing secure microservices. Technological diversity in microservices can lead to increased complexity and a broader attack surface, especially due to third-party vulnerabilities. For instance, deploying 300 unique microservices using 5 programming languages will likely result in a greater number of libraries and software packages compared to a monolithic application that utilizes a single programming language. On the flip side, if these 300 microservices are built using the same technology stack and are horizontally scaled, the homogeneity of these services could facilitate shared-code vulnerabilities, potentially leaving every microservice vulnerable to the same attacks.

7.2. Mitigation Strategies

In the area of mitigation strategies, we observed more mature research, as opposed to Chapter 5 and were able to collect more detailed and valuable content pertinent to facilitation of security measures and mitigation of security threats. Specifically, topics like *Secure Code, Design Patterns, and Architecture* (see Section 6.3), *User Authentication and Authorization* (see Section 6.1) and *Infrastructure Defense and Intrusion Mechanisms* (see Section 6.9) were covered by a large number of evaluated publications.

For topics addressing mitigation strategies, we also observed more mature and detailed research, e.g., for the subtopics of runtime diversification of services, as addressed in the works by Yarygina *et al.* [42] and Otterstad *et al.* [82] in Section 6.9. Additionally, it was noted that many of the publication authors used conceptual frameworks to address microservice security. These frameworks span a variety of approaches, such as design patterns, security tactics, and mitigation decisions (refer to Section 6.3), reflecting efforts to systematically address and mitigate inter-service security threats through well-defined mitigation strategies.

7.3. Systematic Map of Primary Security Focus

To analyze the primary security focus in the reviewed publications, we created a systematic map that maps the primary security focus (either security threats or mitigation strategies) of each publication with its corresponding research and contribution facets, as portrayed in Figure 7.1.

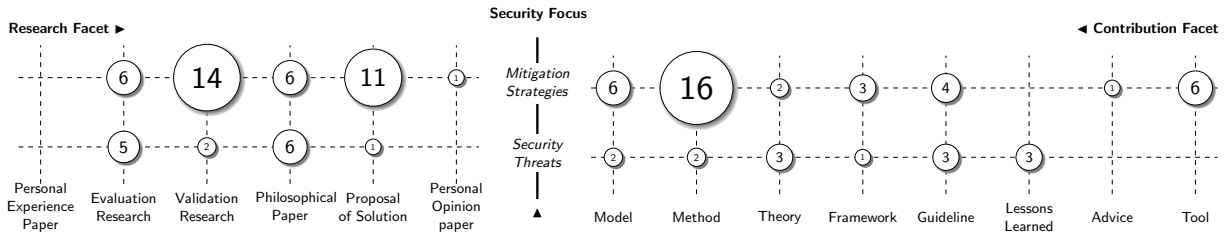

Figure 7.1.: Systematic map of publications main security focus

The systematic map reveals a predominant focus on mitigation strategies over security threats in most publications. Furthermore, within this area of security focus, *Validation Research* and *Proposal of Solution* emerge as the most commonly pursued research facets. This trend indicates a greater emphasis on developing and validating solutions to address security threats, rather than merely identifying or analyzing security threats in the microservice architecture. For publications focusing primarily on inter-service security threats,

we observe that the majority of them are *Philosophical Papers* and *Evaluation Research* contributing several types of work, with the vast majority of it being *Lessons Learned* and *Theory*.

8. Threats to Validity

8.1. Internal Validity

Hannousse *et al.* [57] characterizes internal validity threats as issues potentially arising from the data collection process from the evaluated studies. To mitigate such threats, particularly those related to selection bias and the use of specific terminology in database searches, we employed reverse snowballing. This method was used to address the risk of missing relevant studies due to the varied use of the term "inter-service" and its derivatives, which may not be uniformly used by all authors in the field. This process, as detailed in Section 3.5 and Section 3.6, aimed to ensure a more comprehensive coverage of the literature. Additionally, the search strategy and terms employed in this thesis were meticulously designed to enhance reproducibility. We also integrated and adapted selected methods from the PRISMA framework, further strengthening the reproducibility and reliability of our research.

Publications that were not relevant or lacked sufficient detail on the research topic were initially labeled as "Irrelevant." However, upon a closer review of these exclusions, it was evident that a majority of the papers fell into this category. To enhance transparency and address potential threats to validity, we re-evaluated these excluded papers, categorizing them under three more specific criteria: "Insufficient Microservices Focus", "Limited Security Discussion", and "Off-topic Domains". The categorization based on the research and contribution facets was performed by a single reviewer, due to the nature of Master's theses. Hence, there is a bias in the categorization of the publications for Chapter 4. The same applies to the categorization of publications for Chapter 5 and Chapter 6. Ideally, multiple authors should have categorized the publications and calculated the mean difference between the categorization results.

8.2. External Validity

Othmane [90] defines external validity as the extent to which results can be generalized. The categories used for grouping publications in this thesis are derived directly from the topics discussed in these publications. As the field of microservices security evolves, new categories may need to be created or existing ones modified. Additionally, the focus on inter-service security in this thesis may not align directly with the broader context of microservices security. Emphasizing the inter-service aspect means that the results and guidelines presented here might not be entirely applicable to the general domain of microservices security.

9. Conclusion and Future Work

This section includes a conclusive summary of the topics covered throughout the thesis, as well as our identified research gaps and recommendations for future work in the area.

9.1. Conclusion

The microservices architecture provides a convenient and modular approach for configuring large software systems, but also provides complexity that needs to be handled and addressed, especially since this complexity also extends to the challenges in implementing security effectively. As communication transitions from local to inter-service, new security challenges arise, requiring the adoption of suitable security mechanisms to ensure resilience. The field of inter-service security, unique to distributed architectures like the microservice architecture, presents distinct challenges for developers and security professionals accustomed to monolithic architectures. They may therefore encounter difficulties in understanding the specific security threats related to distributed systems and in identifying effective strategies for mitigating and reducing these risks.

This thesis presents the findings of a systematic literature review focused on the security threats and mitigation strategies specific to inter-service communication in microservice architecture. We conducted an analysis of 52 publications filtered according to defined criteria. The publications were retrieved from five databases and further complemented by one iteration of reverse snowballing to include all relevant studies. These publications were categorized on the basis of their research facets, contribution facets, publication venues, and publication channels, as well as their year of publication. Our findings indicate that most of these publications primarily contribute methods, models, and guidelines. We observed that most publications involved research with validation & evaluation, and we anticipate a continued increase in publications in the field of microservice security. Furthermore, we noted that conference papers make up the bulk of

these publications, with a wide variety in publication venues, reflecting the diverse platforms used for disseminating research in this area.

In our analysis of the selected publications, we derived several categories related to inter-service security threats and mitigation strategies. Each publication was assigned to one or more of these categories. Regarding the security threats reported, we found that a significant number of these threats are connected to the security perimeters and attack surface of the microservice architecture, as well as aspects of containerization and orchestration. Moreover, a notable theme in several publications is the insufficient implementation of monitoring and intrusion detection mechanisms. Additionally, we categorized the publications based on their level of focus on specific topics to better gauge the granularity of research conducted in various areas. We found that the overall level of focus on reported security threats in the assessed publications is relatively low, indicating a shortfall in comprehensive analysis of these threats. This is especially true in the realms of inter-service authentication & authorization, and inter-service communication, where in-depth exploration is particularly lacking.

Regarding the identified mitigation strategies, our analysis indicates a generally higher level of focus on the various derived mitigation categories, in comparison to the categories related to security threats. In particular, topics such as infrastructure defense, monitoring, tracing and logging, and secure coding, design patterns, and architecture have been extensively discussed in several publications, each receiving considerable attention and focus. In relation to mitigation strategies, we also identified more in-depth and advanced research, including sophisticated methods for modifying microservices systems at run-time to enhance their resilience against attacks.

To gain a clearer understanding of the research conducted on security threats and mitigation strategies in microservice architectures, we also developed a systematic map. This map assessed the main security focus of the publications. The results revealed that the vast majority of publications primarily concentrate on mitigation strategies. Within this focus, most studies are categorized as validation research and solution proposals, with their key contribution being the development of methods.

9.2. Future Work

Future research could greatly benefit from conceptualizing identified security threats and mitigation strategies in relation to each other, similar to the categorization performed by Ponce *et al.* [18], where security

issues and mitigation means were classified as security smells and refactoring with clear relationships. Although time constraints prevented this in our current study, such an analysis would be valuable in determining effective mitigation strategies for each specific security threat. Further exploration could involve validating these strategies through simulated experiments, interviews with microservices practitioners, or other methods to establish a hierarchy of effectiveness. In addition, it would be insightful to evaluate the identified security threats based on predefined criteria, such as the potential risk they pose to organizations and the effectiveness, complexity and potential side effects of the corresponding mitigation strategies.

During our research, we also identified a research gap in the use and categorization of terminology. We observed that certain concepts are differently categorized by various authors, and what one may define as a design pattern another might classify as a security tactic, and often the rationale behind these categorizations is not explicitly stated but rather assumed. Developing a comprehensive framework that consolidates all these tactics, design patterns, methods, etc., related to microservices security, and then re-categorizing them with clear explanations for their classification, would be beneficial. Such a framework would simplify the process of developing secure microservices by providing clearer guidance on what principles and practices to adhere to.

List of Figures

List of Tables

Acronyms

ADD	Architectural Design Decisions
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
Behavior Assets	Assets used to operate system
BFF	Backend for frontend
CI/CD	Continuous Integration and Continuous Delivery
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
Data Assets	Valuable data to organisation
DoS	Denial-of-Service
FTPS	File Transfer Protocol Secure
gRPC	gRPC Remote Procedure Calls
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
Inter-service communication	Communication between two or more microservices
IRS	Intrusion Response Systems

JSON	JavaScript Object Notation
JWT	JSON Web Token
Kubernetes	Open-source container orchestration system
MITM	Man-in-the-middle
MQTT	Message Queuing Telemetry Transport
mTLS	Mutual TLS
NAT	Network Address Translation
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PRISMA	Preferred Reporting Items for Systematic reviews and Meta-Analyses
REST	Representational State Transfer
SDLC	Software Development Lifecycle
Security Tactics	Techniques or measures to increase security. More fine-grained than security design patterns.
SEP	Security Enforcement Point
SOA	Service-oriented architecture
SSDLC	Secure Software Development Lifecycle
TLS	Transport Layer Security
VM	Virtual Machine

XACML eXtensible Access Control Markup Language

Bibliography

[1] Adam Gordon Bell and Jan Machacek, *CoRecursive: Coding stories*. [Online]. Available: https://corecursive.com/014-micro-service-architectures-with-jan-machacek/.

[2] Sam Newman, *Building microservices: designing fine-grained systems*, Second Edition. Sebastopol, CA: O'Reilly Media, 2021, 586 pp., OCLC: on1263799754, ISBN: 978-1-4920-3402-5.

[3] Robert C. Martin, *Clean architecture: a craftsman's guide to software structure and design* (Robert C. Martin series). London, England: Prentice Hall, 2018, 404 pp., OCLC: on1004983973, ISBN: 978-0-13-449416-6.

[4] Martin Fowler, *Patterns of enterprise application architecture* (The Addison-Wesley signature series). Boston: Addison-Wesley, 2003, 533 pp., ISBN: 978-0-321-12742-6.

[5] Chris Richardson, *Microservices patterns: with examples in Java*. Shelter Island, New York: Manning Publications, 2019, 490 pp., OCLC: on1002834182, ISBN: 978-1-61729-454-9.

[6] The Linux Foundation. "Kubernetes." (2024), [Online]. Available: https://kubernetes.io/ (visited on 01/10/2024).

[7] The Kubernetes Authors. "Production-grade container orchestration," Kubernetes. (2024), [Online]. Available: https://kubernetes.io/ (visited on 11/28/2023).

[8] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Software*, vol. 35, no. 3, pp. 24–35, May 2018, Conference Name: IEEE Software, ISSN: 1937-4194. DOI: 10.1109/MS.2018.2141039.

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina, "Microservices: Yesterday, today, and tomorrow," in *Present and Ulterior Software Engineering*, Manuel Mazzara and Bertrand Meyer, Eds., Cham: Springer International Publishing, 2017, pp. 195–216, ISBN: 978-3-319-67425-4. DOI: 10.1007/978-3-

319-67425-4_12. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12 (visited on 09/20/2023).

[10] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 22–32, Sep. 2017, ISSN: 2325-6095. DOI: 10.1109/MCC.2017.4250931. [Online]. Available: http://ieeexplore.ieee.org/document/8125558/ (visited on 10/17/2023).

[11] Francisco Ponce, Gaston Marquez, and Hernan Astudillo, "Migrating from monolithic architecture to microservices: A rapid review," in *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, Concepcion, Chile: IEEE, Nov. 2019, pp. 1–7, ISBN: 978-1-72815-613-2. DOI: 10.1109/SCCC49216.2019.8966423. [Online]. Available: https://ieeexplore.ieee.org/document/8966423/ (visited on 10/24/2023).

[12] Prabath Siriwardena and Nuwan Dias, *Microservices security in action*. Shelter Island, NY: Manning Publications Co, 2020, 588 pp., OCLC: on1190759452, ISBN: 978-1-61729-595-9.

[13] Docker Inc. "Docker." (2024), [Online]. Available: https://www.docker.com/ (visited on 01/10/2024).

[14] Ramaswamy Chandramouli, "Security strategies for microservices-based application systems," National Institute of Standards and Technology, Gaithersburg, MD, NIST SP 800-204, Aug. 2019, NIST SP 800–204. DOI: 10.6028/NIST.SP.800-204. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204.pdf (visited on 01/23/2024).

[15] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, Dec. 2018, ISSN: 01641212. DOI: 10.1016/j.jss.2018.09.082. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121218302139 (visited on 09/13/2023).

[16] Anelis Pereira-Vale, Eduardo B. Fernandez, Raúl Monge, Hernán Astudillo, and Gastón Márquez, "Security in microservice-based systems: A multivocal literature review," *Computers & Security*, vol. 103, p. 102 200, Apr. 2021, ISSN: 01674048. DOI: 10.1016/j.cose.2021.102200. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404821000249 (visited on 09/06/2023).

[17] Davide Berardi, Saverio Giallorenzo, Jacopo Mauro, Andrea Melis, Fabrizio Montesi, and Marco Prandini, "Microservice security: A systematic literature review," *PeerJ Computer Science*, vol. 7, e779, Jan. 5, 2022, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.779. [Online]. Available: https://peerj.com/articles/cs-779 (visited on 08/26/2023).

[18] Francisco Ponce, Jacopo Soldani, Hernán Astudillo, and Antonio Brogi, "Smells and refactorings for microservices security: A multivocal literature review," *Journal of Systems and Software*, vol. 192, p. 111 393, Oct. 1, 2022, ISSN: 0164-1212. DOI: 10.1016/j.jss.2022.111393. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122200111X (visited on 01/31/2024).

[19] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman, "Systematic literature reviews in software engineering – a systematic literature review," *Information and Software Technology*, Special Section - Most Cited Articles in 2002 and Regular Research Papers, vol. 51, no. 1, pp. 7–15, Jan. 1, 2009, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.09.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584908001390 (visited on 01/23/2024).

[20] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher, "The PRISMA 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, n71, Mar. 29, 2021, ISSN: 1756-1833. DOI: 10.1136/bmj.n71. [Online]. Available: https://www.bmj.com/lookup/doi/10.1136/bmj.n71 (visited on 12/05/2023).

[21] Leonardo Roever, "PICO: Model for clinical questions," *Evidence-Based Medicine*, Aug. 9, 2018. DOI: 10.4172/2471-9919.1000115.

[22] Claes 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*, London England United Kingdom: ACM, May 13, 2014, pp. 1–10, ISBN: 978-1-4503-2476-2. DOI: 10.1145/2601248.2601268. [Online]. Available: https://dl.acm.org/doi/10.1145/2601248.2601268 (visited on 06/19/2023).

[23] Philipp Haindl and Reinhold Plösch, "Focus areas, themes, and objectives of non-functional requirements in DevOps: A systematic mapping study," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Aug. 2020, pp. 394–403. DOI: 10.1109/SEAA51224.2020.00071. [Online]. Available: https://ieeexplore.ieee.org/document/9226285 (visited on 01/31/2024).

[24] Gastón Márquez and Hernán Astudillo, "Identifying availability tactics to support security architectural design of microservice-based systems," in *Proceedings of the 13th European Conference on Software Architecture - Volume 2*, ser. ECSA '19, event-place: Paris, France, New York, NY, USA: Association for Computing Machinery, 2019, pp. 123–129, ISBN: 978-1-4503-7142-1. DOI: 10.1145/3344948.3344996. [Online]. Available: https://doi.org/10.1145/3344948.3344996.

[25] Anelis Pereira-Vale, Gaston Marquez, Hernan Astudillo, and Eduardo B. Fernandez, "Security mechanisms used in microservices-based systems: A systematic mapping," in *2019 XLV Latin American Computing Conference (CLEI)*, Panama, Panama: IEEE, Sep. 2019, pp. 01–10, ISBN: 978-1-72815-574-6. DOI: 10.1109/CLEI47609.2019.235060. [Online]. Available: https://ieeexplore.ieee.org/document/9073967/ (visited on 08/26/2023).

[26] Muhammad Waseem, Peng Liang, and Mojtaba Shahin, "A systematic mapping study on microservices architecture in DevOps," *Journal of Systems and Software*, vol. 170, p. 110 798, Dec. 2020, ISSN: 01641212. DOI: 10.1016/j.jss.2020.110798. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121220302053 (visited on 09/14/2023).

[27] Martin Fowler. "Microservices," martinfowler.com. (Mar. 25, 2014), [Online]. Available: https://martinfowler.com/articles/microservices.html (visited on 09/20/2023).

[28] Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah, "Fine-grained access control for microservices," in *Foundations and Practice of Security*, Nur Zincir-Heywood, Guillaume Bonfante, Mourad Debbabi, and Joaquin Garcia-Alfaro, Eds., Series Title: Lecture Notes in Computer Science, vol. 11358, Cham: Springer International Publishing, 2019, pp. 285–300, ISBN: 978-3-030-18418-6 978-3-030-18419-3. DOI: 10.1007/978-3-030-18419-3_19. [Online]. Available: http://link.springer.com/10.1007/978-3-030-18419-3_19 (visited on 08/24/2023).

[29] Antonio Nehme, Vitor Jesus, Khaled Mahbub, and Ali Abdallah, "Securing microservices," *IT Professional*, vol. 21, no. 1, pp. 42–49, Jan. 2019, ISSN: 1520-9202, 1941-045X. DOI: 10.1109/MITP.

2018.2876987. [Online]. Available: https://ieeexplore.ieee.org/document/8657392/ (visited on 11/05/2023).

[30] Aaron Parecki. "OAuth 2.0 — OAuth." (2024), [Online]. Available: https://oauth.net/2/ (visited on 01/10/2024).

[31] OpenID Foundation. "OpenID - OpenID foundation." (2024), [Online]. Available: https://openid.net/ (visited on 01/10/2024).

[32] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin, "Promises and challenges of microservices: An exploratory study," *Empirical Software Engineering*, vol. 26, no. 4, p. 63, Jul. 2021, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-020-09910-y. [Online]. Available: https://link.springer.com/10.1007/s10664-020-09910-y (visited on 09/12/2023).

[33] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111 061, Dec. 2021, ISSN: 01641212. DOI: 10.1016/j.jss.2021.111061. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121221001588 (visited on 10/24/2023).

[34] Patric Genfer and Uwe Zdun, "Avoiding excessive data exposure through microservice APIs," in *Software Architecture*, Ilias Gerostathopoulos, Grace Lewis, Thais Batista, and Tomáš Bureš, Eds., Series Title: Lecture Notes in Computer Science, vol. 13444, Cham: Springer International Publishing, 2022, pp. 3–18, ISBN: 978-3-031-16696-9 978-3-031-16697-6. DOI: 10.1007/978-3-031-16697-6_1. [Online]. Available: https://link.springer.com/10.1007/978-3-031-16697-6_1 (visited on 09/09/2023).

[35] Uwe Zdun, Mirko Stocker, Olaf Zimmermann, Cesare Pautasso, and Daniel Lübke, "Guiding architectural decision making on quality aspects in microservice APIs," in *Service-Oriented Computing*, Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, Eds., Series Title: Lecture Notes in Computer Science, vol. 11236, Cham: Springer International Publishing, 2018, pp. 73–89, ISBN: 978-3-030-03595-2 978-3-030-03596-9. DOI: 10.1007/978-3-030-03596-9_5. [Online]. Available: http://link.springer.com/10.1007/978-3-030-03596-9_5 (visited on 08/27/2023).

[36] Ali Rezaei Nasab, Mojtaba Shahin, Seyed Ali Hoseyni Raviz, Peng Liang, Amir Mashmool, and Valentina Lenarduzzi, "An empirical study of security practices for microservices systems," *Journal of Systems and Software*, vol. 198, p. 111 563, Apr. 2023, ISSN: 01641212. DOI: 10.1016/j.jss.

2022.111563. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121222002394 (visited on 08/30/2023).

[37] Kevin Walsh and John Manferdelli, "Mechanisms for mutual attested microservice communication," in *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, Austin Texas USA: ACM, Dec. 5, 2017, pp. 59–64, ISBN: 978-1-4503-5195-9. DOI: 10.1145/3147234.3148102. [Online]. Available: https://dl.acm.org/doi/10.1145/3147234.3148102 (visited on 08/27/2023).

[38] Qingyang Zeng, Mohammad Kavousi, Yinhong Luo, Ling Jin, and Yan Chen, "Full-stack vulnerability analysis of the cloud-native platform," *Computers & Security*, vol. 129, p. 103 173, Jun. 2023, ISSN: 01674048. DOI: 10.1016/j.cose.2023.103173. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167404823000834 (visited on 08/30/2023).

[39] Nacha Chondamrongkul, Jing Sun, and Ian Warren, "Formal security analysis for software architecture design: An expressive framework to emerging architectural styles," *Science of Computer Programming*, vol. 206, p. 102 631, Jun. 2021, ISSN: 01676423. DOI: 10.1016/j.scico.2021.102631. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167642321000241 (visited on 09/07/2023).

[40] Uwe Zdun, Pierre-Jean Queval, Georg Simhandl, Riccardo Scandariato, Somik Chakravarty, Marjan Jelic, and Aleksandar Jovanovic, "Microservice security metrics for secure communication, identity management, and observability," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, pp. 1–34, Jan. 31, 2023, ISSN: 1049-331X, 1557-7392. DOI: 10.1145/3532183. [Online]. Available: https://dl.acm.org/doi/10.1145/3532183 (visited on 09/14/2023).

[41] Sahil Suneja, Ali Kanso, and Canturk Isci, "Can container fusion be securely achieved?" In *Proceedings of the 5th International Workshop on Container Technologies and Container Clouds*, Davis CA USA: ACM, Dec. 9, 2019, pp. 31–36, ISBN: 978-1-4503-7033-2. DOI: 10.1145/3366615.3368356. [Online]. Available: https://dl.acm.org/doi/10.1145/3366615.3368356 (visited on 09/13/2023).

[42] Tetiana Yarygina and Anya Helene Bagge, "Overcoming security challenges in microservice architectures," in *2018 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, Mar. 2018, pp. 11–20. DOI: 10.1109/SOSE.2018.00011.

[43] Kennedy A. Torkura, Muhammad I.H. Sukmana, and Christoph Meinel, "Integrating continuous security assessments in microservices and cloud native applications," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, Austin Texas USA: ACM, Dec. 5, 2017, pp. 171–180, ISBN: 978-1-4503-5149-2. DOI: 10.1145/3147213.3147229. [Online]. Available: https://dl.acm.org/doi/10.1145/3147213.3147229 (visited on 08/27/2023).

[44] Dalton A. Hahn, Drew Davidson, and Alexandru G. Bardas, "MisMesh: Security issues and challenges in service meshes," in *Security and Privacy in Communication Networks*, Noseong Park, Kun Sun, Sara Foresti, Kevin Butler, and Nitesh Saxena, Eds., Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 335, Cham: Springer International Publishing, 2020, pp. 140–151, ISBN: 978-3-030-63085-0 978-3-030-63086-7. DOI: 10.1007/978-3-030-63086-7_9. [Online]. Available: https://link.springer.com/10.1007/978-3-030-63086-7_9 (visited on 07/12/2023).

[45] Angeliki Aktypi, Dimitris Karnikis, Nikos Vasilakis, and Kasper Rasmussen, "Themis: A secure decentralized framework for microservice interaction in serverless computing," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ser. ARES '22, New York, NY, USA: Association for Computing Machinery, Aug. 23, 2022, pp. 1–11, ISBN: 978-1-4503-9670-7. DOI: 10.1145/3538969.3538983. [Online]. Available: https://dl.acm.org/doi/10.1145/3538969.3538983 (visited on 09/21/2023).

[46] Amine El Malki and Uwe Zdun, "Guiding architectural decision making on service mesh based microservice architectures," in *Software Architecture*, Tomas Bures, Laurence Duchien, and Paola Inverardi, Eds., Series Title: Lecture Notes in Computer Science, vol. 11681, Cham: Springer International Publishing, 2019, pp. 3–19, ISBN: 978-3-030-29982-8 978-3-030-29983-5. DOI: 10.1007/978-3-030-29983-5_1. [Online]. Available: http://link.springer.com/10.1007/978-3-030-29983-5_1 (visited on 08/27/2023).

[47] R. Alboqmi, S. Jahan, and R. F. Gamble, "Toward enabling self-protection in the service mesh of the microservice architecture," in *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, Journal Abbreviation: 2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), Sep. 19, 2022, pp. 133–138. DOI: 10.1109/ACSOSC56246.2022.00047.

[48] Istio. "Istio / the istio service mesh," Istio. (2024), [Online]. Available: https://istio.io/latest/about/service-mesh/ (visited on 01/10/2024).

[49] HashiCorp. "Consul by HashiCorp," Consul by HashiCorp. (2024), [Online]. Available: https://www.consul.io/ (visited on 01/10/2024).

[50] Buoyant. "Linkerd." (2024), [Online]. Available: https://linkerd.io/2.14/overview/ (visited on 01/10/2024).

[51] Kennedy A. Torkura, Muhammad I.H. Sukmana, Anne V.D.M. Kayem, Feng Cheng, and Christoph Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," in *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Melbourne, Australia: IEEE, Dec. 2018, pp. 932–939, ISBN: 978-1-72811-141-4. DOI: 10.1109/BDCloud.2018.00137. [Online]. Available: https://ieeexplore.ieee.org/document/8672278/ (visited on 08/27/2023).

[52] Nuha Alshuqayran, Nour Ali, and Roger Evans, "A systematic mapping study in microservice architecture," in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, Macau, China: IEEE, Nov. 2016, pp. 44–51, ISBN: 978-1-5090-4781-9. DOI: 10.1109/SOCA.2016.15. [Online]. Available: http://ieeexplore.ieee.org/document/7796008/ (visited on 08/27/2023).

[53] Mohsen Ahmadvand and Amjad Ibrahim, "Requirements reconciliation for scalable and secure microservice (de)composition," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Beijing, China: IEEE, Sep. 2016, pp. 68–73, ISBN: 978-1-5090-3694-3. DOI: 10.1109/REW.2016.026. [Online]. Available: http://ieeexplore.ieee.org/document/7815609/ (visited on 08/26/2023).

[54] Kennedy A. Torkura, Muhammad I. H. Sukmana, Feng Cheng, and Christoph Meinel, "CAVAS: Neutralizing application and container security vulnerabilities in the cloud native era," in *Security and Privacy in Communication Networks*, Raheem Beyah, Bing Chang, Yingjiu Li, and Sencun Zhu, Eds., Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 254, Cham: Springer International Publishing, 2018, pp. 471–490, ISBN: 978-3-030-01700-2 978-3-030-01701-9. DOI: 10.1007/978-3-030-01701-9_26. [Online]. Available: http://link.springer.com/10.1007/978-3-030-01701-9_26 (visited on 09/08/2023).

[55] Francesco Minna and Fabio Massacci, "SoK: Run-time security for cloud microservices. are we there yet?" *Computers & Security*, vol. 127, p. 103 119, Apr. 1, 2023, ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103119. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823000299.

[56] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, and Muhammad Ali Babar, "Understanding and addressing quality attributes of microservices architecture: A systematic literature review," *Information and Software Technology*, vol. 131, p. 106 449, Mar. 2021, ISSN: 09505849. DOI: 10.1016/j.infsof.2020.106449. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584920301993 (visited on 09/07/2023).

[57] Abdelhakim Hannousse and Salima Yahiouche, "Securing microservices and microservice architectures: A systematic mapping study," *Computer Science Review*, vol. 41, p. 100 415, Aug. 1, 2021, ISSN: 1574-0137. DOI: 10.1016/j.cosrev.2021.100415. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000551.

[58] Priyanka Billawa, Anusha Bambhore Tukaram, Nicolás E. Díaz Ferreyra, Jan-Philipp Steghöfer, Riccardo Scandariato, and Georg Simhandl, "SoK: Security of microservice applications: A practitioners' perspective on challenges and best practices," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, ser. ARES '22, event-place: Vienna, Austria, New York, NY, USA: Association for Computing Machinery, 2022, ISBN: 978-1-4503-9670-7. DOI: 10.1145/3538969.3538986. [Online]. Available: https://doi.org/10.1145/3538969.3538986.

[59] Amjad Ibrahim, Stevica Bozhinoski, and Alexander Pretschner, "Attack graph generation for microservice architecture," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, Limassol Cyprus: ACM, Apr. 8, 2019, pp. 1235–1242, ISBN: 978-1-4503-5933-7. DOI: 10.1145/3297280.3297401. [Online]. Available: https://dl.acm.org/doi/10.1145/3297280.3297401 (visited on 08/27/2023).

[60] Xing Li, Yan Chen, Zhiqiang Lin, Xiao Wang, and Jim Hao Chen, "Automatic policy generation for inter-service access control of microservices," *30th USENIX Security Symposium, USENIX Security 2021*, Proceedings of the 30th USENIX Security Symposium, pp. 3971–3988, 2021, Publisher: USENIX Association. [Online]. Available: http://www.scopus.com/inward/record.url?scp=85114505734&partnerID=8YFLogxK (visited on 08/11/2021).

[61] The Linux Foundation. "Containerd." (2024), [Online]. Available: https://containerd.io/ (visited on 01/10/2024).

[62] opencontainers. "Opencontainers/runc." original-date: 2015-06-05T23:30:45Z. (Jan. 10, 2024), [Online]. Available: https://github.com/opencontainers/runc (visited on 01/10/2024).

[63] The Linux Foundation. "Kubectl." (2024), [Online]. Available: https://kubernetes.io/docs/reference/kubectl/ (visited on 01/10/2024).

[64] The Linux Foundation. "Kubelet." Section: docs. (2024), [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ (visited on 01/10/2024).

[65] Tetiana Yarygina and Christian Otterstad, "A game of microservices: Automated intrusion response," in *Distributed Applications and Interoperable Systems*, Silvia Bonomi and Etienne Rivière, Eds., Series Title: Lecture Notes in Computer Science, vol. 10853, Cham: Springer International Publishing, 2018, pp. 169–177, ISBN: 978-3-319-93766-3 978-3-319-93767-0. DOI: 10.1007/978-3-319-93767-0_12. [Online]. Available: http://link.springer.com/10.1007/978-3-319-93767-0_12 (visited on 07/12/2023).

[66] Francisco Ponce, Jacopo Soldani, Hernán Astudillo, and Antonio Brogi, "Microservices security: Bad vs. good practices," in *Software Architecture. ECSA 2022 Tracks and Workshops*, Thais Batista, Tomáš Bureš, Claudia Raibulet, and Henry Muccini, Eds., Series Title: Lecture Notes in Computer Science, vol. 13928, Cham: Springer International Publishing, 2023, pp. 337–352, ISBN: 978-3-031-36888-2 978-3-031-36889-9. DOI: 10.1007/978-3-031-36889-9_23. [Online]. Available: https://link.springer.com/10.1007/978-3-031-36889-9_23 (visited on 08/24/2023).

[67] Peter Nkomo and Marijke Coetzee, "Software development activities for secure microservices," in *Computational Science and Its Applications – ICCSA 2019*, Sanjay Misra, Osvaldo Gervasi, Beniamino Murgante, Elena Stankova, Vladimir Korkhov, Carmelo Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O. Apduhan, and Eufemia Tarantino, Eds., Series Title: Lecture Notes in Computer Science, vol. 11623, Cham: Springer International Publishing, 2019, pp. 573–585, ISBN: 978-3-030-24307-4 978-3-030-24308-1. DOI: 10.1007/978-3-030-24308-1_46. [Online]. Available: http://link.springer.com/10.1007/978-3-030-24308-1_46 (visited on 07/12/2023).

[68] Yuqiong Sun, Susanta Nanda, and Trent Jaeger, "Security-as-a-service for microservices-based cloud applications," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, Vancouver, BC, Canada: IEEE, Nov. 2015, pp. 50–57, ISBN: 978-1-4673-9560-1. DOI: 10.1109/CloudCom.2015.93. [Online]. Available: http://ieeexplore.ieee.org/document/7396137/ (visited on 08/27/2023).

[69] Xing Li, Yan Chen, and Zhiqiang Lin, "Towards automated inter-service authorization for microservice applications," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, Beijing China: ACM, Aug. 19, 2019, pp. 3–5, ISBN: 978-1-4503-6886-5. DOI: 10.1145/3342280.3342288. [Online]. Available: https://dl.acm.org/doi/10.1145/3342280.3342288 (visited on 08/27/2023).

[70] Loic Miller, Pascal Merindol, Antoine Gallais, and Cristel Pelsser, "Towards secure and leak-free workflows using microservice isolation," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, Paris, France: IEEE, Jun. 7, 2021, pp. 1–5, ISBN: 978-1-66544-005-9. DOI: 10.1109/HPSR52026.2021.9481820. [Online]. Available: https://ieeexplore.ieee.org/document/9481820/ (visited on 08/31/2023).

[71] Mohsen Ahmadvand, Alexander Pretschner, Keith Ball, and Daniel Eyring, "Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework," in *Software Technologies: Applications and Foundations*, Manuel Mazzara, Iulian Ober, and Gwen Salaün, Eds., Series Title: Lecture Notes in Computer Science, vol. 11176, Cham: Springer International Publishing, 2018, pp. 573–588, ISBN: 978-3-030-04770-2 978-3-030-04771-9. DOI: 10.1007/978-3-030-04771-9_43. [Online]. Available: http://link.springer.com/10.1007/978-3-030-04771-9_43 (visited on 08/24/2023).

[72] Ning Xi, Jin Liu, Yajie Li, and Bojun Qin, "Decentralized access control for secure microservices cooperation with blockchain," *ISA Transactions*, S0019057823003257, Jul. 2023, ISSN: 00190578. DOI: 10.1016/j.isatra.2023.07.018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0019057823003257 (visited on 09/09/2023).

[73] Muhammad Waseem, Peng Liang, Aakash Ahmad, Mojtaba Shahin, Arif Ali Khan, and Gastón Márquez, "Decision models for selecting patterns and strategies in microservices systems and their evaluation by practitioners," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, May 2022, pp. 135–144. DOI: 10.1145/3510457.3513079.

[74] Mohammad Reza Saleh Sedghpour and Paul Townend, "Service mesh and eBPF-powered microservices: A survey and future directions," in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, ISSN: 2642-6587, Aug. 2022, pp. 176–184. DOI: 10.1109/SOSE55356.2022.00027.

[75] Simon Schneider and Riccardo Scandariato, "Automatic extraction of security-rich dataflow diagrams for microservice applications written in java," *Journal of Systems and Software*, vol. 202, p. 111 722, Aug. 1, 2023, ISSN: 0164-1212. DOI: 10.1016/j.jss.2023.111722. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121223001176.

[76] Francisco Ponce, "Towards resolving security smells in microservice-based applications," in *Advances in Service-Oriented and Cloud Computing*, Christian Zirpins, Iraklis Paraskakis, Vasilios Andrikopoulos, Nane Kratzke, Claus Pahl, Nabil El Ioini, Andreas S. Andreou, George Feuerlicht, Winfried Lamersdorf, Guadalupe Ortiz, Willem-Jan Van Den Heuvel, Jacopo Soldani, Massimo Villari, Giuliano Casale, and Pierluigi Plebani, Eds., Series Title: Communications in Computer and Information Science, vol. 1360, Cham: Springer International Publishing, 2021, pp. 133–139, ISBN: 978-3-030-71905-0 978-3-030-71906-7. DOI: 10.1007/978-3-030-71906-7_11. [Online]. Available: http://link.springer.com/10.1007/978-3-030-71906-7_11 (visited on 08/26/2023).

[77] Olaf Zimmermann, Daniel Lübke, Uwe Zdun, Cesare Pautasso, and Mirko Stocker, "Interface responsibility patterns: Processing resources and operation responsibilities," in *Proceedings of the European Conference on Pattern Languages of Programs 2020*, Virtual Event Germany: ACM, Jul. 2020, pp. 1–24, ISBN: 978-1-4503-7769-0. DOI: 10.1145/3424771.3424822. [Online]. Available: https://dl.acm.org/doi/10.1145/3424771.3424822 (visited on 08/31/2023).

[78] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*, 3rd edition. Upper Saddle River, NJ: Addison-Wesley Professional, Sep. 25, 2012, 624 pp., ISBN: 978-0-321-81573-6.

[79] Nuno Mateus-Coelho, Manuela Cruz-Cunha, and Luis Gonzaga Ferreira, "Security in microservices architectures," *Procedia Computer Science*, vol. 181, pp. 1225–1236, 2021, ISSN: 18770509. DOI: 10.1016/j.procs.2021.01.320. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050921003719 (visited on 06/01/2023).

[80] HashiCorp. "Vault by HashiCorp," Vault by HashiCorp. (2024), [Online]. Available: https://www.vaultproject.io/ (visited on 01/10/2024).

[81] Microsoft. "Key vault | microsoft azure." (2024), [Online]. Available: https://azure.microsoft.com/en-us/products/key-vault (visited on 01/10/2024).

[82] Christian Otterstad and Tetiana Yarygina, "Low-level exploitation mitigation by diverse microservices," in *Service-Oriented and Cloud Computing*, Flavio De Paoli, Stefan Schulte, and Einar Broch Johnsen, Eds., Series Title: Lecture Notes in Computer Science, vol. 10465, Cham: Springer International Publishing, 2017, pp. 49–56, ISBN: 978-3-319-67261-8 978-3-319-67262-5. DOI: 10.1007/978-3-319-67262-5_4. [Online]. Available: https://link.springer.com/10.1007/978-3-319-67262-5_4 (visited on 07/30/2023).

[83] D. C Kalubowila, S. M Athukorala, B. A. S Tharaka, H. W. Y. R Samarasekara, Udara Srimath S. Samaratunge Arachchilage, and Dharshana Kasthurirathna, "Optimization of microservices security," in *2021 3rd International Conference on Advancements in Computing (ICAC)*, Dec. 2021, pp. 49–54. DOI: 10.1109/ICAC54203.2021.9671131.

[84] Patric Genfer and Uwe Zdun, "Identifying domain-based cyclic dependencies in microservice APIs using source code detectors," in *Software Architecture: 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings*, Berlin, Heidelberg: Springer-Verlag, Sep. 13, 2021, pp. 207–222, ISBN: 978-3-030-86043-1. DOI: 10.1007/978-3-030-86044-8_15. [Online]. Available: https://doi.org/10.1007/978-3-030-86044-8_15 (visited on 01/18/2024).

[85] Microservice-API-Patterns. "Microservice-API-patterns/LakesideMutual at spring-term-2020," GitHub. (Apr. 19, 2021), [Online]. Available: https://github.com/Microservice-API-Patterns/LakesideMutual (visited on 01/18/2024).

[86] .NET Foundation and Contributors. "Dotnet-architecture/eShopOnContainers." (Apr. 29, 2022), [Online]. Available: https://github.com/dotnet-architecture/eShopOnContainers/tree/main (visited on 01/18/2024).

[87] Nacha Chondamrongkul, Jing Sun, and Ian Warren, "Automated security analysis for microservice architecture," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Salvador, Brazil: IEEE, Mar. 2020, pp. 79–82, ISBN: 978-1-72817-415-0. DOI: 10.1109/ICSA-C50368.2020.00024. [Online]. Available: https://ieeexplore.ieee.org/document/9095669/ (visited on 08/27/2023).

[88] Steven J. Brams and Morton D. Davis. "Game theory | definition, facts, & examples | britannica." (Dec. 5, 2023), [Online]. Available: https://www.britannica.com/science/game-theory (visited on 01/21/2024).

[89] OWASP. "Microservices security - OWASP cheat sheet series." (2024), [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Microservices_Security_Cheat_Sheet.html (visited on 01/22/2024).

[90] Daniela S. Cruzes Othmane Lotfi ben, "Threats to validity in empirical software security research," in *Empirical Research for Software Security*, Num Pages: 26, CRC Press, 2017, ISBN: 978-1-315-15485-5.

[91] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson, "Systematic mapping studies in software engineering," in *Proceedings of the 12th international conference on Evaluation and Assessment in Software Engineering*, ser. EASE'08, Swindon, GBR: BCS Learning & Development Ltd., Jun. 26, 2008, pp. 68–77. (visited on 01/12/2024).

[92] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland, "Requirements engineering paper classification and evaluation criteria: A proposal and a discussion," *Requirements Engineering*, vol. 11, no. 1, pp. 102–107, Mar. 1, 2006, ISSN: 1432-010X. DOI: 10.1007/s00766-005-0021-6. [Online]. Available: https://doi.org/10.1007/s00766-005-0021-6 (visited on 01/12/2024).

[93] M. Shaw, "Writing good software engineering research papers," in *25th International Conference on Software Engineering, 2003. Proceedings.*, Portland, OR, USA: IEEE, 2003, pp. 726–736, ISBN: 978-0-7695-1877-0. DOI: 10.1109/ICSE.2003.1201262. [Online]. Available: http://ieeexplore.ieee.org/document/1201262/ (visited on 01/13/2024).

[94] Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, vol. 56, no. 10, pp. 1200–1218, Oct. 1, 2014, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2014.04.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000950 (visited on 01/13/2024).

A. Research Facets

Category	Description
Personal experience paper	A list of lessons learned from one or more projects. The study reflects the personal experience of the authors. The study often describes the use of tools/techniques in practice, with less focus on research methodology.
Evaluation research	A technique is implemented in real life and an evaluation of the implemented technique is performed. The pros and cons of implementation and relevant industry problems are also taken into account.
Validation research	The study investigates techniques that have not yet been implemented in practice. Research methods such as lab experiments & work, simulations, etc. can be used.
Philosophical paper	Sketches a new way of looking at something, for example, by forming a taxonomy or conceptual framework.
Proposal of solution	Proposes a solution for a problem but does not completely validate it. The proposal of the solution is emphasized with a small proof of concept, such as an example or argumentation.
Personal opinion paper	Personal opinion of the author on whether something is good or bad, such as a technique.

Table A.1.: Research facets, adapted from Petersen *et al.* [91] and Wieringa *et al.* [92]

B. Contribution Facets

Category	**Description**
Model	An abstraction of the authors observed in reality using conceptualization.
Method	Method or approach for dealing with inter-service security threats or applying mitigation strategies in microservice architectures.
Theory	Explanation for why certain results occur, based on cause-effect relationships.
Framework	Combination of multiple methods. Specifies conditions in which the use of methods is applicable, including the required input parameters and possible outcome.
Guideline	List of advices based on research results.
Lessons Learned	List of outcomes based on research results (including recommendations).
Advice	Generic recommendations based on personal opinions.
Tool	Some technology or software application/program used for dealing with security threats in microservice architectures.

Table B.1.: Contribution facets, adapted from Shaw [93] and Paternoster *et al.* [94]

C. Categorization of Publications

Table C.1: Summary of Research Papers

Publication	**Research Facet**	**Contribution Facet**	**Security Focus**
[38]	Evaluation Research	Method	Security Threats
[36]	Evaluation Research	Guideline	Mitigation Strategies
[16]	Evaluation Research	Guideline	Security Threats
[55]	Philosophical Paper	Guideline	Security Threats
[79]	Personal Opinion paper	Advice	Mitigation Strategies
[56]	Philosophical Paper	Model	Mitigation Strategies
[18]	Evaluation Research	Framework	Security Threats
[57]	Validation Research	Guideline	Security Threats
[52]	Philosophical Paper	Model	Mitigation Strategies
[39]	Validation Research	Tool	Mitigation Strategies
[15]	Philosophical Paper	Theory	Security Threats
[72]	Validation Research	Framework	Mitigation Strategies
[32]	Philosophical Paper	Lessons Learned	Security Threats
[28]	Validation Research	Tool	Mitigation Strategies
[71]	Proposal of Solution	Framework	Mitigation Strategies
[66]	Evaluation Research	Guideline	Mitigation Strategies
[54]	Validation Research	Tool	Mitigation Strategies

Continued on next page

Table C.1: Summary of Research Papers (Continued)

Publication	Research Facet	Contribution Facet	Security Focus
[34]	Validation Research	Method	Mitigation Strategies
[83]	Proposal of Solution	Method	Mitigation Strategies
[74]	Philosophical Paper	Theory	Mitigation Strategies
[40]	Validation Research	Method	Mitigation Strategies
[45]	Validation Research	Tool	Mitigation Strategies
[58]	Evaluation Research	Theory	Mitigation Strategies
[24]	Validation Research	Model	Security Threats
[73]	Validation Research	Guideline	Mitigation Strategies
[41]	Proposal of Solution	Method	Mitigation Strategies
[33]	Evaluation Research	Lessons Learned	Security Threats
[44]	Evaluation Research	Lessons Learned	Security Threats
[65]	Proposal of Solution	Method	Security Threats
[25]	Philosophical Paper	Model	Mitigation Strategies
[53]	Proposal of Solution	Method	Mitigation Strategies
[42]	Philosophical Paper	Framework	Mitigation Strategies
[17]	Philosophical Paper	Theory	Security Threats
[67]	Evaluation Research	Guideline	Mitigation Strategies
[76]	Validation Research	Tool	Mitigation Strategies
[82]	Proposal of Solution	Method	Mitigation Strategies
[51]	Validation Research	Method	Mitigation Strategies
[69]	Proposal of Solution	Method	Mitigation Strategies
[59]	Proposal of Solution	Method	Mitigation Strategies
[46]	Evaluation Research	Model	Mitigation Strategies
[52]	Philosophical Paper	Model	Security Threats

Continued on next page

Table C.1: Summary of Research Papers (Continued)

Publication	**Research Facet**	**Contribution Facet**	**Security Focus**
[68]	Validation Research	Method	Mitigation Strategies
[87]	Proposal of Solution	Method	Mitigation Strategies
[35]	Evaluation Research	Model	Mitigation Strategies
[43]	Proposal of Solution	Method	Mitigation Strategies
[37]	Proposal of Solution	Method	Mitigation Strategies
[70]	Proposal of Solution	Method	Mitigation Strategies
[77]	Philosophical Paper	Model	Mitigation Strategies
[60]	Validation Research	Tool	Mitigation Strategies
[75]	Validation Research	Method	Mitigation Strategies
[47]	Validation Research	Method	Mitigation Strategies
[29]	Philosophical Paper	Theory	Security Threats