Securing Service Mesh from the Bottom-Up: Adoption Concerns, Vulnerabilities and their Mitigation

by

Yihao Chen

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

September 2023

Abstract

As service mesh architectures rise in popularity, there is a growing need for a comprehensive understanding of the challenges found by DevOps engineers adopting them, as well as their interplay with package-based deployment methods such as Helm Charts. The sophisticated infrastructure required by service mesh architectures is prone to misconfiguration, leading to prevalent security concerns. Meanwhile, Helm Charts, while simplifying deployment processes, can leave vulnerabilities unnoticed. These risks are especially concerning in the context of service mesh frameworks, which often operate on a large-scale and follow a zero-trust model, where all components in the system must be secured. This thesis carries out two large-scale empirical studies on the adoption concerns and security vulnerabilities present in service mesh frameworks and Helm Charts. By doing so, it lays the foundation for evidence-based strategies to enhance security from the bottom-up.

Objective: This thesis employs two mixed-methods empirical investigations to analyze adoption concerns and security vulnerabilities in service mesh frameworks and Helm Charts. By analyzing practitioner questions from service mesh forums and a large dataset of Helm Charts and their maintenance repositories on GitHub, we identify prevalent concerns, trends, and theories to facilitate safer deployments of service mesh systems.

Method: Our approach combines Dynamic Topic Modeling (DTM) and open card sorting for service mesh-related question analysis, corroborated by domain expert verification. Concurrently, we construct a Grounded Theory (GT) to discern the prevalence and mitigation strategies of Common Vulnerability Enumerations (CVEs) in Helm Charts, shedding light on the reasons for their high prevalence in such a security-critical context.

Results: Our study uncovers persistent infrastructure-related concerns and heightened security and observability issues surrounding service mesh frameworks. Interestingly, most service mesh-related errors are remediable with minimal configuration changes. We find that Helm Charts often harbor unfixed but fixable vulnerabilities, posing a significant risk to the security of service mesh deployments. However, the effectiveness of CVE mitigation strategies in Helm Charts is often impeded by maintainer considerations such as incentives, trade-offs, and trust among stakeholders.

Conclusion: The findings advocate for consistent documentation, practical automation of service mesh deployment, and improved container orchestration. We propose a shift towards a shared responsibility model, emphasizing the importance of proactive CVE mitigation to strengthen the security of service mesh frameworks from the ground up. This study offers valuable insights for practitioners, maintainers, and

researchers, creating pathways for the safer adoption and deployment of service mesh frameworks.

Acknowledgments

I would like to express my deepest gratitude to Professor Bram Adams and Professor Ahmed E. Hassan for their continuous support and invaluable contributions to my work. Professor Bram, your truly brilliant ideas and feedback have illuminated my path. I could not have grasped the key to research without your guidance. Professor Ahmed, your encouragement to pursue the most challenging problems in the domain has motivated me to excel. This journey and my future would not have been possible without your support.

I owe immense thanks to Dr. Eduardo Fernandes and Dr. Jiahuei Lin, for being great mentors, friends, and co-authors. Your collective patience and wisdom have shaped my approach to research, and I am thankful for the invaluable skills I have learned from both of you.

I would like to thank Jaskirat Singh for his thoughtful suggestions on the revision of this thesis. Additionally, I extend my gratitude to my lab mates for all their assistance with my work.

My sincere gratitude goes to the Apache SkyWalking community for their assistance in helping me grasp some of the most complex problems in the domain. In particular, I would like to thank Sheng Wu and Zhenxu Ke for being my mentors, my friends and my life-long role models in the open-source world.

I would like to thank Yuhao Chen for being my best buddy, and no matter what happens, I can always find support and understanding.

I would like to thank PiPiFox Hu for her invaluable friendship. Our bond and journey over the past years has been truly wonderful.

I would like to especially thank YGL "Xiao Yang", for supporting me and having faith in me. You will always mean the world to me.

I would like to dedicate this thesis to my family. My grandparents and parents have invested immensely in my journey and supported me throughout my Master's program. Words cannot express the depth of my gratitude and love.

Co-authorship

For all the thesis chapters and related publications, my contributions are designing and refining the research plan; conducting background literature reviews and related work elicitation; collecting datasets; conducting corresponding experiments and data analysis; and writing and polishing the manuscripts.

The work presented in this thesis are listed below:

- Yihao Chen, Eduardo Fernandes, Bram Adams, and Ahmed E. Hassan. 2023. On Practitioners' Concerns when Adopting Service Mesh Frameworks. Journal of Empirical Software Engineering (EMSE). Accepted.

- Yihao Chen, Jiahuei Lin, Bram Adams, and Ahmed E. Hassan. 2023. Why Not Fixing the Fixables? An Empirical Study on Understanding the Unfixed Vulnerabilities in Helm Charts. Journal of Empirical Software Engineering (EMSE). Under submission.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

The advent of service mesh architectures has instigated a significant shift in the management of microservices and infrastructures (70). A service mesh is a dedicated infrastructure layer for handling service-to-service communication, providing a platform to connect, manage, and secure microservices. With advanced features like service discovery, traffic management, and observability, it addresses previous challenges associated with the complex orchestration of microservices (39). The popularity of service mesh architectures has attracted over 40% of organizations with a microservices application implement service mesh according to a survey in 2022 (21).

However, the growing popularity of service mesh introduces complex mechanisms at the infrastructure level. Without proper configuration, these systems can become susceptible to security vulnerabilities. For instance, misconfiguration could leave internal communication channels exposed, making them potential targets for data breaches. Furthermore, improper traffic management settings could inadvertently allow unauthorized access to certain services. These risks are particularly concerning in a "zero-trust" environment, which demands stringent security measures for every component and step of the deployed systems (14; 23; 95).

Given the intricacy of the infrastructure supporting service mesh architectures, novel deployment methods like Helm Charts have emerged as a convenient solution. Simply put, Helm Charts are pre-configured packages for deploying heterogeneous applications on Kubernetes with minimal manual work from DevOps engineers. While Helm Charts simplify the deployment and management of cloud-native infrastructures, they can inadvertently broaden the attack surface to Common Vulnerability Enumerations (CVEs) due to their single-command deployment style (i.e., “Helm install chartName”) of complex software, prepackaged by a third party (25; 127). This trend illustrates a tight coupling between service mesh application architectures and the underlying infrastructure. Such an interplay reinforces the need to conduct an in-depth investigation into potential vulnerabilities, not only in service mesh architectures themselves but also in the underlying package-based deployment infrastructure like Helm Charts.

This dual-sided narrative forms the crux of our thesis. As the first exploratory study in the service mesh domain, we investigate adoption concerns related to two dominant open-source service mesh frameworks based on 5,497 practitioner questions from generic (Stack Overflow) and framework-specific question-and-answer forums. Simultaneously, we study Helm Charts to understand the nature and reasons behind the high prevalence of unfixed yet fixable vulnerabilities (CVEs). We present a Grounded

Theory (GT) based on our findings, which uncovers convoluted reasons behind these unfixed vulnerabilities and guides future researchers and chart maintainers towards more secure large-scale distributed systems such as service mesh.

This thesis focuses on presenting a comprehensive and concise view of the prevalent adoption concerns, vulnerabilities, and vulnerability mitigation strategies. It aims to provide vital insights that could help address the infrastructure-related concerns and unattended vulnerabilities in service mesh frameworks and Helm Charts, respectively. The practical takeaways from our research not only inform and guide practitioners in the domain, but also inspire further investigation into improving the adoption and security of service mesh frameworks and Helm Charts. In the wake of an increasingly dynamic and complex software landscape, securing the service mesh from the bottom-up becomes an imperative task for the sustainable and safe deployment of microservices and cloud-native infrastructures.

1.1 Thesis Statement

> DevOps engineers adopting and securing large scale distributed systems like service mesh and their underlying infrastructures encounter a wide range of challenges, of which security makes up a large proportion.

1.2 Thesis Overview

In this section, we provide an outline of this thesis:

1.2.1 Chapter 2: Background and Related Work

Chapter 2 delves into the foundational concepts of service mesh and Helm Charts, elucidating how they interact and complement each other in modern microservices architectures. This chapter also provides a comprehensive review of existing literature that has explored these technologies, providing the context for our subsequent empirical investigation in Chapter 3 and Chapter 4.

1.2.2 Chapter 3: On Practitioners' Concerns when Adopting Service Mesh Frameworks

The emerging service mesh architecture tries to simplify microservices by delegating crucial tasks to dedicated infrastructure. However, service mesh introduces new notions and enables complex capabilities such as sidecar proxies that inevitably bring major adoption concerns.

We investigate the adoption concerns in two dominant open-source service mesh frameworks via a mixed-methods empirical investigation of the past, current and evolution of 5,497 practitioner questions posted on generic and framework-specific question-and-answer fora.

We first mine the topics of questions with the help of Dynamic Topic Modeling (DTM). Then, we identify evolution by applying topic modelling to time periods and aggregating topics into macro-topics. We conduct a qualitative analysis to understand the three major types of questions and to generalize common fix patterns for the extracted error symptoms. We consulted a service mesh domain expert to provide feedback on our findings and discuss implications.

We found that about half of the questions are error-related and mined 18 topics, covering service mesh traffic, infrastructure, security, observability and application. We discovered a drastic decline in traffic-related concerns while finding persisting infrastructure concerns and a rise in security and observability concerns. We identified 54 error symptoms from two popular service mesh frameworks and generalized 9 common fix patterns. We found complex symptom-to-fix relationships, yet, surprisingly, minimal configuration changes were able to fix most symptoms.

Providing consistent documentation and practical automation that assists customization of service mesh deployment and functionalities is crucial in the current service mesh domain, given the diversity of discovered intentions, goals and symptoms. Furthermore, there should be more work towards better container orchestration to deploy service mesh frameworks and reliable customization of security and observability service mesh features.

1.2.3 Chapter 4: Why Not Fixing the Fixables? An Empirical Study on Understanding the Unfixed Vulnerabilities in Helm Charts

The vast adoption of Kubernetes has led to the rise of Helm as a package manager for deploying and managing cloud-native infrastructures due to its simplicity. Nonetheless, Helm's packaging format, namely “Charts" widens the attack surface to vulnerabilities due to the “one-command" deployment of complex software prepackaged by a third party. Among the vulnerabilities disclosed, many remain unfixed despite being fixable. Our mixed-methods empirical study seeks to shed light on the nature and reasons behind such unfixed vulnerabilities. We comprehensively explore the adoption and opinions over current CVE mitigation strategies to answer why chart maintainers keep fixable vulnerabilities unattended.

This study aims to investigate the prevalent vulnerabilities, the factors that relate to the vulnerabilities, current effective mitigation strategies, and the roles of strategies in mitigating vulnerabilities in Helm Charts and suggest practical takeaways to chart users, maintainers and researchers in the domain. We conduct an empirical investigation on a large dataset of Helm charts, gathered from Artifact Hub and their corresponding maintenance repositories on GitHub.

We employ a mixed-methods methodology that backs our quantitative findings with a qualitative grounded theory investigation. We first identify the prevalence of CVEs in 11,035 Helm charts and analyze contextual factors contributing to their prevalence based on chart officiality and category. Subsequently, we review the adoption of current CVE mitigation strategies by analyzing discussions, issues, and pull requests and automation on GitHub.

Our findings highlight image count and time constraint is not among the factors that contribute to unfixed CVEs. We find that practitioners should not rely on contextual factors such as officiality to determine the security aspects of a chart. Most importantly, we formulate a grounded theory based on 12 effective CVE mitigation strategies, and highlight three key concepts: Incentive, Trade-off and Trust, which leads to a low utilization of available mitigation strategies and drives maintainers away from security best practices.

Current effective CVE mitigation strategies are under-utilized in the Helm Charts domain. We find that strategies often receive mixed maintainer stance given their trade-offs between maintenance effort, security and other non-functional requirements. We also find that chart maintainers lack the incentive to mitigate CVEs given a unclear boundary of mitigation responsibilities. Practicing the shared responsibility model would help shifting security to the left and result in safer charts for end users.

1.3 Thesis Contributions

This thesis investigate the intertwined realms of service mesh architectures and Helm Charts, which co-evolve with the evolving intricate infrastructure requirements in modern cloud-native systems. Our exploration illuminates the most pressing concerns for service mesh practitioners and the high security vulnerability risks within such technologies, offering empirically-grounded suggestions for their mitigation. Crucially, our findings of security concerns in service mesh are supported and supplemented by an expert in the domain, whose insights have guided our research direction and deepened our understanding. Our thesis contributes valuable insights towards the goal of enhancing security in a zero-trust environment where each component of the deployed system is well-secured:

- We conduct two large scale mixed-methods empirical studies of the adoption concerns of service mesh frameworks and unfixed vulnerabilities in Helm Charts. We study 5,497 practitioner questions and answers of two service mesh frameworks, security reports of 11,035 Helm Charts and 90 Helm Charts maintenance repositories.

- We reveal that error-related concerns occupy a majority (48.3%) of the attention for service mesh practitioners, signalling a pressing need for automated solutions and greater domain expertise. Furthermore, the challenges associated with deploying and managing service mesh frameworks, driven by the intricacies of underlying infrastructure layers like Kubernetes, emerge as significant obstacles through our qualitative study.

- In collaboration with a service mesh expert, we identify a substantial shift from service mesh traffic-related concerns towards infrastructure-related concerns. Concurrently, we recognize underestimated concerns related to security and observability.

- Our study uncovers the complex relationships between error symptoms and their corresponding fixes in the service mesh domain. These intricate relationships make the mitigation process challenging and underscore the need for enhanced error messages from framework designers.

- We discover a high prevalence of vulnerabilities across Helm Charts (14,448 CVEs), with many remaining unattended (84%) for extended periods (more than 100 days), underscoring the urgent need for more efficient mitigation processes and strategies.

- Our analysis also suggests that contextual factors alone cannot serve as a reliable predictor or filter for vulnerable Charts given the effect sizes are negligible.

- We comprehensively investigate and categorize 12 mitigation strategies discovered in chart maintenance repositories. Despite their potential for addressing security vulnerabilities, these strategies are significantly underused. Our grounded theory suggests the need for a shared responsibility model and greater use of informative strategies to prevent shifting the security burden to end users of Helm Charts.

CHAPTER 2

Background and Related Work

2.1 Background

2.1.1 Service Mesh Architecture

Microservices are designed for large-scale deployments with complex network conditions, involving the distribution of heterogeneous services across networks with numerous interactions between said services (7). However, with the ever-increasing requirement for user experience, large organizations face increased costs to support millions of service instances deployed around the globe[1]. Since each service is managed by an individual team, any decision to release a new version of one service could have a

[1] https://www.youtube.com/watch?v=CZ3wIuvmHeM

ripple effect throughout the ecosystem of deployed microservices. Therefore, it is crucial to understand how such complex systems are assembled before securing them.

As a variant of the microservices architecture, service mesh implements the network infrastructure used by microservices to communicate with each other differently. Previous works (52; 70; 102) define service mesh as a dedicated layer of infrastructure that manages the communication of microservices over heterogeneous networks. Essentially, service mesh separates microservices development from operational tasks such as deployment and monitoring, no longer required to physically alter the source code of a service just to enable debugging or other observability tasks.

Figure 2.1 presents an architectural comparison between a traditional microservices architecture based on shared Software Development Kits (SDKs) for inter-service communication and a typical service mesh architecture based on Istio. The left figure shows a tight relationship between individual services with a shared SDK that implements the inter-service communication features, for example, traffic monitoring. The service and the SDK are packaged within the same container, meaning minor modifications to either part would inevitably lead to a rebuild and redeployment, even if they were not meant to impact each other. Furthermore, once a service uses a given SDK, its implementation is locked into that SDK, complicating the reuse of the microservices in different settings.

On the other hand, microservices with service mesh address the above problem by adopting the "Sidecar Pattern". A sidecar container is considered a reusable unit that enhances the functionalities provided by the microservices' containers[2]. At the

[2] https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns/#example-1-sidecar-containers

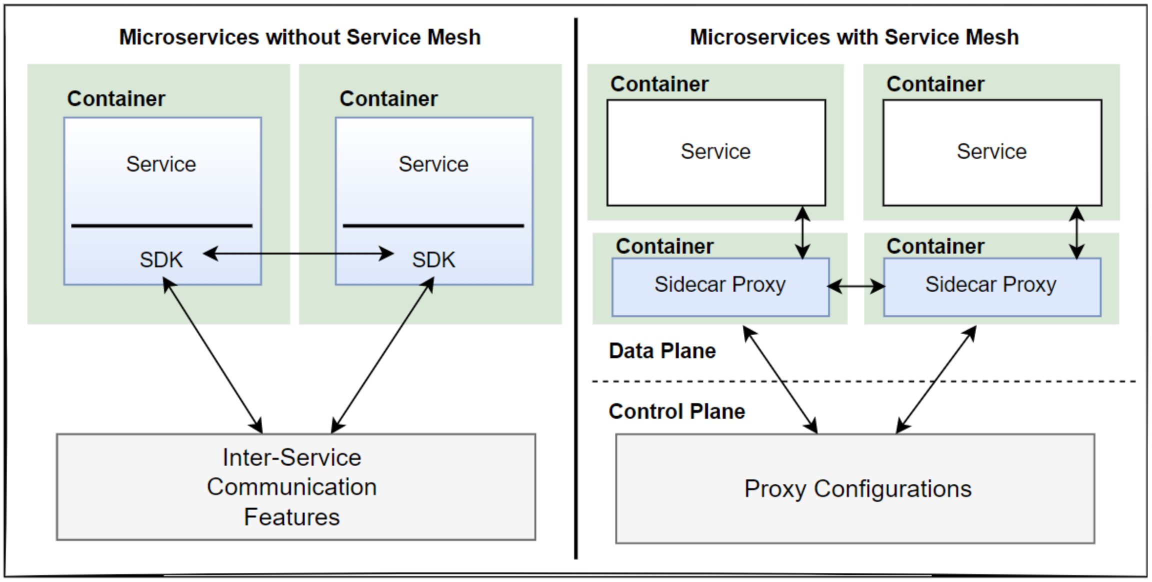

Figure 2.1: Architectural Differences between Traditional Microservices and Service Mesh

same time, they can be developed, maintained and deployed individually by the DevOps team. As shown in the figure, we observe a sidecar container deployed against each service, forcing all communication between microservices to pass between their respective sidecars, forming a mesh of nodes ("service mesh architecture"). Such sidecar containers serve as network proxies, intercepting and further processing communication based on rules set by system admins.

Figure 2.2 presents a typical service mesh-based microservices architecture consisting of two major components: a control plane and a data plane. According to the architecture definition of Istio[3], the control plane is a set of services that manages and monitors the service mesh proxies according to service mesh configurations

[3] https://istio.io/latest/docs/ops/deployment/architecture/

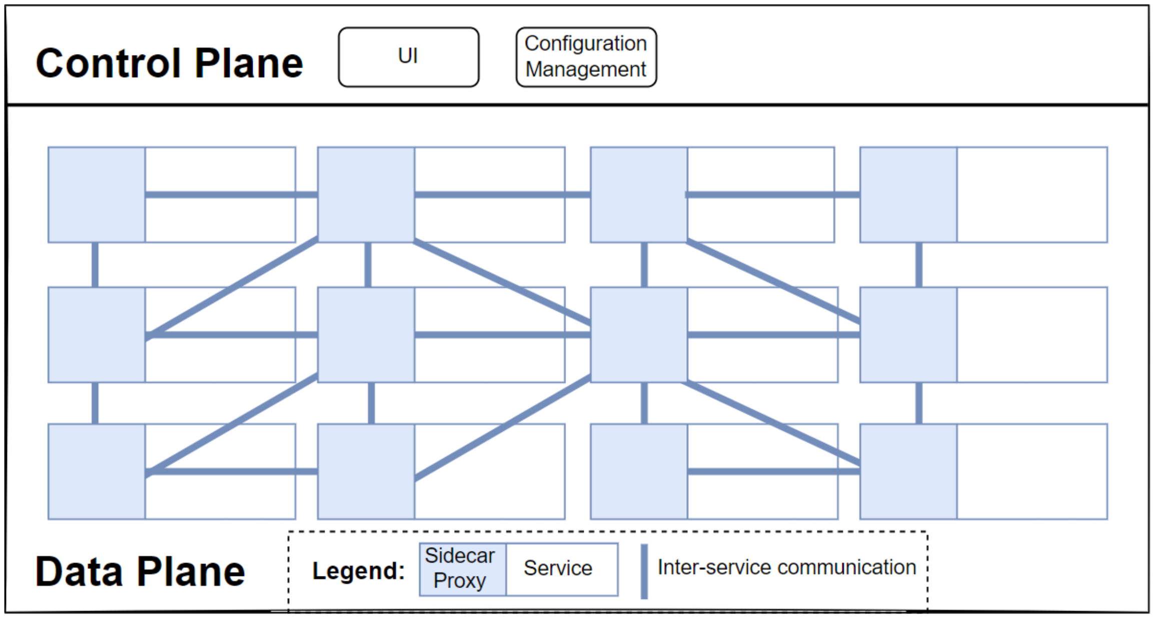

Figure 2.2: Overview of a Service Mesh Architecture

provided by DevOps engineers. The data plane accepts the aforementioned configurations and enforces the configurations by forcing communication through the microservices' proxies.

In other words, the service mesh proxies serve as a dedicated mechanism to perform advanced features, including but not limited to monitoring, routing and load balancing. Monitoring often covers the so-called "three pillars of observability data" (85), including (I): "Traces" - the distributed call graph of the monitored services, which helps to understand the topology and detect abnormal behaviour. (II): "Metrics" - the numerical measurement of specific aspects of the health of services. (III): "Logs" - the system logs gathered from distributed services, which record the detailed system events at each time. Routing serves the system to direct the traffic from outside the network to internal services based on user strategies and system logic. On the other hand, load balancers usually follow fixed strategies to distribute workload evenly among identical instances of each service (27).

2.1.2 Service mesh and Helm Charts

As service mesh architectures continue to gain traction for managing complex microservices interactions, the deployment methods employed for these architectures are evolving in parallel. Helm has emerged as a prevalent tool for streamlining the deployment and management of service mesh architectures on container orchestration platforms like Kubernetes. While service meshes provide robust solutions for microservices communication, they introduce a sophisticated layer of infrastructure that is susceptible to misconfigurations and subsequent security vulnerabilities (70). In contrast, Helm, designed to simplify deployment processes, can inadvertently mask such vulnerabilities, leading to security loopholes in the system (127).

Helm provides a way to define, install, and manage reusable applications such as databases and web servers as packaged manifests, namely Helm charts. Using YAML syntax, Helm chart maintainers encapsulate all the necessary Kubernetes resources, predefined configurations, and their dependencies required to deploy complex applications in a single package that could be distributed and imported into any Kubernetes environment. Figure 2.3 depicts a typical flow of deploying reusable (i.e., databases) and customize applications (i.e., user microservices) into service mesh through the Kubernetes API via prepackaged Helm Charts. In this way, DevOps engineers no longer have to deploy resources and their dependencies via manual command-line operations across a large service mesh cluster.

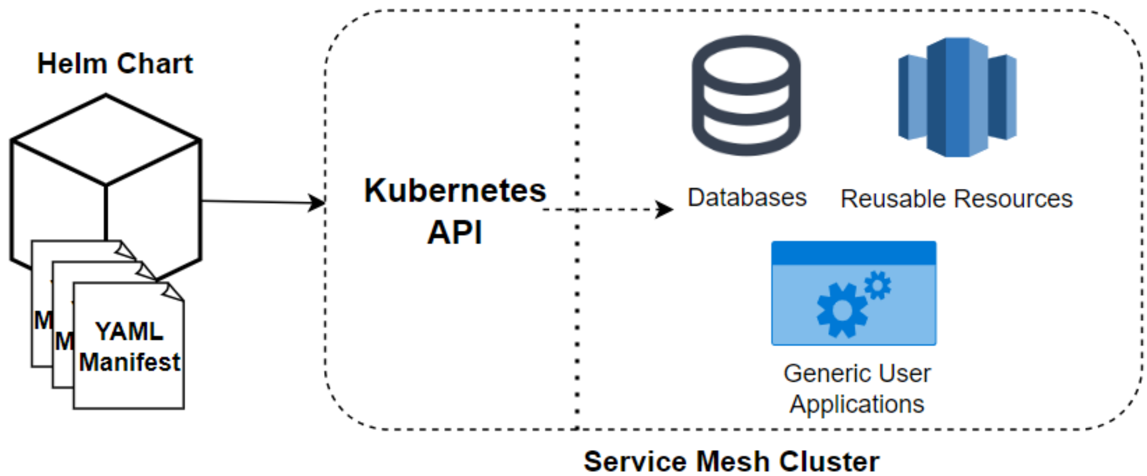

Figure 2.3: Helm Chart and Service Mesh Deployment

2.1.3 Helm Charts and Dependency Vulnerabilities

Helm charts vary in complexity depending on the application they represent. Some Helm charts are relatively simple, encapsulating a single component, while others may encompass an entire application stack composed of multiple interconnected components maintained by multiple parties and released on diverse schedules. With the number of components increasing, a Helm Chart could be exposed to drastically more vulnerabilities such as Common Vulnerabilities and Exposures (CVEs) in the dependencies, leading to a higher possibility of attack (81).

Table 2.1 depicts a matrix to differentiate the four possible vulnerability statuses in the Helm Charts domain. On the one hand, we have vulnerabilities that are fixable; Upstream dependencies with such vulnerabilities already disclose a public mitigation method and therefore could be fixed by applying patches to dependencies or updating the impacted chart dependencies. If these fixable vulnerabilities have already been addressed in the chart, we refer to them as "Already Mitigated". On the other hand, if they are yet to be handled, they are classified as “Unmitigated by Chart Maintainers,” highlighted in red, emphasizing that these require immediate attention. On the contrary, some vulnerabilities are unfixable; upstream projects have yet to mitigate such vulnerabilities, and a public way to fix them is not disclosed. Therefore any downstream projects such as Helm Charts could not patch them with reasonable effort. These are often described as “Open Vulnerabilities" when they remain unfixed[4]. It is worth mentioning that there is no concept of “fixed” for unfixable vulnerabilities. Hence the corresponding cell is marked as “N/A".

In chapter 4, We focus on the vulnerabilities that remain “Unmitigated by Chart Maintainers" since empirical evidence and effective strategies to aid such a phenomenon are yet to be extensively explored in academia and industry (127). If they remain unfixed, such fixable vulnerabilities could pose a significant risk to deployed systems (125). With a better understanding, it will allow researchers to categorize and prioritize efforts in the realm of Helm Chart vulnerability management.

	Fixable	**Unfixable**
Fixed	Already Mitigated	N/A
Unfixed	Unmitigated by Chart Maintainers	Open Vulnerability

Table 2.1: Table to differentiate between fixable vulnerability, unfixable, fixed, and unfixed

2.1.4 Artifact Hub, GitHub and Security Reports

As applications and their dependencies evolve over time, keeping Helm charts up to date is crucial to ensure compatibility and security. Like traditional software artifacts,

[4] https://docs.bitnami.com/kubernetes/open-cve-policy/

Helm charts require regular maintenance and updates to address issues and introduce new features. This maintenance process involves monitoring for vulnerabilities, addressing bug fixes, and accommodating changes in the underlying dependency stack. In this study, we use the centralized artifact registry (Artifact Hub) and the maintenance platform (GitHub repositories) to conduct a mixed-methods empirical study involving quantitative and qualitative analysis.

Helm charts follow a versioning scheme, allowing for controlled releases and updates. Version numbers help track the evolution of Helm charts and provide a mechanism for users to specify the desired version during installation or upgrade. Currently, Artifact Hub[5] is the de-facto public repository to register such Helm Charts, where it hosts over 10,000 chart artifacts, each with multiple versions.

On the one hand, we find over 50% of the charts in Artifact Hub disclose "Security Reports", which is an automatically generated report with Trivy (aquasecurity), an open-source image vulnerability scanner. Such a report contains comprehensive information on all existing CVEs in the chart dependencies while indicating their fixed version in an upstream dependency (if available). We use this information to draw quantitative insights, such as the prevalence of vulnerabilities and the impact of contextual factors, such as the application domain of the chart.

On the other hand, Helm provides commands and tools for managing Helm repositories, which serves the process of maintaining and releasing Helm charts on GitHub. Therefore, we consider GitHub a rich source of information that encodes the maintenance activities such as CVE mitigation strategies and maintainer stance over such strategies, which could enrich the quantitative data we collect from Artifact Hub.

[5] `https://artifacthub.io/`

2.1.5 CVEs, CVSS Scoring and CVSS Metrics

Similar to previous work (71), we rely on the CVSS score and CVSS vectors to measure the exploitability and impact of vulnerabilities. We collect CVSS base scores (V3) as it is the latest standard that came into effect in 2019 (45) from the National Vulnerability Database (NVD). We only consider the base scores since the temporal and environmental scores that differ within each application are beyond reasonable effort to collect, given the large number of charts (11,035 Helm Charts) in the dataset.

The CVSS v3 base score is a quantitative representation of the severity of a disclosed vulnerability. It provides a numerical score ranging from 0 to 10, with 10 being the most severe. Specifically, the score is calculated through a set of eight categorical metrics (5 exploitability metrics and 3 impact metrics) assigned by security experts to estimate the corresponding CVSS scores, each evaluating a unique characteristic of the vulnerability (45):

- Attack Vector (AV) - Through which means the vulnerability can be exploited, such as locally or remotely.
- Attack Complexity (AC) - The complexity of the attack required to exploit the vulnerability.
- Privileges Required (PR) - The level of privileges required for an attacker to exploit the vulnerability.
- User Interaction (UI) - Whether user interaction is required to exploit the vulnerability.
- Scope (S) - Whether a vulnerability in one component impacts resources beyond its scope.

- Confidentiality (C) - The impact on confidentiality of a successfully exploited vulnerability.
- Integrity (I) - The impact on system integrity of a successfully exploited vulnerability.
- Availability (A) - The impact on availability of a successfully exploited vulnerability.

2.2 Related Work

2.2.1 Empirical Studies of Service Mesh

Table 2.2 summarizes the main contributions of 11 related papers on service mesh. Next, we briefly go over the relevant publications and focus on an in-depth discussion of the three empirical studies that relate the most to this thesis.

A recent study (70) surveyed the emerging service mesh technology and the developing frameworks, including Istio and Linkerd covered in our work. They inspected service mesh capabilities and identified three high-level challenges regarding the quality attributes of a service mesh technology: performance, adaptability, and robustness. The authors suggested the direction of researching service mesh's capabilities regarding edge computation and data analysis. Previous studies in the software engineering domain have covered these aspects, focusing on the adoption of service mesh in edge computation scenarios (38; 48; 60) and on discovering the automatic operation of service mesh frameworks with machine learning technologies (98; 99; 102; 120). However, as shown in Table 2.2, empirical evidence and studies are lacking to prove and pinpoint

Table 2.2: Previous Research on Service Mesh

Publication	Year	Type	Contributions
(70)	2019	Survey	Three challenges of service mesh features and future research directions.
(38)	2020	Novel Architecture	A service mesh architecture for Internet of Things (IoT).
(60)	2020	Novel Architecture	A data driven approach to service mesh for IoT.
(48)	2020	Novel Algorithm	A service mesh management algorithm for IoT.
(99)	2020	Novel Algorithm	New algorithms for service coordination.
(98)	2021	Novel Algorithm	Deep learning algorithm for service coordination.
(102)	2021	Novel Algorithm	Adaptive service mesh circuit breaker.
(120)	2021	Novel Algorithm	Adaptive Kubernetes scheduler powered by service mesh.
(52)	2020	Empirical Study (Controlled Experiments)	Assessed the impact of flawed security features in service mesh frameworks and proposed a new threat model.
(47)	2021	Empirical Study (Controlled Experiments)	Identified the challenges to enforce performance constraints in service mesh and the need for reliable benchmark.
(96)	2022	Empirical Study (Controlled Experiments)	Identified effective configurations for Istio circuit breaking mechanisms.
This study	2023	Empirical Study (Quantitative and Qualitative Case Study)	A systematic study of existing practitioner questions, identifying rising concerns around service mesh security, infrastructure, and observability. Suggesting the need for enhanced methodologies and automation in areas of concern.

such challenges and adoption concerns of service mesh. Our work is the first empirical study to systematically elicit and understand real-world concerns faced by service mesh practitioners by analyzing user-seeded posts on Q&A platforms. Next, we cover the three empirical studies most closely related to our study.

Service mesh traffic management: A previous work (96) proposes an empirical study exploring the effects of traffic management-related functionalities in Istio. The authors manually searched and identified a comprehensive set of parameters related to network communication and collected performance metrics in controlled experiments by pinning a single parameter while tuning the remaining. The authors concluded that system administrators should take extra care when configuring the traffic policies because such features could visibly affect each other even with carefully adjusted parameters, leading to degraded performance. The authors also advise against using such mechanisms when traffic overload is not likely to persist since they will inevitably introduce adverse side effects like the connection failure symptoms we discovered in RQ3's Table 3.8. Similar to our findings, the authors remarked on the difficulty of getting the configuration values right based on actual needs and traffic characteristics, leading to a requirement for concrete suggestions and a reliable source of information.

Our study's findings on diverse service mesh error symptoms and fix patterns support some claims regarding recurrent configuration challenges. While the general trends of traffic-related work drastically declined over the last two years (see Figure 3.3), we believe that managing traffic remains a concern as the aggregated category still takes up around 20% of the total topic prevalence. Although previous work (96) derived insights from numerous controlled experiments, an exhaustive search for the best parameters could be unfeasible for practitioners as they cannot deploy their projects into production environments and experiment with the best traffic management policies on a trial-and-error basis. On the other hand, deployment into testing environments requires an accurate understanding of the ever-changing traffic volume and system specification, which is also challenging to provide valuable hints on configuring traffic policies and handling traffic overload. The conclusion supports our call for empirically verified best practices and indicative error messages whenever an unexpected behaviour occurs in the system.

Service mesh security: Another previous work (52) proposes a controlled experiment focused on service mesh security. The authors indicate that a heavy burden of security overhead was introduced by adopting service mesh frameworks that might offset the benefits of additional workload for system admins. The study tested the service mesh frameworks against adversarial scenarios and discovered design flaws that current service mesh frameworks have. Furthermore, the authors found that even experienced experts could not prevent security issues from happening in over half of the cases. The authors call for future studies to focus on security and amend the security flaws in today's state-of-the-art service mesh frameworks.

Part of our taxonomy regarding security concerns in Table 3.8 supports the previous work's claims (52) on the pressing need for more attention in this area. Our topic evolution findings within RQ2 (Figure 3.4) reinforce the authors' conclusions, as we have discovered that the security concern has maintained prevalence over the years. In RQ3, our process to derive the symptoms and fix patterns suggests that such problems in service mesh are hard to pinpoint. Practitioners without deep domain knowledge can only observe the apparent symptoms and understand the root causes from a diverse but scattered selection of knowledge resources (i.e., blogs, books and documentation). Therefore, we support the claim presented by the previous work (52) stating that expert engineers could still face difficulty dealing with mission-critical configurations (i.e., deployment, routing, security) due to inherent software flaws and a lack of reliable best practices.

Service mesh performance: A previous work (47) proposes a study focused on performance (traffic management-related topics) in a context where high-performance networking is essential for microservices to operate normally. Therefore, the researchers conduct controlled experiments to understand computing resource utilization and performance impacts, such as latency, when adopting service mesh in specialized computation domains. The authors concluded that service mesh frameworks face performance challenges when high network performance is one of the top development priorities and call for more action. However, the challenges elicited are derived from the author's experience as a practitioner in the field rather than empirical evidence derived from a systematic study.

Our study derived a list of 18 topics for the service mesh domain. Interestingly, performance was considered an implicit requirement that embeds many of the symptoms that could link to performance problems, such as cross-data-center service mesh, load balancing and proxy delays, which span across three major categories that we aggregated. We realize that service mesh frameworks require more work to optimize when serving fundamental purposes in areas with unique software requirements. As more practitioners adopt the technology, more work should be done to better understand the difficulty of transforming general-purpose service mesh frameworks into specialized solutions for those areas, given practical suggestions on practices and the correct configurations to use.

2.2.2 Helm Chart Vulnerabilities

Previous studies have started to uncover the complex landscape of Helm chart vulnerabilities. One previous work (17) introduces a methodology to graph and quantify the attack risks associated with Helm charts. They discovered that Helm charts can be vulnerable through sophisticated attack paths due to their complexity. Our work further empirically substantiates their findings, revealing a high prevalence of detectable vulnerabilities in Helm charts. Furthermore, we observed a general reluctance among maintainers to engage in sophisticated vulnerability prevention when producing Helm chart artifacts. Rather, they frequently depend on automation to update to the latest versions, which in over 50% of the studied cases, still contain vulnerabilities of high to critical severity.

Another empirical study (127) investigated the growth, reuse, dependencies, as well as outdatedness of the container images used within them. Their research highlighted that despite the exponential growth of Helm charts, most lack popularity and do not have declared open-source licenses. About 90% of the charts were found to be exposed to vulnerabilities affecting their images, often due to the use of outdated images. Our study builds on these findings by delving deeper into the reasons behind the persistence of these vulnerabilities and the inefficiencies in their mitigation through a comprehensive mixed-methods study, with statistical analysis and a qualitative grounded theory approach.

2.2.3 Package Manager Security Research

Extensive studies have been conducted across various package management ecosystems, such as npm (124), Ruby (86), and PyPI (4), among others, revealing inherent security challenges. Package managers, due to their reliance on dependencies, can often act as vectors for vulnerabilities. Helm charts, unlike traditional package managers, package up entire applications along with their configurations for deployment on Kubernetes clusters. This high level of complexity introduces new potential attack vectors that heighten security risks. Furthermore, Helm charts serve a crucial role in the software supply chain, functioning as the final artifact prior to deployment. With Helm chart's loosely assembled ecosystem and dependencies maintained by different parties, vulnerability management becomes even more challenging. Our research extends this line of research by focusing on these unique aspects of Helm and their implications for vulnerability management.

2.2.4 Supply Chain Security in Infrastructure as Code (IaC)

Supply chain security in IaC systems represents another key dimension of our study. IaC systems, by enabling the definition and provisioning of complex system components through code, bring new perspectives on the problem of security vulnerabilities. Previous studies(51; 68; 87) have explored inherent challenges of security and compliance in IaC scripts, such as those used in Terraform and Chef. Similarly, we focus on Helm charts as a form of IaC for Kubernetes environments, which remains unexplored. By combining insights from these research areas, we provide a novel and comprehensive investigation of vulnerability management in the context of Helm chart ecosystems and provide practical takeaways for researchers, chart maintainers and users.

CHAPTER 3

On Practitioners' Concerns when Adopting Service Mesh Frameworks

Microservices is an architectural pattern for heterogeneous applications that gained popularity in recent years (39). Microservices allow distributed development and deployment of software components that together form a large application. In a traditional microservices architecture, software components handling the communication between services are known as shared Software Development Kits (SDKs). With the help of SDKs, services written in different languages and frameworks follow the same standards to keep their functionalities consistent throughout a microservices network[1].

However, despite their wide adoption, traditional microservices architectures that

[1] https://www.redhat.com/en/topics/microservices/what-is-a-service-mesh

rely on language and framework-specific SDKs (70). Custom-built SDKs that suit large projects would require careful development efforts and are prone to inconsistencies due to human errors. For example, if one would like to enforce a security measure in a microservices network, the developers must carefully maintain and release multiple SDKs across their microservices to avoid security vulnerabilities led by SDK inconsistency (43). In addition to the security cost, releasing SDKs into thousands of microservices in the production environment also requires sophisticated control since different versions of SDKs may not correctly communicate with each other.

One of the major technologies trying to reduce these costs are so-called service meshes. A service mesh is a dedicated infrastructure layer that deploys and manages a list of proxies alongside each service to intercept network communication (70). By doing so, a service mesh network could monitor and manipulate traffic towards large deployments of heterogeneous services just by making configurations changes, without changing the source code of said services. Such proxies essentially eliminate the maintenance overhead of SDKs (65) and promote the reuse of microservices. According to the 2022 Annual APIs and Integration Report (21), over 40% of organizations with a microservices application implement service mesh, although the same report indicates that only 9% of those organizations' respondents claim they face "no challenges" while adopting service mesh frameworks.

As an emerging technology, service mesh may not suffice to address the adoption concerns in the microservices domain or may even lead to new critical concerns regarding their usability, performance, and runtime stability (70). For instance, service mesh proxies require dedicated management effort and are prone to occasional failure. Additionally, previous work (52) suggests that advanced features in a service mesh framework could confuse new adopters. Unfortunately, to our knowledge, existing empirical studies (47; 52; 96) on service mesh primarily focus on controlled experiments regarding framework-specific functionalities, such as the exploration of optimal traffic control configurations. Thus, empirical knowledge on service mesh adoption concerns remains scarce, if not non-existent.

This chapter introduces a mixed-method empirical study of 5,497 posts on Stack Overflow and on public discussion fora hosted by open-source service mesh frameworks. Our goal is to understand practitioners' concerns when adopting service mesh frameworks, with a particular focus on question types (e.g., "how-to" and "error-related"), topics (e.g., "Access Control" and "Machine Management"), error symptoms (e.g., "Certificate mismatch") and their common fixes (e.g., "Correct certificate signing"). We target open-source service mesh frameworks that previous work commonly studied (47; 52; 70; 96), namely Istio, and Consul. We validate and discuss the findings of our study with a service mesh expert who has over five years of cloud computing and open-source experience in the service mesh domain.

Regarding the study methodology, we use keyword heuristics to characterize the different types of questions and their distribution in the dataset. We then adopt Dynamic Topic Models (DTMs) (18) to mine the topics of the analyzed questions in terms of service mesh capabilities that concern the practitioners. We next apply aggregation and regression analysis to topic time frames of 3 months to reveal the evolution of topics and identify observable trends (e.g., increase or decrease of topic prevalence) over time. We compare our service mesh findings to traditional microservices in all these analyses. Finally, we conduct a qualitative analysis based on representative samples of knowledge-transfer, how-to and error-related questions to understand question intentions, adoption goals and error symptoms. For error symptoms, we additionally assess common fix patterns for each symptom.

Our main study results and their implications include the following:

- Error-related concerns are predominant and represent 48.3% of all concerns faced while adopting service mesh, against 29.9% for traditional microservices. On the other hand, service mesh practitioners post surprisingly few questions (6.3%) on the general concepts and practices of the technology, compared to 19.8% in the traditional microservices domain. Because service mesh practitioners are primarily concerned with fixing encountered error symptoms, practitioners may benefit from automated suggestions of common fix patterns for encountered errors. Unfortunately, the answer acceptance rate of 40.6% on Stack Overflow suggests a lack of expertise in the domain, which may be mitigated over the years with the increasing popularity of the service mesh frameworks.

- Service mesh, while aiming to be a dedicated layer to reduce operational overhead (103; 113), seems challenging to deploy and manage. We find that most (27.4%) of the questions in the service mesh domain relate to the underlying infrastructure layer based on container orchestration platforms such as Kubernetes[2], similar to the traditional microservices domain (22.8%). On the other hand, traffic-related features unique to service mesh frameworks also attract significant concerns, while traditional microservices practitioners benefit from more mature technologies such as API gateways and load balancers, leading to less concern.

[2] https://kubernetes.io/

- We discover a significant decline in the prevalence of service mesh traffic-related topics representing the technology's core functionalities, such as "Traffic and Gateway" and "Routing and Services" regarding handling and directing microservices traffic. Topics on infrastructure rebounded after an initial decline and rose along with the rapid growth of studied service mesh frameworks and underlying modern container orchestration platforms. On the other hand, we discovered and verified with the interviewed service mesh expert that the rising concerns regarding service mesh security and observability could be underestimated, which should draw more research attention.

- A majority of knowledge-transfer questions (58%) in the service mesh domain are about understanding the availability of functionalities. Carried-over knowledge from traditional microservices does not necessarily translate to a good understanding of service mesh features. While both studied frameworks are feature-rich and highly configurable, there are specific areas in each framework such as external system integration, customized routing and installation/upgrading that raise most concerns and thus should draw research attention. Providing more documentation and support in these areas could help service mesh practitioners overcome frequent concerns.

- Service mesh practitioners face diverse yet hard-to-pinpoint error symptoms and unintuitive error messages. Among the 54 error symptoms we mine from Istio and Consul, we find that 39 of the symptoms appear across the frameworks and even within different areas of concern for each framework separately. Additionally, in 29 cases, we encounter complex error-to-fix relationships, such as multiple fixes being recorded for a single error symptom, while in 15 cases, a single fix could also fix many different symptoms. It is challenging for typical practitioners to follow a systematic approach to find common fix patterns to a problem due to a general lack of consistent one-on-one relationship between an error symptom and a common fix. We encourage framework designers to enhance the error messages to facilitate smoother adoption.

- Minimal modifications in configuration manifests could fix considerable error symptoms that service mesh practitioners face. For example, modifying a simple port name according to the naming convention of Istio could fix a symptom of a defective security policy, which does not provide any error message that links to a port naming problem. We encourage researchers to explore approaches that could recommend proper configurations and examples based on unique user requirements.

- We found abundant knowledge resources in question answers that could serve as hints for fixing or preventing the analyzed error symptoms. Yet overall, we observed (I): a lack of reliable and well-documented practices in configuring service mesh systems linked to user requirements; (II): a lack of effective error diagnosis mechanisms, which render the knowledge resources less useful since practitioners cannot effectively use the knowledge within such resources to solve unclear problems.

- We provide the replication package of our study with the main study artifacts, including the data collection scripts, data processing and filtering scripts, topic models and data spreadsheets (31).

We organize the remainder of this chapter as follows. Section 3.1 motivates our study with specific research questions to be answered. We also follow each detailed step with analysis protocols and data collection techniques. Sections 3.2 to 3.4 discuss the evaluation and results discussions for each research question. Section 3.5 summarizes our key findings and further gives actionable results to researchers and practitioners in the field. Section 3.6 acknowledges and presents the potential threats to our work and justifies the methodologies we adopted to best complement some of the shortcomings. Finally, Section 3.7 concludes our findings and looks into promising research focuses in the future.

3.1 Methodology

3.1.1 Goal and Research Questions

We rely on the Goal/Question/Metric template (12) to define our study goal as follows: *to analyze* the questions and answers software practitioners posted online; *for the purpose of* revealing the recurrent concerns practitioners face while adopting service mesh in industry settings, and comparing these concerns with those observed in the traditional microservices domain; *with respect to* understanding service-mesh-related discussion topics, adoption goals, symptoms, fixes and the remaining concerns for practitioners and researchers to address; *from the point of view of* service mesh and traditional microservices practitioners and software engineering researchers; *in the context of* common open-source service mesh frameworks and their related questions and knowledge resources on Stack Overflow and dedicated Q&A fora hosted by the open-source frameworks.

We introduce the three research questions (RQs) below to achieve the above goal.

RQ1: *What types of questions about service mesh are practitioners asking?* – Throughout the rapid emergence of service mesh frameworks and products, the adoption process has not been trivial for practitioners and companies. Consequentially, many questions regarding the deployment, configuration and operation of a service mesh network exist in today's questions and answers platforms. To the best of our knowledge, such questions generated from practitioners' experiences have yet to be explored, since empirical evidence on service mesh adoption concerns is scarce. We aim to shed light on the main concerns that pose obstacles to adopting the service mesh frameworks.

In RQ1, we focus on understanding the question types and topics in the service mesh domain by gathering and analyzing concerns embedded in question post titles and bodies. We define question types to represent the nature of the inquiry. For example, the "error-related" type represents a practitioner's question on an explicit error or implicit failure while using a service mesh framework. On the other hand, topics are the top keywords expressing the central concern of a group of questions. We derive topics with the help of topic models. We aim to provide an exploratory understanding and definition for each such topic. We also compare these to the top operational concerns of the traditional microservices domain. In the end, we summarize and shed initial insights into the overall state of the service mesh domain by analyzing the question type, acceptance rate and topic definitions.

RQ2: *How did the questions of concern evolve since the emergence of service mesh?* – The earliest service mesh frameworks date to around 2016 and are still under rapid development. To understand how the topics of concern derived from RQ1 have evolved over recent years, we choose to apply a finer-grained topic analysis over time. We do this by analyzing topics across time frames and observing topic persistence, emergence and disappearance. This step allows us to depict critical insights into the characteristics of evolution and identify potential problems. Ultimately, we aim to find actionable suggestions for focuses and concerns on which to base our future work.

RQ3: *What are the question intentions, adoption goals, error symptoms, and fix patterns of service mesh problems?* – In order to extract insights from the questions and answers practitioners seed while using each service mesh framework, we use qualitative methodologies on the question posts and accepted answers for studied open-source service mesh frameworks. In order to cross-reference our findings with previous RQs, we sample questions for each question type and extract practical insights based on the characteristics of the question types. We extract question intentions for the knowledge-transfer questions to understand what conceptual knowledge practitioners expect to gain. We extract adoption goals for the 352 studied how-to questions. Finally, we analyze the error symptoms from the 475 studied error-related questions and uncover the common fixes to symptoms. We define an error symptom as the explicit failure situation a service mesh framework users face. At the same time, we enumerate possible fixes to such symptoms and define a process of fixing as a common fix if it can cure more than 50% of the particular symptom.

Inspired by previous studies (33; 73; 122) that systematically derived taxonomies to capture problems in their domain, RQ3 conducts an explorative qualitative analysis in the domain of service mesh networks by building taxonomy trees on sampled questions of each target service mesh framework and evaluate the differences and similarities. We also extract common fix patterns from the qualified answers and construct an error-to-fix mapping table. We include additional manual analysis to explore the embedded knowledge resources practitioners used in the discussions and reveal complexities that require urgent attention in the service mesh domain.

3.1.2 Service Mesh Frameworks Explored in This Work

To our knowledge, there are more than ten service mesh frameworks in today's market. Since commercial product teams provide comprehensive customer support for their products, users are less likely to ask questions on public platforms. Hence we exclude commercial frameworks during our analysis because of the potential difference in question nature and the shortage of data. Furthermore, we only collect and analyze service mesh frameworks with more than 100 posts combined on Stack Overflow and their own fora. Table 3.1 depicts the resulting four open-source service mesh frameworks that this study explores, and we briefly introduce each framework.

Table 3.1: Project Metrics of Open-Source Service Mesh Frameworks (CP: Control Plane, DP: Data Plane

Name	Feature Focus	#Stars	#Open Issues	#Closed Issues	#Past Commits	#Past Releases
Envoy	DP*	20,691	1,223	6,582	12,064	91
Istio	CP/DP*	31,675	573	15,654	18,619	262
Consul	CP/DP*	25,531	991	3,910	18,260	239
Linkerd	CP/DP*	8,960	134	925	1,433	83

Envoy: Envoy[3], first developed and open-sourced by Lyft, is designed to provide high-performance process isolation to enhance the transparency of service networks. As Lyft claimed a complete transformation from a monolithic architecture to a service mesh architecture with the help of Envoy in 2017, the project was quickly adopted by service mesh frameworks covered in this work as a base implementation of their data

[3] https://github.com/envoyproxy/envoy

plane proxies/agents (65). We do not consider Envoy as a full-service mesh framework because it often only serves as a data plane. Yet, since both Istio and Consul fully support Envoy, and questions usually involve Envoy-related discussions, we include it in our quantitative study of RQ1 and RQ2.

Istio: Istio[4] is an open-source service mesh framework backed by the community. It uses Envoy as its data plane proxy and owns a custom control plane with dashboards. Istio is considered flexible and has gained over 47% market share according to a user survey in 2020[5]. To our knowledge, Istio is the most dominant framework in the market and owns most closed issues, commits and releases over time (see Table 3.1).

Consul: Consul[6] is an open-source service mesh framework developed by HashiCorp. Unlike Istio, Consul started its evolution in 2014 (version 0.1) as a framework for complex service communications. However, the service mesh components were not formally developed and defined until late 2017. In our study, we discard any data before the framework designer of Consul redefined it as a service mesh framework in 2017. As shown in Table 3.1, Consul owns 3,910 closed issues with 18,619 commits and 239 historical releases.

Linkerd: Linkerd[7] is an open-source service mesh that graduated from the Cloud Native Computing Foundation and has since gained popularity. However, we only consider and analyze Linkerd for the first two research questions as it has gone through a complete rewrite and framework renaming (the original Linkerd implementation is

[4] `https://github.com/istio/istio`
[5] `https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf`
[6] `https://github.com/hashicorp/consul`
[7] `https://github.com/linkerd/linkerd2`

deprecated). As a result, during the data collection phase, we extracted only 81 questions with accepted answers on Stack Overflow from the Linkerd dataset. Nonetheless, the questions are further divided into two Linkerd implementations (Linkerd and Linkerd2). We could not gather sufficient data for a meaningful manual analysis to generalize practical insights. Thus, we exclude the Linkerd dataset from the qualitative analysis.

Other open-source service mesh frameworks: Our study cannot cover all open-source service mesh frameworks mainly due to the scarcity of public discussion data (fewer than 100 discussion posts after initial filtering described in Section 3.1.4). Apart from the dominant status of Istio and Consul in the industry, both frameworks provide advanced features that other lightweight service meshes do not offer out-of-the-box. Specifically, Istio and Consul offer advanced ability to customize traffic management, security and integration features. On the other hand, Kuma[8] and Traefik Mesh[9] are considered lightweight service mesh solutions, where limited features are offered in comparison to Istio and Consul. For example, secure service-to-service communication is not readily achievable in other frameworks without compromising security constraints.

On the other hand, we aim to focus on directly comparable service mesh frameworks based on similar technologies and architectures. For example, as a newcomer in the open-source world, Cilium mesh[10] relies heavily on eBPF mechanisms that were not involved in other popular service mesh frameworks such as Istio and Consul. Due to its service mesh functionality not appearing until late 2022, we excluded Cilium mesh from the data collection process.

[8] https://kuma.io/
[9] https://traefik.io/traefik-mesh/
[10] https://cilium.io/

3.1.3 Data Sources

Previous studies in software engineering (6; 8; 29; 114; 116) have used Stack Overflow and/or public fora as a source of mining practitioner concerns and perceptions. We choose to combine Stack Overflow with public fora, since a previous study (107) suggests that particular question types could be discouraged at Stack Overflow, leading some projects to prefer maintaining hybrid support channels, in our case, public Q&A fora.

Table 3.2 presents a general view of the available data on Stack Overflow and the project-specific fora of the commonly studied (47; 52; 70; 96) open-source service mesh frameworks of Table 3.1. As mentioned earlier, these frameworks have more than 100 posts combined on Stack Overflow and their own fora. The second column shows the question count on Stack Overflow, and the third column shows the question count on the official Q&A fora of the particular framework. Since practitioners post more questions in the official, project-specific fora than on Stack Overflow, our study uses questions from both types of venues. One complication is that forum discussions lack answer acceptance mechanisms such as voting. Hence in RQ3, we have to manually process the discussion posts based on domain knowledge and the state of discussions within a question thread to determine concluding answers. We exclude the forum data when analyzing answer acceptance in RQ1 due to the aforementioned reason. In summary, Table 3.2 presents a sufficiently large body of questions applicable to our analysis' data input.

Table 3.2: Count of Relevant Question Posts

Name	**# Questions Stack Overflow**	**# Questions Official Forum**	**Link to Forum**
Istio	2,283	4,054	discuss.istio.io
Consul	1,092	1,489	discuss.hashicorp.com/c/consul
Envoy	645	N/A	N/A
Linkerd	81	189	discourse.linkerd.io

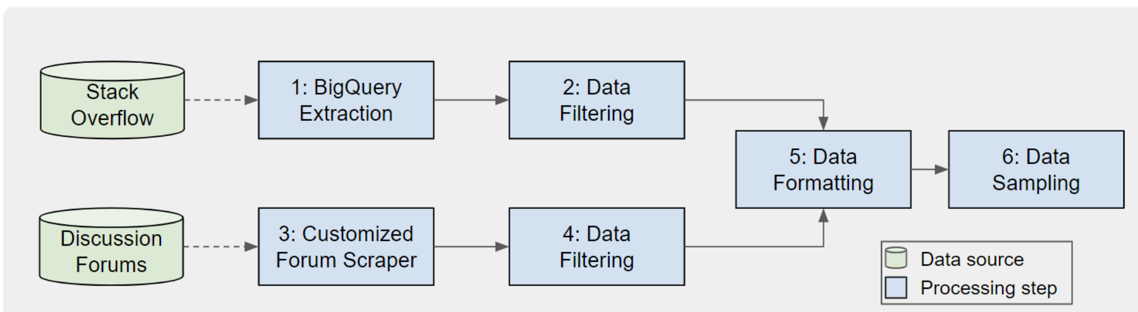

Figure 3.1: Data Collection Steps

3.1.4 Data Collection

Figure 3.1 depicts the steps we follow to collect data from Stack Overflow, and project-specific discussion fora. [11]:

Step 1: *BigQuery Extraction* – We retrieve the Stack Overflow dataset (4,101 questions) between 2017 and 2022 from the Google Big Query platform[12], which covers the earliest emergence of Service Mesh technology to the recent year. We utilize regular expressions that match the name of open-source service mesh frameworks and closely related terms "service mesh" or "mesh" in question tags, title or body. In order to enable comparison against an established domain, we also collect a separate dataset (7,498 questions) for the broader traditional microservices domain by filtering

[11] The scripts to collect, process, filter and analyze data are available in the replication package (31)
[12] https://cloud.google.com/bigquery

for the tag "microservices" and excluding the posts related to service mesh, covering data from its emergence in 2015 (111) to 2022.

Step 2: *Data Filtering* – Solely filtering based on regular expression is prone to false positives, especially in cases where irrelevant tags are attached to questions on Stack Overflow (73). Therefore, we iteratively add conditions to exclude specific keywords from question tags, titles and bodies. For example, we exclude "laravel" since it owns a tool named "Laravel Envoy" unrelated to service mesh.

Step 3: *Customized Forum Scraper* – All public fora of interest use the Discourse open-source forum software[13]. Over the course of several days, we collected public forum data (5,543 posts) starting the creation date of Istio and Consul until January 2022, the latest available data hosted on BigQuery for Stack Overflow, avoiding any interruptions of the normal operations of the websites.

Step 4: *Data Filtering* – We first apply a filtering technique similar to the Stack Overflow filtering process of Step 2. Collected data is also filtered based on labels strictly related to service mesh to exclude irrelevant posts regarding other software products covered by the forum. We also manually remove threads that are considered announcements or casual discussions.

Step 5: *Data Formatting* – The raw text goes through common text preprocessing steps, including converting to lower-case, removing non-words, removing stopwords and stemming (11). To enrich the later topic modelling process with more context, we further generate trigrams, which gather every three consecutive words into a group (e.g., "traffic routing issue") instead of considering only individual terms (e.g., "issue"). In RQ2, the posts are additionally divided based on their creation timestamps into timeframes 3 months apart.

[13] https://www.discourse.org/

Step 6: *Data Sampling* – As the input of our qualitative study in RQ3, we filter out the forum posts that do not have concluding answers, i.e., we exclude post threads that leave a question open or receive objections from other users. We also remove down-voted questions, since such questions are often considered malformed or lower quality. We discard 758 questions (13.8%) that do not belong to a specific question type by applying a keyword-based classifier introduced in Section 3.2.1. We sample the remaining question posts based on a 95% confidence level to conduct manual analysis. We split the sample size between the Stack Overflow and official fora data sources based on the ratio of available data after filtering. Table 3.3 shows the count of questions available for RQ3's analysis after filtering. Since knowledge-transfer questions are scarce after filtering, we evaluate every post instead of focusing on a sample, to provide complete insights.

Table 3.3: Count of Filtered Question Posts

Framework Name	**# Questions Stack Overflow**	**# Questions Official Forum**	**# Questions Sampled**
Istio (Error)	280	559	264
Consul (Error)	201	257	210
Istio (How-to)	201	351	197
Consul (How-to)	110	164	155
Istio (Knowledge)	18	28	46 (all)
Consul (Knowledge)	7	9	16 (all)

3.1.5 Analysis Steps

This section introduces the study protocol we follow to conduct our analyses, which is illustrated in Figure 3.2 (along with the specific output artifacts from each step).

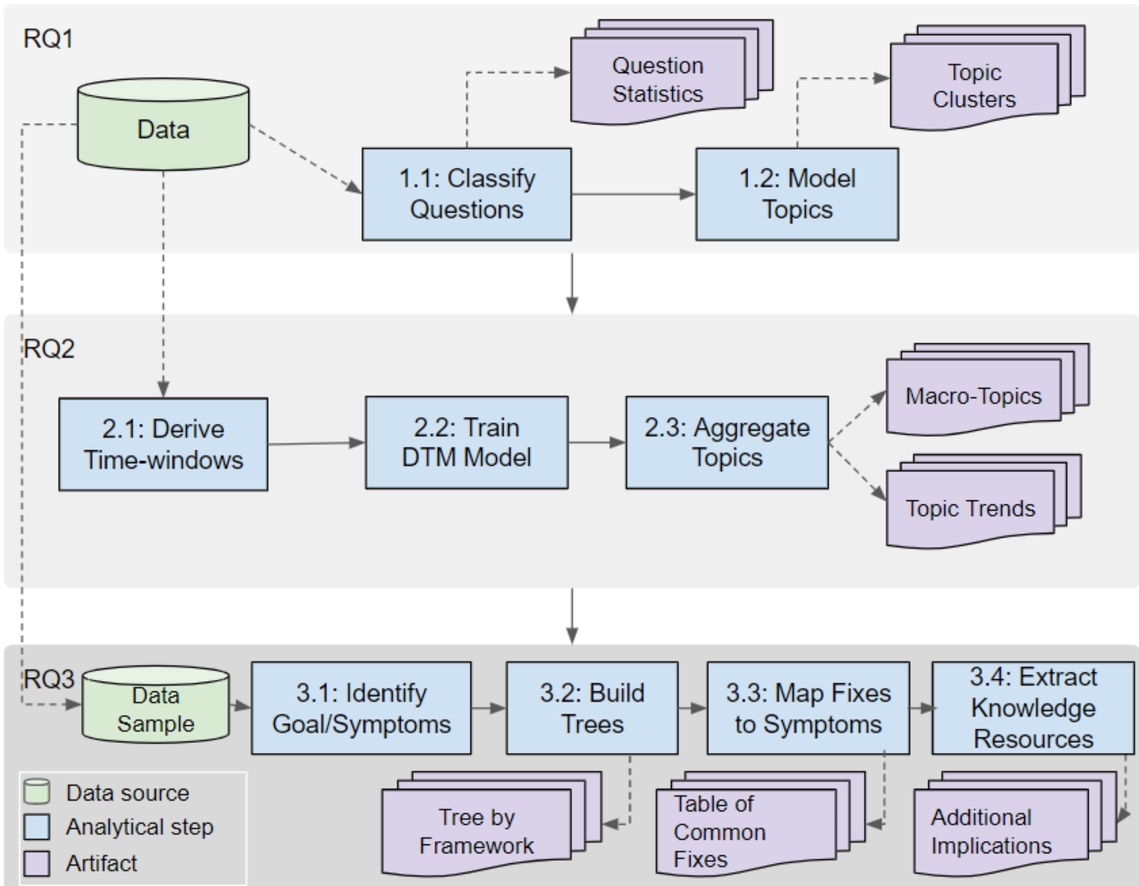

Figure 3.2: Study Protocol

Step 1.1: Classify Questions – Before analyzing the question semantics, we first identify the high-level question types via a keyword filtering process based on heuristics, which sorts questions into four types - "How-to," "Error-related," "Knowledge-transfer," and "Misc" questions. The classification is based on conditional filtering for specific keywords that we derive through iterative assessments of the questions in the domain and how practitioners form their questions. For example, we classify a question with both title and body containing "fatal" or "exception" into "Error-related" questions. The full list of keywords is available in the replication package. We only assign one question type to each question in our analysis. Step 1.1 produces the "Question Statistics" artifact, as shown in the figure. We also calculate the acceptance percentage of question posts on Stack Overflow to understand the expertise in the domain.

Step 1.2: Model Topics – With the understanding of differences in question types from the previous step, we extract topics from question posts across all question types combined. We apply an enhanced topic understanding technique called Dynamic Topic Modeling (DTM), which specializes in generating consistent topics across time intervals (18) and has been used by previous studies (8; 29; 100) to understand the adoption and concerns of new technologies. The choice of DTM is due to our intention to keep topics consistent throughout the study without needing manual topic merging, which could potentially bias our study. For fair comparisons, we apply DTM on the entire set of questions irrespective of time in RQ1 for both the service mesh domain and the traditional microservices domain.

To produce better accuracy and lower the interference from unrelated information, we implemented a threshold to filter out additional words that appear only within less than ten documents or more than 50% of the entire dataset. This step, which essentially expands the stop-word list, helps to eliminate many natural language words that do not present significant meanings.

We tune the hyper-parameters for DTM, aiming for coherent topic clusters. Since the optimal value for hyper-parameters can vary across different datasets, we follow for each domain the so-called "Elbow method" (109). Through iterative experiments (92), we minimize the marginal returns of the topic coherence **c_v** score, which is empirically shown to be an effective coherence metric for topic modelling.

We use the top keywords produced by the topic modelling process to understand the focus, and label a given topic cluster, which is a common practice used by previous studies (11; 56). In order to best depict the topic in a representative way, we collaborate with a service mesh expert to help labelling the topics based on the mined keywords and analyze representative questions. We give a concrete example for each topic and provide its definition, and provide links to all quoted examples in the Appendix A.1. Step 1.2 produces the "Topic Clusters" artifact, as shown in the figure.

Step 2.1: Derive Time-windows – In RQ2, we apply a time-framed analysis to both domains to understand topic evolution and trends. Previous studies take different approaches to understanding the evolution of topics over time. For example, studies commonly (6; 56) train individual topic models for each interval and involve manual inferences to depict continuous trends. With the time-aware capability of DTM, we map the topic occurrences to each time interval by retraining the models with additional information on the question creation time. Thus, we conduct trials at different time intervals ranging from 1 month to 12 months. We select 3 months since it best depicts trends over time while suppressing month-to-month fluctuations.

Step 2.2: Train DTM Model – We retrain the DTM models produced in Step 1.2 with additional time intervals derived in Step 2.1. We utilize the DTM model's prediction function to assign a topic to each document and plot their prevalence over time (114). In our case, the DTM model would provide consistent, framework-agnostic topics across time intervals and thus allow us to depict evolution traits explicitly.

Step 2.3: Aggregate Topics – The application of topic modelling over timestamps yields a trend for each topic over time. In order to depict clear evolution trends considering topic semantics and functionality, we further aggregate similar topics to "macro-topics" according to their definitions from Step 1.2. We sum the aggregated topic prevalence over time and visualize the shifting prevalence of concerns via a moving average trend line. Step 2.3 produces the "Macro-Topics" and the "Topic Trends" artifacts shown in Figure 3.2.

Step 3.1: Identify Question Intentions, Adoption Goals and Error Symptoms – During the qualitative analysis in RQ3, we start from our understanding of question types in RQ1 and perform detailed analysis to compare and extract user concerns in today's dominant open-source frameworks: Istio and Consul. We conduct our analysis separately on the three major question types (knowledge-transfer, how-to and error-related questions) since they embed different levels of concerns. For knowledge-transfer questions, we focus on understanding the intention to gain certain knowledge in the service mesh domain. For how-to questions, we focus on understanding practitioners' adoption goals, such as customization and performance tuning. For error-related questions, we focus on understanding their symptoms and deriving common fixes.

Since we aim to extract information from a complex domain, we utilize the open card sorting technique (121), which does not rely on pre-determined labels to understand domain knowledge and to identify sampled data. We involve two researchers (a master's student and a postdoctoral researcher), each with over three years of experience in software engineering, to assign macro-topics and to identify intentions, adoption goals, error symptoms and their fixes. We invited an additional domain expert to look at our analysis results to validate and ensure the findings are relevant according to industry experience.

Specifically, we adopt an iterative approach during card sorting to extend our understanding of the service mesh domain and minimize analytical biases by researchers. We start our analysis by using domain-specific knowledge of service meshes and individually assigning labels to each post. Throughout later iterations, we incorporate new understandings of how practitioners form their discussions and refine our knowledge. Finally, we merge and reassign labels when both researchers can achieve a strong agreement to reduce personal biases. Furthermore, if multiple non-related goals/symptoms exist in a question post, we separate them into multiple entries in the final tree. We generalize and combine similar goals/symptoms derived during the first run into fewer and more uniform formats.

Step 3.2: Build Trees – The resulting trees represent and consist of the identified adoption goals and error symptoms from Step 3.1. We assign each goal and symptom node to a parent node denoting a "macro-topic", indicating the main area of concern. We finalize the list of goals/symptoms obtained from the third open card sorting iteration in Step 3.1 to form one tree for each open-source framework to discover the differences and commonalities of goals/symptoms across service mesh frameworks. We visualize each entry with a short descriptive text. Step 3.2 produces the "Tree by Framework" artifact, as shown in the figure.

Step 3.3: Map Fixes to Symptoms – Since practitioners provide useful fixes to errors in the collected dataset, we map the fixes to symptoms by systematically analyzing the question answers. We record the specific approaches to fix each symptom and generalize them into descriptive texts. While some symptoms have diverse fixes, others could have a "common fix" if a particular symptom appears more than once and could be fixed in a common way. We treat a suggested and accepted fix in over 50% of the question posts with the same symptom derived in Step 3.2 as a "common fix pattern" (73). By the end of Step 3.3, we produce the "Table of Common Fixes" artifact, as shown in the figure.

Step 3.4: Extract Knowledge Resources – Among the posted answers, we identified abundant URLs directing to external documentation websites, blogs and posts. We additionally sample 360 questions out of 5,497 service mesh-related questions (across all question types) from the unfiltered dataset to analyze the knowledge resources practitioners used to answer the questions. We record and examine the content of the sampled knowledge resources and investigate if they serve as direct hints for answering the questions. Based on the results, Section 3.5 presents additional implications of our study, which hint at future research opportunities.

3.1.6 Validation of Findings

To validate our findings in the service mesh domain, we contacted an expert with over five years of experience in cloud computing who has contributed significantly to a service mesh startup and the Apache Software Foundation. We provided the expert with the latest draft of our study that included our findings for each research question. We requested the service mesh expert to provide feedback about our findings and possible confounding factors based on their real-world experience. Furthermore, based on the service mesh expert's feedback, we adjusted some of the textual representations of the mined topics (topic names) and of the qualitative analysis results (symptom and fix names) to provide better consistency and clarity.

To avoid bias, we first asked the expert to give insights about our RQ findings without discussing with the authors or co-workers. The service mesh expert took time to read our manuscript and write down the feedback for each major finding. We next conducted an online interview with the expert, where we discussed the practical implications of the service mesh development and adoption process. We exchanged ideas for practical automation and unified implementation standards based on recent developments and trends in related technology.

In this chapter, we will refer to the expert as "the service mesh expert." We will discuss the service mesh expert's opinions about our findings in each research question and provide implications backed by practical experience on service mesh technology in Section 3.5.

3.2 Topic Analysis on Questions (RQ1 Results)

3.2.1 Question Types and Acceptance Ratio of Answers

Almost half of the questions in the service mesh domain are error-related, compared to 30% for the traditional microservices domain. On the other hand, conceptual questions are rare in the service mesh domain.

Table 3.4 summarizes the 4 question types classified by our heuristic-based approach. We conduct a Chi-Squared test on the question type distributions of both domains (76), which yields a low p-value (0.008374) compared to a threshold of 0.05. Therefore we reject the null hypothesis that the two groups have the same distribution of question types, accepting the alternative hypothesis of a significant difference in question type distribution. Within the service mesh domain, the percentage of error-related questions at 48.3% is significantly higher than that of how-to questions (31.6%) and eight times higher than that of knowledge-transfer questions (6.3%).

Table 3.4: Question Types of Service Mesh and Traditional Microservices

Question Type	**Service Mesh**	**Acceptance Rate**	**Traditional Microservices**	**Acceptance Rate**
How-to	31.6%	40.4%	36.9%	40.3%
Error-related	48.3%	40.6%	29.9%	37.2%
Knowledge-transfer	6.3%	39.1%	19.8%	43.4%
Misc	13.8%	40.4%	13.4%	41.6%
Total*	**5,497**	**1,297**	**7,168**	**2,559**

*Total accepted answers only cover Stack Overflow data

Surprisingly, only a minority of questions are concerned with general concepts and design choices. We assume that many service mesh practitioners already know basic conceptual knowledge when adopting service mesh, therefore, producing fewer questions on concepts. In comparison, the traditional microservices domain appears more evenly distributed in terms of question types. How-to questions (36.9%) in the traditional microservices domain moderately outnumber error-related questions (29.9%), while knowledge-transfer questions also take up a significant percentage of the total at 19.8%.

Below, we define the four types of questions in the context of service mesh.

How-to questions indicate the user's confusion on the way to solve a problem within the technology involved. Example P01 demonstrates a typical how-to question on integrating a commercial monitoring tool into Istio.

> [P01] *I am new to Istio service mesh. I have to integrate/configure AppDynamics in Istio. I have no clue how to do that. Anything related to this would help. Any example*

or related links or video...anything.

Error-related questions either indicate an explicit error or implicit failure in the service mesh. Example P02 demonstrates a typical error-related question on overcoming an error thrown by Istio.

[P02] *Following blog post [URL] I am trying to deploy this sample service on my AWS k8s through Istio gives me "error: no objects passed to apply"*

Knowledge-transfer questions indicate the intention to understand the meaning or existence of a general concept/practice while not digging deep into technical details. Example P03 demonstrates a typical knowledge-transfer question where the user tries to understand why Istio provides an mTLS feature.

[P03] *I try to understand why Istio has the mTLS feature? It enables mutual TLS authentication between all the services in a cluster via automatically issued certificates.*

Misc questions indicate either a combination of previous question types due to a mixture of keywords or questions that do not fit any type. The indications are likely implicit and indirect, meaning our heuristics failed to classify these questions into a specific type. While it is hard to provide a typical example for this question type, any question that does not belong to the above three categories is automatically resolved to this type.

For each question type, we also calculate the percentage of questions on Stack Overflow with accepted answers. This analysis does not consider forum posts because they do not explicitly indicate acceptance status. As shown in Table 3.4, we arrive at a similar and relatively low acceptance rate of around 40% for both domains, independent from question types, meaning most questions lack a satisfying answer. We presume the service mesh domain in general suffers from a lack of expertise amongst the Q&A platform participants.

Feedback from service mesh expert The service mesh expert mentioned that they are not very surprised that service mesh practitioners ask few questions regarding the conceptual and high-level information of service mesh frameworks, "*Service mesh is a new technology but is derived from and extends the solution to traditional microservices architecture, many of the concepts are similar to those in traditional microservices.*" In terms of the high number of error-related questions and low acceptance rate, the expert pointed out that the increasing prevalence of error-related concerns among service mesh users could be partially attributed to the fact that many cloud vendors build their own products on top of open-source implementations, which often introduce various optimizations, limitations, and tightly integrates with their own commercial products, "*This can cause the examples on the internet to work on one vendor but not work on the other, leading to confusion and frustration for practitioners.*"

3.2.2 Topics, Macro-Topics and Definitions

We identified 18 topics in the studied service mesh questions across all types, belonging to 5 macro-topics.

Table 3.5 depicts the topic clusters and top ten keywords that form a particular topic derived from the combined dataset across all question types. We rank the topic clusters in terms of the count of documents categorized into the topic, as obtained via the prediction method of the DTM topic model. With the help of the service mesh expert, we assign one short phrase as the name of each topic based on top keywords and by analyzing typical examples in the dataset.

We manually aggregate the topics into "macro-topics" based on topic definitions (semantics) to provide deeper insights into service mesh adoption concerns. These "macro-topics" reflect a more general area of concerns that are agnostic of framework and technology. The last column of Tables 3.5 and 3.6 show the macro-topics that each topic belongs to given their definitions and samples. A topic could be assigned with two macro-topics if the topic consists of questions that relate to multiple concerns.

In the following, we illustrate each topic with a real-life example question from our dataset. We utilize the inference method of the trained dynamic topic model to pick the question samples programmatically by assigning a topic of highest probability for each question post in the dataset. Some question quotes are slightly modified to hide lengthy code snippets and URLs.

Traffic and Gateway: The *Traffic and Gateway* topic comprises issues regarding "ingress" gateways, which describe a component on the edge of a service mesh that handles incoming traffic. Example P04 demonstrates a typical how-to question about a newcomer to the service mesh domain, asking about ingress gateway concepts and basic usages in the context of service mesh.

> [P04] *I have recently got into Istio and trying to wrap my head around the gateway concept.*
> *so fundamentally, I get what it is: an entryway into the service-mesh. However, I do not understand how best to use the gateways.*

Routing and Services: The *Routing and Services* topic comprises issues regarding directing and orchestrating service traffic flow through the microservices architecture. As discussed in Section 2, a modern microservices architecture involves many

Table 3.5: The Topic Distribution of the Service Mesh Domain

ID	Prev.	Topic	Top Keywords	Macro-Topic
1	9.8%	*Traffic and Gateway*	access, deploy, ingress, namespace, kubernete, yaml, install, apply, ingressgateway, operator	Traffic
2	8.0%	*Routing and Services*	route, traffic, request, virtualservice, rule, host, limit, destination, match, version	Traffic
3	7.8%	*Kubernetes and Deployment*	helm, kubernete, version, install, chart, installation, deploy, setup, enable	Infrastructure
4	7.6%	*Networking and Sidecar*	sidecar, traffic, namespace, proxy, egress, enable, inject, container, injection, kubernete	Infrastructure
5	6.5%	*Machine Management*	client, agent, node, register, address, nod, connect, machine, instance, configuration	Infrastructure
6	6.4%	*Configuration Management*	value, file, template, command, configuration, store, update, config, pass, parameter	Infrastructure
7	6.3%	*Access Control*	policy, authentication, authorization, user, access, jwt, request, deny, rbac, rule	Security
8	6.1%	*Request Monitoring*	request, header, call, trace, dns, grpc, http, com, domain, query	Observability, Traffic
9	5.5%	*Docker and Containers*	container, docker, host, image, machine, compose, swarm, traefik, instance, command	Infrastructure
10	5.2%	*TLS Certificates*	certificate, tls, cert, client, enable, https, configure, manager, sign, ingress	Security
11	5.2%	*Application Deployment*	app, proxy, connect, nginx, log, com, web, frontend, kubernete, container	Application, Infrastructure
12	5.1%	*Spring and Service Discovery*	spring, boot, property, cloud, register, configuration, bootstrap, class, yml, discovery	Application, Traffic
13	4.0%	*Infrastructure Monitoring*	metric, upgrade, plane, control, istiod, version, endpoint, component, configuration, monitor	Observability, Infrastructure
14	3.8%	*Health Check*	health, fail, log, agent, status, leader, info, message, state, watch	Observability
15	3.2%	*Envoy Filters*	envoy, filter, source, connection, debug, proxy, info, pilot, http, tcp	Traffic
16	3.2%	*Node Ports*	port, ingressgateway, tcp, system, default, nodeport, connection, listen, expose, host	Traffic
17	3.1%	*Load Balancing*	load, balancer, balance, release, support, test, ingress, cpu, documentation, configure	Traffic
18	2.5%	*Secret Management*	vault, connect, reset, connection, datum, disconnect, secret, storage, token, failure	Security

frequently interacting services. Therefore, guiding traffic within the network is essential to serve complete services to external users. Example P05 demonstrates a typical how-to question on routing traffic to an external website.

> [P05] *I am trying to set up a Virtual Service such that any traffic on "/" gets routed to google.com. I can get Virtual Services to work with any in-cluster pods/services, but I cannot seem to configure Istio to route to anything outside the cluster.*

Kubernetes and Deployment: The *Kubernetes and Deployment* topic comprises issues regarding installing and deploying the service mesh frameworks in a Kubernetes (25) container environment. Example P06 demonstrates a typical error-related question on installing Istio into a container cluster.

> [P06] *I am trying to install istio using helm. I get an error "forbidden: attempt to grant extra privileges." I am using Azure AKS cluster..*

Networking and Sidecar: The *Networking and Sidecar* topic comprises issues regarding sidecar containers, which involve the proxies in service mesh frameworks. Example P07 demonstrates a typical error-related question on adding a sidecar to an existing Kubernetes pod.

> [P07] *I am trying to manually inject istio sidecar into an existing deployment according to the instructions here: [Istio website url] I am getting the following error, [Error trace-back]*

Machine Management: The *Machine Management* topic comprises issues regarding the registration of service nodes (clients) in a service mesh framework and relates to the overall infrastructure. Example P08 demonstrates a typical how-to question on connecting clients to service mesh server nodes.

[P08] *I'm testing a consul server cluster. I am using the go client for this. How do I enter multiple servers for the client to connect to?*

Configuration Management: The *Configuration Management* topic comprises issues regarding the value of specific configuration options to set up a service mesh, for instance, setting up routing rules through YAML manifests. Example P09 demonstrates a typical how-to question on configuring Istio after installation.

[P09] *Every document I found only tells you how to enable/disable a feature while installing a new Istio instance. But I think in a lot of cases, people need to update the Istio configuration. Accessing External Services, in this instance, it says I need to provide "flags-you-used-to-install-Istio", but what if I don't know how the instance was installed? Address auto allocation, in this instance, it doesn't mention a way to update the configuration. Does it imply this feature has to be enabled in a fresh installation?*

Access Control: The *Access Control* topic comprises issues regarding service mesh policy-related features that control the network's security policies. Example P10 demonstrates a typical error-related question on setting up JWT auth for their service backend in Istio.

[P10] *I have implemented an istio policy so that users will need a JWT token to access my backend and admin-backend services. However, it is not letting me through with a valid token. I am running istio-demo on Minikube and have done nothing with my deployment but configure an egress for auth0. Then when I go to apply my policy, I can no longer access these services with my requests.*

Request Monitoring: The *Request Monitoring* topic comprises various issues regarding requests and request header interception. Though this topic is closely related to service traffic management, it also comprises questions on traffic monitoring. Therefore, we assign both macro-topics to this topic. Example P11 demonstrates a typical knowledge-transfer question on the header used for tracing in Istio and Envoy.

[P11] *I was trying to understand the tracing in istio. According to istio documentation, x-request-id can be used for tracing purposes.(https://istio.io/latest/docs/tasks/observability/distributed-tracing/overview/) I see different behaviour in Istio vs pure envoy proxy.*

Docker and Containers: The *Docker and Container* topic comprises issues regarding container orchestration outside of Kubernetes. Example P12 demonstrates a typical question on controlling the startup order of service mesh containers.

[P12] *I am trying to implement a service mesh to a service with Kubernetes using Istio and Envoy. I was able to set up the service and istio-proxy, but I am not able to control the order in which the container and istio-proxy are started.*

TLS Certificates: The *TLS Certificates* topic comprises issues regarding the use of encrypted communication inside a service mesh network. Example P13 demonstrates a typical how-to question on setting up end-to-end TLS in the Consul service mesh.

[P13] *I'm having some difficulty understanding Consul end-to-end TLS. For reference, I'm using Consul in Kubernetes (via the hashicorp/consul Helm chart). Only one data center and Kubernetes cluster - no external parties or concerns.*

Application Deployment: The *Application Deployment* topic comprises issues regarding the applications deployed to a service mesh. Since this topic covers both the application side and infrastructure (Kubernetes deployment), we assign both macro-topics. Example P14 demonstrates a typical error-related question on deploying Istio's BookInfo example application.

> [P14] *I am trying to evaluate istio and trying to deploy the bookinfo example app provided with the istio installation. While doing that, I am facing the following issue.*

Spring and Service Discovery: The *Spring and Service Discovery* topic comprises issues regarding Spring-based applications deployed to a service mesh and the discovery of services. To our knowledge, Spring is a dominant microservices framework that frequently integrates with service mesh while providing service discovery functionalities. Since this topic covers both the application side and traffic (service discovery), we assign both macro-topics. Example P15 demonstrates a typical error-related question on setting up a SpringBoot application in Istio.

> [P15] *I have an application running in Minikube that works with the ingress-gateway as expected. A spring boot app is called, the view is displayed, and a protected resource is called via a link. The call is forwarded to Keycloak and is authorized via the login mask, and the protected resource is displayed as expected. With Istio, the redirecting fails with the message: "Invalid parameter: redirect-uri."*

Infrastructure Monitoring: The *Infrastructure Monitoring* topic comprises issues regarding service mesh observability that do not concern the observability of actual user applications. Infrastructure monitoring often relies on monitoring metrics such as CPU utilization and memory, which are the performance indicators of a system. This topic includes questions related to the infrastructure of the service mesh frameworks. Therefore, we assign both macro-topics to this topic. Example P16 demonstrates a typical knowledge-transfer question on the possibility of replacing an out-of-box metric in Istio with the desired metric.

> [P16] *Is there any way to replace istio-tcp-connections-closed-total metric with istio-requests-total for outbound traffic going through egress gateway?*

Health Check: The *Health Check* topic comprises issues regarding the health-checking mechanisms commonly used to determine if a service is prepared to accept traffic. Example P17 demonstrates a typical error-related question on unexpected behaviour when querying the Consul health-check endpoint.

> [P17] *I have a consul agent server on localhost:8500. Then I have a simple HTTP server, which first registers in Consul [snippet], And it answers on getting a request for "healthCheck" [Code Snippet] But Consul's health checker still says that Check is now critical for my service.*

Envoy Filters: The *Envoy Filters* topic comprises Envoy proxies and proxy filtering issues. Example P18 demonstrates a typical error-related question on the connection to a database hosted behind Envoy proxy in Consul.

> [P18] *I have a MariaDB database installed directly on a host and a Nomad cluster hosting a phpMyAdmin. Both hosts are inside the same Consul cluster. I'm having some issues trying to connect the phpMyAdmin to my database. phpMyAdmin returns the error:*

Node Ports: The *Node Ports* topic comprises port number-related networking issues in a service mesh regarding inter-communicating components. In the case of Consul and Istio, a node port defines a static port every node in a cluster listens on. Example P19 demonstrates a typical knowledge-transfer question on why Istio opened random ports.

> [P19] *Who/what assigns these port numbers? It seems so magical, and I don't like Istio to open up random ports on my nodes; this is a security concern to me!*

Load Balancing: The *Load Balancing* topic comprises issues regarding traffic load distribution in a service mesh architecture. Example P20 demonstrates a typical how-to question on intercepting load balancer traffic using Istio.

> [P20] *I want to control/intercept the load balancer traffic using Istio. Istio gives you the ability to add a mixer on a service level, but I want to add some code on a higher level just before the request traffic rules get executed. Thus instead of adding actions per service, I want to have some actions executed just after the request was received from the load balancer.*

Secret Management: The *Secret Management* topic comprises issues regarding the storage of secrets in service mesh networks since the large number of microservices and the infrastructure itself require credentials to communicate with each other. Example P21 demonstrates a typical error-related question on integrating HashiCorp Vault with Istio.

> [P21] *The first issue I had is Vault, and Istio sidecar is not running properly, and the application can not be able to init as below. I tried to use the below annotations to init the first vault, but it did not solve the below issue.*

In order to put the service mesh topics in perspective, Table 3.6 presents the topics for the traditional microservices architecture, where practitioners dominantly use a hybrid development framework such as SpringCloud[14] to create and deploy applications. While the microservices topic clusters contain a combination of operational

[14] https://spring.io/projects/spring-cloud

and application-side concerns, the latter generally do not apply to service mesh environments because service mesh is intended to solve operational complexities rather than development complexities. As a result, we focus on understanding the prevalence of similar operational concerns (Security, Infrastructure, Traffic and Observability) in the microservices domain and do not discuss in detail the strictly application-related topics (Topic 2,3,4,6,9,10,11,12,13,15,16). We highlight the comparable topics (Topic 1,5,7,8,14) in bold.

Next, we discuss the subset of traditional microservices topics that are closely related to operational concerns, i.e., macro-topics “Traffic,” “Security” and “Infrastructure”.

Access Control (ID 1): The *Access Control* topic comprises issues regarding the authentication and authorization features within a microservices system. This topic overlaps with the service mesh topic “Access Control”. Example P22 demonstrates a typical knowledge-transfer question on the best practices of introducing authentication into microservices.

[P22] *Is it a best practice to have auth as a separate service in micro-service architecture application? I saw in some microservices app, the authentication is part of each micro-services as inbuilt Thanks*

Kubernetes and Deployment (ID 5): The *Kubernetes and Deployment* topic comprises issues regarding the overall containerization problems involving Kubernetes. This topic overlaps with the service mesh topic “Kubernetes and Deployment” due to a common focus on Kubernetes container orchestration and underlying infrastructure. Example P23 demonstrates a typical how-to question on deploying pods in Kubernetes.

Table 3.6: The Topic Distribution of the Traditional Microservices Domain

ID	Prev.	Topic	Top Keywords	Macro-Topic
1	10.2%	**Access Control**	authentication, access, authorization, token, jwt, security, oauth, session, resource, auth	Security
2	10%	Method and Error Handling	method, error, code, response, exception, class, rest, controller, endpoint, value	Application
3	8.3%	Message Queue	communication, consumer, queue, kafka, topic, rabbitmq, connection, consume, publish, bus	Application
4	7.2%	RESTful Design	pattern, design, read, understand, example, implement, resource, book, rest, document	Application
5	6.5%	**Kubernetes and Deployment**	kubernete, cluster, pod, deploy, azure, route, fabric, node, nginx, expose	Infrastructure
6	6%	Spring Framework	module, boot, property, dependency, class, configuration, package, build, repository, error	Application
7	5.9%	**Docker and Container**	docker, container, image, compose, machine, host, error, build, deploy, port	Infrastructure
8	5.9%	**Service Discovery and Load Balancing**	instance, eureka, load, register, discovery, boot, cloud, registry, configuration, balancer	Application, Traffic
9	5.8%	Account Management	customer, store, transaction, account, source, state, command, update, example, system	Application
10	5.7%	Database Transactions	product, table, update, query, information, store, record, data, fetch, item	Application
11	5.5%	Testing and Integration	test, process, job, thread, response, memory, testing, worker, status, integration	Application
12	5.4%	Entity and Domain Logic	entity, model, domain, layer, object, function, business, cache, logic, validation	Application
13	5.1%	Web Development	app, web, frontend, component, page, core, build, backend, react, asp	Application
14	4.5%	**Version and Pipeline**	version, deploy, environment, team, development, deployment, build, production, pipeline, manage	Infrastructure
15	4.2%	Shared Libraries	share, approach, system, code, schema, option, library, dependency, graphql, practice	Application
16	3.6%	Logging and Debugging	log, com, jar, http, error, localhost, port, web, springframework, core	Application

[P23] *I use RPC protocol app as a micro-service and an API Gateway in front of them as a proxy. My question is how could I directly access k8s's pod, because I must build a connection pool to hold these connections, for example, I have three RPC micro-service pod.*

Docker and Containers (ID 7): The *Docker and Containers* topic comprises issues regarding similar concerns with Kubernetes. Unlike service mesh frameworks that rely heavily on Kubernetes, early microservices adopters heavily used Docker to deploy applications, which is also reflected in the "Docker and Container" topic of service mesh. Example P24 demonstrates a typical error-related question on migrating services from localhost to docker swarm.

[P24] *I have spring-cloud app with 3 services: eureka-server + gateway(Zuul) + user-service (2 instances). In localhost, everything is working, and I can access and get a simple string response, but when I deploy the app in docker swarm, I get an exception when accessing this endpoint.*

Service Discovery and Load Balancing (ID 8): The *Service Discovery and Balancing* topic comprises issues regarding traffic-related problems such as service discovery and load balancing strategies. This topic could map to the "Load Balancing" topic in the service mesh domain due to the fact that service mesh frameworks typically rely on existing service discovery solutions that can be used independently. Example P25 demonstrates a typical error-related question on service discovery and load balancers.

[P25] *Now, the issue is that once I give Eureka URL as [URL] in the microservice, it registers only on one instance depending on which Eureka instance the load-balancer has redirected the request. But even though peer-to-peer awareness is setup, microservice is not getting registered on the other eureka instance.*

Version and Pipeline (ID 14): The *Version and Pipeline* topic comprises issues regarding the versioning, isolation and deployment of services so that it does not impact user experience. Although it does not map to a specific topic in the service mesh domain, such capabilities are greatly enhanced when service mesh routing components offer advanced deployment models such as Canary Deployment and Blue-Green Deployment[15]. Example P26 demonstrates a typical knowledge-transfer question on best practices regarding service versioning in a traditional microservices architecture.

> [P26] *I go into microservices architecture based on docker, and I have three microservices, which together create one product, for example, "CRM system." Now, I want my client to be able to upgrade his product whenever he wants to. I have 3 different versions of my microservices; which one should the client see? I guess the product version should be independent of microservices because copying one of the microservices versions would make me go into more trouble than having no version at all. So is there any pattern, idea to handle such a situation?*

3.2.3 Macro-Topic Prevalence and Comparison with Traditional Microservices

Infrastructure-related questions are dominating both in the domain of service mesh and traditional microservices.

The "Traffic" macro-topic in Table 3.5 taking up to 24.1% topic prevalence is closely related to various aspects of traffic management functionalities involving gateways, routing rules and load balancers (70). Such high prevalence reflects that traffic management is at the core of service mesh design, since rerouting traffic is essentially what

[15] https://istio.io/latest/docs/ops/deployment

a service mesh network is supposed to do. Understandably, users generate many questions regarding such core features.

While the traffic-related macro-topic concerns the core abstractions and functionalities provided by the service mesh technology (103; 113), to our surprise, the "Infrastructure" macro-topic has the highest prevalence of 27.4%. This macro-topic is agnostic to specific service mesh frameworks, mainly concerning container orchestration platforms that serve as the underlying infrastructure of service mesh frameworks. Although it is known that service mesh frameworks heavily rely on containerization to provide flexibility and extensibility (39; 70), our findings suggest that such dependency on container orchestration platforms involves a much higher degree of complexity than the actual service mesh technology itself. This suggests that service mesh and related emerging technologies would benefit from more research efforts to ease the use of related deployment and containerization technologies as, apparently, they often are problematic for out-of-the-box usage (113).

The "Security" macro-topic relates to access control, TLS certificates and secret management, which covers 15.3% of the total questions, indicating a considerable focus on safeguarding service mesh networks and deployed services. Among the topics concerning security, "Secret Management" related concerns show less prevalence than other topics. In reality, secret storage is often deployed as an external service, such as Hashicorp Vault[16], meaning that part of the questions could be directed to their dedicated platforms without mentioning service mesh, causing an underestimation of the actual importance of this concern.

The "Observability" macro-topic covers questions on monitoring system telemetries as introduced in Section 3.1.2. Since observability is a software system property

[16] https://www.vaultproject.io/

that could apply to either application traffic or service mesh internal components, we find that the “Request Monitoring” topic (ID=8) and “Infrastructure Monitoring” topic (ID=14) each relate to the macro-topics of traffic and infrastructure, respectively. We keep both macro-topics in the table to distinguish different focuses of observability in the domain.

The “Application” macro-topic considers the software projects hosted by the service mesh network rather than the service mesh functionalities. The macro-topic adds up to 10.3% of the total questions, indicating a minor concern. Since issues related to application development are not amongst the design goals of the service mesh technology, we do not further investigate this macro-topic in the study.

While service mesh frameworks are deployed as a dedicated layer to solve operations-related complexities, traditional microservices frameworks are intended as a hybrid of development and operations. As suggested in previous discussions, we only compare to the relevant macro-topics, namely “Infrastructure,” “Security,” and “Traffic”.

The "infrastructure" macro-topic covers 22.8% of all questions in the traditional microservices domain, with a focus on containerization technologies such as Kubernetes and Docker. This macro-topic has significant prevalence compared to other possible aggregations. Infrastructure issues, including containerization and deployment versioning, are presumed to be some of the core operational complexities that concern microservices adoption.

We only observe one security-related topic in the traditional microservices domain, yet “Access Control" ranks as the most prevalent individual topic (10.2%). On the other hand, service mesh security topics are more diverse, including topics on “TLS Certificates" and “Secret Management". Such a difference can be explained by service mesh frameworks' ability to adopt the "zero-trust architecture" pattern, which is a popular emerging security model to counter the surge of cyberattacks and assumes all communication channels between all entities should not be trusted and therefore should enforce encryption (95). However, implementing a zero-trust architecture requires the ability to intercept the traffic between each service and component, which is only available in the service mesh domain.

Surprisingly, traffic-related concerns are scarce in the microservices domain (5.9%) relative to its application-related questions. To the best of our knowledge, traditional microservices rely on API gateways and load balancers to handle traffic management, yet they do not offer the detailed service-to-service level features brought by sidecar proxies. The maturity around API gateways and load balancers could result in such a low prevalence. Conversely, service mesh traffic management features are more complex and cloud expose more problems as a new technology.

Summary of RQ1: A high percentage at 48.3% of error-related posts in the service mesh domain indicates that most questions are related to unexpected problems. Despite the core concerns around traffic, service mesh still faces major infrastructure concerns involving containerization.

3.3 Evolution Analysis of Question Topics (RQ2 Results)

3.3.1 Evolution of Service Mesh Topics

The evolution of service mesh domain topics indicates gradually fewer questions on core (traffic-related) functionalities, while concerns regarding security and infrastructure persist, and observability became an emerging concern.

Figure 3.3 shows the resulting trends of the traffic-related macro-topic (and its constituent topics). We represent each topic with a solid line with different colours, and the green dots represent their summed values. In addition, we calculate the moving average of summed prevalence values to suppress short term fluctuations in the trends. As a result, we observe in the traffic macro-topic a gradual decrease from 35% to 15% in moving average over the last five years. The impactful topics "Traffic and Gateway" and "Routing and Services" have significantly shrunk in 2019 and 2020. On the other hand, the "Node Port" and "Load Balancing" topics have persisted steadily over time.

To support our findings, we calculate the mean time to question acceptance from the creation of the question post to question acceptance on Stack Overflow of the topics, resulting in 274 hours for the port-related questions, compared to 353 hours over the entire domain. In our understanding, questions related to ports are more straightforward regarding error messages, thus, easier to fix in a service mesh network. Similarly, load-related questions own a mean acceptance time of 272 hours. Load balancers are often considered a mature technology with well-known strategies (27), possibly contributing to a more limited number of new questions. From Figure 3.3, we can confirm that, despite a bump in 2020, traffic-related questions diminish over time and pose less concern to practitioners as the service mesh frameworks gradually mature.

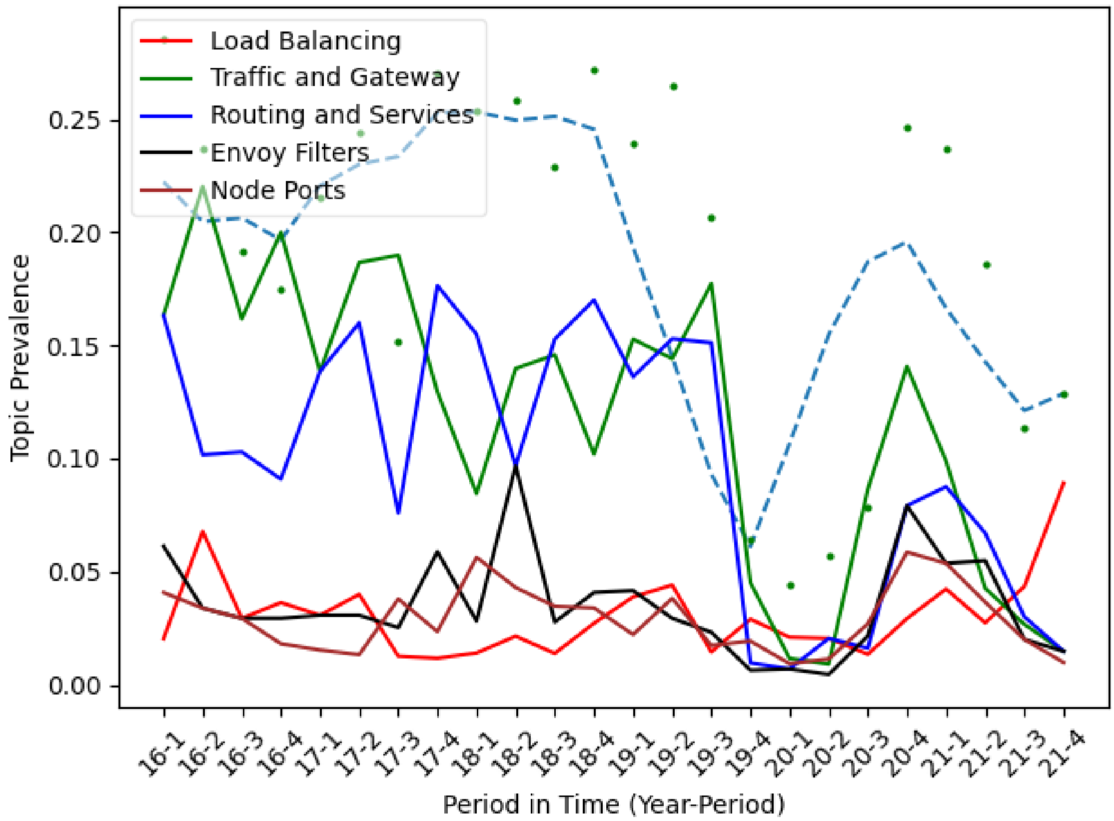

Figure 3.3: Evolution of the "Traffic"Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

Figure 3.4 reveals the trends within the security-related macro-topic. We observe a spiking outlier of policy-related questions in the second quarter of 2021 (21-2), which seems to coincide with the publication of the first security audit report of the Istio project by the NCC Group[17]. The report reveals several sensitive vulnerabilities regarding Istio and indicates the absence of security-related documentation. In response to the report, Istio issued multiple security patches and published official blogs on enhancing concerns about safety in a service mesh architecture, which could explain the sudden focus of discussions over security policies in this quarter.

[17]https://istio.io/latest/blog/2021/ncc-security-assessment/

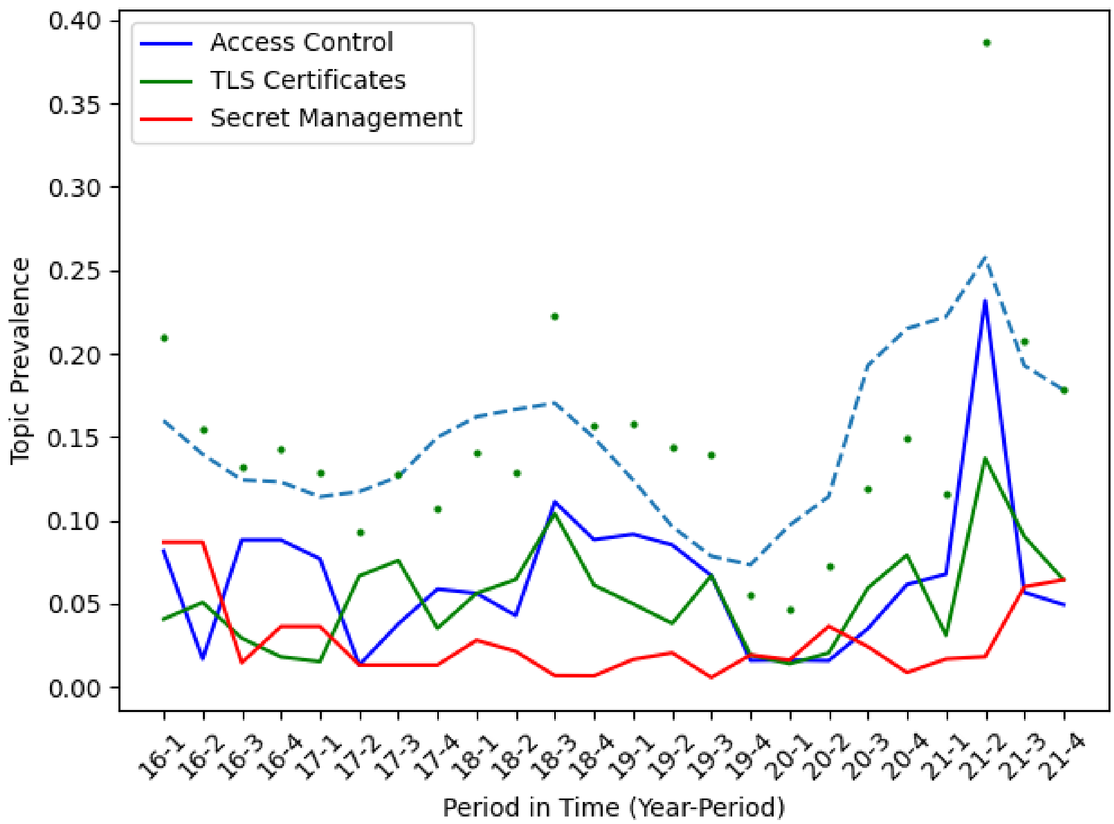

Figure 3.4: Evolution of the "Security" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

When considering the trends beyond the outlier, we observe that the security macro-topic started at 10% prevalence overall and has considerably increased over time to surpass the prevalence of the traffic macro-topic. As a result, we can identify security-related concerns as a rising trend that attracts a significant percentage of questions from service mesh practitioners. One previous work (47) points out that the security-related capabilities of service mesh enable a safer network without introducing additional complexities. Yet, another previous work (52) points out that security features in service mesh are fundamentally vulnerable to attacks. Based on the rising trend of security concerns and perceived importance in adoption, we suggest framework maintainers and software engineering researchers pay more attention to service mesh's security concerns to enable practitioners to secure their applications effectively.

Figure 3.5 reveals the growing trends within the observability-related macro-topic. We want to point out that, as described in Section 3.2.3, two of the plotted topics belong to mixed macro-topics, which consists of the observability of service mesh infrastructure (Infrastructure monitoring) and the user applications (Requests monitoring). Overall, we observe a steady and slightly increasing moving average on topic prevalence, matching the growing technical hype on observability solutions[18]. To our knowledge, service mesh-based observability drastically differs from traditional microservices that rely on SDKs and automatic instrumentation agents (69). Since the service mesh sidecar proxies can only intercept network-level traffic, it is difficult for developers to gain insight into code-level problems at runtime without compromising the non-intrusive nature of sidecars. The difficulty of utilizing such tools in service mesh leads to the requirement for truly non-intrusive ways to monitor system performance, which could be implemented with Extended Berkeley Packet Filter (eBPF), a novel technology to intercept and monitor Linux kernel operations (79).

Figure 3.6 reveals the trends within the aggregated infrastructure-related macro-topic. At the early emergence of service mesh frameworks in 2017, practitioners served microservices from hybrid network infrastructures, including virtual machines (VMs), Docker, and Kubernetes (25). Practitioner preferences gradually shifted focus to Kubernetes-like clusters in today's large-scale systems. We observe a gradual decline in the prevalence of infrastructure-related questions until 2018, when service mesh frameworks with underlying containerization platforms rapidly evolved at the earliest adoption

[18] https://blogs.gartner.com/andrew-lerner/2021/10/11/networking-hype-cycle-2021/

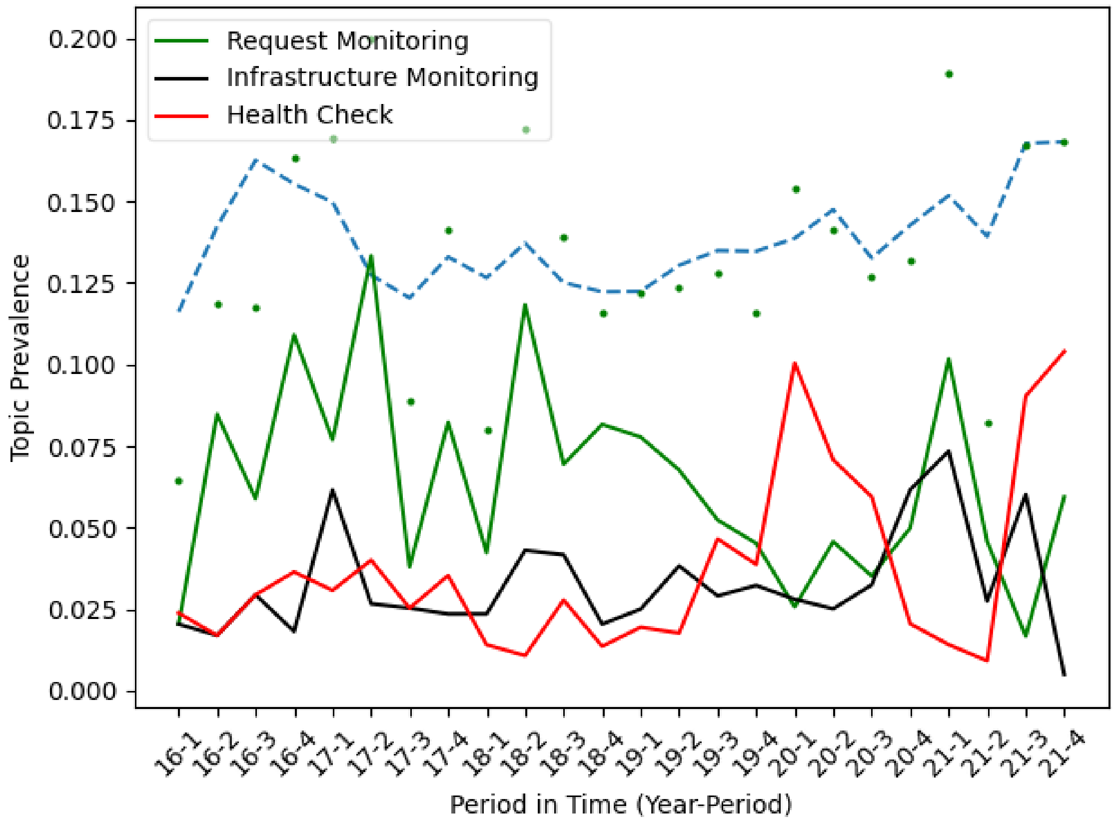

Figure 3.5: Evolution of the "Observability" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

stage. Surprisingly, with the rapid growth of common service mesh frameworks, the prevalence of infrastructure-related questions rebounded and increased again. The model output shows outliers during 2019 and 2020 as the two topics have swapped prevalence. Despite the outlier period, we observe that the overall prevalence of infrastructure-related questions has increased compared to 2017. The high prevalence of infrastructure-related topics persists even though dominant mesh frameworks release new versions monthly, as shown in Table 3.1. Next, we compare this trend to the corresponding trend for infrastructure-related questions of traditional service mesh frameworks.

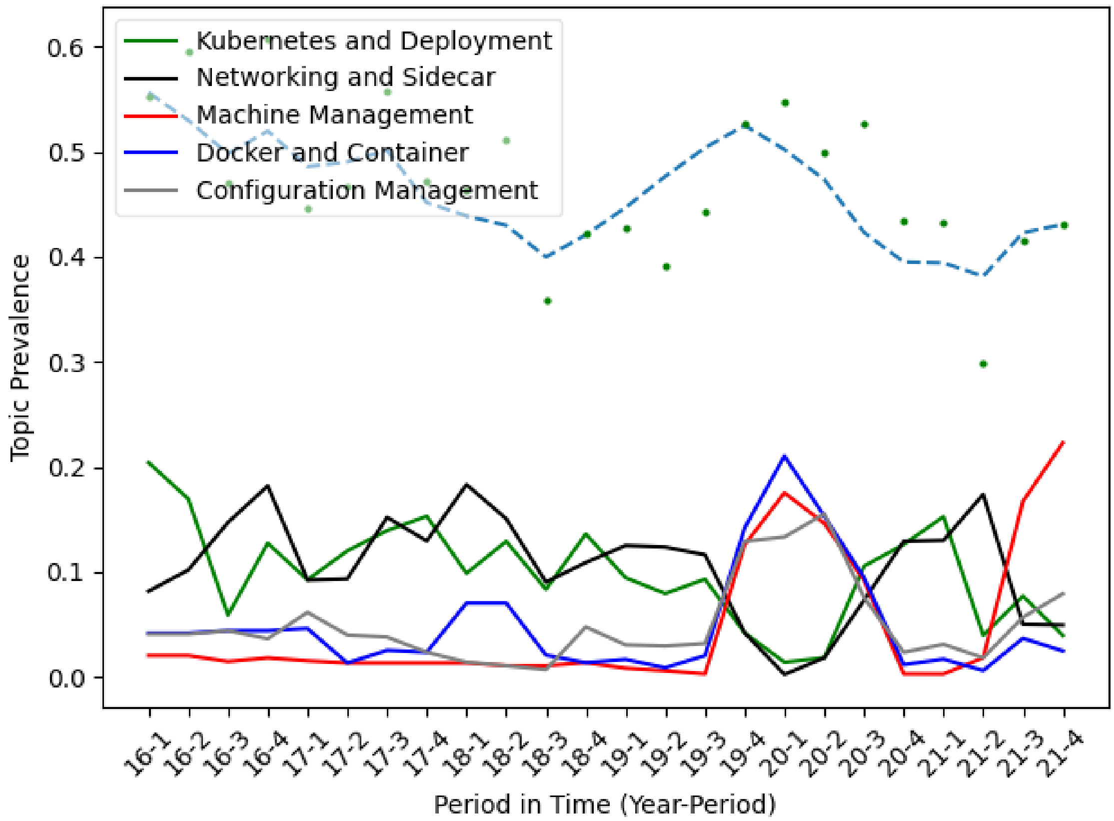

Figure 3.6: Evolution of the "Infrastructure" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

Figure 3.7 depicts the trend of application-related topics in the service mesh domain. We observe a significant drop of "Spring and Service Discovery" topic prevalence from 20% to less than 5%. We cross-reference this finding with RQ1's analysis of the corresponding service discovery topic in traditional microservices, which represents a minimal prevalence (5.9%). As a result, we confirm that such topics in both domains, after years of evolution, now cause minimal practitioner concerns.

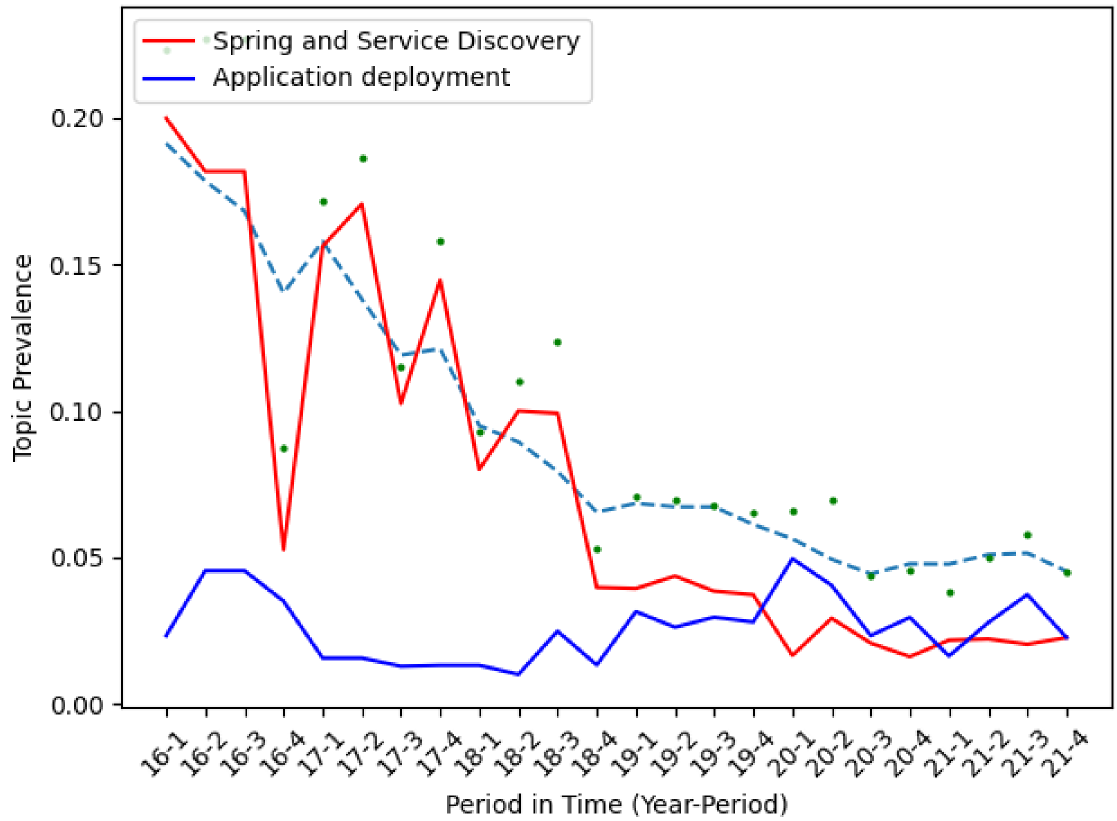

Figure 3.7: Evolution of the "Application" Macro-Topic for Service Mesh: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

3.3.2 Comparison to the Traditional Microservices Infrastructure Domain

The evolution of the infrastructure-related topics in the traditional microservices domain persists, while the service mesh domain sees a minor increase in prevalence over time.

Using a similar time-related analysis as for the service mesh data, and focusing exclusively on the infrastructure-related macro-topic, Figure 3.8 presents the evolution

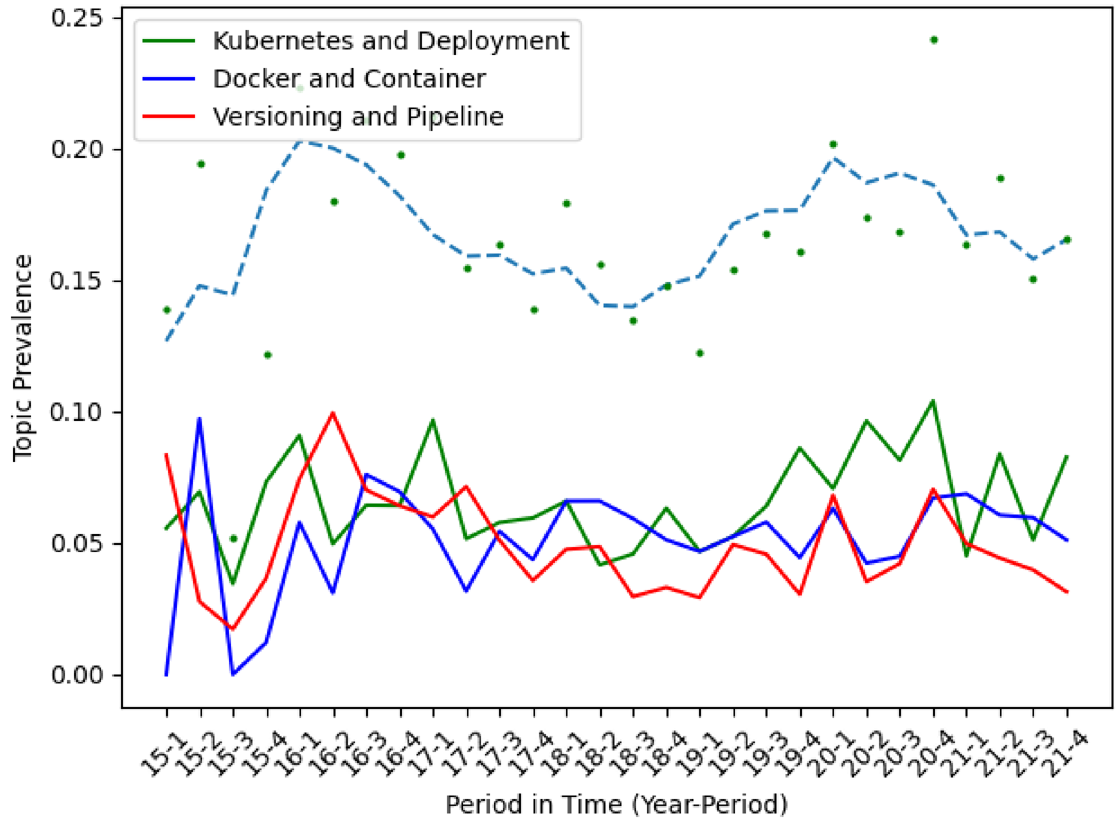

Figure 3.8: Evolution of the "Infrastructure" Macro-Topic for Traditional Microservices: Aggregated Moving Average (Blue Dashed Line over Green Dots) and Individual Topics (Other Lines)

of the prevalence of aggregated infrastructure topics for traditional microservices. To our surprise, in the relatively mature domain of traditional microservices, where well-known frameworks like Spring Cloud have been dominating for years, infrastructure-related concerns have not reduced in prevalence. Hence, infrastructure-related concerns remain relevant over time for the overall microservices landscape, regardless of the adoption of newer mesh architecture or the evolution of frameworks.

In our topic analysis, keywords including "install," "configuration" and "compose" frequently appear at the top along with those related to containerization such as "kubernete" and "container" as presented in Table 3.5. The top keywords suggest most of the infrastructure-related topics relate closely to the usability issues of the installation, update and configuration process in a containerized scenario. We define usability as the easiness of understanding the system setup and correctly configuring the system to work according to user requirements (15). Given our insights on the evolution of containerization-related topics across time, the persisting infrastructure-related concerns in both domains indicate a pressing need for more empirical understanding to improve the usability of such containerization platforms and ease the adoption of service mesh. To our knowledge, empirical studies on enhancing the usability of modern container orchestration platforms are scarce. An early study on the infrastructure-as-code ecosystem puppet (36) has proposed methods of identifying bad configuration smells. Future research could adopt such methods to modern container orchestration platforms such as Kubernetes, which have quickly evolved into the de facto standard (44).

Feedback from service mesh expert The service mesh expert adds a new perspective on the security trend in the service mesh domain. While supporting our finding that security is an area of increasing importance, the expert indicates the analysis in this research question is likely only the "tip of the iceberg", as many security-related inquiries are likely being conducted through enterprise support channels and were not discussed in the public domain, even if the practitioners use open-source frameworks. When talking about the specific area to focus on, the expert notes that achieving a zero-trust architecture as explained in RQ1 has become a common goal among service mesh vendors in recent years, and they believe that this direction should draw more research attention as it is challenging to implement and a successful adoption will bring the most out of the service mesh paradigm.

When asked about more insights on the observability domain of service mesh, the expert notes that "*Nearly every cloud vendor has its monitoring/logging product that behaves differently, and some like OpenShift have more strict security policies that can further complicate the adoption and usage of such monitoring technology.*"

> **Summary of RQ2:** Traffic-related topic prevalence fades as service mesh core functionalities mature over time. However, persisting topics in infrastructure indicate remaining concerns that require extra attention. The rise of security and observability concerns indicates future opportunities for novel ideas. Infrastructure-related topics are prevalent in both service mesh and traditional microservices domains.

3.4 Question Intentions, Adoption Goals, Symptoms and Their Fixes (RQ3 Results)

3.4.1 Understanding Knowledge Intentions of Istio and Consul from Knowledge-transfer Questions

The intention distribution of the 62 studied knowledge-transfer questions effectively reflects a gap between the carried-over experience from traditional microservices and corresponding service mesh features.

Figure 3.9 shows the distribution of question intentions among the 62 knowledge-transfer question posts manually labelled based on the question title and body context. Our analysis identified four types of intentions of knowledge-transfer questions:

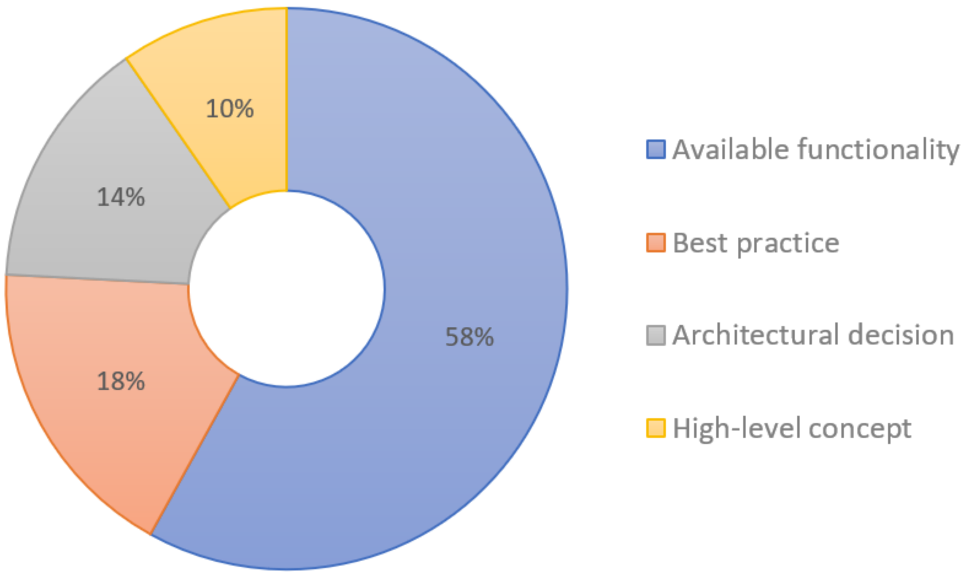

Figure 3.9: Question Intentions of Knowledge-transfer questions

- **Available functionality**: We identified 36 questions in this category. These questions are intended to understand the features and capabilities of the service mesh frameworks, such as traffic management, security, and telemetry.

- **Best practice**: We identified 11 questions in this category. These questions aim at learning the recommended practices and configurations for using the frameworks effectively and correctly.

- **Architectural decision**: We identified 9 questions in this category. These questions involve the architecture of a service mesh network, such as which components to use and how to structure the mesh to optimize for certain use cases.

- **High-level concept**: We identified 6 questions in this category. Such questions seek to understand the fundamental concepts and principles within the service mesh domain.

We find a dominating 58% occurrence of questions trying to understand the availability of functionalities. In RQ1, we find a low number of knowledge-transfer questions in the service mesh domain in contrast to traditional microservices. During the interview, the service expert indicates that service mesh practitioners benefit from carried-over knowledge from the traditional microservices domain. Nevertheless, the high ratio of inquiries on available functionalities indicates that carried-over knowledge does not necessarily imply a good understanding of service mesh features.

On the other hand, questions regarding best practices and architectural decisions often require a high level of expertise to resolve. Although less dominant than the inquiry for functionality, together they form over 32% of the distribution together. Only 10% of questions ask a high-level concept, confirming our RQ1 findings. In the end, service mesh frameworks own many flexible components, and they can be assembled in many ways in various use cases. Our findings suggest a need for automating the suggestion of the best combination of such components to inform practitioners of available features and their best practices.

3.4.2 Understanding Adoption Goals of Istio and Consul from How-to Questions

352 studied how-to questions embed 66 unique adoption goals, focusing on integration, customization, and installation. Specifically, Istio practitioners face significant installation and upgrading concerns.

Figures 3.10 and 3.11 show the adoption goals for the 197 and 155 analyzed how-to question posts from the open-source Istio and Consul frameworks. Since each service mesh framework involves many adoption goals, here we selectively highlight the frequently co-occurring goals in each macro topic and frequent framework-specific goals. We present the key findings of each macro-topic below:

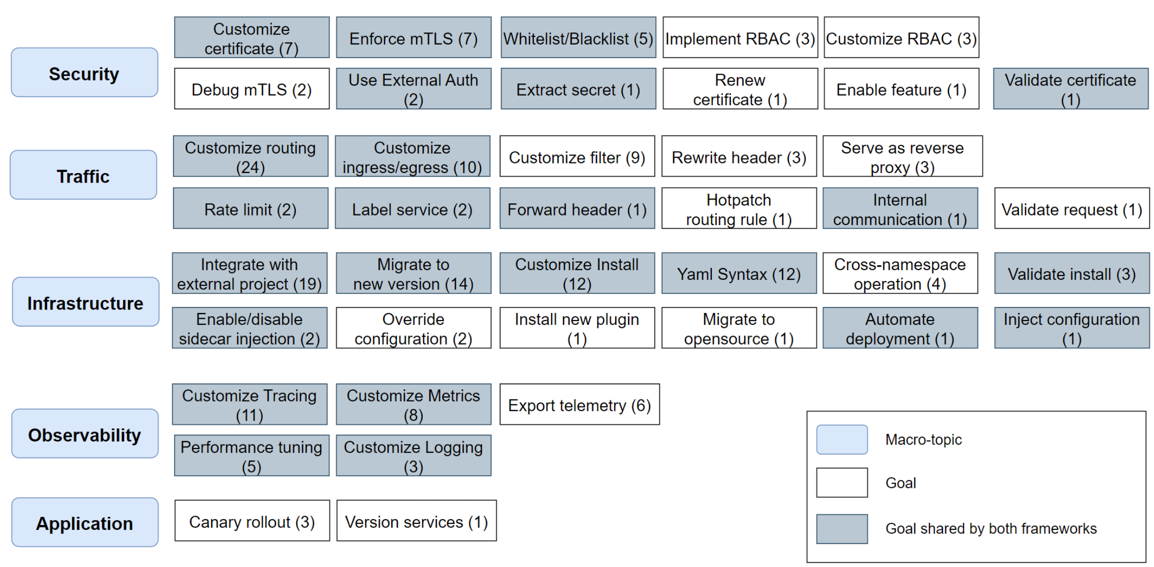

Figure 3.10: Adoption Goals of the 197 Studied Istio How-to Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

- Security: The ACL (Access Control List) system is unique to the Consul service mesh; Istio implements the RBAC (Role-based Access Control) system instead.

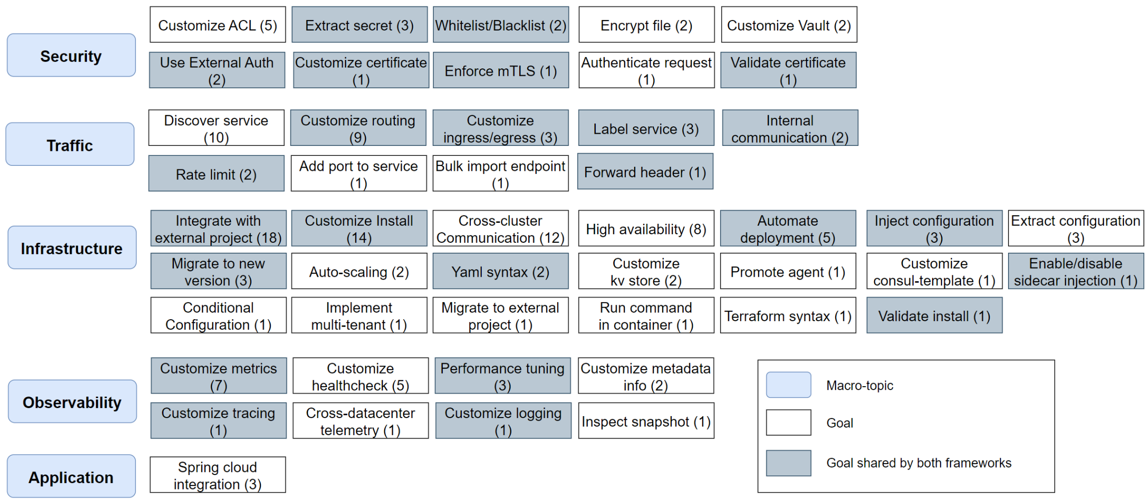

Figure 3.11: Adoption Goals of the 155 Studied Consul How-to Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

While both systems work towards securing service-to-service communication, we observe a slightly higher (4%) ratio of security-related how-to questions in Istio. When cross-referencing our findings with RQ1, we find that although the prevalence of security-related topics is considerably lower than traffic-related topics, both questions present an equal number of adoption goals.

- Traffic: Interestingly, the traffic topic's adoption goals are highly skewed in Istio, with 24 questions on customized routing based on specific use cases, indicating that practitioner goals cannot be easily achieved with the default settings and configurations provided by Istio. On the other hand, Consul provides out-of-the-box support of "service discovery" features, while Istio relies on external integration to accomplish the goal. As a result, Consul users have the most concerns with this functionality (10 cases).

- Infrastructure (Integration): In both Istio and Consul (19/28 cases), we observe open inquiries regarding the ability to integrate service mesh frameworks with third-party projects such as cloud object storage and legacy applications.

- Infrastructure (Installation): Apart from integration, practitioners of both frameworks raise frequent (12/14 cases) concerns about customization of framework installation spanning various installation methods. Interestingly, only Istio practitioners raise significant number of migration inquiries (14 cases), indicating confusion while attempting to upgrade the Istio version seamlessly. On the side of Consul, we additionally observe significantly more diverse adoption goals (20 of 48 cases) than for Istio (12 out of 41 cases), even though the Consul question samples are smaller, which could indicate more customization needs in the Consul domain.

- Infrastructure (Scaling): The high number of questions related to scaling, such as cross-cluster communication and high availability in Consul (20 cases), suggests that it could be a challenging aspect of using Consul. Surprisingly, cross-cluster questions do not represent a visible distribution in the sampled Istio questions, even though rich multi-cluster functionalities are available according to Istio's documentation.

- Observability: Both Istio and Consul practitioners express customization goals regarding tracing, logging and metrics data. By observing the question content, we find that practitioners generally ask for various telemetry customization that reflects their requirements. In several cases, the users are concerned about the performance impact of collecting telemetry from service mesh, which requires advanced sampling techniques to limit the data size.

3.4.3 Understanding Error Symptoms of Istio and Consul from Error-related Questions

Construction of symptom trees from the 474 studied error-related posts for Istio and Consul service mesh yields 54 unique symptoms, among them 39 affect both frameworks. On the other hand, both frameworks show frequent unique symptoms that rarely occur in their counterparts.

Figures 3.12 and 3.13 show the error symptom trees for the 264 and 210 error-related posts from the open-source Istio and Consul framework. The trees contain error symptoms from the posts that are either explicitly stated as an error or considered misbehaviour of the framework. The left nodes in blue represent the macro-topics whose questions were manually labelled given the actual context of each question and domain knowledge. Each child node represents a generalized symptom directly observed from the question body. Within each symptom tree, we underline symptoms that reappear across multiple macro-topics. To enable a clear comparison between the two service mesh frameworks, we mark symptoms shared by both frameworks in grey, along with the frequency of each symptom.

Within the Istio symptom tree, we generalize 41 error symptoms distributed among four macro-topics of concerns. Similarly, we generalize 43 error symptoms in the Consul case across five macro-topics. Table 3.8 defines each error symptom. We cross-reference the number of symptoms and their frequencies with RQ1 data to understand whether the most frequent macro-topics found in RQ1 have been significantly discussed in terms of symptoms/fix patterns. In particular, security-related how-to questions account for a lower prevalence in the sampled data (16.7% and 12.9% for Istio and Consul, respectively) and are not comparable with the traffic and infrastructure

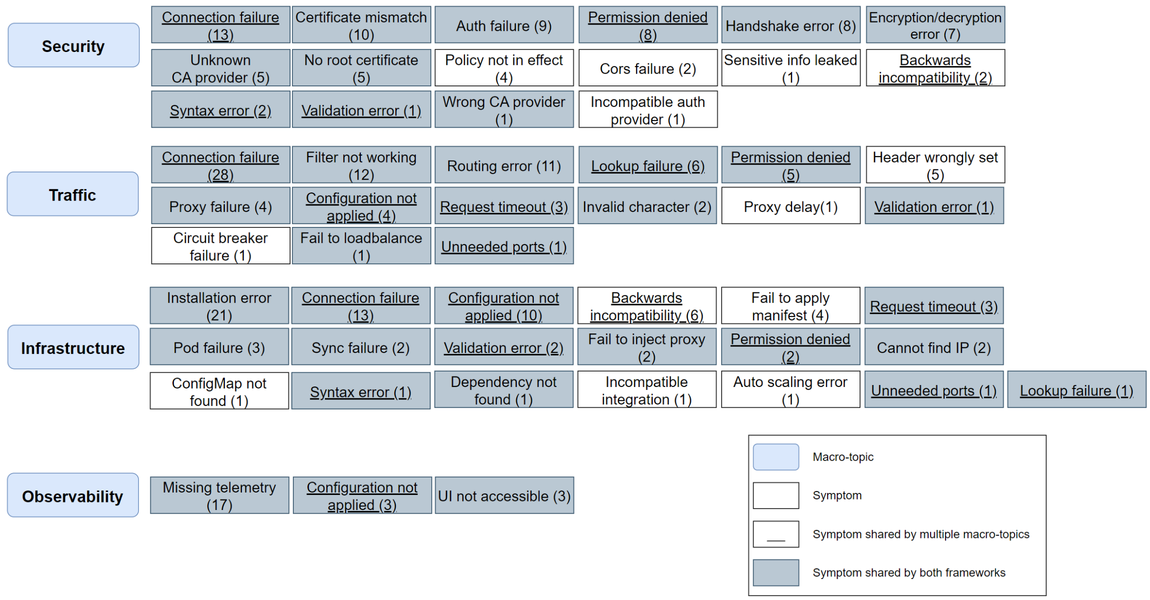

Figure 3.12: Error Symptom Tree of the 264 Studied Istio Error-related Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

macro-topic. Yet, security-related error symptoms take significant prevalence (29.9% and 24.3%) in the error symptom analysis. Such imbalance indicates security-related concerns appear more in the form of failures and unexpected behaviour, which is worth further investigation by security researchers.

Specifically, we observe the most appearances of "connection failure", "permission denied," and "Configuration not applied" symptoms. "Connection failure" symptoms account for 54 and 34 cases in Istio and Consul, respectively, meaning the only direct symptom that service mesh practitioners receive in the error log points to a failed network request. "Permission-denied" symptoms occur in 15 and 16 cases in Istio and Consul, respectively, indicating a frequent error on failed attempts to access service mesh network components due to lack of access. Finally, the "Configuration not applied" symptom occurs in 17 cases for both frameworks and dominantly in the infrastructure macro-topic.

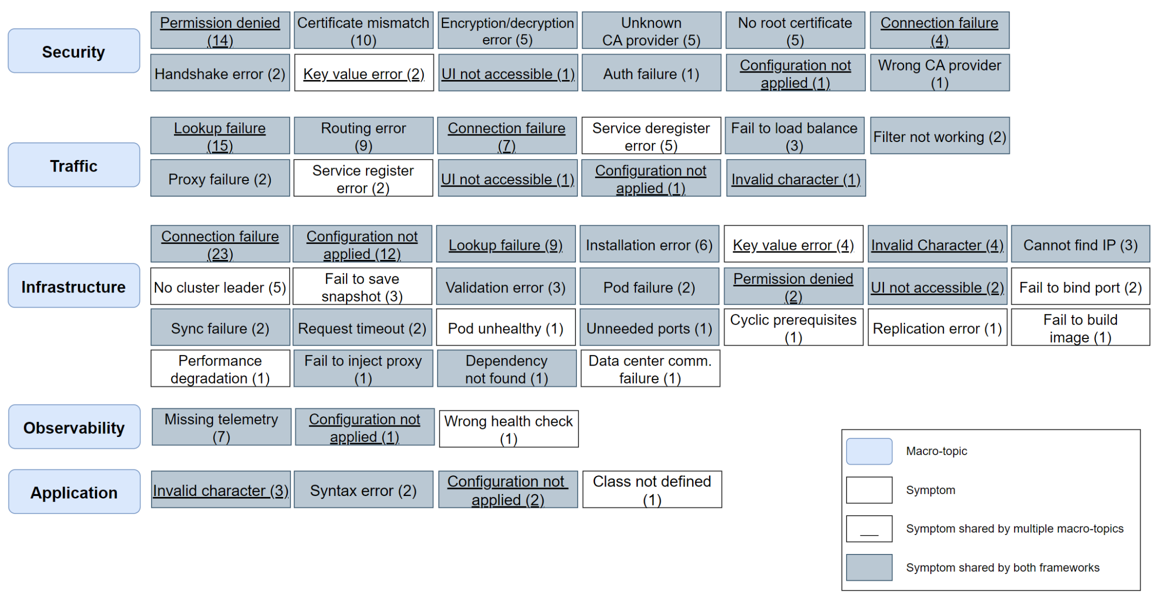

Figure 3.13: Error Symptom Tree of the 210 Studied Consul Error-related Questions (Number of Occurrences Between Parentheses), Across the 5 Macro-Topics of RQ1

Since both frameworks are considered feature-rich and cover common use cases in the microservices domain, the two service mesh frameworks share 39 common symptoms among their union of 54 symptoms across five macro-topics. Most interestingly, the error symptoms also exhibit duplication within different macro-topics of each framework. For instance, the underlined "connection failure" symptoms occur repetitively across three macro-topics in both frameworks. Based on our understanding of the question answers and service mesh frameworks, an erroneous configuration on either traffic routing or underlying deployment infrastructure could cause the same apparent symptom of connection failure. Installation errors, container-related errors and connection failures occur regardless of the framework, supporting our insights from RQ2 on framework-agnostic infrastructure-related concerns. Previous work on Kubernetes (113) points out that applying default configurations in such distributed systems does not automatically grant error-free environments. Unfortunately, finding suitable configurations in a highly configurable system like service mesh is a non-trivial task (123).

We also observe drastically different dominant error symptoms over macro-topics; we present the key findings of each macro-topic below:

- **Security:** With both frameworks facing diverse error symptoms, Istio practitioners face unique errors that could negatively impact adoption. Although rare (1 case), Istio practitioners report a potential leak of sensitive information by sidecar proxy when intercepting request headers on a default configuration. Such a problem could indicate a potential security flaw. According to the accepted answer, users cannot easily fix the problem by adjusting any configuration, and it is still pending fixing after one year.

  Additionally, we notice 8 cases where Istio practitioners report backwards-incompatible behaviour when upgrading Istio to a newer version. Specifically, 2 cases break the security-related functionalities, while 6 cases break the proper function of the underlying infrastructure. We do not observe similar error symptoms in the sampled Consul framework posts.

- **Traffic:** Practitioners using the Istio framework dominantly face connection failure symptoms (28), yet Consul practitioners face more failures that relate to service lookup (15), registration (2) and deregistration (5). Istio does not provide comprehensive service discovery and registration features as Consul does. It is possible that such a difference is introduced due to the feature difference between the two frameworks. However, the connection failures representing 32% of Istio traffic-related error symptoms indicate a dominant concern that prevents normal service-to-service communication.

- **Infrastructure (Cross-Cluster):** Practitioners using both frameworks face the most diverse symptoms regarding infrastructure among all five macro-topics. With fewer samples (210 vs 264), Consul practitioners face 25 different infrastructure-related symptoms, which accounts for 58% of all the symptoms in Consul. Apart from connection failures originating from various root causes, we observe unique symptoms such as "No cluster leader", "Replication error, " and "Data center communication failure" related to high availability and cross-cluster communication. When cross-referencing the unique set of cross-cluster adoption goals in Consul, we observe such topics attract major concerns since they cannot be achieved without errors.

- **Infrastructure (Installation):** Surprisingly, while infrastructure-related symptoms are less diverse in Istio, practitioners mostly face installation errors (21 cases) that do not generally occur in the Consul framework (6 cases). Apart from the backwards incompatibility symptoms, Istio practitioners are likely to face errors even in new installations. For example, a user faces an error when installing Istio in WSL2 (Windows Subsystem for Linux):

  [P27] *I managed to create a cluster successfully. Then I try to install Istio using helm following the documentation. Everything looks fine till I check the status of Istio pods using for which I get the response [status] The pods continue to stay in ContainerCreating status...*

- **Observability:** While practitioners from both Istio and Consul face errors related to missing telemetry, it is significantly more frequent in the Istio framework (17 cases). Given that Istio provides out-of-the-box distributed tracing for applications while Consul relies on third-party integration, Istio practitioners may face more errors in return for native support for observability.

3.4.4 Constructing Fix Patterns for Error Symptoms

Less than 50% of the error symptoms have a common fix pattern, and we observe many-to-many error-fix relationships created by unintuitive error messages, causing extra difficulties in root cause understanding.

Table 3.8 defines each symptom from the above symptom trees within their specific contexts. In the final column of the table, we provide the identifier of a common fix pattern if a given method could resolve over 50% of the questions of the same symptom. Previous work (73) has used 50% as the threshold to define a common fix pattern. We consider a fix as a common fix pattern only if the fixed error symptom appears more than two times and it could be resolved by a particular fix pattern in over 50% of the cases. In other words, only a fix occurring more than once is considered a "common fix pattern".

We briefly explain each fix pattern based on service mesh domain knowledge:

- **Recreate CA**: Although the root cause of an invalid Certificate Authority (CA) can vary, this fix pattern recreates/reloads the certificate authority to ensure it can be properly recognized by both client and server side. This fix pattern can effectively resolve unknown and wrong certificate authority symptoms.

- **Correct certificate signing**: Correcting certificate signing to ensure service mesh components can recognize it. Such fixes can often resolve certificate mismatch errors since the symptom directly hints at the root cause.

Table 3.7: Symptoms and Their Fix Patterns (Part 1 - Security and Traffic)

Macro-topic	**Symptom**	**Definition**	**Ref**
Security	Permission denied	Permission is not granted for access	S04
	Certificate mismatch	The certificates cannot be used	S08
	Encryption/Decryption error	Failure in end-to-end encryption	
	Unknown CA provider	The certificate authority (CA) is unknown	S01
	No root certificate	No root certificate can be found	
	Connection failure	Access should be granted but denied	
	Handshake error	SSL failure with handshake error	
	Key value error	Failure in key-value secret store	
	UI not accessible	UI related to security not available	
	Auth failure	Authentication failed to work	S03
	Wrong CA provider	The framework indicates wrong CA is used	S01
	Policy not in effect	Security policy not enforced	
	Cors failure	Blocked by cross-origin request policy	
	Sensitive info leaked	Request header leaks sensitive info	
	Syntax error	Configuration syntax not recognized	S07
	Validation error	Fail to validate configuration manifest	S07
	Backwards incompatibility	Security components fail after upgrading	
	Incompatible auth provider	External authentication not working	
Traffic	Connection failure	Connection rejected in traffic routing	
	Filter not working	Envoy filter not working as expected	
	Routing error	Routing is not working at all	
	Lookup failure	Service lookup failure	S05
	Permission denied	Cannot access service due to permission	S04
	Header wrongly set	Wrong manipulation of request header	
	Proxy failure	Proxy container failed	
	Configuration not applied	Config regarding traffic not in effect	
	Request timeout	Request timed out reaching limit	
	Invalid character	Invalid character in RPC communication	
	Proxy delay	Proxy experiencing delays	
	Validation error	Fail to validate configuration manifest	S07
	Circuit breaker failure	Fail to break traffic when triggered	
	Fail to load balance	Load balancer is not working as expected	
	Unneeded ports	Service mesh exposing unnecessary ports	
	Service register error	Service cannot properly register	
	Service deregister error	Service cannot properly deregister	

Legend:
S01: Recreate CA
S02: Correct monitoring configuration
S03: Complete missing policy rules/scopes
S04: Check permission policies
S05: Check service routing rules
S06: Use proper commands and scripts
S07: Fix configuration syntax
S08: Correct certificate signing
S09: Create ConfigMap

Table 3.8: Symptoms and Their Fix Patterns (Part 2 - Infrastructure, Observability and Application)

Macro-topic	**Symptom**	**Definition**	**Ref**
Infra.	Connection failure	Connection rejected regarding infrastructure	
	Configuration not applied	Configuration provided but not in effect	
	Lookup failure	DNS lookup failure	
	Installation error	Service mesh component installation failure	S06
	Key value error	Failure in key-value secret store	
	Invalid character	Invalid character in RPC communication	
	Cannot find IP	Cannot find Ip address of service	
	No cluster leader	Cannot reach cluster leader	
	Fail to save snapshot	Cannot backup snapshot	
	Validation error	Fail to validate configuration manifest	S07
	Pod failure	Kubernetes Pod failure	
	Permission denied	Cannot access service due to permission	S04
	UI not accessible	UI components not accessible	
	Fail to bind port	Cannot bind to port in system	
	Sync failure	Fail to sync status between nodes	
	Request timeout	HTTP request timeout	
	Pod unhealthy	Container not in healthy status	
	Unneeded ports	Service mesh exposing unnecessary ports	
	Cyclic prerequisites	Cyclic requirements between data centres	
	Replication error	Cross data center replication error	
	Fail to build image	Fail to build container images of component	
	Performance degradation	Component become slower overtime	
	Fail to inject proxy	Fail to inject proxy into the pod	
	Dependency not found	Framework building dependency not found	
	Datacenter comm. failure	Data centre cannot reach each other	
	Backwards incompatibility	Infra components fail after upgrading	
	Fail to apply manifest	Fail to apply configuration manifest	S06
	ConfigMap not found	ConfigMap cannot be found or reached	S09
	Incompatible integration	Integration with third party not working	
	Autoscaling error	Failure in auto-scaling components	
Obs.	Syntax error	Configuration syntax not recognized	S07
	Missing Telemetry	Telemetry is not properly reported	S02
	Configuration not applied	Configuration provided but not in effect	
	Wrong health check	Health check return unexpected result	
App.	UI not accessible	UI related to telemetry not available	
	Invalid character	Invalid character in RPC communication	
	Syntax error	Configuration syntax not recognized	S07
	Configuration not applied	Configuration provided but not in effect	
	Class not defied	Cannot find Java class definition	

Legend:
S01: Recreate CA
S02: Correct monitoring configuration
S03: Complete missing policy rules/scopes
S04: Check permission policies
S05: Check service routing rules
S06: Use proper commands and scripts
S07: Fix configuration syntax
S08: Correct certificate signing
S09: Create ConfigMap

- **Check permission policies**: Verifying the access permissions for resources to ensure permissions are not too restrictive. Permission-denied symptoms often hint at restrictive policies.

- **Complete missing policy rules/scopes**: Missing policy rules can impact security measures in the service mesh network, allowing unauthorized access. Adding missing policy rules or flags ensures that security policies are enforced as intended.

- **Check service routing rules**: Correct service discovery/routing to make sure lookup can proceed to resolve service. A wrong destination/missing route often leads to various error symptoms, including an apparent connection failure.

- **Use proper commands and scripts**: When practitioners face unexpected results while installing/upgrading service mesh components, correcting the commands/flags in the command line interface (CLI) can usually avoid unexpected results.

- **Fix configuration syntax**: Although configuration validation tools exist for Istio, they cannot verify all scenarios, given their rule-based nature, leading to runtime failures. The fix includes but is not limited to fixing YAML indentation and specific malformed fields. Fixing configuration syntax can avoid validation errors and syntax errors.

- **Correct monitoring configuration**: This error symptom is caused by using default telemetry settings that report incorrect telemetry data and by wrong interpretation of monitoring settings. Fixing the monitoring configuration can often ensure that correct telemetry is reported.

- **Create ConfigMap**: Although this symptom only appears once in Consul, we find this error obvious error symptom exists in service mesh and the overall Kubernetes domain. This problem can always be fixed by recreating the ConfigMap object in the correct place. However, the root cause can vary due to multiple reasons, such as the namespace causing the ConfigMap to be created in the wrong context.

As a result, we could only derive 9 common fix patterns for 12 out of 54 unique symptoms, indicating that most error symptoms, although resolved, do not share a clear connection with their root causes. Additionally, several question authors directly point out their frustrations that framework error messages or current documentation are hard to interpret and do not contribute to direct error resolution. The following quote demonstrates a typical case of user confusion while encountering multiple layers of errors during Consul installation.

> [P28] *I can get clients started that don't have docker installed, however on the docker nodes I am unable to get them to start. The first error I got was there are multiple private networks… blah blah… So I edited the config to use the interface name as suggested in discussions. Now I get this error.*

Despite the difficulties in deriving common fix patterns, only a few fix patterns we identified form a one-to-one relationship with an identified error symptom. We observe 29 cases where multiple fix patterns can fix a single error symptom. That is, if the error symptom occurs more than once in our sample, practitioners have provided different ways to fix them. We also find 15 cases where a fix pattern applies to two error symptoms.

These two observations combined form a many-to-many relationship. For example, in the security symptoms of Istio, both "permission denied" and "policy not in effect" can be fixed by "correcting the port mapping naming convention." We can also find symptoms such as "missing telemetry," which can be solved by two fixes. The most extreme example is the "connection failure" symptom that occurs in multiple places in both symptom trees, which can be fixed in more than 17 ways by observing our limited samples.

Feedback from service mesh expert When asked about our finding on error symptoms, the service mesh expert noted, "*This is really an interesting finding. Service mesh deployments employ the "eventual consistency" paradigm, failing deployments may or may not recover after restarted, depending on the error types, and errors in service mesh frameworks look like Domino Blocks, one error in an upstream node can cause another in a downstream, if the root cause of the errors is not found and fixed, users have to look all errors for possible answers, and if the root cause is found and fixed, all other types of errors disappeared too, this is one of the reasons why the error-fix relationship is not one-to-one exactly.*" With such observations, the service mesh expert emphasized the importance of enhancing error diagnosis mechanisms to help practitioners diagnose and fix complex problems more effectively.

3.4.5 Analyzing Configuration Changes to Fix Error Symptoms

Finally, we find that in 70% of the cases, minimal changes to the configuration manifest of a specific service mesh functionality could fix the error symptoms.

Service mesh configuration files are written in the YAML format (13), a compact language commonly serving as the orchestration language of modern containerization platforms. Such configuration manifests provide a high-level abstraction of complex operations hidden by service mesh frameworks. Therefore, changing a single line could introduce an impact throughout the service mesh system. We observe that most error symptom fixes (70%) require only minimal changes of fields or lines in the configuration files. In some cases, modifying one configuration line was able to fix a complex error symptom with over 100 lines of error messages. For example, changing a simple port name to "http" of a user's service definition could fix an error symptom of a defective security rule for a question that took 24 follow-up discussions over 13 months before obtaining a viable fix[19].

Previous work (97) has systematically identified the main challenges and activities regarding configuration engineering in practice and provided practical recommendations to address such challenges. However, in the context of service mesh and similar modern containerization technologies, a practitioner could orchestrate and scale hundreds of heterogeneous services to form a large-scale system through simple configurations. Although service mesh and its underlying container orchestration platform have exposed configuration through simple APIs, it also allows the impact of simple configuration changes to propagate beyond the traditional software/hardware environments towards heterogeneous environments across clusters, data centers, and even clouds.

Previous work (53) suggests using crowd knowledge to build adaptive configuration suggestions for users of complex software systems. However, in the case of the service mesh domain, users seek the flexibility to customize and extend system capacity to fulfill higher non-functional and functional requirements in today's large-scale

[19]https://discuss.istio.io/t/jwt-policy-does-not-take-affect/141/25

systems. Based on our findings in Section 3.2.1, a crowd-based solution can be challenging to adopt since expertise is likely scarce in the domain. Furthermore, minor modifications in a YAML manifest could lead to significant architectural differences in container topology and software states at runtime[20], producing new challenges beyond applications with predetermined topology. We believe a further understanding of such compact configuration manifests and the adoption of novel ways to recommend suitable configurations that follow best principles (101) (perhaps inspired by those proposed for monolithic systems (129; 130)) will be essential for today's popular container orchestration platforms.

Feedback from service mesh expert During the interview, the service mesh expert notes that "*Although many cloud-native engineers jokingly call themselves "YAML engineers", YAML is just a markup language and not a programming language after all. It can define the configuration structures and data types, but it doesn't verify the correctness by itself, leaving the verification to runtime components.*" When asked about the difficulty of coming up with simple configuration changes given the observed symptoms, the service mesh expert adds, "*Sometimes it's even hard to find the corresponding runtime component that consumes a YAML by looking at the symptom and its logs, not to say fixing the error symptoms.*" When talking about the diversity of adoption goals mined, the expert emphasized the need for advanced automation that generates configuration/fixes based on customized scenarios since practitioners of service mesh frameworks tend to aim at a wide range of customization goals. Any valid reference configuration could significantly reduce the risk of errors.

[20] https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/

Summary of RQ3: Diverse intentions, goals and symptoms exist in the service mesh domain, exposing concerns of service mesh practitioners around specific topics. We find that service mesh error symptoms do not often have common fixes, since error messages generally do not relate one-to-one to root causes. In many cases, the same error symptom could occur across service mesh macro-topics and receive multiple possible fixes. We also find that container orchestration platforms and configuration engineering play a significant role in error fixing.

3.5 Study Implications

3.5.1 For Researchers and Framework Designers

Researchers should investigate service mesh security feature impacts and their adoption complexities: In RQ1 and RQ2, we find that observability and security topics in the service mesh domain attract increasingly more concerns. During the interview with the service mesh expert, we also collected insight from the industry that security concerns could be greatly underestimated due to their nature. The expert adds that zero-trust architectures are prevalent in the service mesh domain amid increasing cyberattacks. Previous work (23) has suggested the superiority of zero-trust architectures. However, the authors suggest that strict security features introduce a non-negligible impact on their users that needs to be explored in academia. In our study, we indeed observe many inquiries regarding the customization of security features of service mesh frameworks. On the other hand, errors frequently occur when practitioners try to apply such features. Future researchers could conduct studies to evaluate the usability of service mesh security-related features, investigate potential usability issues, and understand the source of complexity. This could help identify potential barriers to service mesh adoption and inform the design of more user-friendly security features.

Researchers should investigate novel service mesh observability approaches: The expert also indicates that current service mesh observability features often vary when choosing different vendors and frameworks, leading to adoption complexity and concerns. Given current service mesh features in Istio and Consul, practitioners of both frameworks face observability-related concerns that target application (request monitoring) and service mesh framework themselves (infrastructure monitoring). Practitioners raise a high number of inquiries on customization of telemetry data and face error symptoms in missing telemetry, especially in the Istio framework.

When asked about the observability features, the service mesh expert adds that current service mesh solutions rely on different frameworks that enable tracing, logging, metrics and event telemetry. Since there is no standard to define a healthy service, practitioners are forced to customize the configuration to keep service impact minimal while enabling a complete view of their applications. It is valuable for future observability domain researchers to focus on accurate modelling of system conditions according to industry needs. In this way, practitioners could easily investigate best practices for adjusting observability features given clear requirements.

Researchers should find effective methodologies in configuring complex distributed systems like service mesh: As a highly configurable system that relies on flexible orchestration of software components, service mesh, like machine learning systems (101), comes with high maintenance costs if its configuration problems are not well understood. Service mesh systems rely on compact configuration manifests to define, orchestrate and manage many components. However, when cross-referencing our RQ1 and RQ3 results on error rates, symptoms and fixes, we find that such high-level abstraction of configurations currently leads to diverse error symptoms that inexperienced practitioners could not quickly fix. Given the service mesh expert feedback in RQ3, configuration validation is hard to achieve and often leads to implicit errors that require expertise currently lacking in the service mesh domain. Our study findings in RQ1 and RQ3 confirm a need for better methodologies and automation to help suggest practical configuration principles linked to practitioners' adoption goals. Future research should focus on understanding the unique challenges that complex distributed systems face in configuration to prevent introducing configuration debts while benefiting from a high-level and compact format.

During the interview with the service mesh expert, we discussed the topic of "What automation is needed and is practical to address the practitioner's diverse concerns given the observation of errors and diverse fixes". Given the recent advances of machine learning techniques, the service mesh expert noted that they believe more work could be done in this area. "*For me, the main culprit of this issue is that errors and fixes are not a one-to-one relationship. And because many users who ask the questions have described their requirements in detail, and the answers have fixes for those errors, I have imagined there were tools that can generate complete and runnable examples for the user requirements, with the errors fixed. With the emerging Large Language Models (LLM), I think these kinds of tools are feasible and would be very helpful for practitioners.*" We conclude the existence of a pressing need for automation that can provide multiple service mesh implementation/configuration options with their advantages, disadvantages, and caveats. Such tool should help practitioners choose the most appropriate service mesh features and usage based on their needs.

Service mesh framework designers should focus on enhancing the framework's error diagnosis capabilities: We identified individual fix patterns that can cure multiple service mesh framework error symptoms due to unintuitive error messages. We also identified various fixes that could cure a single symptom. Therefore, we could not generalize a common fix pattern for most symptoms. We suggest that framework designers systematically identify and better hint at the root causes of error symptoms, eventually enhancing the error messages and eliminating implicit failures (i.e., failures without obvious errors). This way, typical service mesh practitioners could quickly troubleshoot independently or provide precise diagnostic details while asking for help on Q&A platforms. It will also be easier for researchers to generalize common fix patterns and empirically verify best practices that could lead to faster incident recovery.

Another common technique the industry adopts is implementing a unified standard for a range of similar products. For example, OpenTelemetry[21] aims to provide a unified standard for observability data. Nonetheless, when discussing such feasibility in the overall service mesh domain, the service mesh expert points out that unified standards would require collaboration among many experts to exhaust as many usage scenarios as possible. This task is not trivial as service mesh cannot be defined as observability data formats can. When asked what is the best solution to this problem, the expert adds that "*It can be challenging for a single enterprise or organization to gather all the necessary expertise for such implementations, unless they are open-sourced*". The expert believes an open-source standard would benefit practitioners by reducing the

[21] https://opentelemetry.io/

need for comparison and decision-making processes. It can also attract more developers to work on the same implementation, ultimately improving service mesh technology's overall adoption and usage.

Service mesh framework designers should practice documentation prioritization: We additionally take a sample of 360 question posts following Step 3.4 in our study protocol of Section 3.2 to understand more about the available resources that could help with practitioners' concerns. We find 158 external knowledge resources embedded in the question answers, which are essential sources of information to understand the state of the practice regarding specific service mesh frameworks and related technologies. Most interestingly, among 158 external resources, 68 pieces are directly hosted from the official documentation of corresponding service mesh frameworks, meaning that service mesh practitioners who posted the questions could not effectively locate them as helpful information. We speculate that this is due to the highly configurable nature of service mesh systems, resembling a "Too many knobs" scenario (123) that confuses new adopters.

According to our observation of the 158 extracted knowledge resource contents, explicit links between possible symptoms and their fixes (or practices to follow) are scarce, meaning such knowledge is hard to comprehend fully by inexperienced practitioners. Unfortunately, our results on the low answer acceptance rate of the service mesh questions (40%) and a long mean time to acceptance (353 hours) indicate that expertise is generally lacking for the service mesh domain.

Hence, we suggest that service mesh framework designers propose and practice better documentation prioritization techniques to assist new adopters in complex configuration scenarios. For example, one could gather the most frequently asked questions and answers into a FAQ and extract explanations of advanced features into secondary documentation as they can confuse new adopters (52). We also suggest a review of existing configurable knobs as suggested by previous literature (123) to prevent over-configurable systems.

3.5.2 For Practitioners Interested in Adopting Service Mesh

New service mesh adopters should take extra caution while exploring advanced capabilities: As the 2021 Gartner report on enterprise networking[22] shows, service mesh has left its "Peak of Inflated Expectations" and entered the "Trough of disillusionment" phase, where it could be challenging for early adopters to determine use cases and put service mesh into production environments. Our qualitative analysis suggests that typical practitioners could face complex error symptoms that lead to frustration, which also reflects in the low answer acceptance rate and long mean time to answer acceptance. Practitioners could also face unclear error messages and confusion from available technical resources.

Specifically, based on our analysis in RQ3, Istio adopters tend to face diverse traffic-related and observability-related errors. At the same time, errors related to infrastructure spanning the installation, configuration and connection of service mesh networks occur in both Istio and Consul. Practitioners should familiarize themselves with the error symptoms and common fixes of open-source service mesh frameworks to avoid pitfalls in adoption.

[22] https://blogs.gartner.com/andrew-lerner/2021/10/11/networking-hype-cycle-2021/

When asked about suggestions to practitioners, the service mesh expert highlights that sometimes a seemingly simple solution to complex problems may involve a higher cost or even lead to building new projects around the solution. Therefore, practitioners should be cautious in adopting advanced features they do not fully understand, as this may lead to technical debt in the long term. The expert also emphasizes the importance of carefully evaluating different implementation options and selecting those that best fit the team's expertise to further reduce the possibility of technical debt.

3.6 Threats to Validity

Below, we discuss threats to the study validity and the strategies we applied to mitigate these threats, based on literature guidelines (119).

Construct Validity: The heuristic-based classification technique used in Section 3.2.1 could misclassify some question posts because it is not possible to compose a comprehensive set of filtering keywords. However, the classification process is applied to both the service mesh domain and the traditional microservices to produce an unbiased comparison. Our filtering conditions only used natural language words that do not involve technical details. The potential bias is expected to be minimal and does not contribute to the significant differences in resulting metrics that we discuss in section 3.2.1.

Another threat is that our topic model is tuned based on the common practices of previous works using an iterative method and evaluation metric (92). We acknowledge that using other possible hyperparameters could reveal additional information not elaborated in this study. We extensively tested hyperparameters according to the

DTM model package documentation and previous literature. It is also possible that using a more advanced clustering model that considers question semantics would yield results that we have yet to discover. However, to our best knowledge, such models require massive data sizes and are hard to implement. We choose to utilize the DTM model as it has been widely applied in recent studies for various topic modelling purposes and is proposed as an alternative to the original Latent Dirichlet Allocation (LDA) algorithm (18; 19).

Internal Validity: One threat to our study is that qualitative analysis can be biased due to its subjective nature. To further enhance our study's validity, we carefully conducted multiple iterations of analysis to refine previously classified questions using newly-learnt domain knowledge. It was not feasible to conduct a full-scale manual analysis of the collected question dataset. As described in the study steps 3.1.5, we applied sampling with 95% confidence to best preserve its representations. We admit that, by sampling, we could not exhaust every possible fix pattern because of the highly diverse failure symptoms. Yet, the richness of the symptoms and fix patterns that we identified, as well as the fact that towards the end of analyzing our sample no new symptoms/fixes were encountered, provide confidence about the degree of completeness of the qualitative analysis.

Another potential threat to the validity of our study is that we only consulted one service mesh domain expert. While the expert has extensive experience in the domain and has contributed significantly to the community, their opinions and insights may not represent the broader service mesh community. This could lead to biased feedback and limit the generalizability of our study's findings. To mitigate this threat, we conducted an in-depth discussion with the expert. We exchanged our ideas from the perspectives of researchers and practitioners for each implication regarding their practicalness and feasibility. To provide better consistency and clarity, we optimized the textual representation of the topic names and the qualitative labels (without altering the topics' meanings) by incorporating the service mesh expert's feedback. Future research may benefit from soliciting feedback from multiple experts with diverse perspectives.

Conclusion Validity: In the qualitative analysis, we identified the many-to-many relationships between multiple error symptoms and multiple possible fixes. However, it is possible that one can generalize more common fix patterns given a comprehensive knowledge of the framework and eliminate the many-to-many relationship by understanding the root causes of each symptom. Our findings do not necessarily suggest that a common fix pattern cannot be generalized from a given complex error symptom. Still, with the domain knowledge of a typical service mesh practitioner or researcher, they cannot effectively pinpoint a fix pattern in case a root problem can cause multiple errors and lead to confusing error messages.

External Validity: During the RQ3 analysis, we only considered the top two open-source service mesh frameworks because the other frameworks tend to involve minimal discussions. According to market share reports[23] and justifications from Section 3.1, the analyzed frameworks correspond to the top two open-source solutions in the service mesh domain. Due to the nature of commercial service mesh frameworks and paid support services, our findings possibly may not generalize to them.

Another threat is that we base our study on Stack Overflow and dedicated fora data. We admit that user discussion could happen in alternate sources, such as mail lists and IRC channels. However, such data sources are less structured, making it difficult

[23] https://youtu.be/Du8ImGRd2TI

to determine the "accepted" answer for a given question. Therefore, we choose Stack Overflow and use dedicated fora as supplemental data, since both include sufficient data and are among common practitioner preferences while establishing online support channels (107).

3.7 Chapter Summary

This chapter presents an in-depth study of 5,497 service mesh questions on popular question-and-answer platforms, including Stack Overflow and two public fora of open-source service mesh frameworks. While previous works focused on exploring and addressing technical security and traffic-related challenges through controlled experimentation, our study has systematically evaluated existing practitioner question topics, intentions, adoption goals, error symptoms and fixes in the service mesh domain. We provided backgrounds that verify the presence of rising concerns around service mesh security, infrastructure and observability.

We first identified the current state of question types, acceptance and topics via topic modelling. We then explored the evolution of aggregated macro-topics over time and discovered the persisting infrastructure and security-related concerns. We conducted a systematic qualitative analysis, inspecting more than 800 question posts to cross-validate previous research question findings. We compared two popular service mesh frameworks to uncover practitioner knowledge intentions, adoption goals and error symptoms. We reviewed accepted answers and knowledge resources to derive common fix patterns and understand the difficulties in solving the symptoms from a practitioner's perspective. We also uncovered insights on many-to-many error-fix relationships and the impact of minimal configuration manifest changes in highly configurable systems like service mesh.

We suggest researchers to explore the usability and adoption complexities of security-related features to implement secure software architectures. We also identified a need for more user-friendly observability features, which should be modelled accurately according to industry needs. Additionally, we highlight the need for enhanced methodologies and automation to assist practitioners in configuring service mesh infrastructure through configuration engineering. Academia and industry should invest more effort to drive such advances, which could significantly improve the adoption of heterogeneous systems such as service mesh.

Recognizing the crucial role of Kubernetes and Helm Charts in the deployment of service mesh architectures, our findings advocate for a comprehensive investigation into Helm Charts' security. Kubernetes, being the underlying orchestration platform, and Helm Charts, a popular deployment tool, contribute significantly to the service mesh infrastructure's complexity. This complexity, combined with a large number of unfixed vulnerabilities in Helm Charts, could broaden the attack surface, making service mesh systems susceptible to security risks. The next chapter of this thesis delves into this critical aspect of service mesh deployment - the security vulnerabilities within Helm Charts. We aim to unveil the prevalence of unfixed yet fixable vulnerabilities (CVEs), their potential impacts, and mitigation strategies. This approach aligns with our goal to enhance service mesh security from the ground up, ensuring that both the deployment process and the deployed systems are secure and reliable.

CHAPTER 4

Why Not Fixing the Fixables? An Empirical Study on Understanding the Unfixed Vulnerabilities in Helm Charts

The rise of containerization technologies has fuelled the rapid adoption of container orchestration platforms such as Kubernetes as a de-facto industry standard. Containers allow practitioners to deploy complex and heterogeneous applications into artifacts that can be efficiently managed at scale (25). However, configuring and managing large Kubernetes deployments remains complex, creating adoption concerns that we analyze in the previous chapter.

Such concerns have led to the rise of Helm as a pivotal tool for defining, installing, and managing Kubernetes applications through reusable packages called Helm Charts[1].

[1] https://helm.sh/docs/topics/charts/

The Helm Charts encapsulate all necessary Kubernetes resources like services, deployments, and configurations into a single, reusable bundle (49). Modern heterogeneous distributed systems such as service mesh, with their intricate orchestration of microservices, particularly benefit from Helm's streamlined deployment approach. However, this ease of deployment also widens the attack surface for service mesh systems, where security is a critical requirement, if vulnerabilities exist in Chart dependencies(81). By prepackaging intricate systems into one-click infrastructure deployments, malicious actors could exploit existing unfixed vulnerabilities to attack critical systems deployed via Helm Charts.

However, this ease of deployment presents a double-edged sword. While Helm Charts enhance operational efficiency, they simultaneously broaden the attack surface for critical systems, such as service mesh architectures, that require heightened security. If vulnerabilities exist within Helm Chart dependencies, these one-click installations become potential access points for malicious actors (81).

Despite these risks, research on vulnerability management in Helm Charts is currently scarce. While previous work (127) have examined vulnerabilities in individual Charts in terms of their counts, no holistic understanding of why such vulnerabilities exists across the widespread Helm ecosystem and most importantly: why do they remain unfixed despite being fixable. Past related work has explored vulnerabilities in Docker containers(36), infrastructure-as-code systems(62), and supply chain attacks(84). However, the unique nuances of the Helm ecosystem warrant dedicated investigation given they serve as the final artifact before deployment and are loosely assembled by numerous dependencies maintained by different parties (49).

This mixed-methods study addresses this gap through an in-depth analysis of 11,035

Helm Charts from Artifact Hub and 90 associated GitHub repositories. We quantify vulnerability prevalence, correlate contextual factors with vulnerability severity, and probe vulnerability mitigation practices via a grounded theory methodology that commonly used in previous empirical studies to understand complex phenomenon (2; 28; 40; 46; 58; 108). Our qualitative grounded theory approach complements the quantitative analysis by eliciting practitioner perspectives (chart maintainers and end users).

Our main study results and their implications include the following:

- We find a high prevalence of vulnerabilities, with a total of 14,448 unique CVEs identified across the 6,203 analyzed Helm Charts with a security report. This indicates widespread vulnerabilities across the ecosystem. Notably, only 56% of the Chart artifacts utilize security reports and 1% use security fix indicators, highlighting insufficient attention to security from maintainers.

- We find the distribution of CVEs per Chart to be skewed (skewness coefficient of 30.375), deviating significantly from normality based on Shapiro-Wilk and Anderson Darling normality tests (p-value < 0.05). Interestingly, the number of CVEs of a chart significantly correlates with the total package count within container images (partial correlation coefficient of 0.625, p<0.001) but not the with the count of container images themselves (partial correlation of 0.073, p<0.001). This suggests that practitioners should not assume charts with fewer dependency container images to be more secure.

- We find that the median age of unfixed CVEs exceeds 100 days across Charts, reaching up to 537 days for non-critical CVEs in unofficial charts. This prolonged timeframe, even for serious vulnerabilities, indicates inefficient mitigation processes, with delays propagating vulnerabilities downstream along the software supply chain.

- Statistical analysis shows significant differences in CVE severity distributions across Chart categories ($p<0.05$) and officiality ($p<0.05$). However, effect sizes are negligible based on Cliff's Delta and Cramer's V values, indicating contextual factors alone cannot practically predict or filter out vulnerable Charts.

- Our qualitative analysis uncovers 12 mitigation strategies across 3 categories in 90 chart maintenance repositories: Ad-hoc (6 strategies), Automated (3 strategies), and Informative (3 strategies). We find an under-use of strategies like CVE scanners and customized images due to misaligned incentives, lack of expertise, and trade-offs between non functional requirements such as stability and security. Adopting a shared responsibility model and informative strategies could improve utilization as we discover a "shift to the right" phenomenon that risks the systems security of end chart users.

- We provide the replication package of our study with the main study artifacts, including the data collection scripts, data analysis scripts and spreadsheets for the grounded theory process (32).

We present the remainder of this chapter as follows. Section 4.1 motivates our study with three research questions to be answered in a progressive manner. We also follow each detailed step with analysis protocols, data collection techniques and corresponding artifacts. Sections 4.2 to 4.4 discuss the evaluation and results discussions for each research question. Section 4.5 summarizes the key findings and further gives practical and actionable suggestions to researchers in the domain and practitioners who are looking to use/maintain Helm Charts with better security in mind. Section 4.6 acknowledges and presents the potential threats to our study and justifies the methodologies we adopted to best complement some of the shortcomings. Finally, Section 4.7 concludes our findings and looks into potential research opportunities in the future.

4.1 Methodology

4.1.1 Goal and Research Questions

We rely on the Goal/Question/Metric template (12) to define our study goal as follows: *to analyze* Helm Chart CVEs and the mitigation strategies employed by Chart maintainers; *for the purpose of* understanding the prevalence of fixable but unfixed CVEs in Helm Charts; *with respect to* identifying the characteristics of infrastructure-level CVEs, severity distributions among contextual factors, and currently available strategies to mitigate such CVEs in the maintenance process; *from the point of view of* Helm Chart maintainers and software engineering researchers; *in the context of* Helm Charts and their associated publicly disclosed vulnerabilities and the maintenance history of Helm artifacts on open-source code hosting platforms such as GitHub.

We introduce the three research questions (RQs) below to achieve the above-mentioned goal.

RQ1: How prevalent are CVEs in Helm Charts, and how long does releasing a fix take for the CVEs?

Motivation: This research question stems from the growing concerns regarding the prevalence of CVEs (Common Vulnerabilities and Exposures) in the containerization domain. As Helm has become a pivotal tool in the Kubernetes ecosystem, the security of its Charts has become a significant pain point. Before diving into the reasons behind unaddressed CVEs, it is critical to understand how widespread such vulnerabilities are within Helm Charts. Moreover, it is unclear what factors impact the typical time users have to wait before a release with CVE mitigation is shipped. Hence, we explore this time, which could provide indication of the prevalence of the unfixed CVEs and their fixing time.

Approach: We collect and analyze a comprehensive dataset of all publicly available Helm Charts on Artifact Hub, extracting their versions and identifying their associated CVEs through corresponding security reports. By evaluating the frequency of CVEs, we establish a set of measures of their prevalence in Helm Charts. Furthermore, we track the time between the disclosure of the vulnerability and the release time of the analyzed charts to gauge the time chart users have to wait before a fixable CVE is mitigated. This investigation provided insights into the security landscape of Helm Charts.

RQ2: How do contextual factors impact CVE severity in Helm Charts?

Motivation: Previous empirical work (128) suggests that contextual factors such as application domain play a significant part in understanding the metric distribution in software engineering maintenance. Understanding the impact and criticality of vulnerabilities based on contextual factors such as chart officiality (i.e., maintainer ownership) and category (i.e., application domain) could help researchers and practitioners to pinpoint a subset of charts with higher security concerns. In addition, we can provide empirical evidence on whether users could rely on such contextual factors to search for charts that are more secure in terms of vulnerability severity.

Approach: Artifact Hub offers two contextual filtering conditions for chart users (available on the UI), namely the officiality and categories of Helm Charts. Artifact Hub considers a chart "official" if the publisher of the chart owns the core dependencies deployed in the chart. Furthermore, based on the application domain of core dependency, Artifact Hub sorts the charts into 8 distinct categories (e.g., AI/Machine Learning). To investigate the severity distributions of fixable CVEs in Helm Charts, we collect the CVSS V3 scores and CVSS V3 metrics. We conduct a series of statistical tests to uncover distribution differences between the different aggregations based on contextual factors and shed light on their practical effects. To ensure the validity of our statistical tests, we randomly sample 353 charts based on a 95% confidence level and manually verified the labelling is faithful to the true application domain of the charts. We conduct such verification by checking available information online (grey literature and official websites) against the core dependency as indicated in the Helm Chart metadata.

RQ3: What strategies do Helm Chart maintainers employ to mitigate fixable CVEs?

Motivation: Previous work (34) states that current security research relying on statistics is mostly flawed due to statistical biases. Hence, we additionally conduct a grounded theory study (131) to provide qualitative insights from the perspective of multiple stakeholders of the Helm Chart domain. Understanding the mitigation strategies employed by chart maintainers is crucial for stakeholders (researchers and maintainers) to understand whether a strategy is effective and widely accepted by others, therefore allowing future improvements. By examining the measures taken to address fixable CVEs and corresponding discussions on GitHub, we can identify deep insights into why CVEs remain unfixed. This knowledge will enable us to provide evidence-backed recommendations and guidelines for maintainers to effectively mitigate vulnerabilities in Helm Charts.

Approach: To investigate how maintainers mitigate fixable CVEs in Helm Charts, we analyze the maintenance activities of chart artifacts on GitHub. Previous studies in the domain of empirical software engineering have successfully used the Grounded Theory (GT) approach to investigate complex strategies in terms of developer activities and strategies. Therefore, in this study, we use a grounded theory approach to investigate current CVE mitigation strategies in a comprehensive way and aim to answer the question of why a large number of fixable CVEs remain unfixed from in a qualitative perspective. We rely on the qualitative findings to compensate previous quantitative results on the time and severity of the unfixed, yet fixable, CVEs.

By examining the commit history, release notes, and related discussions, we identify prevailing approaches and key takeaways from successful CVE mitigation efforts. We also record attempts to apply mitigation strategies that receives a negative stance from maintainers. Such opinions represent the maintainers' stance implementing a particular mitigation strategy from a socio-technical perspective, which previous empirical study has found crucial in understanding software maintenance (78). In the individual observation section of each mitigation strategy, we explain the stance factor with specific examples. In the end, we formulate a grounded theory that could answer the question "Why do fixable CVEs remain unfixed in the Helm Charts?".

4.1.2 Data Sources

To understand why fixable vulnerabilities remain unfixed, we first assemble a comprehensive understanding of the current state of vulnerabilities in the Helm Charts artifacts. During the initial data exploration on the Artifact Hub, we identified a sufficiently large number of publicly available Helm Charts (11,035), along with informative security reports (6,202), which are suitable for obtaining a complete view of the state of vulnerabilities. We perform a in-depth quantitative analysis by extracting and parsing the chart metadata and CVE information within the security reports.

However, solely relying on the chart artifacts and security reports remains insufficient to draw practical conclusions on the prevalence of unfixed yet fixable CVEs. This is because it is also important to understand the current CVE mitigation strategies and their adoption throughout the maintenance activities of the charts. We find that such information for mitigation is not inherently provided in the Artifact Hub data, given that only the final artifacts (chart files) are hosted.

Information related to mitigation strategies is naturally encoded in the GitHub repositories that serve as the source of the Helm Chart releases. Previous studies (1; 24; 59) use GitHub data to understand how developers handle security-related issue reports and provide valuable empirical insights. In our case, we find that information related to CVE mitigation exists beyond issue reports, inside CVE-related pull requests, commit messages, automation, and changelogs in maintenance repositories. Such data is naturally suitable for grounded theory exploration, since no existing guidelines could suggest where to mine such strategies.

We use multiple iterations of open coding and axial coding to extract reliable observations and provide evidence-backed theories. We use the theories to answer the open questions remaining after quantitative analysis and to ultimately provide an explanation about why fixable CVEs remain unfixed in the Helm Charts landscape. In the remainder of this manuscript, we refer to the GitHub repositories as “maintenance repositories”, in contrast to the Artifact Hub “artifact repositories”.

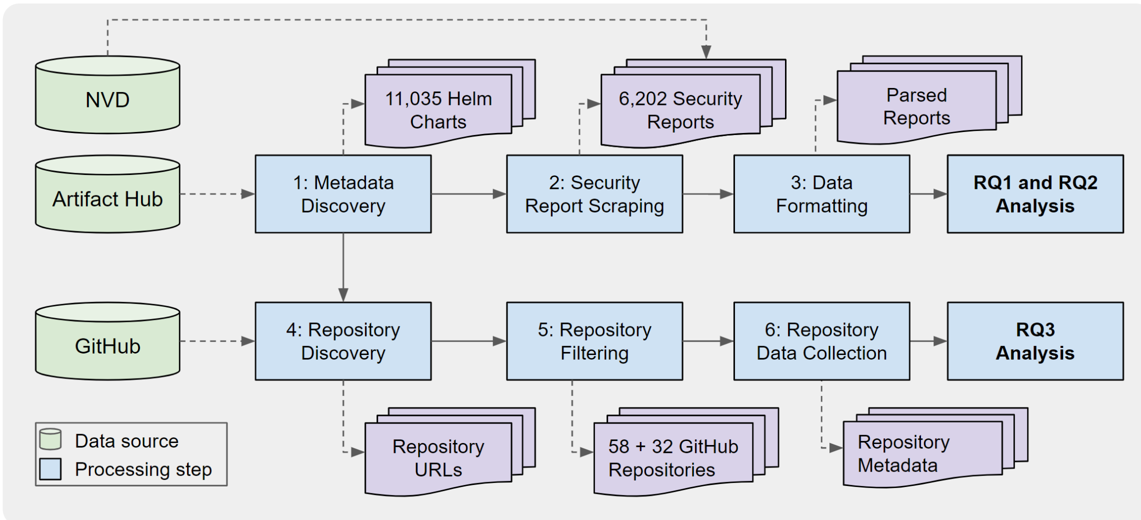

Figure 4.1: Data Collection Steps

4.1.3 Data Collection

Figure 4.1 depicts the steps we follow to collect the Helm Chart security reports and the GitHub dataset (along with the specific output artifacts from each step). All scripts and data artifacts involved in the collection, filtering and analysis steps are available in the replication package (32):

Step 1: *Metadata Scraping* – We retrieve a comprehensive dataset (11,035 Helm Charts) from the Artifact Hub public API[2], which covers all Helm Charts published to the Artifact Hub registry since its establishment in 2020[3] until May 2023.

Step 2: *Security Report Scraping* – With the package IDs retrieved from the metadata of each Helm Chart, we extract its latest security report through the public Artifact Hub API[4], if available. As a result, we compose a dataset containing 6,202 comprehensive security reports containing CVE data for the 11,035 Helm Charts. To enable our

[2] https://artifacthub.io/docs/api/
[3] https://www.cncf.io/projects/artifact-hub/
[4] https://artifacthub.io/docs/api/

analysis on the age of CVEs in RQ1, we extract the metadata of 12,071 fixable and unfixed CVEs from the National Vulnerability Database (NVD) (20).

Step 3: *Data Formatting* – We keep records of all 11,035 Helm Charts (230 official) and 6,202 security reports as the raw input. Yet, the complexity of security reports (>100KB in JSON) for direct interpretation is large. As a result, we use SQLite (Hipp) and Python to parse each security report and extract metrics required for further analysis, including the category and dependencies of each Helm Chart to the specific characteristics of each unfixed CVE and their impacted packages, etc.

Step 4: *Repository Discovery* – As introduced in Section 2, Artifact Hub is the registry for actual Helm Chart Artifacts, yet the daily development and release pipelines of the charts remain dominantly on GitHub[5]. We analyze both unofficial and official Helm Chart GitHub repositories in this study through a grounded theory qualitative study.

During the initial data exploration phase, we find that unofficial repositories are largely hosted under a non-predictable name different from the chart name. Therefore, it is impossible to backtrack all unofficial charts to their development source. Since unofficial charts represent 97.9% of all Helm Chart artifacts, we use a stratified sampling method (2; 28; 40; 108; 131) considering the imbalanced ratio of official and unofficial charts (230 and 10,805 samples) and randomly pick a representative sample of the unofficial charts based on a 95% confidence level. As a result, we obtain a sample of 32 unofficial GitHub repositories to serve as the initial dataset of our grounded theory analysis. If saturation would not be reached after the open coding process, this dataset would be extended. We reuse the definition of saturation used by previous empirical software engineering study (58; 131), i.e., saturation is said to occur as further data collection and analysis reaches diminishing results in terms of new strategies and strategy

[5]https://github.com/

observations.

In addition to unofficial Helm Charts, we also initially use all the official charts (230 charts) as the dataset for open coding. Given their explicit links to their original maintenance repositories, we rely on a exhaustive manual inspection to extract and backtrack the actual source repository for the official Charts. We ensure the Helm Chart to GitHub repository mapping is accurate by first attempting to extract a GitHub reference in the Chart's README from the Artifact Hub dataset. In cases where a chart source reference is missing, we manually search on GitHub and the Google search engine for chart repositories maintained by the same owner of the core dependency in the Helm Chart. Through this process, we notice that it is common for multiple Helm Charts to share a standalone repository if the same owner maintains an ecosystem of projects under the same domain. We filter out 3 cases where the repository is unavailable, as the main dependency of the Helm Chart is under a commercial license, and thus the source is not publicly hosted. In the end, we discover 90 GitHub repositories for 230 official Helm Charts through the GitHub RESTful API[6].

Step 4: *Repository Filtering* – Upon gathering the GitHub repositories, we conduct filtering based on repository metadata. We ensure the quality of each repository mined by following common guidelines for Mining Software Repositories (MSR) (37; 55).

Specifically, we first apply a quantitative filtering for repositories to ensure that each analyzed repository is not a fork (except template repository) or mirror. We ensure that the repositories contain at least 10 commits and have released more than one chart version, which serves as evidence for meaningful development activities. Since we conduct a grounded theory study in RQ3 that analyzes various types of information contained in a repository, we do not enforce strict filtering criteria such as a minimal

[6] https://docs.github.com/en/rest

amount of issues and pull requests, since other data types, such as automation and release notes in the repository, may also provide useful insights.

We focus on repositories corresponding to a standalone Charts repository on GitHub that could include multiple Helm Charts maintained by the same group. Such a filter reduces the false positives for non-chart related maintenance activities and further strengthens the validity of our study. This is because we search for vulnerability reports that target the actual released Helm Charts instead of targeting the development related activities such as tooling, build systems etc. At the end of this step, our dataset includes 58 official and 32 unofficial standalone Helm Chart maintenance repositories.

Step 5: *Repository Data Collection* – We filter for vulnerability-related issue reports and pull requests by searching for a collection of keywords connected by the Boolean OR condition, including “CVE”, “GHSA”[7], “NSWG”[8], “SUSE”[9], and “vulnerability”. We include the latter three terms, since we find that a number of CVEs were initially assigned with an alternative alias according to their origin (GitHub, NodeJS and SUSE Linux distribution).

Since we use a grounded theory approach that builds theories upon the Helm Chart maintenance repositories, we do not place a priori constraints about specific data types to use for the qualitative analysis. Instead, we use any data that appears in a GitHub repository, which typically involves issues, pull requests, source code, continuous integration pipeline definitions and metadata. We use theoretical sampling (40) to gradually extend our data analysis over multiple data types in order to explore CVE mitigation strategies and maintainer stance, and to mine extra observations that could uncover the effectiveness of such strategies.

[7]https://github.com/advisories
[8]https://github.com/nodejs/security-wg
[9]https://www.suse.com/security/cve/

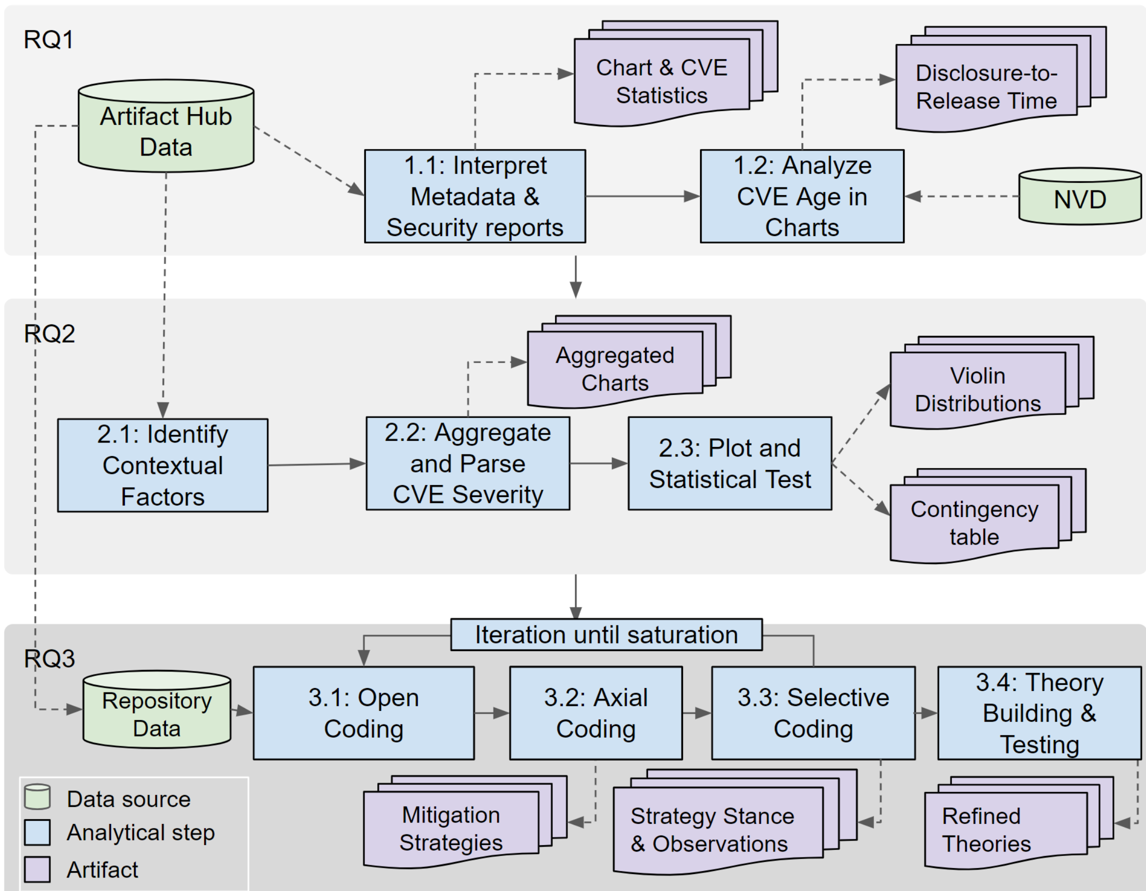

Figure 4.2: Study Protocol

4.1.4 Analysis Steps

This section introduces the study protocol we follow to conduct our analyses in each RQ, which is illustrated in Figure 4.2 (along with the specific output artifacts from each step, if available).

Step 1.1: Interpret Metadata and Security Reports – Before diving into the details of unfixed but fixable CVEs in Helm Charts, we analyze the landscape with high-level metrics and visualize their distributions in order to understand the prevalence of vulnerabilities in the Helm Chart ecosystem. We use the rich metadata shipped with the latest release of the 11,035 Helm Charts, including the dependency and CVE information. We also investigate the usage of out-of-the-box security features provided by Artifact Hub, which includes automatic security scanner reports generated by Trivy (the source of 6,202 security reports) and an indicator for security-related updates in a release. We determine the usage data by counting the number of Helm Chart artifacts that enables such a feature and populates corresponding data (CVE information and changelogs). The purpose is to understand whether such features are widely used to keep end users informed of the vulnerabilities in Helm Charts. Step 1.1 produces the "Chart & CVE Statistics" artifact, as shown in the figure.

Step 1.2: Analyze CVE Age in Charts – We then conduct a time-frame data analysis to analyze the time chart users have to wait before the first CVE fix is available through a chart release. Previous work has used the disclosure time of a CVE to measure the mitigation time (9). While it is not possible to collect data on the actual "fix date" of each CVE in each chart, we measure the wait time by calculating the number of days between the first disclosure time of a fixable CVE and the release time of the latest version of the Helm Chart (latest snapshot). We consider this time-frame as an underestimation of the actual time taken to fix a CVE in the charts since it remains unfixed in the latest release.

We also consider the severity levels of the CVEs, paying special attention at high to critical severity CVEs in order to understand if high severity CVEs are more likely to remain unfixed. We split the dataset by the "officiality" of the charts, which is an indicator of whether the maintainer of the chart also owns the core dependency deployed by the chart. We verify if officially maintained charts demonstrate significant difference in terms of CVE mitigation time. Step 1.2 produces the "Disclosure-to-Release

Time" artifact, as shown in the figure.

Step 2.1: Identify Contextual Factors – In RQ2, we dive further based on the observations of the RQ1 results. We expand the CVE analysis by identifying several key contextual factors that could help us understand the severity characteristics of unfixed CVEs in the Helm Chart ecosystem. Previous work has defined contextual factors as external factors that could impact the distribution of the studied object, such as software maintainability, application domain and system size (128). In our case, we find two contextual factors that are available for chart users as a filter on Artifact Hub, namely the officiality and category of Helm Charts, for further analysis in next steps.

- Officiality: An indicator of whether the core dependencies of a Helm Chart are maintained by the organization developing those.
- Category: An indicator of which domain the core dependency of a Helm Chart is targeted at. For example, Machine Learning and AI.

Step 2.2: Aggregate and Parse CVE Severity – We measure the severity of the CVEs with the CVSS V3 base score and CVSS V3 vector, which are the latest standard for measuring the general severity of the CVEs (45). We do not measure the temporal and environmental scores of the CVEs, given that such data is not available for collection in the scope of this study and the Helm Chart domain (5). We use the CVSS V3 Scores and CVSS V3 Vectors that are attached in the security reports of each chart. To visualize the distributions of CVE severity, we aggregate the Helm Charts based on the contextual factors determined in the previous step. Step 2.2 produces the "Aggregated Charts" artifact of the figure.

Step 2.3: Plot and Statistical Test – We first use two ways to visualize the data to provide a reliable interpretation of the analysis results. Since CVSS V3 scores are numerical and are assigned with a value ranging 0 to 10, we use split violin plots to visualize the distributions when aggregated by officiality and categories. On the other hand, CVSS V3 Vectors are a set of 8 categorical metrics of exploitablity and impact used to calculate the base scores[10]. We use a contingency table to understand if the contribution of each factor to the CVE severity could vary based on officiality of the Helm Charts. Finally, we conduct non-parametric statistical tests to measure if the plotted distributions show statistically significant differences. We use the Mann-Whitney U Test for numerical distribution and Chi-Square Test for categorical data (76; 83). Step 2.3 produces the “Violin Distributions” and “Contingency Tables” artifact, as shown in the figure.

Step 3.1: Open Coding (Iterative) – In RQ3, we begin our grounded theory building process following a rich set of existing guidelines. Previous empirical software engineering studies (2; 28; 40; 108) have shown grounded theory to be a reliable approach to investigate the complex behaviour involved in software engineering activities. In our case, we find that the maintenance practices of Helm Charts vary among the repositories, where a grounded theory analysis could explain such differences and understanding of the reasoning behind the strategies.

We start the process by conducting an open coding of the GitHub repository data (32 unofficial and 58 official) based on all their maintenance history. In the first iteration, we investigate all data types including issues, pull requests, source code, continuous integration runs and release notes to extract any strategies applied to mitigate CVEs. Specifically, for each repository, we manually investigate the following data:

- Repository metadata (README, Link to Documentation): Gain insight in potential mitigation strategies for unfixed vulnerabilities, applied in the repository.

[10] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

- Issues, pull requests and discussions: Specific discussions and change attempts to mitigate CVEs, maintainer stance over such changes and proposals.
- Security advisory tab: Repository policy on CVE disclosure and mitigation.
- Automation (i.e., GitHub Actions[11]): Automation definition and results to detect CVEs and apply automatic patches.
- Code Changes (Commits): Detailed Chart manifest changes to release.
- Changelogs: Changes between releases, mention of CVE fixes and notes.

We write down memos of existing mitigation strategies and mark the usage of such strategy for the current repository. If a strategy is unseen before, we add it to the full list. In later iterations, we gradually merge strategies if they present similar actions and we write down details regarding each strategy to prepare the basis of empirical insights.

Throughout this process, we also write notes on whether the mentioned mitigation strategy applies to all situations, or whether the maintainers expressed mixed opinions including rejection or hesitation. One such example is that some maintainers are against the usage of image scanners due to false positives. We record other observations that could cross findings with previous RQs such as the lack of use of informative security features.

Step 3.2: Axial Coding (Iterative) – In this step, we follow the guidelines of grounded theory (108) to group our noted strategies into categories. We refine the categories iteratively through our memos to come up with finite categories and assign strategies to each of them based on the characteristics of the strategy action. Based on the open

[11] https://github.com/features/actions

coding results where we record the occurrence of strategies in each repository, we revisit the strategies to confirm that our observations remain up-to-date after the previous full iteration. We consider this step completed when all strategies could be fit into a category. Step 3.2 produces the "Mitigation Strategies" artifact of figure 4.2.

Step 3.3: Selective Coding – In this step, we proceed by focusing on the core categories that we identified in the axial coding phase. We draw interconnections between categories, focusing on the mitigation strategies adopted by maintainers for handling CVEs in Helm Charts.

By integrating and refining the categories and their properties around this main phenomenon, we construct a theoretical framework. We build a narrative around how the discovered mitigation strategies are used, along with their conditions and consequences, with specific examples. This yields the "Strategy Stance & Observations" artifact.

Step 3.4: Building the Theory – In this step, we synthesize the insights obtained from the selective coding process to develop a grounded theory. The constructed grounded theory explains the maintainers' strategies in terms of CVE mitigation for Helm Charts. At this stage, we propose an actionable theory that guides future researchers and practitioners in the domain, by providing recommendations on current practices and strategies for mitigating CVEs in Helm Charts. We also bring extra insights to observations that could not be explained via the quantitative results in RQ1 and RQ2. Step 3.4 results in the "Refined Theories" artifact.

4.2 How prevalent are CVEs in Helm Charts, and how long does releasing a fix take for the CVEs? (RQ1 Results)

4.2.1 Prevalence of Helm Charts Vulnerabilities

We discover 14,448 unique unfixed CVEs among 6,202 Helm Charts in their latest snapshot, yet we find a low utilization of informative security features such as security reports for CVE alerts (56%) and security update indicators (1%) for informing end users of CVE fixes.

We find that only 56% of the charts (6,202) choose to disclose a security report, while the remaining charts explicitly disable such a feature even if one could be effortlessly generated with Trivy [12] on a periodical basis. The high number of missing CVE reports underscores the need for pre-deployment detection and mitigation for vulnerabilities within Helm Charts artifacts. Additionally, although over 50% of the studied charts have published more than one versions, we find only 80 charts (1%) populate the "has_security_fix" indicator at least once in their release changelogs to disclose mitigation actions regarding CVEs. The low utilization of such informative security measures (security report and fix indicators) hints at insufficient attention of chart security from maintainers.

Table 4.1 presents 14,448 publicly disclosed unique CVEs identified across the 6,202 analyzed Artifact Hub Helm Charts with a security report. CVEs represent various security vulnerabilities that potentially impact diverse components of the deployed images in a Helm Chart. Though the vulnerabilities may not actually be exploitable to

[12] https://github.com/aquasecurity/trivy

Table 4.1: CVEs and Available Security Reports of 11,035 Helm Charts

Metric	Count
Total #CVEs	14,448
Total #Security Reports	6,202

external attackers, previous work (42) suggests that it is inherently difficult even for experienced engineers to prove that a vulnerability is unexploitable in a complex system like Kubernetes. We assume that a subset of the CVEs could introduce severe security consequences if not mitigated properly when deployed into mission-critical systems.

4.2.2 Image, Package and CVE Distributions in Helm Charts

We find that CVEs and package dependencies in Helm Charts do not follow a normal distribution and are highly skewed. Package count significantly correlates with the occurrence of CVEs, while image count does not.

While each dependency container image could contain multiple layers, each layer could introduce additional dependency packages (106), which in combination serves as a indicator for system size (128). We define a dependency package as the OS-level or Language-level dependencies installed from a corresponding package manager, such as “Apt-get" and “NPM" (4; 26). Given the high prevalence of CVEs in the Helm Charts ecosystem, we analyze three high-level metrics regarding existing unfixed CVEs of the collected charts and conduct statistical tests to shed light on the overall distribution of the CVEs across the dependency packages and images involved in the 6,202 Helm Chart security reports. Figure 4.3, Figure 4.4, and Figure 4.5 depict the distributions of the count of CVEs, dependency packages, and images involved in the Helm Charts, respectively. From the histograms (x-axis as log scale), we observe that the variance of

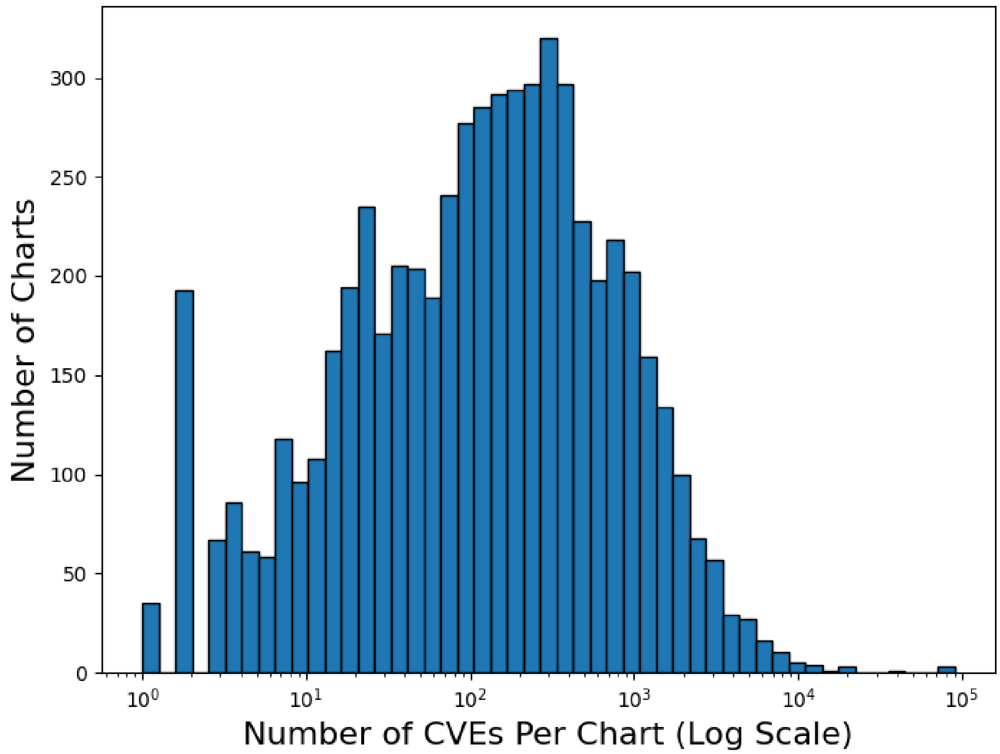

Figure 4.3: CVE Count Distribution in Helm Charts (6,202 Charts)

CVEs and dependency packages in Helm Charts is high, ranging from one to extreme values. In 13 cases, we even discover over 10,000 vulnerabilities (non-unique) in each of the charts [13].

Table 4.2 presents the skewness coefficients and results of the normality tests (Shapiro-Wilk Test and Anderson Darling Test (90)) for the three distributions. Overall, we observe that all three metrics indicate a high level of skewness (30.375, 29.615, 37.023), providing strong evidence to reject the null hypothesis of normality. In particular, we

[13] https://artifacthub.io/packages/helm/appscode/kubedb-one

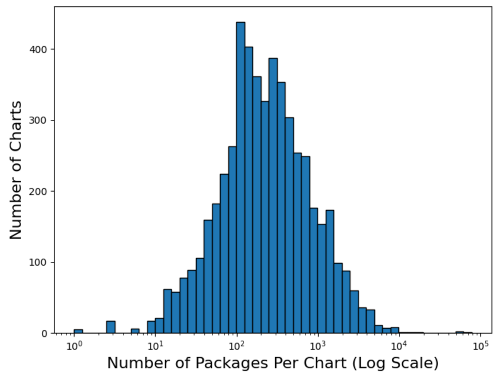

Figure 4.4: Dependency Package Count Distribution in Helm Charts (6,202 Charts)

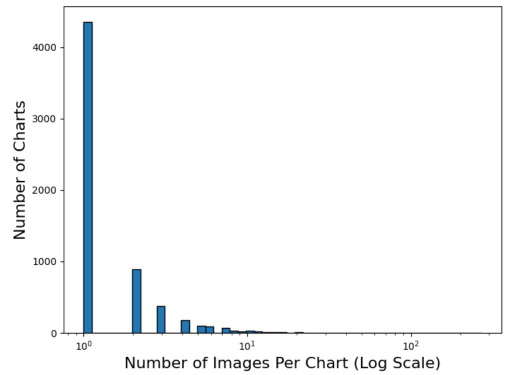

Figure 4.5: Image Count Distribution in Helm Charts (6,202 Charts)

Table 4.2: Skewness and Normality Test Results For 6,202 Helm Charts

Metric	**Skewness**	**Shapiro-Wilk Test**	**Anderson-Darling Test**
cve_per_chart	30.375	0.133	1415.392
package_per_chart	29.615	0.168	1098.653
image_per_chart	37.023	0.099	1605.349

observe that the number of images per chart demonstrates a significantly higher skewness coefficient of 37.023 than that of package count. Based on this, we further conduct a partial correlation test with package count and image count as independent variables and the CVE count in a chart as dependent variable.

The resulting partial correlation coefficients 0.625 and 0.073 ($p<0.001$) support our observation that the high number of CVEs involved in a chart does not necessarily relate to the number of images deployed by a Helm Chart. Rather, it is significantly related to the number of actual packages in a Helm Chart dependency. As a previous study suggests that container images could be stacked with a large number of layers (106), we further confirm that image count in the charts could be a deceiving characteristic that mislead practitioners' view over chart security since few images does not imply a lower security risk. Therefore, we suggest that practitioners should not rely on the number of images in a chart to assume the "safety" of the chart.

4.2.3 Exploring the Unfixed CVEs

We only find 42 Helm Charts that are not vulnerable to any CVEs. While a common set of unfixed CVEs impact a significant number of Helm Charts (40%), they differ significantly (75%) from the top 20 most frequent CVEs that impact multiple dependencies within the same Helm Charts.

Table 4.3 presents the top 20 unfixed public CVEs sorted by the number of impacted Helm charts. We find that the top 20 CVEs impact over 40% of the currently available Helm Charts on Artifact Hub, indicating that a large number of vulnerabilities are actually commonly shared, although remaining unfixed. Interestingly, we observe only five CVEs disclosed in the current year (2023), meaning a majority of the highly-prevalent CVEs remaining unfixed for prolonged periods. Although it is possible that Helm Charts remain secure given that a CVE may not be triggered in a chart component since the impacted features are not used. This could add informational overhead (94) to the end users since it could lead to misunderstanding over the integrity of the Helm Charts when they are evaluated with CVE scanners that are prone to false positives (80).

On the other hand, we also calculate and rank the total occurrence of CVEs across all dependency packages specified in the Helm Charts since image reuse is common in today's containerization ecosystems and therefore images frequently share similar vulnerabilities (36). Table 4.4 presents the CVEs sorted by total occurrence. In contrast to the previous Table 4.3, we count the number of most frequently occurring CVEs

Table 4.3: Top 20 CVEs Sorted by #Impacted Helm Charts

CVE ID	**#Charts**	**CVE ID**	**#Charts**
CVE-2023-0464	3,133	CVE-2013-4235	2,673
CVE-2023-0465	3,129	CVE-2020-16156	2,646
CVE-2016-2781	3,030	CVE-2021-33560	2,625
CVE-2017-11164	2,911	CVE-2023-0286	2,563
CVE-2022-29458	2,792	CVE-2023-0466	2,527
CVE-2022-1304	2,777	CVE-2022-3219	2,523
CVE-2021-36084	2,683	CVE-2019-20838	2,512
CVE-2021-36085	2,683	CVE-2023-0215	2,503
CVE-2021-36086	2,683	CVE-2022-4304	2,503
CVE-2021-36087	2,680	CVE-2022-4450	2,496

since common dependencies often reoccur within the same Helm Chart through different images. For example, OS-level packages such as “CURL” ship with diverse base images, whereas language-level packages such as “PyYAML” are frequently used as a configuration parser. Interestingly, the resulting CVEs show a drastic difference from the previous findings in Table 4.3, with only 5 CVEs in common with the previous table. Such a discrepancy contributes to the prevalence of vulnerabilities among Helm Charts, resulting in security risks. Specifically, the difference in vulnerability ranking between the two tables suggests that some vulnerabilities may impact specific Helm Charts more frequently but have a lower overall occurrence throughout the ecosystem. On the other hand, vulnerabilities with a lower frequency within individual Helm Charts may have a more widespread impact when considering the entire ecosystem. This variation highlights the importance of considering the broader context when assessing the prevalence and impact of vulnerabilities.

Table 4.4: Top 20 CVEs Sorted by Total Occurrence

CVE ID	Frequency	CVE ID	Frequency
CVE-2022-0563	28,048	CVE-2023-0464	11,313
CVE-2022-29458	20,206	CVE-2023-29383	9,854
CVE-2021-37600	15,233	CVE-2019-1010022	9,763
CVE-2022-1304	14,608	CVE-2010-4756	9,737
CVE-2022-42898	14,568	CVE-2018-20796	9,737
CVE-2022-3219	14,440	CVE-2019-1010023	9,737
CVE-2018-5709	13,804	CVE-2019-1010024	9,737
CVE-2021-39537	13,177	CVE-2019-1010025	9,737
CVE-2023-29491	12,103	CVE-2019-9192	9,737
CVE-2023-0465	11,320	CVE-2023-0466	8,905

Table 4.5: Comparison of the number of vulnerabilities and Median CVE Age for unofficial and official Helm chart CVEs

Type	#CVEs	Median Age
unofficial-rest	8,062	537
unofficial-high	2,805	368
unofficial-critical	701	532
official-rest	720	150
official-high	472	109
official-critical	95	179

4.2.4 Time to Mitigate Fixable CVEs

A majority of the CVEs (84%) in Helm Charts are fixable. We find that the median time until users could access a fixed chart release for such CVEs exceeds 100 days. Specifically, we noticed that CVEs with a critical severity often remain unmitigated for extended periods for both types of officiality (median 532, 179 days), surpassing even the high-severity CVEs (median 368, 109 days).

In Section 2, we define a fixable CVE as a vulnerability that has a public patch available that could be applied to chart dependencies. We find 12,701 CVEs are actually fixable by updating the dependency version, while only 2,377 CVEs are unfixable. Notably, of the fixable CVEs, 8,757 were disclosed before 2022. Furthermore, a considerable portion (21.4%) of the fixable CVEs disclosed prior to 2022 are of high to critical severity. This finding underlines a vast majority of fixable CVEs (72.5%) remain unfixed for at least 12 months regardless of their severity.

To understand the time users have to wait for CVE mitigation, we calculate the duration between a CVE's public disclosure and the corresponding chart release. A previous study (115) suggests that the prolonged time required to release a fix for the CVE could be attributed to the need for systematic security patch assessment by the vendor and asset owner, which in our context, would be the dependency providers and chart maintainers. To corroborate this hypothesis, we organized the dataset by the "officiality" of Helm Charts, a label indicating whether the core dependency and the particular chart are maintained by the same organization, which could potentially reduce the time needed for duplicate assessment efforts.

Table 4.5 depicts the median number of days elapsing between the disclosure date of fixable CVEs and the date of a chart release that contains such a fix. We observe that the officially maintained charts exhibit a significantly younger set of CVEs (median 150, 109 and 179 days) than unofficial charts (median 537, 368 and 532 days). Overall, we find in all cases that the median age of CVEs exceeds 100 days regardless of severity and officiality. Notably, the median time-to-release for critical CVEs is surprisingly longer than that of high-severity ones, which could be attributed to the greater impact of critical-severity CVEs and thus more difficult to mitigate in time. As a result, such evidence hints that the current CVE mitigation process could be inefficient, considering the high percentage of fixable CVEs that remain unaddressed and the prolonged mitigation periods.

Summary of Research Question 1

A common set of 20 CVEs impacts over 40% of Helm Charts, whereas we find a different set of 20 CVEs that affect multiple components within the same Chart. We find a low utilization of informative security features even though they are built into the Artifact Hub. It is evident that Helm Charts are prone to security risks since they are vulnerable to a high number (12,701) of fixable CVEs and experiencing prolonged wait time (median over 100 days) between CVE disclosure and chart release.

4.3 How do contextual factors impact CVE severity in Helm Charts? (RQ2 Results)

4.3.1 Distribution of CVSS Scores

We find statistically significant differences in the distribution of CVSS severity across different Helm Chart categories and officiality, yet with negligible effect sizes (Cliff's Delta$\delta < 0.147$ **and and Cramer's V**< 0.05**)).**

Figure 4.6 visualizes the comparison of CVE score distribution across officiality. Figure 4.7 shows the difference in CVSS distributions in each of the 8 chart categories using split violin plots (split by officiality). In order to draw reliable insights, we test the differences with a set of statistical methods.

We first perform Shapiro-Wilk tests for data normality, and we find that severity

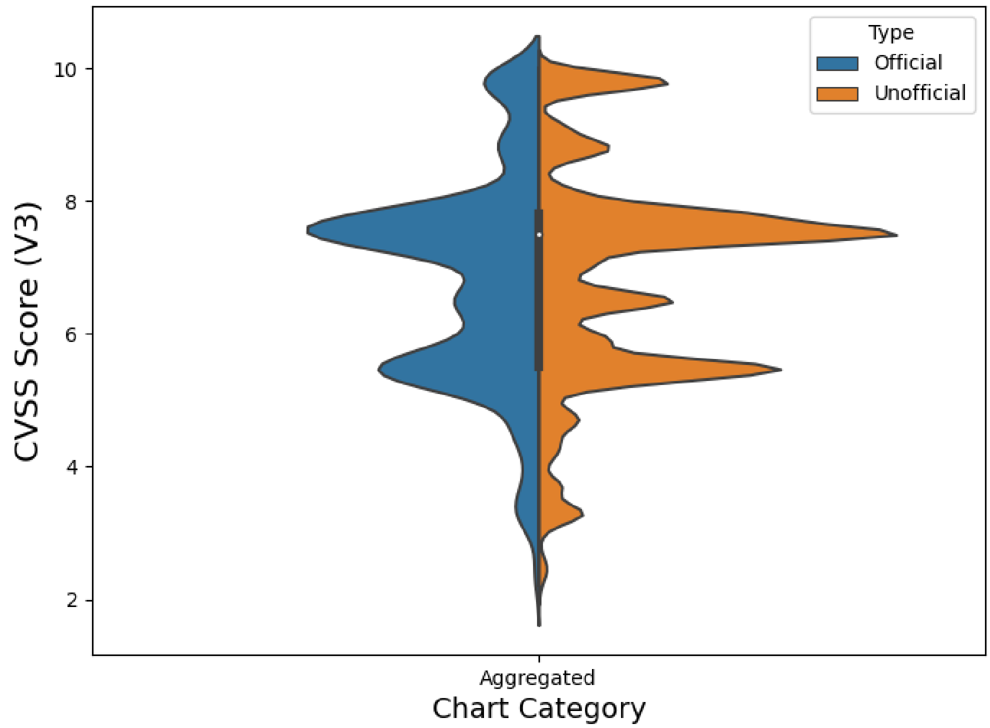

Figure 4.6: Aggregated Distribution of CVSS V3 Score (Split by Officiality)

distributions do not follow a normal distribution (p-value < 0.05) in any of the groups. As a result, we apply the non-parametric Mann-Whitney U Test (p-value < 0.05) for pair-wise comparisons between each category and the aggregation of the remaining data (i.e., we remove the current category data from the overall dataset) (105). In other words, we effectively measure if certain chart categories stand out in terms of CVE severity score distribution. We adjust the p-values with Bonferroni correction by dividing the test count to reduce risks of false positives (128). We formulate a null hypothesis as follows:

[Null Hypothesis 1]: There is no significant difference in the severity distribution of

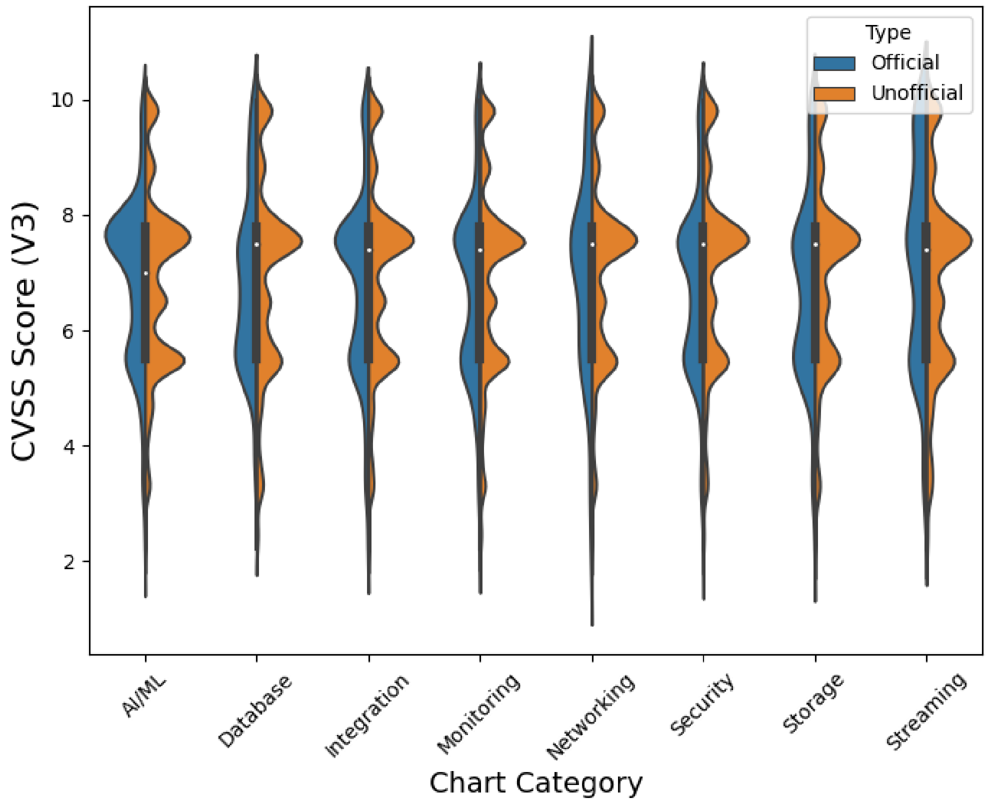

Figure 4.7: Categorical Distribution of CVSS V3 Score (Split by Officiality)

the comparisons considering different contextual factors.

Table 4.6 depicts the test results aggregated by chart officiality and categories, the blue violin represents the severity distribution of official Charts, while the orange violin represents unofficial charts. On the other hand, Table 4.7 shows the test results across different officiality groups within each category, which allows a direct comparison between CVE severity distribution of charts across different categories. In both tables, we report the adjusted p-value and the Cliff's Delta effect size. From Table 4.6, we observe that in the case of official charts, significant differences in CVSS scores were found only in the 'AI/ML' and 'Streaming' categories compared to the aggregated category. Furthermore, the effect sizes, as indicated by Cliff's Delta, are considered negligible ($\delta < 0.147$) referencing previous literature (93). In contrast, for unofficial charts, all categories except 'Database' and 'Streaming' show significant differences. However, similar to the official charts, the effect sizes were also negligible in terms of practicalness. From Table 4.7, we observe that there are significant differences in CVSS scores between the official and unofficial charts in the 'AI/ML,' 'Monitoring,' 'Security,' 'Storage,' and 'Streaming' categories, as well as in the aggregated data. However, the effect sizes are consistently negligible across all categories. The 'Database' categories do not show any statistically significant difference in CVSS scores either within or across officiality.

In essence, our statistical findings support the rejection of Null Hypothesis 1, as we observe significant differences in the CVSS score distributions in charts, aggregated by officiality and category. However, the negligible effect sizes imply that the actual impact of these differences on real-world security evaluation is likely to be minimal. As a result, from a security standpoint, practitioners are unlikely to benefit from choosing official charts over unofficial charts, in none of the categories.

4.3.2 Categorical Distribution of CVSS Metrics

We confirm that users generally do not have reliable measures to avoid vulnerable charts using existing filtering measures of Artifact Hub, since unexplored factors exist in the ecosystem. As introduced in Section 2, we further analyze the 8 fine-grained

Table 4.6: Adjusted P-values for Mann-Whitney U Test Comparisons and Cliff's Delta Against Aggregated Category

Category	**Official P-value**	**Official Cliff's δ**	**Unofficial P-value**	**Unofficial Cliff's δ**
AI/ML (1)	3.01e-02	0.073 (NG*)	7.64e-25	-0.040 (NG)
Database (2)	NS*	-	NS	-
Integration (3)	NS	-	9.93e-04	-0.013 (NG)
Monitoring (4)	NS	-	1.53e-02	-0.009 (NG)
Networking (5)	NS	-	1.89e-08	0.025 (NG)
Security (6)	NS	-	3.33e-23	0.039 (NG)
Storage (7)	NS	-	5.29e-05	0.027 (NG)
Streaming (8)	4.10e-03	0.108 (NG)	NS	-

*NS: Non-Significant adjusted P-value *NG: Negligible effect size < 0.147

CVSS metrics that compose the base CVSS score, which approaches the severity of vulnerabilities from a more detailed dimension. Given that CVSS base scores are calculated based on eight distinct metrics, we formulate a second, more refined null hypothesis as follows:

[Null Hypothesis 2]: There is no significant association between any of the eight CVSS metrics and contextual factors.

Table A.3 and Table A.4 in the appendix depict detailed frequency data aggregated by category and officiality. We populate these tables with the frequencies of the ordinal values of the 8 metrics. Each table column represents an independent chart category/officiality, and each row represents a possible value of the metric. We apply Pearson's Chi-square test on each of the 8 metrics (76; 105) to measure if the observed distribution of metric values differs significantly from the expected distribution under the null hypothesis. In other words, we reject the null hypothesis if a metric value's distribution significantly varies depending on the chart category/officiality. We calculate

Table 4.7: Adjusted P-values for Mann-Whitney U Test Comparisons and Cliff's Delta Against Opposite Officiality

Category	**Official vs. Unofficial P-value**	**Official vs. Unofficial Cliff's δ**
AI/ML (1)	3.73e-02	0.065 (NG)
Database (2)	NS	-
Integration (3)	NS	-
Monitoring (4)	3.73e-02	-0.061 (NG)
Networking (5)	NS	-
Security (6)	4.80e-05	-0.102 (NG)
Storage (7)	5.23e-03	-0.104 (NG)
Streaming (8)	3.73e-02	0.082 (NG)
Aggregated	5.09e-03	-0.030 (NG)

Cramer's V effect size (0-1) to measure the practical impact of the differences if Chi-square tests indicate statistical significance (64; 105).

Table 4.8 depicts the impact of chart category on individual CVSS metrics in terms of Chi2 statistics and Cramer's effect sizes. We observe that all metrics except for Scope (S) exhibit significance with p-values equal to 0. Table 4.9 depicts the impact of chart officiality on individual CVSS metrics in terms of Chi2 statistics and Cramer's effect sizes. We observe that all metrics except for Attack Complexity (AC) and Scope (S) show p-values equal to or less than 0.006 (p-value < 0.05).

We reuse the definition of effect size from previous work (3): A Cramer's V effect size above 0.05 is considered a weak association, while a value above 0.3 is considered a strong association. While the majority of metrics do exhibit statistical significance under the two contextual factors, Cramer's V values for both tables generally indicate weak associations (< 0.05). Notably, the Attack Vector (AV) and the Privileges Required (PR) metrics show the largest effect size when charts are aggregated by category (0.027

Table 4.8: CVSS v3 Severity Metrics (By Category)

Metric	Cramer	Chi2	P-value
AV	0.027	1703.365	0.000
AC	0.011	86.668	0.000
PR	0.026	1075.095	0.000
UI	0.008	43.420	0.000
S	-	10.720	NS
C	0.011	171.804	0.000
I	0.008	95.849	0.000
A	0.015	359.417	0.000

Table 4.9: CVSS v3 Severity Metrics (By Officiality)

Metric	Cramer	Chi2	P-value
AV	0.004	12.576	0.006
AC	-	0.089	NS
PR	0.006	32.722	0.000
UI	0.008	51.200	0.000
S	-	0.0318	NS
C	0.008	46.666	0.000
I	0.007	40.911	0.000
A	0.005	18.492	0.000

and 0.026), though we consider that such a weak association would still be impractical to use from a practitioner's standpoint.

In summary, our analysis rejects null hypothesis 2 for seven of the eight CVSS metrics. Although our analysis has unveiled statistically significant relationships, the weak strength of these associations suggests that there exist unexplored factors that contributing to the concerning CVEs regardless of chart category and officiality. In RQ1, we find that high-to-critical CVEs widely exist in today's Helm Charts, while in this RQ, we find that the distribution does not vary across categories and officiality, suggesting that users may lack reliable measures to avoid vulnerable charts through filtering measures on Artifact Hub.

4.3.3 Cross-Referencing RQ1 and Remaining Questions:

As we uncover the distributions of the severity of unfixed CVEs in the charts, we find surprisingly negligible CVE severity differences across the two contextual factors. Notably, officiality is often brought up in previous empirical studies when investigating the Docker image ecosystem. Previous work (118) studies vulnerabilities in the Docker ecosystem and finds that the official Docker Hub images are the most secure regarding the number of high-to-critical vulnerabilities. Another work (61) on Docker Compose also considers the officiality of images as evidence that could attract more usage.

However, our findings from the analysis of CVSS metrics, categories, and officiality, along with the observed CVE severity distributions, point towards a complex landscape of security vulnerability mitigation in Helm Charts that differs from the Docker ecosystem. The negligible difference in CVE severity distributions across categories and officiality suggests that the mitigation strategies currently adopted may not be fully addressing the risks of vulnerabilities. Even though such a finding would benefit chart users as it suggests users should always exercise caution when finding Helm Charts to use, we find that it leads to the following intriguing question: "Why are official charts not safer than unofficial charts in terms of CVE severity?".

In RQ1, we find that charts often deploy multiple dependencies maintained by multiple parties, likely reducing the officiality effect and leading to practically insignificant effect sizes. Given that such quantitative data does not suggest concrete rationales, we leave the open questions to be answered through the RQ3 qualitative findings.

Summary of Research Question 2

Despite the p-values signifying statistically significant associations between CVSS score and most CVSS metrics regarding chart category and officiality, the strength of these relationships, as conveyed by Cliff's Delta and Cramer's V values, is indeed weak in practice. We find a nuanced relationship beyond the apparent characteristics of currently available contextual factors. Practitioners should not solely rely on contextual factors to evaluate whether a chart is secure, as our findings suggest differences are practically negligible when category and officiality vary.

4.4 What strategies do Helm Chart maintainers employ to mitigate fixable CVEs? (RQ3 Results)

Table 4.10 presents the 12 mitigation strategies extracted through open coding iterations from the 90 (58 official and 32 unofficial) Helm Chart maintenance repositories. Through the iterative axial coding phase, we assign the strategies into three distinct categories: "Ad-hoc", "Automated", and "Informative." Next, we present and discuss the empirical observations of the mitigation strategies in a top-down fashion. We first introduce the categories of mitigation strategies assigned via axial coding. Then we discuss the individual mitigation strategies through our observations, backed by a real-life example and a discussion around the maintainer stance over the strategy. The quoted examples are slightly modified to remove real usernames and correct common typos. We present links to quoted examples as a table in Appendix A.2.

Table 4.10: CVE Mitigation Strategies in 90 Helm Chart Repositories (58 official, 32 unofficial)

Category	Strategy	Official	Unofficial
Ad-hoc	[S01] Manual version bump on CVE report	20 (34%)	11 (34%)
	[S02] Patch application configuration	10 (17%)	3 (9%)
	[S03] Switch to another image variant	5 (9%)	1 (3%)
	[S04] Customize base OS image	3 (5%)	3 (9%)
	[S05] Minimize functionality for attack surface	3 (5%)	2 (6%)
	[S06] Enforce use of image digest instead of tag	0 (0%)	1 (3%)
Automated	[S07] Always release latest dependencies	17 (29%)	12 (38%)
	[S08] Utilize CVE scanners	0 (0%)	2 (6%)
	[S09] Vulnerability scanner as a component	0 (0%)	1 (3%)
Informative	[S10] Define security policy on CVE handling	17 (29%)	5 (16%)
	[S11] Meaningful security changelogs	6 (10%)	5 (16%)
	[S12] Provide CVE mitigation notes	2 (3%)	0 (0%)

4.4.1 Ad-hoc Strategies:

We define an *ad-hoc* strategy as one action that requires significant manual work in order to prevent, detect and mitigate the impact of CVEs. We mine a total of 6 ad-hoc strategies applied in 36 chart maintenance repositories.

[S01] Manual version bump on CVE report: This strategy is a reactive action, which involves a typical step of manually incrementing the chart dependency/sub-chart version to a safe dependency version when a stakeholder identifies a known CVE in the chart. This strategy is applied in 20 (34%) official and 11 (34%) unofficial chart repositories.

Observation: We find this strategy as the most commonly used (31 cases) among all strategies. We observe 2 cases where the initially opened pull request is insufficient due to an impacted dependency being referenced multiple times in a chart. Specifically, we find that such a strategy entirely depends on the availability of CVE reports and the availability of maintainers. In addition, maintainers cannot guarantee the timeliness of CVE mitigation because the reports, in many cases, are created by an end user of the already released charts. High utilization of this strategy supports our finding in RQ1, where the median CVE mitigation time exceeds 100 days. In 3 cases, we find that maintainers conduct a manual investigation to determine if a CVE is truly impacting the charts before executing the bump up. [Q01] demonstrates a typical case where a chart user reports vulnerabilities to the maintainers and asks for a bump up and re-build.

> [Q01] *Bugs: GHSA-vvpx-j8f3-3w6h (CVE-2022-41723) & CVE-2022-28391 exists in the current prometheus image version being used. The bugs have been resolved in prometheus docker image version v2.43.0. The prometheus chart should be re-built with this newer docker image...*

[S02] Patch application configuration: This strategy involves actions to manually patch the vulnerable deployment configurations, once identified. This strategy is applied in 10 (17%) official and 3 (9%) unofficial chart repositories.

Observation: Throughout the analysis, we find that the most frequently (4 cases) patched CVE in this strategy is the Log4Shell vulnerability[14]. Specifically, we find that this strategy not only appears as a reactive measure to user reports but also appears as a part of regular security hardening activity, which requires dedicated knowledge of the chart dependencies and typically involves multiple rounds of conversation before a patch is merged. [Q02] demonstrates a typical case where a chart maintainer patches a CVE by specifying a new container configuration.

> [Q02] *Add missing NET_RAW capability to the system-probe container. This capability is now required with CRIO due to a patch remediation for the CVE-2020-14386...*

[14] https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[S03] Switch to another image variant: We consider this strategy as a proactive action to reduce potential CVEs. It is applied when a maintainer finds that an existing image used by the chart could be replaced entirely by another variant, which is considered less vulnerable to CVEs. This strategy is applied in 5 (9%) official and 1 (3%) unofficial chart repositories.

Observation: We find this strategy particularly useful when maintainers find an image variant that is frequently vulnerable to OS-level package CVEs but is not used by the actual application but shipped by the default OS image. A part of such vulnerabilities, although unfixed, are considered less exploitable, given that they cannot be easily triggered. As a result, such CVEs receive patches from different OS distributions under diverse priority[15]. We find that there are two types of changes involved in this strategy. First, the maintainer switches to an equivalent dependency provided by a third party that is free of the detected vulnerabilities. Another possibility is to use a safer variant of the same image, for example, changing the image tag from Debian to Ubuntu.

We observe that the first method of this strategy is not always well-received by maintainers, given that the difference in maintenance of the images and default configurations could introduce incompatibility. [Q03] demonstrates a typical case where the maintainer indicates that a switch to an equivalent image is not a good idea but rather prefers the contributor to update the dependency upstream.

> [Q03] *Contributor: This PR overrides elasticsearch-exporter tag with latest releases available in docker, whereas main repo is not updated in few months. Maintainer reply: I don't think it's a good idea to change to the bitnami image by default, if you want an update, add it upstream.*

[15] linkedin.com/posts/danlorenc_cve-2023-38408-activity-7088845085701742592-L7rw

[S04] Customize base OS image: This strategy, similar to S03, is a proactive action to mitigate OS-level CVEs. The typical step involves migrating a dependency (application) onto a maintainer-customized OS image that is free of potential CVEs. This strategy is applied in 3 (5%) official and 3 (9%) unofficial chart repositories.

Observation: While it is safe in most situations to simply update and incorporate the CVE fixes from an upstream OS distribution that frequently receives CVE patches (71), in 2 cases, we find that maintainers manually remove and use alternative packages in order to avoid frequently vulnerable packages like "Curl". This strategy requires building images "in-house", which demands skills beyond simply switching an image tag to another variant like S03.

Particularly, in 3 cases, we find that contributors suggest rebuilding the OS images to a "distroless" variant, which is a set of language-specific container images that are reduced to contain bare minimum dependencies (Google). However, one maintainer suggests that the use of distroless images is not always feasible because some applications require different system packages to function. Our observation suggests that this is a manual approach that requires caution since changing the base OS image of a chart dependency could lead to incompatibility and breakage.

Nonetheless, we find that "minimal" images (such as distroless) and their adoption have yet to be extensively studied in academia and are not widely used in the industry either. One recent study (54) conducted a correlation analysis between the number of vulnerabilities and drew a conclusion that distroless images, in several cases, own more critical CVEs than their full-size counterpart. A previous empirical study (54) suggests that a smaller image size often leads to a higher number of severe vulnerabilities. Future research could focus on analyzing the adoption of such images in the domain and draw empirical understandings of the practical trade-offs of using such images. [Q04] demonstrates an example that expresses the trade-offs to consider when adopting distroless images.

[Q04] *We are aware of the distroless initiative as well as its pros and cons, in fact, some of the Bitnami containers are distroless. But not always this is an alternative since some applications need different system packages to work so it is needed an underlying distro.*

[S05] Minimize functionality for attack surface: This strategy is a proactive action that involves a typical step of manually evaluating the deployment definition of chart containers and their exposed functionalities to external network such as root access. This strategy is applied in 3 (5%) official and 2 (6%) unofficial chart repositories.

Observation: Previous work (74) defines attack surface as the proportion of resources that is used for an attack on the system (81). By doing a manual evaluation, maintainers use their expertise to remove unnecessary permissions and access given to containers and thus exposes a smaller attack surface to vulnerabilities. We find that this strategy is well-received by chart maintainers and is in alignment with the security best practices discovered in previous empirical studies on Kubernetes (104). However, we find that such security hardening requires extensive knowledge of dependency configuration and mechanisms that often maintained by multiple parties. Only in 1 case did the maintainers mention that they would spend time proactively evaluating each chart to enforce a minimal attack surface. The requirement for significant manual effort and expertise likely contributes to the low occurrence of this strategy in the maintenance repositories. [Q05] demonstrates an example of changing the security context to minimize attack surface for vulnerabilities.

[Q05] *Reducing kernel capabilities available to a container limits its attack surface containers[] .securityContext .readOnlyRootFilesystem == true*

[S06] Enforce the use of image digest instead of tag: This is a special strategy that is only discussed and executed as a standard in one maintenance repository, though we observe multiple repositories supplying image digests along with image tags without specifying its a CVE mitigation strategy. Therefore, we consider this strategy to be applied in 0 (0%) official and 1 (3%) unofficial chart repositories.

Observation: Image digest is a SHA256 hash representation of the layers in a container image. Previous work (84) finds that unpinned dependencies are among the core reasons for supply chain attacks. In the context of containerization, images could provide multiple digests (builds) even if they share the same image tag. We observe that this problem is common when chart maintainers only pin dependencies based on image tags. However, users could receive vulnerable images (built on the same tag) even if the pre-release CVE check against that tag is passed. Carefully engineered attacks could utilize this characteristic and bring critical security consequences. [Q06] demonstrates an action (pull request) to pin a chart dependency by its digest to ensure that a consistent image is used in every deployment.

[Q06] *Using a digest ensures the image being pulled has not been compromised and allows testing release against images without having to tag them.*

4.4.2 Automated Strategies:

We define an *automated* strategy as an action that does not require manual participation beyond the review of pull requests (to approve the actions). Such strategies are inherently automated and can operate on their own to prevent, detect and mitigate the impact of CVEs. We mine a total of 3 automated strategies applied to 29 chart maintenance repositories.

[S07] Always release latest dependencies: This strategy is a proactive action that typically involves automated pipelines that periodically bump up the chart dependencies. This strategy is applied in 17 (29%) official and 12 (38%) unofficial chart repositories.

Observation: This strategy, in a majority of cases, is powered by open-source tools such as Renovate (RenovateBot). Previous empirical work (80) finds that repositories with automated pull requests, although benefiting from a quicker merge of updates, often suffer from other downsides such as accidentally introduced bug and performance regressions from the newer versions. In our case, we find that a large number of automated pull requests simply mention the versions to update but fail to identify the reasons to update (such as fixing the particular CVEs and the impact radius of CVE).

We find such automated pipelines to run in a fully-automatic manner in 13 repositories (4 official, 9 unofficial), meaning a commit is submitted to the main branch without needing maintainer approval. In the remaining cases, we observe that this process requires manual intervention (3 cases), given a bump-up can yield a defective deployment due to version incompatibility among the dependencies. However, we find that using such automated tools does not prevent all fixable CVEs. In fact, we find that among the 20 automated maintenance repositories with a security report, high-to-critical fixable CVEs exist in 18 repositories (90%). In an extreme case, one chart artifact is exposed to 713 fixable CVEs, indicating that always releasing the latest dependencies cannot actually fix all fixable CVEs. Such a phenomenon is due to the fact that chart dependencies could be abandoned or rarely maintained and therefore accumulate large number of CVEs overtime. [Q07] demonstrates an example of automatic bump-up requiring human intervention.

[Q07] *Thanks for letting us know, the new version was detected by our automated test & release pipeline but we found some issues when packaging this new version and we need to manually fix it. At this moment the team is working on it. We will update this thread once we released the new version.*

[S08] Utilize CVE scanners: This strategy is a proactive action that could involve an automated check before the merge event of pull requests, upon chart releases and/or on a scheduled timer that is executed in CI/CD pipelines. This strategy is applied in 0 (0%) official and 2 (6%) unofficial chart repositories.

Observation: We find surprisingly scarce evidence of running CVE scanners in chart maintenance repositories. At first glance, it seems counter-intuitive given their obvious efficiency advantages (22; 89) and their wide adoption in research studies (106; 127). However, our observation identified two major concerns that account for this reluctance: the issue of false positives and the need for manual handling, which, in combination, result in a low adoption rate.

In cases where CVE scanners are discussed, we observe that one repository publishes explicit security guidelines against reporting raw CVE scanning results, with several maintainers indicating the problems of false positives (Q08). Interestingly, we find that the term “false positives" sometimes means that the detected CVE will impact the chart deployment, unlike the claims in previous work (80). Rather, chart maintainers indicate that such CVEs are considered false positives because they are out of maintainers’ responsibility and should be handled upstream by the dependency maintainers.

However, we find it noteworthy that, regardless of the negative stance from the maintainers' perspective, chart users frequently report CVEs with such scanners. Through the open coding phase, we find CVE reports and fixes proposed based on the results of 7 different scanners, indicating that end users of charts relied heavily on such automation before deployment. We find two maintainers to choose to directly reject such CVE reports and instead ask the chart user to propagate the mitigation request upstream (to the dependency sources) even when both sources are under the same owner (official charts). Quoted example [Q08] shows a typical example of false positives via image scanners, where the mentioned Log4Shell CVE has been addressed by patching application configuration (Strategy S02).

> [Q08] *Thanks for a quick response on this, I guess I missed out on this release note, appreciate sharing this. Also we are looking forward for 7.16.2 to mitigate the false positive too from scan results.*

[S09] Vulnerability scanner as a component: This is a special strategy that comes into effect in the post-deployment phase of the charts. This strategy involves deploying a CVE scanner such as Trivy (aquasecurity) into the run-time of the Kubernetes cluster in order to detect new CVEs being disclosed in public vulnerability databases such as the National Vulnerability Database (NVD). This strategy is applied in 0 (0%) official and 1 (3%) unofficial chart repositories.

Observation: We cannot draw insights based on the maintainer stance due to the scarcity of discussion around this strategy. To provide alternative empirical insights, we conducted an ad-hoc literature (grey and academic) search regarding such practices and their adoption in the cloud computing domain. As a result, we find that the concept and tools around continuous vulnerability scanning is emerging in the industry with over 10 mentions of the concept “continuous scanning". Yet, empirical evidence around its adoption, effect, and impact remain unexplored in academia, we find only one study in 2017 (16) that proposes a novel architecture for continuously securing containers in a deployment. Given that critical vulnerabilities such as Log4Shell could be disclosed at an unknown point in time, any existing cluster that previously passed scanning in the CI/CD pipeline could still be impacted. We find such a continuous scanning strategy to be an essential complementary to the existing pre-deployment CVE mitigation strategies. [Q09] demonstrates a document explaining the role of the image scanner in a chart deployment.

> [Q09] *The included trivy scans the running workload in your cluster for CVEs and creates Custom Resources to present the results.*

4.4.3 Informative Strategies:

We define an *informative* strategy as one action that is purely informative with the purpose of informing practitioners on the CVEs (mitigation and awareness). The informative strategies contain important information that guides practitioners through the changes related to CVE mitigation and the channels to discuss and submit CVE reports. We mine a total of 3 informative strategies applied to 27 chart maintenance repositories.

[S10] Define security policy on CVE handling: This strategy is an informative and proactive action, which involves presenting information about how to report, disclose and find information on user-impacting CVEs. This strategy is applied in 17 (29%) official and 5 (16%) unofficial chart repositories.

Observation: Unfortunately, although a considerable number of repositories contain a "SECURITY.md", which is an automatically parsed file to populate the "Security" tab of GitHub, we find only 2 security policies providing practical information relevant to Helm Charts. Specifically, among the 22 repositories with such security policy, 20 are directly referencing an organization-wide policy that is not tailored to the Helm Charts. We find such security policies to be generally less informative and to focus solely on the core dependency of the chart (owned by the organization). Example Q2 demonstrates an informative security policy on "Open CVEs" in the Bitnami organization, explaining why all the remaining unfixed CVEs in charts are actually unfixable.

> [Q10] *Open CVEs are the ones that have not been fixed by the Linux Distribution maintainers because they did not work on that yet or they do not consider a critical issue. Bitnami is not able to fix those CVEs since those fixes depend directly on the distribution maintainers.*
> *Is Bitnami Releasing Charts Or Containers That Include CVEs?*
> *All Bitnami containers and Helm charts do not include fixable CVEs.*

[S11] Meaningful security changelogs: This strategy is an informative and reactive action, which involves presenting information around the security changes such as CVE-IDs, impact and suggestions for update. This strategy is applied in 6 (10%) official and 5 (16%) unofficial chart repositories.

Observation: In a majority of cases, we find that chart maintenance repositories do not keep meaningful changelogs that indicates the changes since last chart release. Through our open coding phase, we find that such a phenomenon is mainly attributed to two situations: (1) unawareness of tools (S07) that could help formulate changelogs; (2) the frequent releasing nature of charts implies significant manual effort to keep track of all changes.

In particular, Artifact Hub supports a special annotation to record the changelogs: "annotations: artifacthub.io/changes". Given the low adoption of meaningful changelogs in general, we find it not surprising to extract only 11 repositories among 90 mentioning CVE fixes, while less than 5 repositories use such Artifact Hub features to disclose CVEs. Currently, without meaningful changelogs, users are forced to use the charts without any knowledge of the changes, given that there is no method other than reading all commits and pull requests in the maintenance repository to understand the actual changes. [Q11] shows a real-life usage of the changelog annotation to inform chart users of specific changes in a new chart version.

```
[Q11]
artifacthub.io/changes: |
- kind: security
  description: 'loki-stack: update chart from 2.9.9 to 2.9.10'
  links:
    - name: 'fix: update bats image for CVE-2019-14697'
      url: Commit Hash (Omitted due to length)
```

[S12] Provide CVE mitigation notes: This strategy is an informative and reactive action that provides detailed mitigation methods for particular CVEs discovered in a chart. This strategy is applied in 2 (3%) official and 0 (0%) unofficial chart repositories.

Observation: We find this method to be applied rarely. In two cases, it is used for the well-known Log4Shell vulnerability. One of the now-archived alternatives to Artifact Hub encourages such a strategy by defining a specification named "Security Mitigation Notes"[16]. Figure 4.8 depicts an example of implementing the specification through a file that ships along with the Helm Chart artifact. The specification involves manual

[16] https://jfrog.com/blog/helm-chart-security-mitigation-in-chartcenter/

notes for mitigation strategies and the impact radius of a CVE (version range and package).

We find that such a strategy could provide chart users with confidence when deploying a third-party chart, although this could be a tedious process for chart maintainers in practice and thus receive a low utilization like other ad-hoc strategies. On the other hand, we find such mitigation notes to be a crucial complement to strategy S09 by clarifying the truly impacting CVEs. Future studies could explore the historical adoption rate of such specifications and potentially propose automation to generate such strategies using information obtained from the National Vulnerability Database (NVD) and/or through static/dynamic analysis of the deployed resources/configuration by a Helm Chart. [Q12] demonstrates a quote that explains the reasoning behind the "Security Mitigation Notes" specification.

> [Q12] *This enables Helm chart authors to annotate their charts with notes about the CVEs flagged by their dependencies, letting users know whether and when to be concerned, or if they can mitigate the risk. It helps you to say in response to a CVE, "Yes, but..." and engage in a type of dialogue with the users of your chart.*

4.4.4 Cross-referencing the Strategies: Dependency Bump-up is Not All We Need

We find that a simple bump-up for chart dependencies is a common strategy to ease the manual work of following upstream releases, yet such a strategy alone does not secure Helm Charts. We find multiple strategies that could complement automation to further secure the released chart artifacts.

```yaml
schemaVersion: v1
summary: >
  Security mitigation information for this application is tracked
  by the security-mitigation.yaml file that's part of this Helm chart.
mitigations:
  - cves:
      - CVE-2019-11888
    affectedPackageUri: docker://docker.io/rimusz/security-sample-app
    affectedVersions: <= 0.1.1
    description: >
      This CVE is applicable for applications running on Windows.
      This application currently doesn't support Windows OS, hence this CVE
          does not impact our users.
```

Figure 4.8: A Sample Security Mitigation Note from JFrog

Always releasing the latest dependencies (S07) prevents and mitigates the impact of vulnerabilities as they can operate automatically with minimal human intervention to eliminate many fixable CVEs. Yet, overall, we observe fewer than 50% of the studied chart maintenance repositories using such strategies. Previous work stresses the importance of adopting automation in the domain of DevSecOps (66; 89; 112), yet the studies find evidence through a survey that existing automation is little known to practitioners due to a lack of security-related knowledge and tool standards. We find this phenomenon relevant in the Helm Charts domain, as we observe multiple cases where maintainers are unaware of automation that could assist in maintaining up-to-date dependencies. On the other hand, in S07, we also find 3 cases where automatic bump-ups lead to incompatibility and functional issues, indicating such a strategy is not a "silver bullet".

Nevertheless, our analysis for S07 reveals that many high-to-critical fixable CVEs remain unfixed in those charts that always track the latest dependency versions. In most cases, the chart maintainers do not directly build the dependency images. As a result, waiting (with automation) for potential upstream CVE mitigation becomes the trivial and passive path that a majority of chart maintainers choose to mitigate CVEs. Such observation suggests strong evidence that we need more than simply relying on automation to secure Helm Charts. We also find that automated version bump-ups led to incompatibility issues due to dependencies that could not work with the newer versions, which leads to maintainer overhead (94). This phenomenon indicates that not all vulnerabilities can be addressed through trivial automated fixes, especially when the images are not under the Helm Chart maintainers' direct control. This creates a dilemma of unclear responsibility in the software supply chain.

Throughout the analysis, we find that informational and ad-hoc strategies serve as a necessary complement to automated strategies to secure Helm Charts. We found that image scanners are often prone to false positives and thus work best if chart maintainers provide additional information, such as mitigation notes and changelogs, to convey effective CVE fix news to users. On the other hand, manual work, such as switching to another image variant (S03) and rebuilding the OS image (S04), could proactively mitigate future CVEs originating from the vulnerable images built upstream.

4.4.5 Formulating the Grounded Theory: Incentive, Trade-off and Trust

Figure 4.9 shows the overall flow of our grounded theory derivation process. After mining the 12 CVE mitigation strategies and their categorization into three distinct categories (i.e., Ad-hoc, Automated and Informative), we conduct selective coding to draw commonalities from the strategies before formulating the final theory.

While it is evident that the core subject from the studied data is "CVE mitigation strategies in Helm Chart maintenance repositories", we identified three driving factors

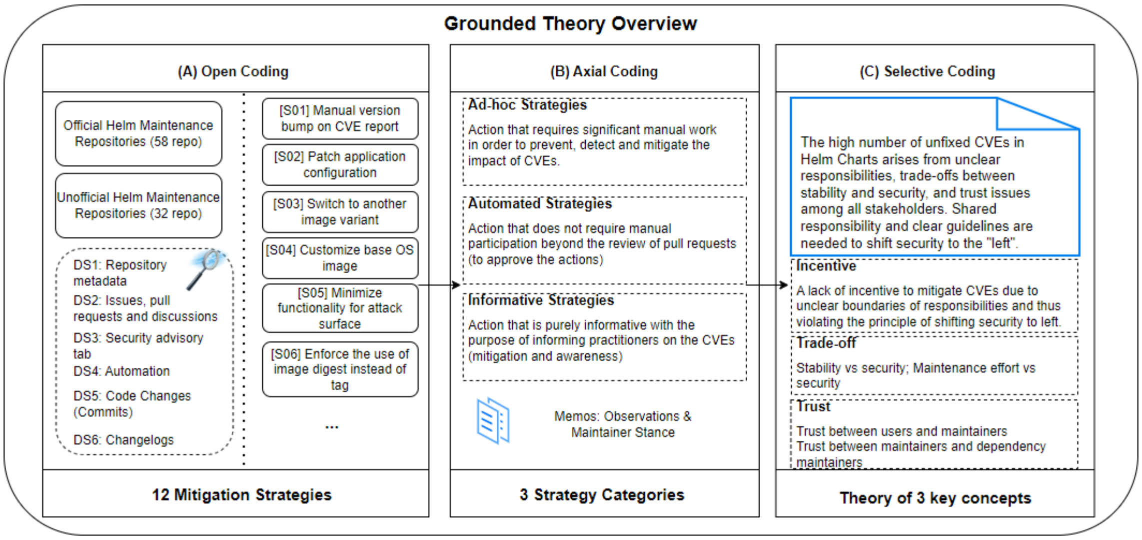

Figure 4.9: Grounded Theory Results

that fundamentally relate to the high prevalence of fixable yet unfixed CVEs: Incentive, Trade-off and Trust, which emerged as central keywords interconnecting most of the observed maintainer stances and strategy observations. We formulate the grounded theory as following: The high number of unfixed CVEs in Helm Charts arises from unclear responsibilities, trade-offs between stability and security, and trust issues among all stakeholders. Shared responsibility and clear guidelines are needed to shift security to the "left".

We find a lack of incentive to mitigate CVEs due to unclear boundaries of responsibilities and thus violating the principle of shifting security to the left.

Despite the fact that it is possible to maintain charts that are free of fixable CVEs (Q10), we find the lack of incentive to be the driving factor why such mitigation strategies and standards are not widely enforced. Other than the easily fixable CVEs by bumping up to the latest dependency versions, many remain unfixed because the dependencies fail to mitigate them. As the final software artifact before deployment, Helm

Charts suffer from an unclear boundary of who should mitigate the CVEs. We find multiple statements from chart maintainers expressing that they are not responsible for addressing the reported CVEs and that the CVEs should be reported upstream instead.

Previous studies in DevSecOps (75; 82) suggest that “Shifting security to the left” and “Shared responsibility” are attracting increasing attention from both academia and the industry. Both principles aim to preemptively mitigate security concerns such as CVEs as early as possible in the software supply chain, with a collective approach involving all stakeholders, which minimizes risk and ensures a more secure end product.

Shifting security to the left: Currently, to mitigate fixable CVEs that remain unfixed in dependencies, maintainers rely on rebuilding and swapping out outdated packages for their safer equivalents. However, given the observed stance, we find that maintainers are generally reluctant to do so due to the manual effort required. Previous work (67) supports our observation through interviews, where the authors find that most developers are unaware of security practices in software maintenance and often consider security-hardening activity an extra overhead. As a result, we find that maintainers suggest users to report CVEs upstream, i.e., to the maintenance repository of dependencies (images and sub-charts). Example [Q13] conveys a strong message regarding the delegation of CVE mitigation to dependencies.

> [Q13] *It doesn’t make any sense to provide these reports for helm chart. Please forward this to identified projects.*

At first glance, such delegation indeed aligns with the principle of “shifting security left” as it assigns the responsibility of mitigating CVEs to an earlier stage in the software development process. However, in RQ1, we find that Helm Charts often involve multiple images (owners), each with unique maintenance practices and security policies. While the delegation of CVE mitigation might be seen as a shift towards "left" within the scope of individual components, it inevitably leads to delays and uncertainties in addressing vulnerabilities from the chart users' perspective given upstream dependencies may not release a timely patch. We consider it a passive strategy that relies on the assumption that upstream will mitigate the vulnerabilities in a timely manner. In RQ1, we also find a high prevalence of fixable CVEs in the released chart artifacts; thus, in real-life settings, the delegation of security responsibility generally results in chart users having to identify, understand and mitigate the unfixed CVEs, essentially "shifting right" in the overall process.

Such delays emphasize the necessity of applying the "shared responsibility" model (75; 112) in the Helm Chart domain, where all stakeholders, including chart maintainers, should proactively participate in securing the charts rather than passing the responsibility upstream. While ad-hoc strategies such as S02, S04, and S05 could require extensive knowledge and effort to implement, we find informative strategies generally trivial to achieve as a side-task before each release. Previous empirical studies (72) find CVE patching to be frequently delayed in Docker images due to a lack of incentive and the complex composition nature of images. Another empirical study (124) finds that attackers often aim at the weakest dependencies to attack, which effectively emphasizes a need to secure all dependencies in a chart.

To understand if shifting security to the left by sharing responsibility is indeed achievable, we estimate the potential impact of incorporating S04 by simply bumping up

OS images without sophisticated package modifications. We choose S04 as we discover among the total instance of impacted packages that 731,628 (87.4%) belong to OS packages that have an actively maintained upstream distribution such as Ubuntu (71), meaning a simple bump-up of the base OS image will likely mitigate a large number of CVEs. Such a simple bump-up may not be sufficient to eliminate all CVEs as removing vulnerable packages would require further image modification. We believe that such a strategy could still mitigate CVEs that were introduced by outdated upstream images.

For this purpose, we investigate the repositories that adopt the "Customize base OS image" strategy (S04) to understand if the complexity of adoption is beyond maintainers' reach. We find 2 representative repositories among the studied 90 maintenance repositories that operate in a fully automated manner. The chart maintainers set up the initial Dockerfiles and could therefore rely on existing automation mentioned in (S07) to update the Ubuntu base images to rebuild chart dependencies. We find such effort to be generally achievable as we only find 7 minor manual changes to adjust minor functionalities among over 1,000 automated commits spanning the year of 2022-2023[17]. In addition, given that all studied chart repositories are hosted on GitHub, we consider the CI resources not among the constraints as it is free-of-charge for public repositories[18].

We find the adoption of CVE mitigation strategies subject to trade-offs. Among the three major categories of CVE mitigation strategies, we observe that no strategy can eliminate all of the fixable CVEs, and most of the strategies receive a mixture of maintainer stances giving the trade-offs between security, resources (time, effort etc.) and non-functional requirements such as stability and compatibility (35). We consider

[17] https://github.com/bitnami/containers/blob/main/bitnami/zookeeper/3.7/debian-11/Dockerfile

[18] https://github.com/features/actions

such trade-offs crucial in the context of Helm Charts since the deployments often scale across large distributed systems like service mesh (70).

Stability vs security: Within two strategies (S03, S04), we find that maintainers favour the stability of Docker images over their potential vulnerability, citing the need to ensure the correct functioning of applications. Previous empirical work (126) finds that Docker images often are outdated in terms of vulnerability and bugs, whereas stable OS images are consistently more vulnerable. Such trade-offs are inevitable whereas security patches often come within newer versions of images. As a result, it is understandable that current charts could be exposed to fixable CVEs that are only patched in a non-stable base image, which requires time to propagate to downstream projects such as Helm Charts. We consider this preference for stability over their potential vulnerability a paradoxical scenario where the very act of ensuring application functionality (via stable images) may expose the application to security risks due to latent vulnerabilities in these dependencies.

Example Q14 shows a statement around such trade-offs between security and stability from both the chart users' and chart maintainers' points of view.

[Q14] *User: Therefore, it would be great if we could do a new 2.8.27 hotfix which simply has no functional changes but pulls in Debian updates.*
Maintainer: I am sorry, but we only release containers for the stable version of the asset.

Maintenance effort vs security: The inherent complexity and rapidly changing nature of Helm charts and their dependencies make proactive strategies challenging to implement and maintain. We consider that such a trade-off originates from a lack of resources and/or expertise within maintenance teams to proactively identify and mitigate vulnerabilities. This trade-off is also magnified by the lack of incentive discussed in the previous section. During the open coding phase, we find that ad-hoc manual inspection and configuration fix-ups cannot be replaced by current automation strategies such as automated bump-ups. As we find CVEs of high-to-critical severity to widely exist in automated chart maintenance repositories, such manual efforts are generally under-utilized during maintenance and contribute significantly to the high prevalence of unfixed CVEs.

We find that Helm Chart maintenance and distribution predominantly operate on a chain of vulnerable trust relationships.

Our analysis of CVE mitigation strategies reveals that Helm chart maintenance activities centre around trust, which indicates a trust for software that is not written by the users (downstream charts, end user of charts) (110; 117). We find that most CVE mitigation strategies are adopted in minimal repositories; in many instances, we find CVE fixes to be ad-hoc, and issue reports become stale due to a lack of prompt response from maintainers. While reducing maintenance effort, this trust-based style evidently contributes to the high number of unfixed CVEs in the charts and higher security risks when deployed into production environments.

Trust between chart users and chart maintainers: We observe that the adoption of CVE scanners (S08) is a contentious subject due to a divergent stance. While users commonly rely on these scanners to identify potential vulnerabilities, maintainers frequently dismiss the generated CVE reports, citing issues such as false positives and increased maintenance effort. As a result, we find large number of CVE reports from scanners perpetuating a negative feedback loop that worsen the maintainers' skepticism in the CVE scanning results.

We find that this negative feedback loop creates a security dilemma where the trust over CVE scanners diminishes over time as more false positives are reported that frustrate the maintainers and worsen the relationship between users and maintainers. Previous empirical work also points out that false positives are among the core reasons that render automated pull requests less useful (80). Evidently, we find that security policies (S10) that clearly define the responsibilities and expectations of chart maintainers and users could ease such fatigue.

We also find that despite the trust issues of CVE scanners, the lack of well-defined security policies (S10), changelogs, and mitigation notes (S11, S12) leave chart users uninformed of the impact of vulnerabilities. As such, chart users have no choice but to use charts without any security assurances, which often leads to post-hoc CVE detection and mitigation. Given some mitigation strategies, such as patching application configuration (S02), require advanced knowledge of the dependency, a typical end user of Helm Chart may not have the knowledge to mitigate CVEs in time. This discrepancy leads to a situation where users that do not have sufficient skills to run comprehensive security evaluation have to rely on maintainers to ensure chart security. Given the reluctance to reinforce security at the last line of defence before deployment, it could cripple the security of Helm Charts as they inherently rely on multiple images that are not owned by the chart maintainers.

Trust between chart maintainers and dependency maintainers:

Another trust relationship we uncovered is between the chart maintainers and the maintainers of upstream dependencies. In RQ1, we find charts often compose containers that involves numerous packages and images. As such, in our qualitative observation, chart maintainers typically delegate the responsibility of CVE mitigation to the maintainers of these upstream dependencies.

However, this delegation of trust is fragile. Previous works (41; 124) have highlighted how vulnerabilities can propagate from upstream dependencies, leading to security risks (supply chain attacks). It underpins the risks of this trust relationship, where vulnerabilities in a single weak point (image) can propagate downstream and risk the entire Helm chart to be vulnerable. Our grounded theory, therefore, emphasize that the trust dynamics between chart maintainers and dependency maintainers should be enhanced by a shared responsibility model introduced in the "Incentive" discussion. This step would strengthen the weak points as chart maintainers provide the quality assurance for potentially under-maintained chart dependencies before the vulnerabilities could reach user environments.

4.4.6 Validating the Theory with Remaining Questions from RQ1 and RQ2

In RQ1 and RQ2, we investigate characteristics and contextual factors that relate to the prevalence of unfixed CVEs in the Helm Chart domain. However, two questions emerged and remained unresolved when we uncovered the quantitative results. With the convergence of concepts in CVE mitigation strategies, we find concrete evidence that could explain such questions and therefore prove the validity of our theory; below section discuss the answer to each question based on the emerged theory:

[Question1] Why are existing informative features rarely used for chart artifacts?: Such informative strategies are generally ignored on the maintenance level. Given that less than 20% of the repositories include meaningful changelogs, and over 30% rely on automation to bump up dependencies, maintainers generally lack the incentive to include meaningful changelogs unless the automation generates one.

We also find a lack of awareness of the necessity of such strategies because current chart maintenance is predominantly based on the trust between users, chart maintainers and upstream maintainers. As we find CVE mitigation to be generally ad-hoc and mitigation responsibilities are often delegated upstream, it becomes even more effort-demanding for chart maintainers to keep a record of security changes as such information could be lost through the supply chain. Lastly, in this RQ, we discover a largely negative stance over the raw results of vulnerability scanners. Particularly, false positives that do not pose a real threat to charts could contribute to the low utilization of the security report feature since they could confuse end users.

[Question2] Why are official charts not safer than unofficial charts in terms of CVE severity?: We find that the under-utilization of CVE mitigation strategies does not necessarily correlate with its officiality. On the one hand, we find evidence that Q13, even within officially maintained charts, maintainers tend to delegate fixable CVEs upstream, except those that are fixable through simple version bump-ups, causing delays in CVE mitigation. On the other hand, we find two unofficial repositories that use comprehensive ad-hoc strategies to patch the security of upstream dependencies. Table 4.10 shows that other than S10, official charts likely do not employ more effective strategies than unofficial charts. We also find that charts of both officialities do not employ more advanced mitigation strategies for CVEs of higher severity beyond the 12 strategies and therefore do not lead to safer Helm Chart artifacts. Therefore, from both quantitative and qualitative perspectives, we conclude that official charts are not inherently safer than unofficial charts and should not serve as the sole criteria for security evaluation.

Summary of Research Question 3

We present a comprehensive discussion of 12 currently adopted CVE mitigation strategies from the perspectives of researchers and maintainers in 90 chart maintenance repositories. We distill our findings into a grounded theory where trust, trade-off and incentive emerge as the three key factors influencing the adoption of such strategies.

4.5 Study Implications

4.5.1 For Researchers and Helm Chart Maintainers

Researchers should investigate guidelines for shifting left and the shared responsibility model for Helm Chart maintenance While we identify that Helm Charts bear significant potential for enhancing the security of distributed systems, we find that a shift-left approach is needed in dependencies. CVEs in Helm Charts primarily originate from their dependencies, indicating that better techniques and tools are needed to ensure the safety of distributed images. Our findings in RQ3 also suggest that chart maintainers could take up some of the shared responsibility in chart security without significant additional effort, shifting the burden of security evaluation from users to the left (chart maintainers).

Researchers should investigate efficient and trust-worthy automated tools These RQ3 findings highlight a complex trade-off between effort and security in automated tools within the Helm Chart community. As the complexity of security threats continues to evolve and the need for scalability becomes pressing in cloud-native technologies such as service mesh (70), securing deployed systems from the bottom up (starting from deployment scripts and images) becomes crucial.

Future researchers and chart maintainers should focus on improving the adoption and effectiveness of automated CVE mitigation strategies, developing standard guidelines on using CVE scanners, improving their accuracy to reduce false positives, and exploring ways to automate the generation of mitigation notes and changelogs. We also suggest investigating the adoption and impact of continuous vulnerability scanning in the cloud computing domain, given its potential to complement pre-deployment CVE mitigation strategies.

Researchers and chart maintainers should work on improving security feature utilization and proactive CVE mitigation We call for more proactive and informative measures from chart maintainers. Our findings in RQ1 indicate a low utilization of security features such as security fix indicators and security reports on Artifact Hub. Given the high number of false positives generated by current vulnerability scanners, users of Helm Charts often need to rely on these scanners due to insufficient information on the vulnerability status of the Helm Chart artifacts. As users typically lack expertise in the systems deployed by Helm Charts, it becomes difficult to accurately assess the impact of detected CVEs without information from the chart maintainers. We thus recommend future researchers and chart maintainers explore ways to dynamically determine vulnerability based on the configuration supplied to a chart dependency. Security mitigation notes are noticeably absent in current Artifact Hub artifacts. We suggest the automation of CVE note detection and the subsequent generation of mitigation notes. This would not require manual effort from maintainers and could potentially be achieved with the help of large language models that dedicates to evaluate Kubernetes manifests (30; K8sGPT). Future researchers should investigate the feasibility of developing a predictive model for identifying potential vulnerabilities in Helm charts and suggest mitigation methods using the CVSS database and known mitigation strategies.

Researchers and chart maintainers should use the mitigation strategies as a checklist In RQ3, we mine a list of 12 current mitigation strategies. Future researchers can use this as a checklist to boost the security of charts by understanding the takeaway from each mitigation strategy and further mine empirical evidence by testing the mitigation strategies on individual CVEs to understand their effectiveness in different scenarios. This allows maintainers to select the suitable ones based on resource constraints and find a balance between maintenance effort and security. On the other hand, our study underscores that automated strategies play an essential role in CVE mitigation. However, they must be complemented with manual and informative strategies to address vulnerabilities effectively. When implementing these mitigation strategies in Helm Chart maintenance, current chart maintainers could use the checklist to balance the trade-offs between security and stability, resource constraints, and effort.

4.5.2 For Chart Users

Chart users should conduct a comprehensive security assessment before deploying public charts into critical systems From our statistical analysis in RQ1, we infer that vulnerabilities can be pervasive in Helm Chart deployments, even when the number of images involved is minimal. In RQ2, we find that officiality and chart category do not suggest a practical difference between the distributions of CVE severity scores and the breakdown distributions of vector metrics (exploitability and impact) in current public charts. Furthermore, in RQ3, we find that Helm Charts could still be exposed to high-to-critical vulnerabilities even with automated strategies. We advise practitioners who deploy charts into critical systems to examine configurations beyond the default values, potentially uncovering unexpected vulnerabilities not accounted for by maintainers' evaluations and scanners. This more comprehensive approach to security would offer a reliable understanding of Helm Chart security and help mitigate security risks more effectively.

4.6 Threats to Validity

Below, we discuss threats to the study validity and the methodologies we applied to minimize these threats, based on literature guidelines (119).

Construct Validity: In our study, we acknowledge a potential threat to construct validity due to the use of multiple rounds of statistical tests, such as the pair-wise Mann Whitney U test, which might lead to misinterpretation of datac (83). To mitigate this threat, we took cautious steps before conducting such tests. We first carried out normality tests, which assisted us in choosing the most suitable tests for our study following best practices as suggested in previous literature (105), aimed to enhance the accuracy of our interpretations and reduce any potential for erroneous results. Additionally, we implemented the Bonferroni correction technique to adjust our significance levels, consequently minimizing the chance of type I errors (77).

Beyond statistical significance, we also incorporated an evaluation of effect sizes to gauge the practical significance of our findings. This dual-layer analysis gave us a comprehensive understanding of the real-world implications of our results. Hence, while we rejected the null hypothesis of significant differences in RQ2, we further explored the effect sizes to draw more nuanced conclusions. Our findings showed negligible effect sizes as measured by Cliff's Delta and Cramer's V, thus reinforcing the robustness of our analysis and enhancing the construct validity of our study.

Internal Validity: In RQ3, we utilized a Grounded Theory (GT) methodology, which, despite being inherently qualitative and subjective, can provide insightful analysis if applied correctly. To address the threat to internal validity, we followed the guidelines and protocols from previous works in the empirical software engineering domain (28; 40; 58; 131). We undertook multiple iterations of coding phases before we arrived at generalized theories, which minimized the risk of interpretation bias. Given the highly technical nature of the CVE mitigation strategies, our grounded theory is less susceptible to biases, as we support every strategy we discuss with relevant quotes and literature references. Appendix A.2 includes all quoted examples and their URLs for future verification. During the theory building phase, we also cross-referenced our findings with previous work to ensure our observations are well-founded and robust.

Conclusion Validity: In the qualitative analysis, it is possible that one could generalize CVE mitigation strategies based on different aspects. In RQ3, we took several measures to reduce this threat. We iteratively conducted axial coding phases to ensure that the categorical labels faithfully and accurately represent the various CVE strategies. Although alternative aggregation of the mitigation strategies could exist, we believe our current categorization is meaningful to draw practical takeaways.

We acknowledge the possibility of an alternative path of evaluating mitigation strategies through the analysis of individual CVEs and tracing their mitigation in each development repositories, which is used by one of the recent studies of vulnerabilities in Linux distributions (71). While such a methodology is effective for studying the mitigation strategies when the scope is well-constrained (Debian vs Fedora), we find the Helm domain to own unique characteristics given the large number of Helm Charts (10,000+), unique CVEs (10,000+) and the untraceable mapping between most of the unofficial Artifact Hub charts to their maintenance repositories as introduced in Section 4.1. Thus, we sampled the repositories with stratified sampling (40) and focused on studying mitigation strategies that can help explain the prevalent existence of unfixed but fixable CVEs.

External Validity: We acknowledge that some CVE issue reports and corresponding fixes might be temporarily kept private due to a responsible disclosure policy (88). However, as a grounded theory research we have comprehensively investigated 90 GitHub chart maintenance repositories and reached theoretical saturation. This ensures that we did not encounter any new CVE mitigation strategies when exploring new repositories beyond our dataset. We believe that this analysis provides a representative understanding of the chart maintenance activities and corresponding mitigation strategies currently employed and discussed. We also verified our observations while cross-referencing with our RQ1 and RQ2 results, which reduces the threat to generalization validity.

Another potential threat to external validity is the reliance on Artifact Hub data for Helm Charts, which may not represent the complete landscape of public Helm Charts. We are aware that charts are also hosted on other platforms and private registries. However, Artifact Hub, as a Cloud Native Computing Foundation (CNCF) sandbox project, is recognized as the de-facto registry for chart artifacts. Other platforms, like JFrog ChartCenter, have recommended migrating to Artifact Hub[19] since its entrance into the CNCF. Therefore, we believe our study, while not exhaustive, provides a broad and representative perspective on Helm Chart vulnerabilities and their mitigation strategies.

4.7 Chapter Summary

This chapter presents a mixed-methods empirical study of 11,035 chart artifacts on the Artifact Hub registry of Helm Charts and 90 corresponding maintenance repositories on GitHub. While previous work suggests that vulnerabilities widely exist in the Helm Chart domain, our work systematically investigates the reasons behind such a phenomenon through a combination of quantitative analysis and qualitative analysis following the Grounded Theory (GT) methodology. Our study has verified the nuance

[19] https://github.com/jfrog/chartcenter

behind the high prevalence of fixable CVEs that remain unfixed in the domain and provides in-depth discussions over the current effective mitigation strategies with empirical evidence and a curated theory that revolves around three concepts including trust, trade-off and incentive, which explains the complex phenomenon.

We conduct statistical tests to understand the prevalence of CVEs in Helm Charts and provide insight based on the utilization of security features and the time to CVE mitigation. We investigate contextual factors, including officiality and categories, to shed light on whether the presence of vulnerability severity varies with the factors as filtering criteria for secured Helm Charts. Lastly, we conduct an in-depth grounded theory qualitative study to uncover the application of three types of mitigation strategies in the current Helm Charts maintenance repositories; we answer the remaining questions from previous RQs with the generated theory and provide practical takeaways for researchers, chart maintainers and chart users.

Our empirical investigation into Helm Chart vulnerabilities is intrinsically tied to the broader implications for the security posture of service mesh architectures. As service mesh deployments frequently depend on Helm Charts, the persistent issue of fixable CVEs remaining unfixed poses a significant threat to service mesh system security. The proactive maintenance of Helm Charts, the use of automated tools to tackle the complexity of security threats, and the thorough security assessment before deploying public charts into critical systems are necessary measures to significantly bolster the security of Helm Charts. By doing so, practitioners could enhance the security at the foundational level of deployment mechanisms, fortifying the security of service mesh components deployed via Helm Charts. Thus, the security concerns and mitigation strategies identified within Helm Charts can be seen as a vital aspect of enhancing service mesh security from the bottom up.

We recommend that researchers and Helm Chart maintainers investigate methods for shifting left the responsibility for Helm Chart maintenance and improving automated tools for addressing the complexity of security threats. Researchers should aim to enhance the adoption and effectiveness of automated CVE mitigation strategies and augment security feature utilization. We also advise chart users to conduct comprehensive security assessments before deploying public charts into critical systems. These combined efforts could improve the security of Helm Charts and propagate such benefits to the distributed systems.

CHAPTER 5

Conclusions and Future Work

This thesis presents a comprehensive study of adoption concerns and potential security vulnerabilities inherent in service mesh frameworks and Helm Charts, two integral components of cloud-native technologies that have co-evolved in recent years. Our research analyzed 5,497 service mesh questions on Stack Overflow and other public fora to understand practitioners' concerns, and studied 11,035 chart artifacts on the Artifact Hub registry of Helm Charts along with corresponding maintenance repositories on GitHub to delve into vulnerabilities and their mitigation strategies.

Through our exploration of two service mesh frameworks (Istio and Consul), we highlighted major concerns related to their adoption, such as infrastructure, configuration, and security. Our findings emphasize the need for improved service mesh infrastructure configuration methods and a more concentrated focus on the adoption of security-related features. These measures can promote wider and more efficient use of service mesh frameworks in heterogeneous systems.

In parallel to this, we investigated the security implications of Helm Charts, a pivotal deployment packaging format often employed in the deployment of service mesh architectures. Our empirical study unveiled the prevalence of fixable CVEs that remain unfixed (84%), posing notable security risks to service mesh systems. To mitigate this, we advocate for more proactive approaches to Helm Chart maintenance by sharing vulnerability mitigation responsibilities and reducing the security assessment burden on end users of charts. By addressing these vulnerabilities in Helm Charts, we enhance the security of service mesh architectures by enforcing a bottom-up security improvement approach.

In conclusion, our research provides a comprehensive view of the intricate dynamics between service mesh adoption concerns and Helm Chart vulnerabilities. We demonstrate that the adoption of service mesh architectures, while offering significant potential benefits, can inadvertently introduce security vulnerabilities through its associated deployment methods such as Helm Charts. The insights gathered from our research not only guide practitioners in the field, but also fuel further investigation into improving the adoption and security of service mesh frameworks and Helm Charts. In the ever-evolving cloud computing landscape, securing service mesh from the bottom-up becomes crucial for the sustainable and safe deployment of microservices and cloud-native infrastructures.

5.1 Thesis Contributions

The main contributions of this thesis are listed below:

- Our work encompasses two extensive, mixed-methods empirical studies that scrutinize the adoption concerns of service mesh frameworks and unfixed vulnerabilities in Helm Charts, involving an analysis of 5,497 practitioner questions and answers related to service mesh, and 11,035 Helm Charts with their maintenance repositories.
- We identify a dominance of error-related concerns (48.3%) among service mesh practitioners and highlight infrastructure complexities as significant obstacles to deploying and managing service mesh frameworks.
- Collaborating with a service mesh expert, we uncover a notable shift in concern focus from service mesh traffic to infrastructure, while recognizing the underplayed issues related to security and observability.
- We detail the complex relationships between error symptoms and fixes in the service mesh domain, stressing the need for enhanced error feedback from framework designers.
- We expose the high prevalence and persistence of vulnerabilities (14,448 CVEs) in Helm Charts, emphasizing the need for more efficient mitigation processes.
- Our findings suggest that contextual factors alone cannot reliably predict or filter vulnerable Charts due to negligible effect sizes.

- We delve into 12 underused mitigation strategies found in chart maintenance repositories, advocating for a shared responsibility model and a greater emphasis on informative strategies to prevent shifting the security burden onto Helm Charts' end users.

5.2 Future Work

Potential directions for future research include:

- Exploring the pressing concerns specific to the configuration of container orchestration platforms like Kubernetes. Our research indicates that these platforms and their configuration manifests are key concerns in adopting emerging cloud-native technologies such as service mesh. Understanding the norms and complexities of these configuration problems could lead to the development of effective recommendation tools, simplifying adoption, and reducing complexities. This would further strengthen the approach of enhancing security from the bottom up.

- Investigating the application of the 12 CVE mitigation strategies to individual vulnerabilities. This can provide valuable insights into whether certain mitigation strategies are more effective for specific types of vulnerabilities, considering CVE severity and impact. Such an investigation could greatly contribute to Helm Chart and service mesh security, providing practitioners with guidelines to choose the most appropriate strategy based on detected vulnerabilities.

Bibliography

[1] Acar, Y., Stransky, C., Wermke, D., Mazurek, M. L., and Fahl, S. (2017). Security developer studies with {GitHub} users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95.

[2] Adolph, S., Hall, W., and Kruchten, P. (2011). Using grounded theory to study the experience of software development. *Empirical Software Engineering*, 16:487–513.

[3] Akoglu, H. (2018). User's guide to correlation coefficients. *Turkish journal of emergency medicine*, 18(3):91–93.

[4] Alfadel, M., Costa, D. E., and Shihab, E. (2023). Empirical analysis of security vulnerabilities in python packages. *Empirical Software Engineering*, 28(3):59.

[5] Ali, A., Zavarsky, P., Lindskog, D., and Ruhl, R. (2011). A software application to analyze the effects of temporal and environmental metrics on overall cvss v2 score. In *2011 World Congress on Internet Security (WorldCIS-2011)*, pages 109–113. IEEE.

[6] Allamanis, M. and Sutton, C. (2013). Why, when, and what: analyzing stack overflow questions by topic, type, and code. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 53–56.

[7] Alshuqayran, N., Ali, N., and Evans, R. (2016). A systematic mapping study in microservice architecture. In *Proceedings of the 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 44–51.

[8] Aly, M., Khomh, F., and Yacout, S. (2021). What do practitioners discuss about iot and industry 4.0 related technologies? characterization and identification of iot and industry 4.0 categories in stack overflow discussions. *Internet of Things*, 14:100364.

[9] Antal, G., Keleti, M., and Hegedŭs, P. (2020). Exploring the security awareness of the python and javascript open source communities. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 16–20.

[aquasecurity] aquasecurity. Trivy. https://github.com/aquasecurity/trivy.

[11] Barua, A., Thomas, S. W., and Hassan, A. E. (2014). What are developers talking about? an analysis of topics and trends in stack overflow. *Empirical Software Engineering*, 19(3):619–654.

[12] Basili, V. and Rombach, H. (1988). The TAME project: Towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, 14(6):758–773.

[13] Ben-Kiki, O., Evans, C., and Ingerson, B. (2009). Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008*, 5:11.

[14] Bertino, E. (2021). Zero trust architecture: does it help? *IEEE Security & Privacy*, 19(05):95–96.

[15] Bevana, N., Kirakowskib, J., and Maissela, J. (1991). What is usability? In *Proceedings of the 4th International Conference on Human-Computer Interaction (HCI International)*, pages 1–5.

[16] Bila, N., Dettori, P., Kanso, A., Watanabe, Y., and Youssef, A. (2017). Leveraging the serverless architecture for securing linux containers. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 401–404. IEEE.

[17] Blaise, A. and Rebecchi, F. (2022). Stay at the helm: secure kubernetes deployments via graph generation and attack reconstruction. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 59–69. IEEE.

[18] Blei, D. M. and Lafferty, J. D. (2006). Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 113–120.

[19] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022.

[20] Booth, H., Rike, D., and Witte, G. A. (2013). The national vulnerability database (nvd): Overview.

[21] Bourne, V. (2022). Annual apis and integration report - 2022. Technical report, Software AG.

[22] Brady, K., Moon, S., Nguyen, T., and Coffman, J. (2020). Docker container security

in cloud computing. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0975–0980. IEEE.

[23] Buck, C., Olenberger, C., Schweizer, A., Völter, F., and Eymann, T. (2021). Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust. *Computers & Security*, 110:102436.

[24] Bühlmann, N. and Ghafari, M. (2022). How do developers deal with security issue reports on github? In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, pages 1580–1589.

[25] Burns, B., Grant, B., Oppenheimer, D., Brewer, E., and Wilkes, J. (2016). Borg, omega, and kubernetes. *Communications of the ACM*, 59(5):50–57.

[26] Cappos, J., Samuel, J., Baker, S., and Hartman, J. H. (2008). A look in the mirror: Attacks on package managers. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 565–574.

[27] Cardellini, V., Colajanni, M., and Yu, P. S. (1999). Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39.

[28] Carver, J. (2007). The use of grounded theory in empirical software engineering. In *Empirical Software Engineering Issues. Critical Assessment and Future Directions: International Workshop, Dagstuhl Castle, Germany, June 26-30, 2006. Revised Papers*, pages 42–42. Springer.

[29] Chakraborty, P., Shahriyar, R., Iqbal, A., and Uddin, G. (2021). How do developers

discuss and support new programming languages in technical q&a site? an empirical study of go, swift, and rust in stack overflow. *Information and Software Technology*, 137:106603.

[30] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[31] Chen, Y., Fernandes, E., Adams, B., and Hassan, A. E. (2022). Replication package of the paper "on practitioners' concerns when adopting service mesh frameworks".

[32] Chen, Y., Lin, J., Adams, B., and Hassan, A. E. (2023). Replication package of the paper "why not fixing the fixables? an empirical study on understanding the unfixed vulnerabilities in helm charts".

[33] Chen, Z., Cao, Y., Liu, Y., Wang, H., Xie, T., and Liu, X. (2020). A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, pages 750–762.

[34] Christey, S. and Martin, B. (2013). Buying into the bias: Why vulnerability statistics suck. *BlackHat, Las Vegas, USA, Tech. Rep*, 1.

[35] Chung, L., Nixon, B. A., Yu, E., and Mylopoulos, J. (2012). *Non-functional requirements in software engineering*, volume 5. Springer Science & Business Media.

[36] Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., and Gall, H. C. (2017). An empirical analysis of the docker container ecosystem on github. In *Proceedings*

of the 14th International Conference on Mining Software Repositories (MSR), pages 323–333.

[37] Dabic, O., Aghajani, E., and Bavota, G. (2021). Sampling projects in github for msr studies. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 560–564. IEEE.

[38] De Sanctis, M., Muccini, H., and Vaidhyanathan, K. (2020). Data-driven adaptation in microservice-based iot architectures. In *Proceedings of the 3rd International Conference on Software Architecture Companion (ICSA-C)*, pages 59–62.

[39] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and Ulterior Software Engineering*, pages 195–216.

[40] Draucker, C. B., Martsolf, D. S., Ross, R., and Rusk, T. B. (2007). Theoretical sampling and category development in grounded theory. *Qualitative health research*, 17(8):1137–1148.

[41] Duan, R., Alrawi, O., Kasturi, R. P., Elder, R., Saltaformaggio, B., and Lee, W. (2020). Towards measuring supply chain attacks on package managers for interpreted languages. *arXiv preprint arXiv:2002.01139*.

[42] Dullien, T. (2017). Weird machines, exploitability, and provable unexploitability. *IEEE Transactions on Emerging Topics in Computing*, 8(2):391–403.

[43] Esposito, C., Castiglione, A., and Choo, K.-K. R. (2016). Challenges in delivering software in the cloud as microservices. *IEEE Cloud Computing*, 3(5):10–14.

[44] Ferreira, A. P. and Sinnott, R. (2019). A performance evaluation of containers running on managed kubernetes services. In *Proceedings of the 11th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 199–208.

[45] FIRST, E. (2015). Common vulnerability scoring system v3. 0: Specification document.

[46] Fitzgerald, S., Simon, B., and Thomas, L. (2005). Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the first international workshop on Computing education research*, pages 69–80.

[47] Ganguli, M., Ranganath, S., Ravisundar, S., Layek, A., Ilangovan, D., and Verplanke, E. (2021). Challenges and opportunities in performance benchmarking of service mesh for the edge. In *Proceedings of the 5th International Conference on Edge Computing (EDGE)*, pages 78–85.

[48] Goethals, T., Volckaert, B., and De Turck, F. (2020). Adaptive fog service placement for real-time topology changes in kubernetes clusters. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 161–170.

[49] Gokhale, S., Poosarla, R., Tikar, S., Gunjawate, S., Hajare, A., Deshpande, S., Gupta, S., and Karve, K. (2021). Creating helm charts to ease deployment of enterprise application and its related services in kubernetes. In *2021 International Conference on Computing, Communication and Green Engineering (CCGE)*, pages 1–5. IEEE.

[Google] Google. Distroless. https://github.com/GoogleContainerTools/distroless.

[51] Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE international conference on software maintenance and evolution (ICSME)*, pages 580–589. IEEE.

[52] Hahn, D. A., Davidson, D., and Bardas, A. G. (2020). Mismesh: Security issues and challenges in service meshes. In *Proceedings of the 13th International Conference on Security and Privacy in Communication Systems (SecureComm)*, pages 140–151.

[53] Hamidi, S., Andritsos, P., and Liaskos, S. (2014). Constructing adaptive configuration dialogs using crowd data. In *Proceedings of the 29th international conference on Automated software engineering (ASE)*, pages 485–490.

[54] Haque, M. U. and Babar, M. A. (2022). Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1066–1077. IEEE.

[55] Hassan, A. E. (2008). The road ahead for mining software repositories. In *2008 frontiers of software maintenance*, pages 48–57. IEEE.

[56] Hindle, A., Godfrey, M. W., and Holt, R. C. (2009). What's hot and what's not: Windowed developer topic analysis. In *Proceedings of the 25th International Conference on Software Maintenance (ICSM)*, pages 339–348.

[Hipp] Hipp, R. Sqlite. https://github.com/sqlite/sqlite.

[58] Hoda, R., Noble, J., and Marshall, S. (2012). Developing a grounded theory to

explain the practices of self-organizing agile teams. *Empirical Software Engineering*, 17:609–639.

[59] Horawalavithana, S., Bhattacharjee, A., Liu, R., Choudhury, N., O. Hall, L., and Iamnitchi, A. (2019). Mentions of security vulnerabilities on reddit, twitter and github. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 200–207.

[60] Houmani, Z., Balouek-Thomert, D., Caron, E., and Parashar, M. (2020). Enhancing microservices architectures using data-driven service discovery and qos guarantees. In *Proceedings of the 20th International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 290–299.

[61] Ibrahim, M. H., Sayagh, M., and Hassan, A. E. (2021). A study of how docker compose is used to compose multi-component systems. *Empirical Software Engineering*, 26:1–27.

[62] Iosif, A.-C., Gasiba, T. E., Zhao, T., Lechner, U., and Pinto-Albuquerque, M. (2021). A large-scale study on the security vulnerabilities of cloud deployments. In *Inernational Conference on Ubiquitous Security*, pages 171–188. Springer.

[K8sGPT] K8sGPT. K8sgpt. https://github.com/k8sgpt-ai/k8sgpt.

[64] Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., and Pohthong, A. (2017). Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, 22:579–630.

[65] Klein, M. (2017). Lyft's envoy: Experiences operating a large service mesh.

[66] Kreitz, M. (2019). Security by design in software engineering. *ACM SIGSOFT Software Engineering Notes*, 44(3):23–23.

[67] Kula, R. G., German, D. M., Ouni, A., Ishio, T., and Inoue, K. (2018). Do developers update their library dependencies? an empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23:384–417.

[68] Lepiller, J., Piskac, R., Schäf, M., and Santolucito, M. (2021). Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities. In *Tools and Algorithms for the Construction and Analysis of Systems: 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings, Part II 27*, pages 105–123. Springer.

[69] Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J., and Liu, X. (2022). Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, 27(1):1–28.

[70] Li, W., Lemieux, Y., Gao, J., Zhao, Z., and Han, Y. (2019). Service mesh: Challenges, state of the art, and future research opportunities. In *Proceedings of the 13th International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225.

[71] Lin, J., Zhang, H., Adams, B., and Hassan, A. E. (2023). Vulnerability management in linux distributions: An empirical study on debian and fedora. *Empirical Software Engineering*, 28(2):47.

[72] Liu, P., Ji, S., Fu, L., Lu, K., Zhang, X., Lee, W.-H., Lu, T., Chen, W., and Beyah,

R. (2020). Understanding the security risks of docker hub. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*, pages 257–276. Springer.

[73] Lou, Y., Chen, Z., Cao, Y., Hao, D., and Zhang, L. (2020). Understanding build issue resolution in practice: symptoms and fix patterns. In *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, pages 617–628.

[74] Manadhata, P. K. and Wing, J. M. (2010). An attack surface metric. *IEEE Transactions on Software Engineering*, 37(3):371–386.

[75] Mao, R., Zhang, H., Dai, Q., Huang, H., Rong, G., Shen, H., Chen, L., and Lu, K. (2020). Preliminary findings about devsecops from grey literature. In *2020 IEEE 20th international conference on software quality, reliability and security (QRS)*, pages 450–457. IEEE.

[76] McHugh, M. L. (2013). The chi-square test of independence. *Biochemia Medica*, 23(2):143–149.

[77] Mecklin, C. J. and Mundfrom, D. J. (2005). A monte carlo comparison of the type i and type ii error rates of tests of multivariate normality. *Journal of Statistical Computation and Simulation*, 75(2):93–107.

[78] Mens, T. (2016). An ecosystemic and socio-technical view on software maintenance and evolution. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–8. IEEE.

[79] Miano, S., Bertrone, M., Risso, F., Tumolo, M., and Bernal, M. V. (2018). Creating complex network services with ebpf: Experience and lessons learned. In *Proceedings of the 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8.

[80] Mirhosseini, S. and Parnin, C. (2017). Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In *2017 32nd IEEE/ACM international conference on automated software engineering (ASE)*, pages 84–94. IEEE.

[81] Mohallel, A. A., Bass, J. M., and Dehghantaha, A. (2016). Experimenting with docker: Linux container and base os attack surfaces. In *2016 international conference on information society (i-Society)*, pages 17–21. IEEE.

[82] Myrbakken, H. and Colomo-Palacios, R. (2017). Devsecops: a multivocal literature review. In *Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings*, pages 17–29. Springer.

[83] Nachar, N. et al. (2008). The mann-whitney u: A test for assessing whether two independent samples come from the same distribution. *Tutorials in quantitative Methods for Psychology*, 4(1):13–20.

[84] Ohm, M., Plate, H., Sykosch, A., and Meier, M. (2020). Backstabber's knife collection: A review of open source software supply chain attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 17th International Conference, DIMVA 2020, Lisbon, Portugal, June 24–26, 2020, Proceedings 17*, pages 23–43. Springer.

[85] Picoreti, R., do Carmo, A. P., de Queiroz, F. M., Garcia, A. S., Vassallo, R. F., and Simeonidou, D. (2018). Multilevel observability in cloud orchestration. In *Proceedings of the 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 776–784.

[86] Prana, G. A. A., Sharma, A., Shar, L. K., Foo, D., Santosa, A. E., Sharma, A., and Lo, D. (2021). Out of sight, out of mind? how vulnerable dependencies affect open-source projects. *Empirical Software Engineering*, 26:1–34.

[87] Rahman, A., Parnin, C., and Williams, L. (2019). The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175. IEEE.

[88] Ramsauer, R., Bulwahn, L., Lohmann, D., and Mauerer, W. (2020). The sound of silence: Mining security vulnerabilities from secret integration channels in open-source projects. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 147–157.

[89] Rangnau, T., Buijtenen, R. v., Fransen, F., and Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines. In *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 145–154. IEEE.

[90] Razali, N. M., Wah, Y. B., et al. (2011). Power comparisons of shapiro-wilk,

kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33.

[RenovateBot] RenovateBot. Renovate. https://github.com/renovatebot/renovate.

[92] Röder, M., Both, A., and Hinneburg, A. (2015). Exploring the space of topic coherence measures. In *Proceedings of the 9th International Conference on Web Search and Data Mining (WSDM)*, pages 399–408.

[93] Romano, J., Kromrey, J. D., Coraggio, J., and Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In *annual meeting of the Florida Association of Institutional Research*, volume 177, page 34.

[94] Rombaut, B., Cogo, F. R., Adams, B., and Hassan, A. E. (2023). There's no such thing as a free lunch: Lessons learned from exploring the overhead introduced by the greenkeeper dependency bot in npm. *ACM Transactions on Software Engineering and Methodology*, 32(1):1–40.

[95] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. Technical report, National Institute of Standards and Technology.

[96] Saleh Sedghpour, M. R., Klein, C., and Tordsson, J. (2022). An empirical study of service mesh traffic management policies for microservices. In *Proceedings of the 38th International Conference on Performance Engineering (ICPE)*, pages 17–27.

[97] Sayagh, M., Kerzazi, N., Adams, B., and Petrillo, F. (2018). Software configuration

engineering in practice interviews, survey, and systematic literature review. *IEEE Transactions on Software Engineering*, 46(6):646–673.

[98] Schneider, S., Khalili, R., Manzoor, A., Qarawlus, H., Schellenberg, R., Karl, H., and Hecker, A. (2021). Self-learning multi-objective service coordination using deep reinforcement learning. *IEEE Transactions on Network and Service Management*, 18(3):3829–3842.

[99] Schneider, S., Klenner, L. D., and Karl, H. (2020). Every node for itself: Fully distributed service coordination. In *Proceedings of the 16th International Conference on Network and Service Management (CNSM)*, pages 1–9.

[100] Scoccia, G. L., Migliarini, P., and Autili, M. (2021). Challenges in developing desktop web apps: a study of stack overflow and github. In *Proceedings of the 18th International Conference on Mining Software Repositories (MSR)*, pages 271–282.

[101] Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in machine learning systems. *Advances in Neural Information Processing Systems*, 28.

[102] Sedghpour, M. R. S., Klein, C., and Tordsson, J. (2021). Service mesh circuit breaker: From panic button to performance management tool. In *Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems (HAOC)*, pages 4–10.

[103] Shadija, D., Rezai, M., and Hill, R. (2017). Towards an understanding of microservices. In *Proceedings of the 23rd International Conference on Automation and Computing (ICAC)*, pages 1–6.

[104] Shamim, M. S. I., Bhuiyan, F. A., and Rahman, A. (2020). Xi commandments of kubernetes security: A systematization of knowledge related to kubernetes security practices. *2020 IEEE Secure Development (SecDev)*, pages 58–64.

[105] Sheskin, D. J. (2003). *Handbook of parametric and nonparametric statistical procedures*. Chapman and hall/CRC.

[106] Shu, R., Gu, X., and Enck, W. (2017). A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 269–280.

[107] Squire, M. (2015). “Should we move to stack overflow?” measuring the utility of social media for developer support. In *Proceedings of the 37th International Conference on Software Engineering (ICSE)*, volume 2, pages 219–228.

[108] Stol, K.-J., Ralph, P., and Fitzgerald, B. (2016). Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International conference on software engineering*, pages 120–131.

[109] Syed, S. and Spruit, M. (2017). Full-text or abstract? examining topic coherence scores using latent dirichlet allocation. In *2017 IEEE International conference on data science and advanced analytics (DSAA)*, pages 165–174. IEEE.

[110] Thompson, K. (1984). Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763.

[111] Thönes, J. (2015). Microservices. *IEEE Software*, 32(1):116–116.

[112] Tomas, N., Li, J., and Huang, H. (2019). An empirical study on culture, automation, measurement, and sharing of devsecops. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–8. IEEE.

[113] Vayghan, L. A., Saied, M. A., Toeroe, M., and Khendek, F. (2018). Deploying microservice based applications with kubernetes: Experiments and lessons learned. In *Proceedings of the 11th International Conference on Cloud Computing (CLOUD)*, pages 970–973.

[114] Venkatesh, P. K., Wang, S., Zhang, F., Zou, Y., and Hassan, A. E. (2016). What do client developers concern when using web apis? an empirical study on developer forums and stack overflow. In *Proceedings of the 13th International Conference on Web Services (ICWS)*, pages 131–138.

[115] Wang, B., Li, X., de Aguiar, L. P., Menasche, D. S., and Shafiq, Z. (2017). Characterizing and modeling patching practices of industrial control systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):1–23.

[116] Wang, W. and Godfrey, M. W. (2013). Detecting api usage obstacles: A study of ios and android developer questions. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR)*, pages 61–64.

[117] Williams, L. (2022). Trusting trust: Humans in the software supply chain loop. *IEEE Security & Privacy*, 20(5):7–10.

[118] Wist, K., Helsem, M., and Gligoroski, D. (2021). Vulnerability analysis of 2500 docker hub images. In *Advances in Security, Networks, and Internet of Things: Proceedings from SAM'20, ICWN'20, ICOMP'20, and ESCS'20*, pages 307–327. Springer.

[119] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering.* Springer Science & Business Media, 1st edition.

[120] Wojciechowski, Ł., Opasiak, K., Latusek, J., Wereski, M., Morales, V., Kim, T., and Hong, M. (2021). Netmarks: Network metrics-aware kubernetes scheduler powered by service mesh. In *Proceedings of the 40th Conference on Computer Communications (INFOCOM)*, pages 1–9.

[121] Wood, J. R. and Wood, L. E. (2008). Card sorting: current practices and beyond. *Journal of Usability Studies*, 4(1):1–6.

[122] Wu, M., Zhang, Y., Liu, J., Wang, S., Zhang, Z., Xia, X., and Mao, X. (2022). On the way to microservices: Exploring problems and solutions from online q&a community. In *Proceedings of the 29th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 432–443.

[123] Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., and Talwadker, R. (2015). Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 307–319.

[124] Zahan, N., Zimmermann, T., Godefroid, P., Murphy, B., Maddila, C., and Williams, L. (2022). What are weak links in the npm supply chain? In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 331–340.

[125] Zerouali, A., Cosentino, V., Mens, T., Robles, G., and Gonzalez-Barahona, J. M.

(2019). On the impact of outdated and vulnerable javascript packages in docker images. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 619–623. IEEE.

[126] Zerouali, A., Mens, T., Decan, A., Gonzalez-Barahona, J., and Robles, G. (2021). A multi-dimensional analysis of technical lag in debian-based docker images. *Empirical Software Engineering*, 26(2):19.

[127] Zerouali, A., Opdebeeck, R., and De Roover, C. (2022). Helm charts for kubernetes applications: Evolution, outdatedness and security risks.

[128] Zhang, F., Mockus, A., Zou, Y., Khomh, F., and Hassan, A. E. (2013). How does context affect the distribution of software maintainability metrics? In *2013 IEEE International Conference on Software Maintenance*, pages 350–359. IEEE.

[129] Zhang, S. and Ernst, M. D. (2013). Automated diagnosis of software configuration errors. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, pages 312–321.

[130] Zhang, S. and Ernst, M. D. (2014). Which configuration option should i change? In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, pages 152–163.

[131] Zimmermann, T. and Falleri, J.-R. (2023). A grounded theory of community package maintenance organizations.

APPENDIX A

Appendix

Table A.1: Selected Sample Post URLs

ID	URL
P01	https://stackoverflow.com/questions/66872969/how-to-integrate-appdynamics-in-with-istio
P02	https://stackoverflow.com/questions/48567474/istio-error-no-objects-passed-to-apply
P03	https://stackoverflow.com/questions/55453117/what-are-the-benefits-of-enabling-mtls-in-istio
P04	https://stackoverflow.com/questions/62585045/how-to-enable-https-on-istio-ingress-gateway-with-kind-service
P05	https://stackoverflow.com/questions/54659041/route-to-external-site-via-istio-virtual-service
P06	https://stackoverflow.com/questions/53095970/helm-install-istio-returns-forbidden-error
P07	https://stackoverflow.com/questions/57488845/istio-manual-sidecar-injection-gives-an-error
P08	https://stackoverflow.com/questions/52183540/consul-go-client-redundant-server-connection
P09	https://stackoverflow.com/questions/70076326/how-to-update-istio-configuration-after-installation
P10	https://stackoverflow.com/questions/55159292/istio-policy-not-authenticating-jwt
P11	https://stackoverflow.com/questions/67386438/override-x-request-id-header-in-istio
P12	https://stackoverflow.com/questions/52936524/starting-a-container-pod-after-running-the-istio-proxy
P13	https://stackoverflow.com/questions/62023421/hashicorp-consul-how-to-do-verified-tls- from-pods-int-kubernetes-cluster
P14	https://stackoverflow.com/questions/48753297/bookinfo-example-app-crashes-on-istio
P15	https://stackoverflow.com/questions/59251833/spring-boot-minikube-istio-and-keycloak-invalid-parameter-redirect-uri
P16	https://stackoverflow.com/questions/65896941/istio-egress-gateway-use-istio-requests-total-metric
P17	https://stackoverflow.com/questions/62173549/consul-health-checker
P18	https://stackoverflow.com/questions/71479142/consul-connect-envoy-can
P19	https://stackoverflow.com/questions/59303619/how-do-i-install-istio-with-fixed-static-nodeport-assignments
P20	https://stackoverflow.com/questions/53152217/control-intercept-load-balancer-traffic-using-istio
P21	https://stackoverflow.com/questions/65735493/istio-and-hashicorpt-vault-agent-sidecar-not-working-properly
P22	https://stackoverflow.com/questions/69326155/microservice-with-auth-as-separate-service
P23	https://stackoverflow.com/questions/47096064/kubernetes-rpc-micro-service-with-api-gateway
P24	https://stackoverflow.com/questions/52230457/spring-microservices-timeout-docker-swarm
P25	https://stackoverflow.com/questions/59489749/microservices-not-registering-on-all-eureka-instances
P26	https://stackoverflow.com/questions/33202053/product-versioning-microservices
P27	https://stackoverflow.com/questions/66527165/installing-istio-on-wsl2-fails-with-failedmount-for-pods
P28	https://discuss.hashicorp.com/t/consul-client-and-docker-wont-start-due-to-ip-config

A.1 Appendix

Table A.2: Representative Examples and URLs from Chart Maintenance Repositories

ID	URL
Q01	https://github.com/prometheus-community/helm-charts/issues/3139
Q02	https://github.com/DataDog/datadog-operator/pull/246
Q03	https://github.com/prometheus-community/helm-charts/pull/2957
Q04	https://github.com/bitnami/charts/issues/13516
Q05	https://github.com/hazelcast/charts/issues/21
Q06	https://github.com/helm/charts/issues/13449
Q07	https://github.com/bitnami/charts/issues/17382
Q08	https://github.com/bitnami/charts/issues/8449
Q09	https://github.com/teutonet/teutonet-helm-charts/
Q10	https://docs.bitnami.com/kubernetes/open-cve-policy/
Q11	https://github.com/adfinis/helm-charts
Q12	https://jfrog.com/blog/helm-chart-security-mitigation-in-chartcenter/
Q13	https://github.com/prometheus-community/helm-charts/issues/2582
Q14	https://github.com/bitnami/charts/issues/12706

Table A.3: Contingency Table of CVSS V3 Metrics (By Category)

		1	2	3	4	5	6	7	8	Aggr.
AV	N	1,941	14,506	14,721	33,418	66,398	66,484	82,463	82,666	82,983
	L	1,364	13,160	13,233	25,747	44,735	44,759	53,829	53,972	54,022
	A	57	497	497	899	1,606	1,607	1,852	1,859	1,861
	P	34	509	509	918	1,532	1,532	1,741	1,746	1,746
AC	L	2,889	24,663	24,917	51,900	96,810	96,908	118,338	118,645	118,943
	H	507	4,009	4,043	9,082	17,461	17,474	21,547	21,598	21,669
PR	N	2,240	17,709	17,948	40,287	76,812	76,907	95,619	95,841	96,113
	L	1,021	9,617	9,655	18,179	32,493	32,509	38,318	38,440	38,530
	H	135	1,346	1,357	2,516	4,966	4,966	5,948	5,962	5,969
UI	N	2,626	21,410	21,585	45,214	85,933	86,015	104,881	105,177	105,512
	R	770	7,262	7,375	15,768	28,338	28,367	35,004	35,066	35,100
S	U	3,290	27,727	28,013	58,894	110,570	110,678	135,433	135,785	136,116
	C	106	945	947	2,088	3,701	3,704	4,452	4,458	4,496
C	N	1,801	14,246	14,417	29,857	55,793	55,855	67,594	67,772	67,959
	H	1,404	12,869	12,973	27,339	50,984	51,024	63,119	63,276	63,444
	L	191	1,557	1,570	3,786	7,494	7,503	9,172	9,195	9,209
I	N	2,123	16,918	17,115	35,659	67,165	67,224	81,363	81,597	81,783
	H	1,100	10,362	10,436	21,962	40,830	40,866	50,715	50,825	50,974
	L	173	1,392	1,409	3,361	6,276	6,292	7,807	7,821	7,855
A	H	2,524	22,249	22,480	45,898	84,888	84,954	103,739	103,991	104,222
	N	694	5,164	5,211	11,863	23,253	23,293	28,641	28,718	28,814
	L	178	1,259	1,269	3,221	6,130	6,135	7,505	7,534	7,576

Table A.4: Contingency Table of CVSS V3 Metrics (By Officiality)

		Official	Unofficial
AV	N	4,550	462,173
	L	2,948	314,479
	A	117	11,430
	P	86	12,075
AC	L	6,513	677,756
	H	1,188	122,401
PR	N	5,214	545,204
	L	2,246	220,362
	H	241	34,591
UI	N	5,984	593,000
	R	1,717	207,157
S	U	7,461	775,555
	C	240	24,602
C	N	3,998	384,773
	H	3,271	363,694
	L	432	51,690
I	N	4,734	466,133
	H	2,528	290,848
	L	439	43,176
A	H	5,591	598,025
	N	1,673	159,910
	L	437	42,222