The Journal of Systems & Software 192 (2022) 111393

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Smells and refactorings for microservices security: A multivocal literature review

Francisco Ponce [a,*], Jacopo Soldani [b], Hernán Astudillo [a], Antonio Brogi [b]

[a] Universidad Técnica Federico Santa María, Valparaíso, Chile
[b] University of Pisa, Pisa, Italy

A R T I C L E I N F O

Article history:
Received 4 November 2021
Received in revised form 24 May 2022
Accepted 1 June 2022
Available online 7 June 2022

A B S T R A C T

Context: Securing microservices is crucial, as many IT companies are delivering their businesses through microservices. If security "smells" affect microservice-based applications, they can possibly suffer from security leaks and need to be refactored to mitigate the effects of security smells therein.
Objective: As the available knowledge on securing microservices is scattered across different pieces of white and grey literature, our objective here is to distill well-known smells for securing microservices, together with the refactorings enabling to mitigate their effects.
Method: To capture the state of the art and practice in securing microservices, we conducted a multivocal review of the existing white and grey literature on the topic. We systematically analysed 58 primary studies, selected among those published from 2011 until the end of 2020.
Results: Ten bad smells for securing microservices are identified, which we organized in a taxonomy, associating each smell with the security properties it may violate and the refactorings enabling to mitigate its effects.
Conclusions: The security smells and the corresponding refactorings have pragmatic value for practitioners, who can exploit them in their daily work on securing microservices. They also serve as a starting point for researchers wishing to establish new research directions on securing microservices.

1. Introduction

Microservices are on the rise for architecting enterprise applications nowadays, with big players in IT (e.g., Amazon, Netflix, Spotify, and Twitter) already delivering their core businesses through microservices (Thönes, 2015). This is mainly because microservice-based applications are cloud-native, thus better exploiting the potentials of cloud hosting, and since they fully twin with DevOps and continuous delivery practices (Balalaie et al., 2016). Microservices also bring various other advantages, such as ease of deployment, resilience, and scalability (Newman, 2015). Together with their gains, however, microservices bring also some pains, and securing microservice-based applications is certainly one of those (Soldani et al., 2018).

Microservice-based applications are essentially service-oriented applications adhering to an extended set of design principles (Zimmermann, 2017), e.g., shaping services around business concepts, decentralization, and ensuring the independent deployability and horizontal scalability of microservices, among others. Such additional principles make microservice-based applications not only service-oriented, but also highly distributed and dynamic. As a result, other than the classical security issues and best-practices for service-oriented applications, microservices bring new security challenges (Soldani et al., 2018). For instance, being much more distributed than traditional service-oriented applications, microservice-based application expose more endpoints, thus enlarging the surface prone to security attacks (Lea, 2015). It is also crucial to establish trust among the microservices forming an application and to manage distributed secrets, whereas these concerns are of much less interest in traditional web services or monolithic applications (Yarygi-nam and Bagge, 2018). Another example follows from the many communications occurring among the microservices forming an application, which —if not properly handled— can result in message data being intercepted and in malicious users inferring business operations from such data (Yu et al., 2019).

Whereas there exist quite much literature on securing microservices, different pieces of literature deal with different aspects of security. As a result, the currently available information on securing microservices is scattered among a considerable amount of books, blog posts, research papers, videos, and whitepapers. This hampers consulting the body of knowledge on the topic, both for academic researchers wishing to delineate

* Corresponding author.
E-mail addresses: francisco.ponceme@usm.cl (F. Ponce), jacopo.soldani@unipi.it (J. Soldani), hernan@inf.utfsm.cl (H. Astudillo), antonio.brogi@unipi.it (A. Brogi).

https://doi.org/10.1016/j.jss.2022.111393
0164-1212/© 2022 Elsevier Inc. All rights reserved.

novel research directions and solutions for securing microservices, and for practitioners daily needing to secure microservice-based applications. To help both researchers and practitioners, this paper tries to organize the scattered knowledge on securing microservices, by answering to the following two research questions:

(RQ1) What are the smells indicating possible security violations in microservice-based applications?

(RQ2) How to refactor microservice-based applications to mitigate the effects of security smells therein?

A security smell can be observed in an application, being it a possible symptom of a bad decision (though often unintentional) while designing or developing the application, which may impact on the overall application's security (Ponce, 2021). The effects of security smells can be mitigated by refactoring the application or the services therein, while at the same not changing the functionalities offered to external clients. Even if applying refactorings requires some efforts to application developers, it is known that they can help mitigating the effects of smells to improve the overall system quality (Almogahed and Omar, 2021; Bass et al., 2012).

To answer to our research questions, in this paper we report on what is being said by researchers and practitioners about known security smells and refactorings enabling to mitigate their effects. We provide a sort of "snapshot" of the state of the art and practice on such smells and refactorings, keeping the level of detail at which they are discussed in literature. We indeed systematically analysed the available literature on securing microservices to elicit security smells well-known among researchers and practitioners, which may possibly result in security violations in microservice-based applications. We also elicited the refactorings proposed by researchers and practitioners that are known to mitigate the effects of such security smells. In particular, following the recommendations by Garousi et al. (2016), we captured both the state of the art and the state of practice in the field by conducting a multivocal review of the existing literature. We analysed both white literature (viz., peer-reviewed papers) and grey literature (viz., blog posts, industrial whitepapers, books, and videos). We carefully selected 58 primary studies published since 2011 (when microservices were first discussed in an industrial workshop Lewis and Fowler, 2014) until the end of 2020, which we then systematically analysed by following the guidelines for conducting systematic reviews (Garousi et al., 2019; Kitchenham and Charters, 2007). As a result, we obtained a taxonomy that organizes the identified security smells together with all refactorings that should be enacted to mitigate their effects. Finally, we exploited the taxonomy to classify the selected primary studies, in order to distill the actual recognition of the identified smells and of their corresponding refactorings.

In this paper, we illustrate the results of our multivocal review. We first present the taxonomy of security smells, including ten security smells and ten refactorings, organized around three security properties defined in the ISO/IEC 25010 standard for software quality (ISO, 2011), viz., integrity, authenticity, and confidentiality. Then we discuss each security smell by illustrating why it can possibly violate the security property it is associated with. For each smell, we also discuss the corresponding refactorings recommended by practitioners, by also explaining how such refactoring enables mitigating its effects.

We believe that the results of our study can provide benefits to both researchers and practitioners interested in microservices. A systematic presentation of the state of the art and practice on well-known security smells for microservices provides a body of knowledge to develop new techniques and solutions, to investigate and experiment research implications, and to set future research directions. At the same time, it can help practitioners to better understand the currently most recognized security smells for microservices, and to enact the corresponding refactorings to mitigate their effects. This is of pragmatic value for practitioners, who can use our study as a starting point for securing their microservice-based applications, as well as a reference in their day-by-day work with microservices.
This article is organized as follows. Section 2 illustrates the process enacted to select the reference primary studies. Section 3 presents the taxonomy of security smells and describes them in detail, together with the refactorings enabling to mitigate its effects. Section 4 discusses the potential threats to the validity of our study and how we dealt with them. Finally, Sections 5 and 6 discuss related work and draw some concluding remarks, respectively.

2. Research Design

We identified microservices' security smells and refactorings by conducting a multivocal review Garousi et al. (2016), namely by searching and classifying both white and grey literature on the topic. Considering grey literature however comes with an intrinsic difficulty. Grey literature is indeed intended as materials and research produced by organizations outside the traditional commercial or academic publishing distribution channels, e.g., technical, research, or project reports, working/white papers, government documents, or videos and evaluations. The use of grey literature is hence risky, since there is often little or no scientific factual representation of data or analyses presented in grey literature itself (Farace and Schöpfel, 2010), and because it lacks independent reviews assessing its quality (Garousi et al., 2019). On the other hand, a growing interest around using grey literature for helping software engineering practitioners, as well as combining it with white literature to determine the state of the art and practice around a topic is gaining a considerable interest in the field of software engineering (Garousi et al., 2016; Soldani et al., 2018).

Given the above, and with the aim of maximizing the validity of our study, we followed a systematic approach based on that by Kitchenham and Charters (2007) for conducting systematic literature reviews in software engineering. We first defined the PICO terms for defining the goal and scope of our study (Table 1), and we then enacted a systematic search and classification of primary studies, considering both white literature and grey literature. As for grey literature, following the guidelines by Garousi et al. (2016) and the lessons learned in our former grey literature review Soldani et al. (2018), we varied the standard approach by Kitchenham and Charters (2007) as follows:

- We exploited general web search engines for searching for grey literature.
- We adopted the effort bounded stopping criteria.
- We fixed the type of relevant grey literature to blog posts, whitepapers, industrial magazines, and videos.

The first variation was motivated by the fact that grey literature is publicly available on the web, but typically not indexed by indexing databases (as in the case of white literature). The latter two variations were instead aimed to limit the number of relevant search hits to consider to a manageable amount, and following the recommendations outlined by Garousi et al. (2019).

We hereafter detail the systematic approach that we followed, by starting from the structuring of the search string and by also describing the triangulation and inter-rater reliability assessment trials that we ran to enforce the validity of our findings.

Table 1
PICO terms of our research problem.

Concern	Explanation
Population	(RQ1) microservices' security smells, (RQ2) refactorings for mitigating security smells in microservice-based applications
Intervention	Characterization, internal/external validation, data extraction, synthesis
Comparison	Comparison based on mapping primary studies to a taxonomy
Outcome	A taxonomy of security smells and refactorings for microservice-based applications

2.1. Search for Primary Studies

The structuring of the search string was done by following the guidelines provided by Kitchenham and Charters (2007). We identified the search string guided by the *Population* terms of our research problem (Table 1), with search keywords taken from each aspect of our research problem. We anyhow decided —differently from what indicated by Kitchenham and Charters (2007)— to not restrict our focus to specific research settings, as research settings are often not explicitly described in grey literature (Garousi et al., 2019; Soldani et al., 2018). As a result, our search string was formed by the following terms:

$$(\texttt{microservice}*) \land (\texttt{security}*) \land \\ (\texttt{smell} * \lor \texttt{antipattern} * \lor \texttt{bad practice} * \lor \texttt{pitfall} * \lor \\ \texttt{refactor} * \lor \texttt{reengineer} * \lor \texttt{restructure}*)$$

(where '*' matches lexically related terms). The search was restricted to primary studies published since the beginning of 2011 (when microservices were first discussed in an industrial workshop Lewis and Fowler, 2014) until the end of 2020.

The search of white literature was carried out by matching the search string against the title and abstract of the white literature indexed by the following databases: ACM Digital Library, DBLP, Google Scholar, IEEE Xplore, ISI Web of Science, Science Direct, Scopus, and SpringerLink. Given the recency of microservice-related studies and the well-known concerns with indexing, Google Scholar played a key role for the initial selection before the inclusion and exclusion stage.

The search for grey literature was instead carried out by exploiting the features natively supported by web search engines, which match search strings against the whole content of websites (including, e.g., Medium, Stack Overflow, and YouTube, often used by practitioners to share their experiences). The search engines we employed were the following: Google, Bing, and DuckDuckGo. Given the amount of results returned by the different combinations of the keywords in the search string (often, hundreds of thousands), and given that each search was repeated on each considered search engine, we adopted the effort bounded stopping criteria suggested by Garousi et al. (2019). In particular, for each search on each search engine, we considered the top 250 search hits (e.g., the first 25 pages of results returned by the Google search engine).

The above search resulted in around 6000 search hits, 5250 out of which were the grey literature matches identified with web search engines. These included multiple hits for the same piece of literature, hitted on different indexing platforms or web search engines, and in a high number of irrelevant studies. This held especially in the case of matching grey literature, since web search engines indeed look for search strings over the whole pages they index. We hence enacted a sample selection based on control factors, which we describe in the following section.

2.2. Selection of Primary studies

We enacted a first refinement of the search hits based on the following inclusion criteria:

(i_1) A study is to be selected if it is written in English.
(i_2) A study is to be selected if it qualifies as white literature, or as a blog post, whitepaper, industrial magazine publication, or video authored by a practitioner.[1]
(i_3) A study is to be selected if it focuses on microservices.
(i_4) A study is to be selected if it focuses on security.

The above criteria were designed to focus on studies written in English, either being white literature or grey literature of the form we decided to consider, whilst at the same time dealing with microservices' security. Such criteria enabled us to reduce the search hits to 136 candidate primary studies, which we further refined to align with our research questions, viz., eliciting well-known security smells in microservices and the refactorings enabling to resolve such smells. We indeed screened the 136 candidate primary studies by means of the following two additional inclusion criteria:

(i_5) A study is selected if it presents *at least one security smell* possibly resulting in a violation of a security property defined by the ISO/IEC 25010 software quality standard (ISO, 2011), such as confidentiality, integrity, and authenticity.
(i_6) A study is selected if it presents *at least one refactoring* for mitigating the effects of a security smell, even if the latter is not explicitly mentioned.

In particular, i_5 was checked by determining whether a primary study discusses a possible security smell, viz., a security issue deriving from bad decisions while designing or developing microservices. i_6 was instead checked by determining whether a primary study discusses a technical solution for resolving the occurrence of one such possible security smell. To reduce possible biases in applying i_5 and i_6, we enacted thematic coding (Basit, 2003). The two criteria were applied to all 136 candidate primary studies by the first two authors of this article, who independently coded such studies as to be included/excluded. The inter-rater agreement on inclusion/exclusion of candidate primary studies was then measured by adopting the Krippendorff Kα coefficient, which measures the agreement between two lists of codes applied as part of content analysis (Krippendorff, 2004). The initial agreement on inclusion/exclusion of studies already reached 88.24%, already above the typical reference score of 80%. The 16 inclusion/exclusion mismatches were then resolved by triangulation, with the last two authors of this article independently coding the corresponding 16 primary studies. A joint discussion session was then organized to agree on whether to finally include each of the 16 primary studies based on the four independent codings.

Finally, we decided to keep only one instance of each representative study, in case they were exactly replicated across other pieces of literature, as this often happen in the case of grey literature (e.g., we removed Gardner (2017a) and Raible (2020a) from consideration, as they were replicating the blog posts Gardner (2017b) and Raible (2020b), respectively).

As a result, 58 primary studies were selected to be analysed further. The selected primary studies are listed in Table 2 by

[1] We combined i_2 with the following criteria to ensure that grey literature was authored by a practitioner. We indeed only considered the primary studies published on IT companies' websites, by the social media accounts of IT companies, or by people whose LinkedIn profiles indicate that they work in IT since 5 + years.

Table 2
References, publication years, colours (viz., *white* or *grey* literature), and types (viz., *blog post*, *book*, *book chapter*, *conference* paper, *journal* article, or *video*) of the selected primary studies.

Reference	Year	Colour	Type
Newman (2015)	2015	grey	book
Lea (2015)	2015	grey	blog post
Perera (2016)	2016	grey	blog post
Newman (2016)	2016	grey	video
Hofmann et al. (2016)	2016	grey	book
Wolff (2016)	2016	grey	book
Esposito et al. (2016)	2016	white	journal
Gardner (2017b)	2017	grey	blog post
Troisi (2017)	2017	grey	blog post
Smith (2017)	2017	grey	blog post
Behrens and Payne (2017)	2017	grey	blog post
da Silva (2017)	2017	grey	blog post
Matteson (2017a)	2017	grey	blog post
Matteson (2017b)	2017	grey	blog post
Sass (2017)	2017	grey	blog post
Mannino (2017)	2017	grey	video
Jackson (2017)	2017	grey	book
Carnell (2017)	2017	grey	book
Ziade (2017)	2017	grey	book
Yaryginam and Bagge (2018)	2018	white	conference
Budko (2018)	2018	grey	blog post
Doerfeld (2015)	2018	grey	blog post
Douglas (2018)	2018	grey	blog post
Gupta (2018)	2018	grey	blog post
Khan (2018)	2018	grey	blog post
Krishnamurthy (2018)	2018	grey	blog post
Gebel and Brossard (2018)	2018	grey	video
Jain (2018)	2018	grey	video
Feitosa Pacheco (2018)	2018	grey	book
McLarty et al. (2018)	2018	grey	book
Indrasiri and Siriwardena (2018)	2018	white	book chapter
Richter et al. (2018)	2018	white	conference
Nehme et al. (2019a)	2018	white	conference
Sahni (2019)	2019	grey	blog post
Abasi (2019)	2019	grey	blog post
Boersma (2019)	2019	grey	blog post
Lemos (2019)	2019	grey	blog post
Smith (2019)	2019	grey	blog post
SumoLogic (2019)	2019	grey	blog post
Wallarm (2019)	2019	grey	blog post
Edureka (2019)	2019	grey	video
Siriwardena (2019)	2019	grey	video
Parecki (2019)	2019	grey	video
Sharma (2019)	2019	grey	book
Chandramouli (2019)	2019	grey	whitepaper
Bogner et al. (2019)	2019	white	conference
Nkomo and Coetzee (2019)	2019	white	conference
Nehme et al. (2019b)	2019	white	journal
Raible (2020b)	2020	grey	blog post
Kamaruzzaman (2020)	2020	grey	blog post
Kanjilal (2020)	2020	grey	blog post
Mody (2020)	2020	grey	blog post
O'Neill (2020)	2020	grey	blog post
Radware (2020)	2020	grey	blog post
Siriwardena (2020)	2020	grey	blog post
Siriwardena and Dias (2020)	2020	grey	book
Mateus-Coelho et al. (2020)	2020	white	conference
Rajasekharaiah (2020)	2020	white	book

providing a reference to their full bibliographic information available in the references listed at the end of this article. The table also classifies each selected primary study by publication year, colour, and type. Notably, the colour of 40 out of the 58 selected primary studies is grey, again witnessing the importance of grey literature in distilling the state of practice on microservices, as already noticed in previous reviews (Soldani et al., 2018; Neri et al., 2020).

It is also worth noting that, despite we searched for primary studies published from 2011 to 2020, only primary studies published from 2015 satisfied our selection criteria, therefore getting included in our analysis. A possible reason for this is that microservices started to spread mainly after Lewis and Fowler formalized the microservice-based architectural style in their blog post (Lewis and Fowler, 2014), dated 2014. Security smells and refactorigs were then getting discussed in grey and white literature only after early adopters of the microservice-based architectural style started experiencing security issues in their applications. The first primary studies actually sharing security smells and refactorings were indeed published only a year after Lewis and Fowler's blog post (Lewis and Fowler, 2014), namely in 2015.

2.3. *Literature analysis*

To obtain the findings discussed in Section 3 we again adopted thematic coding (Basit, 2003) and Krippendorff Kα-based inter-rater reliability assessment (Krippendorff, 2004). The selected primary studies were subject to annotation and labelling with the goal of identifying the security smells and refactoring emerging from the analysed text. This process of analysis was executed in parallel over two 50% splits of the selected primary studies, to ensure avoidance of observer bias. The coders of the two splits (viz., the first two authors of this study) were then inverted and an inter-rater evaluation was enacted between the two emerging lists of security smells and refactorings. Inter-rater reliability was then measured by applying the Kα coefficient to measure the agreement among the emerging lists of security smells and of their corresponding refactorings by the two independent observers who individually coded 100% of the selected primary studies. The result of applying Kα to measure the agreement between the two lists amounted to 91.90% agreement, above the typical reference score of 80%.

To further mitigate possible biases, we also enacted a triangulation step. The other two authors of this study were indeed involved in cross-checking the coding, without any prior knowledge on the coding itself. Feedback sessions were then organized to discuss the cross-checking enacted by the last two authors of this study, by firstly discussing their feedback separately with the first two authors of this study, and by then organizing a plenary feedback sessions where all authors were involved. As a result of the feedback sessions, we obtained the final coding discussed in Section 3.

2.4. *Replication package*

To encourage the repeatability of our study and the verifiability of our findings, a replication package is publicly available on GitHub.[2] The replication package contains the intermediary artifacts and the final results of our study. The selected white and grey literature can instead be accessed online, by recovering the bibliographic information from the references listed in Table 2.

3. Microservices security smells and refactorings

Fig. 1 illustrates a taxonomy for the security smells pertaining to the considered security properties, and for the refactorings allowing to mitigate such smells. We obtained our taxonomy by following the guidelines for conducting systematic reviews in software engineering proposed by Kitchenham and Charters (2007):

[2] https://github.com/ms-security/smells-replication-package.

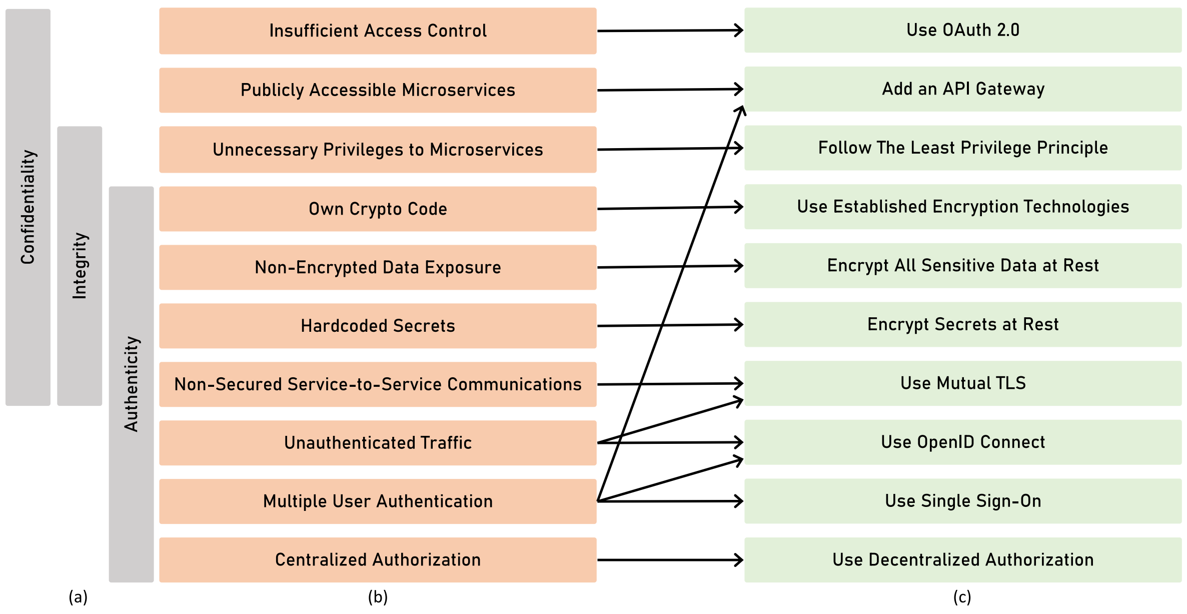
Confidentiality
Integrity
Authenticity
Insufficient Access Control
Publicly Accessible Microservices
Unnecessary Privileges to Microservices
Own Crypto Code
Non-Encrypted Data Exposure
Hardcoded Secrets
Non-Secured Service-to-Service Communications
Unauthenticated Traffic
Multiple User Authentication
Centralized Authorization
Use OAuth 2.0
Add an API Gateway
Follow The Least Privilege Principle
Use Established Encryption Technologies
Encrypt All Sensitive Data at Rest
Encrypt Secrets at Rest
Use Mutual TLS
Use OpenID Connect
Use Single Sign-On
Use Decentralized Authorization
(a)
(b)
(c)

Fig. 1. Taxonomy of microservices security (a) properties, (b) smells, and (c) refactorings. For the sake of readability, the association between security properties and smells is represented by aligning the corresponding boxes, whilst that between smells and refactorings is represented with arrows.

1. We identified the security smells by performing a first scan of the selected primary studies.
2. We excerpted the refactorings directly from the selected primary studies after additional scans.

The obtained security smells and refactorings were then manually organized to obtain a taxonomy. A first version of the taxonomy was obtained by grouping the security smells based on the security properties they pertain to, by taking the security properties defined in the ISO/IEC 25010 standard (ISO, 2011) as a reference. Out of the five properties defined in the ISO/IEC 25010, only three of them resulted to be directly corresponding to some of the identified security smells. These are confidentiality (viz., the degree to which a product or system ensures that data are accessible only to those authorized to have access), integrity (viz., the degree to which a system, product, or component prevents unauthorized modification of computer programs or data), and authenticity (viz., the degree to which the identity of a subject or resource can be proved to be the one claimed). The taxonomy of security properties, smells, and refactorings underwent various iterations among the authors of this study. This resulted in some corrections and amendments to the first version of the taxonomy, which resulted in the final version of the taxonomy displayed in Fig. 1. In the taxonomy, the refactorings associated to a smell should *all* be applied to mitigate its effects.

As we outlined in Section 1, we aim at providing a "snapshot" of the state of the art and practice on smells that may affect the security of microservice-based applications and on refactorings enabling to mitigate such smells' effects. Therefore, despite the smells and refactorings in the taxonomy may be known to appear in distributed systems, our objective here is to analyse on how and how much they affect microservices, by reporting on what researchers and practitioners state in the selected primary studies.

In this perspective, Table 3 shows the classification of all selected primary studies based on the taxonomy in Fig. 1. The table provides a first overview of the coverage of the security smells over the selected primary studies, which is also displayed in Fig. 2. We can observe that researchers and practitioners put more emphasis on securing, authenticating, and authorizing service interactions. Indeed, the more a security smell is related to service interactions, the higher is its coverage in the selected primary studies, as the authors consider it more impacting on the security of a microservice-based application. A reason for this is that microservices-based applications are heavily distributed, with many different services offering endpoints used by external clients and by the application services to interact with each other. The many endpoints and service interactions result in a broader attack surface, whose securing is fundamental (Siriwardena and Dias, 2020). Also, as we discuss hereafter, this is not only when an external service interacts with a frontend service of a microservice-based application, but also when the services forming a microservice-based application interact with each other. Securing, authenticating, and authorizing service interaction occurring between the boundaries of the network of services forming an application is indeed as important as securing, authenticating, and authorizing those coming from external services (Feitosa Pacheco, 2018).

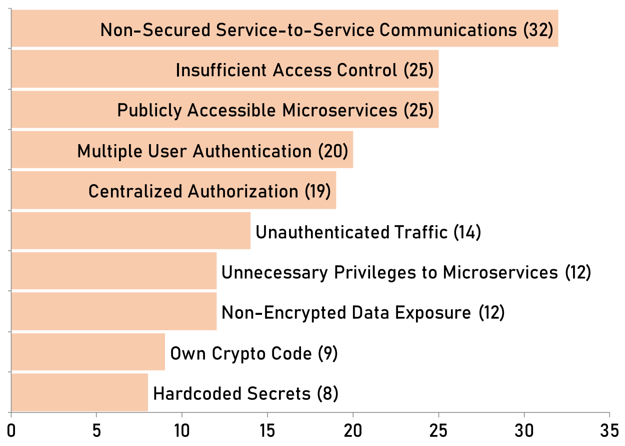

Fig. 2. Coverage of the microservices security smells in the selected studies.

Table 3
Classification of the selected studies according to the taxonomy in Fig. 1.

	Insufficient access control	Publicly accessible microservices	Unnecessary privileges to microservices	Own crypto code	Non-encrypted data exposure	Hardcoded secrets	Non-secured service-to-service communications	Unauthenticated traffic	Multiple user authentication	Centralised authorization
Newman (2015)				✓	✓					✓
Lea (2015)	✓		✓				✓	✓		
Yaryginam and Bagge (2018)							✓			✓
Gardner (2017b)		✓		✓					✓	
Raible (2020b)	✓					✓	✓			
Perera (2016)										✓
Newman (2016)	✓				✓		✓			
Hofmann et al. (2016)	✓			✓	✓	✓		✓	✓	✓
Wolff (2016)	✓						✓			
Esposito et al. (2016)							✓			
Troisi (2017)	✓	✓		✓						
Smith (2017)		✓							✓	
Behrens and Payne (2017)			✓						✓	
da Silva (2017)			✓	✓	✓		✓		✓	
Matteson (2017a)	✓	✓	✓				✓			
Matteson (2017b)			✓				✓			
Sass (2017)						✓	✓			
Mannino (2017)		✓				✓			✓	✓
Jackson (2017)		✓	✓		✓		✓		✓	✓
Carnell (2017)	✓	✓	✓				✓			
Ziade (2017)	✓									✓
Budko (2018)								✓		
Doerfeld (2015)	✓							✓		
Douglas (2018)		✓					✓		✓	✓
Gupta (2018)					✓		✓			
Khan (2018)	✓	✓		✓		✓				✓
Krishnamurthy (2018)	✓	✓			✓					
Gebel and Brossard (2018)	✓	✓						✓	✓	
Jain (2018)			✓		✓	✓	✓			
Feitosa Pacheco (2018)		✓					✓		✓	
McLarty et al. (2018)	✓	✓					✓	✓		✓
Indrasiri and Siriwardena (2018)	✓	✓					✓		✓	✓
Richter et al. (2018)										✓
Nehme et al. (2019a)	✓	✓						✓	✓	✓
Sahni (2019)				✓		✓		✓		
Abasi (2019)	✓	✓	✓					✓		
Boersma (2019)			✓		✓		✓	✓		
Lemos (2019)				✓			✓			
Smith (2019)	✓				✓		✓	✓	✓	
SumoLogic (2019)	✓	✓								✓
Wallarm (2019)								✓		
Edureka (2019)	✓						✓		✓	
Siriwardena (2019)		✓					✓			
Parecki (2019)	✓					✓				
Sharma (2019)	✓						✓			
Chandramouli (2019)							✓	✓	✓	
Bogner et al. (2019)		✓								
Nkomo and Coetzee (2019)							✓	✓	✓	✓
Nehme et al. (2019b)	✓	✓							✓	✓
Kamaruzzaman (2020)		✓								
Kanjilal (2020)		✓	✓				✓		✓	
Mody (2020)			✓				✓			
O'Neill (2020)	✓	✓		✓						
Radware (2020)							✓			
Siriwardena (2020)										✓
Siriwardena and Dias (2020)	✓	✓			✓		✓		✓	✓
Mateus-Coelho et al. (2020)					✓		✓		✓	
Rajasekharaiah (2020)		✓					✓			✓

At the same time, Fig. 2 also shows that all security smells in the taxonomy are significantly recognized by the authors of the selected primary studies, hence making them worthy to get discussed in detail. We hereafter provide a structured presentation of the identified smells. For each smell, we recall the *affected security properties* and provide a *description* of the smell, where we also illustrate how (as per what emerges from the selected primary studies) the smell may possibly result in violating the corresponding security properties. We then illustrate the *suggested refactorings*, also classifying them based on their *type* (viz., use of established security protocols/libraries, design pattern implementation, or data encryption).

3.1. Insufficient access control

Affected Security Properties. Confidentiality.
Description. The Insufficient Access Control smell occurs whenever a microservice-based application does not enact access control in one or more of its microservices, hence possibly violating the Confidentiality of the data and business functions of the microservices where access control is lacking (Troisi, 2017; Carnell, 2017). If this smell is present, microservices can get exposed to, e.g., the "confused deputy problem", with attackers that can trick a service and get data that they should not be able to get (Newman, 2016). At the same time, microservices are not suitable for traditional identity control models, since client details and permissions need to be verified as and when a request is sent (Edureka, 2019), and they need a way to automatically decide whether to allow or reject calls between services (Ziade, 2017). In addition, microservices require development teams to establish and maintain the identity of users without introducing extra latency and contention with frequent calls to a centralized service (Hofmann et al., 2016).
Suggested Refactoring. From the 25 primary studies describing the Insufficient Access Control smell, it turns out that the effects of this smell can be mitigated if development teams Use OAuth 2.0. Open Authorization (OAuth) 2.0 (IETF OAuth Working Group, 2012) is indeed the most used mechanism to manage access delegation. OAuth 2.0 is a token-based security framework for delegated access control that lets a resource owner grant a client access to a certain resource on their behalf. This access is for a limited time and with limited scope (McLarty et al., 2018). OAuth 2.0 is hence a natural candidate to enforce access control in microservice-based application at each level, therein included controlling the accesses to each microservice (Lemos, 2019).

Refactoring Type. Use of established security protocols.

3.2. Publicly accessible microservices

Affected Security Properties. Confidentiality.
Description. The Publicly Accessible Microservices smell occurs whenever the microservices forming an application are directly accessible by external clients (Matteson, 2017a; O'Neill, 2020). Each microservice can be accessed independently through its own API, and it needs a mechanism to ensure that each request is authenticated and authorized to access the set of functions requested (Abasi, 2019). However, if each microservice performs authentication individually, the full set of a user's credentials is required each time, increasing the likelihood of Confidentiality violations (e.g., with the exposure of long-term credentials) and reducing the overall maintainability and usability of the application. Also, each microservice is required to enforce the security policies that are applicable across all functions of the microservice-based application (McLarty et al., 2018).
Suggested Refactoring. The Publicly Accessible Microservices smell is discussed in 25 out of the 58 selected primary studies. The authors of such 25 primary studies highlight that development teams can mitigate the effects of the Publicly Accessible Microservices smell if they Add an API Gateway. This enables to identify a set of microservices to be exposed through the newly introduced API Gateway, while the rest of the microservices are made unreachable from outside this domain. The API gateway centrally enforces security for all the requests entering the microservices application, including authentication, authorization, throttling, and message content validation for known security threats (Siriwardena and Dias, 2020). For instance, as we shall discuss in Section 3.9, the API gateway can be used to implement a single sign-on to the application. In addition, by using this approach development teams can also secure all microservices behind a firewall, allowing the API gateway to handle external requests and then communicate with the microservices behind the firewall (O'Neill, 2020). Since the clients do not directly access the services, they cannot exploit the services on their own (Kanjilal, 2020).

Refactoring Type. Design pattern implementation.

3.3. Unnecessary privileges to microservices

Affected Security Properties. Confidentiality and Integrity.
Description. The Unnecessary Privileges to Microservices smell occurs when microservices are granted unnecessary access levels, permissions, or functionalities that are actually not needed by such microservices to deliver their business functions (Carnell, 2017; Abasi, 2019). The Unnecessary Privileges to Microservices smell is described in 12 out of the 58 selected primary studies, all highlighting how such additional privileges given to microservices would potentially result in Confidentiality and Integrity issues. This happens, e.g., when a service can write or read data stored in databases or messages posted in messages queues, even if such databases or queues are not needed by the service to deliver its business function. As a result, resources are unnecessarily exposed, hence unnecessarily increasing the attack surface for Confidentiality and Integrity leaks: an intruder taking control of a service can indeed start reading or modifying all data and messages the service can access to Lea (2015).
Suggested Refactoring. The authors of the primary studies describing the Unnecessary Privileges to Microservices smell also highlight that development teams can mitigate its effect if they Follow the Least Privilege Principle. The latter recommends that accounts and services should have the least amount of privileges they need to suitably perform their business function (Jackson, 2017). This principle should be used because even if a development team has ensured that the machine-to-machine communication is secured and there are appropriate safeguards with their firewall, there is always the risk that an attacker gets access to the microservice-based application (Jackson, 2017; Boersma, 2019). Development teams should indeed limit service privileges, by providing each service with access to only the resources they actually need to deliver their business functionalities.

Refactoring Type. Design pattern implementation.

3.4. Own crypto code

Affected Security Properties. Confidentiality, Integrity, and Authenticity.
Description. It is well-known that Confidentiality, Integrity, and Authenticity of data in software applications can get violated if development teams use their Own Crypto Code, viz., their own new encryption solutions and algorithms, unless they have been heavily tested (Khan, 2018; O'Neill, 2020). The authors of nine out of the 58 selected primary studies emphasize that microservice-based applications are not the exception: development teams that implement their own encryption solutions may end with improper solutions for securing microservices, which may result

in possible Confidentiality, Integrity, and Authenticity issues. The use of Own Crypto Code may actually be even worse than not having any encryption solution at all, as it may produce a false sense of security (Newman, 2015).
Suggested Refactoring. In all the primary studies describing the Own Crypto Code smell, the authors point out that the way to mitigate this smell is through the Use of Established Encryption Technologies. Development teams should indeed minimize the amount of encryption code they write on their own, but rather maximize the reuse of code coming from libraries that have already been heavily tested by the community (Gardner, 2017a; O'Neill, 2020). Development teams should also avoid the use of experimental encryption algorithms, as they may be subject to various kinds of vulnerabilities, which may be not yet known at the time of their use. Whatever are the programming languages used to implement the microservices forming an application, development teams always have access to reviewed and regularly patched implementations of established encryption algorithms (Newman, 2015).

Refactoring Type. Use of established security libraries.

3.5. Non-encrypted data exposure

Affected Security Properties. Confidentiality, Integrity, and Authenticity.
Description. The Non-Encrypted Data Exposure smell occurs when a microservice-based application accidentally expose sensitive data, e.g., because it was stored without any encryption in the data storage, or because the employed protection mechanisms are affected by security vulnerabilities or flaws (Newman, 2015; Boersma, 2019). When sensitive data is exposed, its Confidentiality and Integrity can get violated because it could be acquired or modified by an intruder who gets direct access to the microservices forming an application. As a result, the intruder may access to or manipulate, e.g., some credentials for accessing other systems or business-critical data (Hofmann et al., 2016).
Suggested Refactoring. From the 12 selected primary studies that describe the Non-Encrypted Data Exposure smell, it emerges that development teams can mitigate the effects of this smell if they Encrypt all Sensitive Data at Rest, viz., when data is not actively moving from device to device or network to network, e.g., when data is stored on a hard drive. The microservice-based architectural style enables development teams to separate functions from data in each of the microservices forming an application (Newman, 2016). As a recommendation, all sensitive data should always be encrypted, and it should be decrypted only when it needs to be used. Most database management systems provide features for automatic encryption, and disk-level encryption features are available at the operating-system level (Siriwardena and Dias, 2020). Application-level encryption is another option, in which the application itself encrypts the data before passing it over to the file system or a database. Finally, if a caching technology is used and the data is encrypted in the database, then development teams need to ensure that the same level of encryption is applied to such caching technology (Jackson, 2017).

At the same time, development teams should also keep in mind that encryption is a resource-intensive operation that could have a considerable impact on the application performance (Siriwardena and Dias, 2020). As not all data needs the same level of security, and since in a microservice-based application it is common to have multiple data stores, development teams must perform a proper analysis to identify the critical ones and encrypt them.

Refactoring Type. Data encryption.

3.6. Hardcoded secrets

Affected Security Properties. Confidentiality, Integrity, and Authenticity.
Description. The Hardcoded Secrets smell occurs when a microservice of an application has hard-coded credentials in its source code, or when credentials are hardcoded in the deployment scripts for a microservice-based application, e.g., as environment variables passing secrets in a Dockerfile or a Docker Compose file (Hofmann et al., 2016; Mannino, 2017). Microservices are indeed likely have secrets to be used for communicating with authorization servers and other services. These secrets might be an API key, a client secret, or credentials for basic authentication (Raible, 2020a). The authors of eight out of the 58 selected primary studies clearly state that development teams should never store sensitive keys and other information in environment variables. In the latter case, the Confidentiality and Integrity of secrets may get violated, as they could be accidentally exposed, e.g., since exception handlers may grab and send the corresponding information to a logging platform. In addition, since child processes duplicate the parent's environment on startup, child processes may be another source of exposure of secrets in application logs, or they may be the reason why secrets could be unintentionally accessed by other services. In all such cases, we would end up with Confidentiality and Integrity leaks (Khan, 2018; Sahni, 2019).
Suggested Refactoring. The authors of the 8 primary studies describing the potential security leaks due to Hardcoded Secrets all agree on saying that development teams can mitigate the effects of this smell if they Encrypt Secrets at Rest. This would indeed help achieving that only authorized resources have access to secrets. In doing so, development teams should also adopt the following best practices: they should never store credentials alongside applications (Hofmann et al., 2016) or in the repositories used to host their source code (Raible, 2020b), nor they should exploit environment variables to pass secrets to applications (Khan, 2018).

Refactoring Type. Data encryption.

3.7. Non-secured service-to-service communications

Affected Security Properties. Confidentiality, Integrity, and Authenticity.
Description. This smell occurs whenever two microservices in an application interact without enacting a secure communication channel, even if they are within the same network (Siriwardena and Dias, 2020; Mateus-Coelho et al., 2020). As microservice-based applications are highly distributed, communication interfaces and channels proliferate, hence increasing the overall application attack surface. Each API exposed by each microservice indeed constitutes a potential attack vector that could be exploited by a malicious intruder, as it does each communication channel between any two microservices (Kanjilal, 2020). Microservices often need to communicate with each other to perform their business functions, and —if the communication channel is not secured— the data transferred can be exposed to man-in-the-middle, eavesdropping, and tampering attacks. This could not only result in Confidentiality issues for service-to-service communications, as it could also break their Integrity and authenticity, e.g., intruders could intercept the communication between two microservices and change the data in transit to their advantage (da Silva, 2017; Siriwardena and Dias, 2020).
Suggested Refactoring. The authors of the 32 primary studies describing the Non-Secured Service to Service Communications

all agree on saying that this smell can be mitigated with the Use of Mutual Transport Layer Security. Mutual TLS (Siriwardena, 2014) is indeed a widely-accepted solution to secure service-to-service communications, which enables encrypting the data in transit and ensuring its integrity and confidentiality. This is going to protect the microservice-based application context from man-in-the-middle, eavesdropping, and tampering attacks providing a bidirectional encryption channel. In addition, when Mutual TLS is used to secure communications between two microservices, each microservice can legitimately identify the microservice it is talking to, which means that microservices can authenticate each other (Indrasiri and Siriwardena, 2018; Siriwardena and Dias, 2020). Hence, whereas there exist other solutions for encrypting data in transit and ensuring its integrity, Mutual TLS is suggested by experienced practitioners as it also helps mitigating the Unauthenticated traffic smell (as we will discuss next).
Refactoring Type. Use of established security protocols.

3.8. Unauthenticated traffic

Affected Security Properties. Authenticity.
Description. The Unauthenticated Traffic smell occurs in a microservice-based application not only when there are unauthenticated API requests coming from external systems, but also when there are unauthenticated requests between the microservices of the application themselves (Abasi, 2019; Boersma, 2019). To ensure Authenticity in microservice-based applications, it is critical to ensure that each request is authenticated and authorized. Therefore, it is necessary to have mechanisms that allow the authentication of traffic coming from outside the application, as well as that corresponding to messages between its microservices. It is indeed crucial each microservice can authenticate each other, especially when the interactions are due to some user transactions and a microservice is passing the logged-in user context to another microservice (Siriwardena and Dias, 2020). The problem here is that no information is typically shared among microservices, and the user context must be passed explicitly from one microservice to another. The challenge is hence to build trust between two interacting microservices, in such a way that the receiving microservice can accept the user context passed from the calling one (Siriwardena and Dias, 2020). Therefore, a way is needed to verify that the user context (and, more generally, the data) passed between microservices has not been modified. If the traffic is not authenticated, there is actually no guarantee that this is the case, and microservices are exposed to security attacks that may result in, e.g., tampering with data, denial of service, or elevation of privileges (Nkomo and Coetzee, 2019).
Suggested Refactoring. The Unauthenticated Traffic smell is discussed in 14 of the selected primary studies, whose authors highlight the refactorings to be enacted to mitigate this smell are to Use Mutual TLS and to Use OpenID Connect. As we already discussed how Mutual TLS (Siriwardena, 2014) enables authentication between any two interacting services (see 3.7), we hereafter focus on the use of OpenID Connect. OpenID Connect is the most used mechanism to manage user authentication. It is based on the use of an ID token, typically a JSON Web Token that contains authenticated user information, including user claims and other relevant attributes (Siriwardena and Dias, 2020). The user ID token enables microservices to verify the user identity based on the authentication performed by an authorization server, as well as to obtain basic profile information about the end-user in an interoperable and REST-like manner. OpenID Connect hence provides a distributed identity mechanism for traffic authentication, which is also recognized to be easy to use, self-contained, and easy to replicate (Doerfeld, 2015).
Refactoring Type. Use of established security protocols.

3.9. Multiple user authentication

Affected Security Properties. Authenticity.
Description. The Multiple User Authentication smell occurs when a microservice-based application provides multiple access points to handle user authentication (da Silva, 2017). Each access point constitutes a potential attack vector that can be exploited by an intruder to authenticate as an end-user, and having multiple access points hence result in increasing the attack surface to violate Authenticity in a microservice-based application (Kanjilal, 2020). The use of multiple access points for user authentication also results in maintainability and usability issues, since user login is to be developed, maintained, and used in multiple parts of the application (Mateus-Coelho et al., 2020).
Suggested Refactoring. The Multiple User Authentication smell is discussed in 21 of the selected primary studies, whose authors point out that development teams can mitigate the effects of this smell if they Use a Single Sign-On. The single sign-on approach suggests having a single entry point to handle user authentication and to enforce security for all the user requests entering the microservice-based application (Kanjilal, 2020). This approach facilitates log storage and auditing tasks (Nehme et al., 2019b), allowing the detection of abnormal situations that may occur. Single sign-on can actually be achieved if (i) we Add an API Gateway acting as a single entry point to the application, if not already there, and if (ii) we Use OpenID Connect to share the user context among the microservices. Both refactorings not only help implementing the single sign-on to mitigate the effects of the Multiple User Authentication smell, but also contribute mitigating the effects of other smells. Indeed, we already discussed how the use of an API gateway helps mitigating the Publicly Accessible Microservices smell, as well as how OpenID Connect enables mitigating the effects of the Unauthenticated Traffic smell (in Sections 3.2 and 3.8, respectively).

Refactoring Type. Design pattern implementation (Use Single Sign-On, Add an API Gateway) and use of established security protocols (Use OpenID Connect).

3.10. Centralized authorization

Affected Security Properties. Authenticity.
Description. In a microservice-based application, authorization can be enforced at the "edge" of the application (e.g., with the API gateway), by each microservice of the application, or both. The Centralized Authorization smell occurs when the microservice-based application only handles authorization in one component, typically at the "edge" of the application, while it does not enact any fine-grained authorization control at the microservices-level. Such a kind of centralized authorization diminishes the advantages of having a distributed solution, such as that given by microservices, and it reduces performances and efficiency, since the central authorization point tends to become a bottleneck (Hofmann et al., 2016; Siriwardena and Dias, 2020). A Centralized Authorization may also result in violating Authenticity in microservice-based applications. For instance, when authorization is managed only at the edge, microservices are exposed to the so-called "confused deputy problem": they trust the gateway based on its mere identity, exposing the microservices in an application to misuse if they are compromised (Nehme et al., 2019a). It is worth noting that the "confused deputy problem" here occurs in a different way from when there is Insufficient Access Control, as here a centralized authorization results in an

Table 4
Summary of the classification of selected primary studies.

Smell	Affected security properties	Refactoring	Refactoring type	Coverage
Insufficient Access Control	Confidentiality	Use OAuth 2.0	Use established security protocols	25/58
Publicly Accessible Microservices	Confidentiality	Add an API Gateway	Design pattern implementation	25/58
Unnecessary Privileges to Microservices	Confidentiality, Integrity	Follow The Least Privilege Principle	Design pattern implementation	12/58
Own Crypto Code	Confidentiality, Integrity, Authenticity	Use Established Encryption Technologies	Use established security libraries	9/58
Non-Encrypted Data Exposure	Confidentiality, Integrity, Authenticity	Encrypt All Sensitive Data at Rest	Data encryption	12/58
Hardcoded Secrets	Confidentiality, Integrity, Authenticity	Encrypt Secrets at Rest	Data encryption	8/58
Non-Secured Service-to-Service Communications	Confidentiality, Integrity, Authenticity	Use Mutual TLS	Use established security protocols	32/58
Unauthenticated Traffic	Authenticity	Use Mutual TLS, Use OpenID Connect	Use established security protocols	14/58
Multiple User Authentication	Authenticity	Use Single Sign-On, Add an API Gateway, Use OpenID Connect	Design pattern implementation, use established security protocols	20/58
Centralized Authorization	Authenticity	Use Decentralized Authorization	Design pattern implementation	19/58

invoked microservice trusting its invoker without enacting any further authorization. In the case of Insufficient Access Control, instead, it is the invoker that has access to too many resources, hence potentially exposing such resources to attacks.
Suggested Refactoring. The Centralized Authorization smell is discussed in 19 out of the 58 selected primary studies. From such 19 primary studies, it emerges that development teams can refactor microservice-based application to mitigate the Centralized Authorization smell by enacting a decentralized authorization approach. They can indeed Use Decentralized Authorization by simply developing a token-based authorization mechanism. This is achieved by transmitting an access token together with each request to a microservice, and access to such microservice is granted to the caller only if a known and correct token is passed (Richter et al., 2018). In this way, authorization can be enforced also at the microservices-level, as it gives each microservice more control to enforce its own access-control policies. Among the 19 primary studies discussing the Use Decentralized Authorization refactoring, JSON Web Token (JWT) is the most used mechanism to implement such a refactoring. A JWT is a standard for safely passing claims or data attributed to a user within an environment (Jackson, 2017), which ensures that a man in the middle cannot change its content because the issuer of the JWT actually signs it (Siriwardena and Dias, 2020).

Refactoring Type. Design pattern implementation.

3.11. Summary

Table 4 shows a summary of the smells and refactorings discussed in this section. This also includes the security properties affected by them and how often they were discuss on the selected primary studies (viz., their *coverage*).

The table again confirm what we observed earlier, namely that researchers and practitioners put more emphasis on the security smells and refactorings related to service interactions. Interaction-related smells can indeed affect confidentiality and authenticity in microservice-based applications, mainly because such application are heavily distributed. Many different services indeed offer endpoints used by external clients and by the application services to interact with each other, hence increasing the surface for potential security attacks (Siriwardena and Dias, 2020). Table 4 also shows that, to mitigate the effects of smells possibly affecting confidentiality and authenticity, researchers and practitioners mainly recommend to leverage of existing solutions, namely to implement well-known design patterns and to use established security protocols (Indrasiri and Siriwardena, 2018; Abasi, 2019).

Leveraging of existing solutions is also widely recommended to deal with integrity-related smells, with researchers and practitioners recommending to use established security protocols and libraries to mitigate their effects (O'Neill, 2020). In this case, a significantly recognized solution is also to encrypt sensitive data (Jain, 2018).

4. Threats to validity

Wohlin et al. (2000) define the potential threats to the validity of studies in empirical software engineering, four of which also apply to our study. These are the threats to the *external*, *internal*, *construct*, and *conclusions* validity, which we discuss hereafter.

4.1. Threats to external validity

The external validity concerns the applicability of a set of results in a more general context (Wohlin et al., 2000). Since the primary studies considered by our multivocal review were selected from a very large extent of online sources, the security smells and the corresponding refactorings may only be partly applicable to the broad area of disciplines and practices on microservices. This may hence result in threatening the external validity of our study.

To reinforce the external validity of our findings, we organized multiple feedback sessions with all authors during our analysis of the selected studies (Section 2.3). The discussions held within and after the feedback sessions resulted in qualitative data, which we exploited to fine-tune the taxonomy of security smells and refactorings resulting from our study, obtaining that presented in Section 3. We also prepared a replication package (Section 2.4) storing the artifacts produced during our study, to make them publicly available to all who wish to deepen their understanding on the data we produced. We believe that this helps in making our results and observations more explicit and applicable in practice.

As for the external validity of our study, one may also consider our selection criteria as "too restrictive". However, such criteria enable focusing only on representative studies, viz., studies discussing at least a security smell or a corresponding refactoring. There is however a potential risk of having missed some relevant literature, as a study might not explicitly mention the security smells in our taxonomy (Fig. 1). To mitigate this potential threat, we carefully checked both our selection criteria against each candidate study. We indeed checked whether a study was discussing the security issues connected to some security smell, and whether it was discussing the concrete changes to apply to a microservice-based application to mitigate the effect of such smell. This enabled us to select also those studies that were not explicitly referring to a smell or refactoring in our taxonomy, but rather reporting on the corresponding security issues or to-be-applied changes.

Finally, another potential threat to the external validity of our study is having missed relevant grey literature. Practitioners may indeed share knowledge by exploiting a different terminology than ours, e.g., a practitioner's blog post may discuss some security smells or refactorings, without explicitly mentioning the terms "smell" or "refactor". To mitigate this threat to validity, we included relevant synonyms in the search string, and we exploited the features natively supported by search engines, such as including related terms in string-based searches.

4.2. Threats to construct and internal validity

Wohlin et al. (2000) define the internal validity of studies as concerning the method employed to study and analyse data, therein included the potential types of bias involved. They instead define the construct validity as the generalizability of the constructs under study.

To mitigate the potential threats to the construct and internal validity of our study, we exploited theme coding and inter-rater reliability assessment (Sections 2.2 and 2.3). These helped limiting potential biases, such as observer and interpretation biases, hence helping us to enhance the validity of the analysis we performed on the data we retrieved. In addition, the taxonomy organizing the emerging lists of smells underwent various iterations among all the authors of this study to further avoid bias by triangulation. The same process was applied to the actual classification of the selected primary studies and to the results of the analysis.

4.3. Threats to conclusions validity

The conclusions validity is defined by Wohlin et al. (2000) as concerning the degree to which the conclusions of a study are reasonably based on the available data.

To mitigate potential threats to the conclusions validity of our study, we exploited the above described inter-rater reliability assessment to limit potential biases in our observations and interpretations. Additionally, the observations and conclusions discussed in this paper were drawn independently by the authors of this paper. They were then discussed and double-checked against the selected primary studies in two joint discussion sessions.

5. Related work

Various secondary studies analyse and classify the state of the art and practice on microservices. For instance, Pahl and Jamshidi (2016) elicit potential research directions on microservices, after discussing agreed and emerging concerns on microservices and positioning microservices with respect to existing cloud and container technologies. Taibi et al. (2018) instead report on common architectural patterns for microservices, by discussing the advantages, disadvantages, and lessons learned of each pattern. However, neither Pahl and Jamshidi (2016) nor Taibi et al. (2018) provide an overview on the smells possibly resulting in security issues in microservice-based applications, nor on the ways to mitigate their effects of such smells.

Other noteworthy examples are the systematic grey literature review by Soldani et al. (2018) and the industrial surveys by Di Francesco et al. (2018) and Ghofrani and Lübke (2018), which all provide an overview on the state of practice on microservices. Soldani et al. (2018) identify the technical and operational advantages and disadvantages of microservices, therein included the difficulties in securing microservice-based applications. Di Francesco et al. (2018) and Ghofrani and Lübke (2018) instead illustrate the results of surveys they conducted with practitioners daily working with microservices, also resulting in distilling the advantages and disadvantages of microservices. The studies by Soldani et al. (2018), Di Francesco et al. (2018), and Ghofrani and Lübke (2018) differ from ours in their objective: they report on challenges and advantages of microservices, whereas we focus on distilling the smells that may possibly result in security issues in microservices, as well as on how to mitigate their effects.

Berardi et al. (2022) instead first report on microservices' security, by analysing white literature to identify the research communities where this is most discussed. Berardi et al. (2022) also distill the currently known security attacks to microservice-based applications, as well as the countermeasures that have been proposed to secure such applications from such attacks. However, review (Berardi et al., 2022) differs from ours in the objectives. They indeed focus on existing solutions to secure microservices from possible attacks. We instead focus on known security smells, namely on bad decisions that may hamper the security of microservice-based applications, as well as on the refactorings that are know to mitigate their effects. Also, Berardi et al. (2022) consider only white literature on securing microservices, while we also consider grey literature to analyse both the state of the art and the state of practice on microservices' security smells.

In this perspective, the industrial survey reported by Taibi and Lenarduzzi (2018) is a step closer to ours, as they first explicitly defined 11 microservice-specific bad smells. The smells defined by Taibi and Lenarduzzi (2018) span from the design of microservice-based application to their actual development, and each smell is equipped with the best practices enabling to avoid incurring in such smell. Our results complement those by Taibi and Lenarduzzi (2018), as none of the smells they define is about securing microservices. We instead precisely distill the refactorings that enable mitigating the effects of well-known security smells in microservice-based applications.

Similar considerations apply to the secondary studies presented by Bogner et al. (2019), Carrasco et al. (2018), and Neri et al. (2020), which all distill architectural smells for microservices. Bogner et al. (2019) present a systematic literature review identifying and documenting architectural smells in SOA-based architectural styles, including microservices. Although the main focus of their review is on the broader SOA, several smells apply also to microservices. Carrasco et al. (2018) and Neri et al. (2020) instead explicitly focus on microservices, with two multivocal reviews distilling architectural smells for microservices as well as architectural refactorings enabling to resolve such smells. It is however worth noting that the focus of Bogner et al. (2019), Carrasco et al. (2018), and Neri et al. (2020) is on ensuring that the architecture of microservice-based applications complies with the design principles defining microservices themselves. We hence complement the results of their studies, as we focus on another architectural aspect than complying with microservices' design principles, viz., securing microservice-based applications.

Finally, it is worth relating our study with the multivocal literature review by Pereira-Vale et al. (2021), and with the grey literature review by Mao et al. (2020), even if they focused on DevOps rather than on microservices. Pereira-Vale et al. (2021) report the state of art and practice of the security solutions that have been proposed for microservices, and they identified the most used ones. Mao et al. (2020) instead report on the state of practice of DevSecOps, by first overviewing the currently existing risks in classical DevOps practices, and by then illustrating the best practices in DevSecOps and how they enable addressing the security risks of DevOps. The reviews by Pereira-Vale et al. (2021) and by Mao et al. (2020) differ from ours because they focus on the solutions proposed to support securing microservices, whereas we focus on distilling smells that may result in security issues in microservices. At the same time, the results in their reviews and ours complement each other, since, e.g., the security solutions they review can be used to implement the refactorings we describe.

In summary, to the best of our knowledge, there is currently no study classifying the smells that can possibly result in security issues for microservice-based applications, nor eliciting the refactorings that enable mitigating their effects. The latter is precisely the scope of our study, which we have presented in this paper.

6. Conclusions

We presented the results of a multivocal review focused on identifying the smells denoting possible security violations in microservice-based applications, and on the refactorings (proposed by practitioners) enabling to mitigate the effects of such smells. More precisely, we presented a taxonomy organizing ten security smells and ten refactorings. The taxonomy associates each smell with the ISO/IEC 25010 (ISO, 2011) security properties it can violate, and with the refactorings that should all be applied to mitigate its effects. We also provided an overview of the actual recognition of each smell in the selected literature, we discussed the effects of such smells in detail, and we showed how each corresponding refactoring enables mitigating their effects.

Our study reports on what is being said by researchers and practitioners about problems in building secure microservice-based applications. We indeed provide a "snapshot" of the security smells that researchers and practitioners already identified, and of the refactoring enabling to mitigate the effects of such smells. Our study hence complements other existing studies on smells and refactorings for microservices, e.g., Carrasco et al. (2018) and Neri et al. (2020), by covering the security aspects of microservice-based applications. This can have a pragmatic value for practitioners, who can exploit the results of our study in their daily work on securing microservice-based applications. It can also serve as a starting point for researchers wishing to study new solutions for securing microservices or to establish future research directions.

For future work, we plan to study how to identify security smells and enact the corresponding refactorings concretely. More precisely, we plan to elicit concrete examples of security smells from existing microservice-based applications, and investigate which technologies and updates can be applied to implement the refactorings known to mitigate the elicited smells. We then plan to exploit the results of our study to develop an enhanced support for mitigating security smells in microservice-based applications. For instance, we plan to conduct an empirical study to evaluate the impact of the identified security smells on existing microservices-based applications, as well as the efforts needed to actually enact the refactoring allowing to resolve them. We also plan to extend existing languages and tools supporting the design of microservice-based applications (e.g., μTOSCA and μFreshener Brogi et al. (2020), or that by Pigazzini et al. (2020)) to enable detecting security smells automatically, as well as to automatically recommend the refactorings that should be enacted to mitigate the effects of identified security smells. We also plan to relate our work to the technical debt one would experience in microservice-based applications if security smells would appear therein.

Another interesting direction for future work is to consider other quality properties for microservices. It is indeed known that different quality properties of a system can either reinforce or contradict each other, hence requiring to find suitable "tradeoffs" (Bass et al., 2012). Therefore, we plan to systematically analyse the existing literature on smells and refactorings pertaining to other quality properties than security. We also plan to develop a support for helping developers in deciding how to refactor their microservice-based applications to resolve/mitigate the effects of the smells therein, based on acceptable tradeoffs among multiple different quality properties.

CRediT authorship contribution statement

Francisco Ponce: Conceptualization, Methodology, Investigation, Data curation, Writing – Original Draft, Writing - review & editing. **Jacopo Soldani:** Conceptualization, Methodology, Investigation, Data curation, Writing – Original Draft, Writing - review & editing. **Hernán Astudillo:** Conceptualization, Methodology, Investigation, Writing - review & editing, Funding acquisition. **Antonio Brogi:** Conceptualization, Methodology, Investigation, Writing - review & editing, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work has been partly funded by the Dirección de Postgrado *y* Programas of the Universidad Técnica Federico Santa María, Chile.

References

Abasi, F., 2019. Securing Modern API- and Microservices-Based Apps by Design. IBM Developer, https://developer.ibm.com/technologies/api/articles/securing-modern-api-and-microservices-apps-1/.

Almogahed, A., Omar, M., 2021. Refactoring techniques for improving software quality: Practitioners' perspectives. J. Inf. Commun. Technol. 20 (4), 511–539. http://dx.doi.org/10.32890/jict2021.20.4.3.

Balalaie, A., Heydarnoori, A., Jamshidi, P., 2016. Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Softw. 33 (3), 42–52. http://dx.doi.org/10.1109/MS.2016.64.

Basit, T., 2003. Manual or electronic? The role of coding in qualitative data analysis. Educ. Res. 45 (2), 143–154. http://dx.doi.org/10.1080/0013188032000133548.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, third ed. Addison-Wesley Professional.

Behrens, S., Payne, B., 2017. Starting the Avalanche: Application DDoS In Microservice Architectures. The Netflix Tech Blog, https://netflixtechblog.com/starting-the-avalanche-640e69b14a06.

Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F., Prandini, M., 2022. Microservice security: a systematic literature review. PeerJ Comput. Sci. 8, e779. http://dx.doi.org/10.7717/peerj-cs.779.

Boersma, E., 2019. Top 10 Security Traps to Avoid When Migrating from a Monolith to Microservices. Sqreen, https://blog.sqreen.com/top-10-security-traps-to-avoid-when-migrating-from-a-monolith-to-microservices/.

Bogner, J., Boceck, T., Popp, M., Tschechlov, D., Wagner, S., Zimmermann, A., 2019. Towards a collaborative repository for the documentation of service-based antipatterns and bad smells. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE Computer Society, United States, pp. 95–101. http://dx.doi.org/10.1109/ICSA-C.2019.00025.

Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A., 2019. Microservices in industry: Insights into technologies, characteristics, and software quality. In: 2019 IEEE International Conference on Software Architecture Companion (ICSA-C). IEEE Computer Society, United States, pp. 187–195. http://dx.doi.org/10.1109/ICSA-C.2019.00041.

Brogi, A., Neri, D., Soldani, J., 2020. Freshening the air in microservices: Resolving architectural smells via refactoring. In: Yangui, S., Bouguettaya, A., Xue, X., Faci, N., Gaaloul, W., Yu, Q., Zhou, Z., Hernandez, N., Nakagawa, E.Y. (Eds.), Service-Oriented Computing – ICSOC 2019 Workshops. Springer International Publishing, Cham, pp. 17–29.

Budko, R., 2018. Five Things You Need to Know about API Security. The New Stack, https://thenewstack.io/5-things-you-need-to-know-about-api-security/.

Carnell, J., 2017. Spring Microservices in Action, first ed. Manning Publications, Shelter Island, NY, United States.

Carrasco, A., Bladel, B.v., Demeyer, S., 2018. Migrating towards microservices: Migration and architecture smells. In: Proceedings of the 2nd International Workshop on Refactoring. In: IWoR 2018, Association for Computing Machinery, New York, NY, USA, pp. 1–6. http://dx.doi.org/10.1145/3242163.3242164.

Chandramouli, R., 2019. Security Strategies for Microservices-based Application Systems. NIST Special Publication 800-204, http://dx.doi.org/10.6028/NIST.SP.800-204.

da Silva, R.C., 2017. Best Practices to Protect Your Microservices Architecture. Medium, https://medium.com/@rcandidosilva/best-practices-to-protect-your-microservices-architecture-541e7cf7637f.

Di Francesco, P., Lago, P., Malavolta, I., 2018. Migrating towards microservice architectures: An industrial survey. In: 2018 IEEE International Conference on Software Architecture (ICSA). IEEE Computer Society, United States, pp. 29–38. http://dx.doi.org/10.1109/ICSA.2018.00012.

Doerfeld, B., 2015. How to Control User Identity Within Microservices. NordicAPIs, https://nordicapis.com/how-to-control-user-identity-within-microservices/.

Douglas, M., 2018. Microservices Authentication & Authorization Best Practice. CodeBurst, https://codeburst.io/i-believe-it-really-depends-on-your-environment-and-how-well-protected-the-different-pieces-are-7919bfa6bc86.

Edureka, 2019. Microservices security: Best practices to secure microservicess. https://youtu.be/wpA0N7kHaDo.

Esposito, C., Castiglione, A., Choo, K., 2016. Challenges in delivering software in the cloud as microservices. IEEE Cloud Comput. 3 (5), 10–14. http://dx.doi.org/10.1109/MCC.2016.105.

Farace, D., Schöpfel, J., 2010. Grey Literature in Library and Information Studies. K.G. Saur.

Feitosa Pacheco, V., 2018. Microservice Patterns and Best Practices, first ed. Packt Publishing, Birmingham, United Kingdom.

Gardner, Z., 2017a. Security in the Microservices Paradigm. Keyhole Software, https://keyholesoftware.com/2017/03/13/security-in-the-microservices-paradigm/.

Gardner, Z., 2017b. Security in the Microservices Paradigm. DZone, https://dzone.com/articles/security-in-the-microservices-paradigm.

Garousi, V., Felderer, M., Mäntylä, M.V., 2016. The need for multivocal literature reviews in software engineering: Complementing systematic literature reviews with grey literature. In: Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. In: EASE '16, Association for Computing Machinery, New York, NY, USA, pp. 1–6. http://dx.doi.org/10.1145/2915970.2916008.

Garousi, V., Felderer, M., Mantyla, M.V., 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. 106, 101–121. http://dx.doi.org/10.1016/j.infsof.2018.09.006.

Gebel, G., Brossard, D., 2018. Securing APIs and Microservices with OAuth, OpenID Connect, and ABAC. Axiomatics, Webinar, https://www.youtube.com/watch?v=TnCPJUV9RnA.

Ghofrani, J., Lübke, D., 2018. Challenges of microservices architecture: A survey on the state of the practice. In: Herzberg, N., Hochreiner, C., Kopp, O., Lenhard, J. (Eds.), Proceedings of the 10th Workshop on Services and their Composition (ZEUS 2018). CEUR-WS.org, Aachen, Germany, pp. 1–8.

Gupta, N., 2018. Security strategies for devops, apis, containers and microservices. Imperva, https://www.imperva.com/blog/security-strategies-for-devops-apis-containers-and-microservices/.

Hofmann, M., Schnabel, E., Stanley, K., 2016. Microservices Best Practices for Java, first ed. IBM Redbooks, United States.

IETF OAuth Working Group, 2012. Open Authorization (OAuth), Version 2.0. https://oauth.net/2/.

Indrasiri, K., Siriwardena, P., 2018. Microservices security fundamentals. In: Microservices for the Enterprise: Designing, Developing, and Deploying. A Press, Berkeley, CA, pp. 313–345. http://dx.doi.org/10.1007/978-1-4842-3858-5_11.

ISO, 2011. Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models. ISO/IEC FDIS 25010, 2011, 1–34.

Jackson, N., 2017. Building Microservices with Go, first ed. Packt Publishing, Birmingham, United Kingdom.

Jain, C., 2018. Top 10 Security Best Practices to secure your Microservices - AppSecUSA 2017. OWASP, https://youtu.be/VtUQINsYXDM.

Kamaruzzaman, M., 2020. Microservice Architecture and its 10 Most Important Design Patterns. Towards Data Science, https://towardsdatascience.com/microservice-architecture-and-its-10-most-important-design-patterns-824952d7fa41.

Kanjilal, J., 2020. 4 Fundamental Microservices Security Best Practices. SearchAppArchitecture, https://searchapparchitecture.techtarget.com/tip/4-fundamental-microservices-security-best-practices.

Khan, A., 2018. How to Secure Your Microservices: Shopify Case Study. Dzone, https://dzone.com/articles/bountytutorial-microservices-security-how-to-secur.

Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01.

Krippendorff, K., 2004. Content Analysis: An Introduction to its Methodology, second ed. Sage Publications, Thousand Oaks, CA, United States.

Krishnamurthy, T., 2018. Transition to Microservice Architecture - Challenges. BeingTechie, https://www.beingtechie.io/blog/transition-to-microservices-challenges.

Lea, G., 2015. Microservices security: All the questions you should be asking. https://www.grahamlea.com/2015/07/microservices-security-questions/.

Lemos, R., 2019. App Security in the Microservices Age: 4 Best Practices. TechBeacon, https://techbeacon.com/app-dev-testing/app-security-microservices-age-4-best-practices.

Lewis, J., Fowler, M., 2014. Microservices. a definition of this new architectural term. https://martinfowler.com/articles/microservices.html.

Mannino, J., 2017. Security In The Land Of Microservices. AppSec EU 2017, https://www.youtube.com/watch?v=JRmWlLY8MGE.

Mao, R., Zhang, H., Dai, Q., Huang, H., Rong, G., Shen, H., Chen, L., Lu, K., 2020. Preliminary findings about DevSecOps from grey literature. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE Computer Society, United States, pp. 450–457. http://dx.doi.org/10.1109/QRS51102.2020.00064.

Mateus-Coelho, N., Cruz-Cunha, M., Ferreira, L.G., 2020. Security in microservices architectures. In: CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies. In: Procedia Computer Science, Elsevier, Amsterdam, Netherlands, pp. 1–12.

Matteson, S., 2017a. How to Establish Strong Microservices Security using SSL, TLS, and API Gateways. TechRepublic, https://www.techrepublic.com/article/how-to-establish-strong-microservice-security-using-ssl-tls-and-api-gateways/.

Matteson, S., 2017b. 10 Tips for Securing Microservice Architecture. TechRepublic, https://www.techrepublic.com/article/10-tips-for-securing-microservice-architecture/.

McLarty, M., Wilson, R., Morrison, S., 2018. Securing Microservices APIs, first ed. O'Reilly, Newton, MA, United States.

Mody, V., 2020. From Zero to Zero Trust. Teleport, https://goteleport.com/blog/zero-to-zero-trust/.

Nehme, A., Jesus, V., Mahbub, K., Abdallah, A., 2019a. Fine-grained access control for microservices. In: Zincir-Heywood, N., Bonfante, G., Debbabi, M., Garcia-Alfaro, J. (Eds.), Foundations and Practice of Security. Springer International Publishing, Cham, pp. 285–300.

Nehme, A., Jesus, V., Mahbub, K., Abdallah, A., 2019b. Securing microservices. IT Prof. 21 (1), 42–49. http://dx.doi.org/10.1109/MITP.2018.2876987.

Neri, D., Soldani, J., Zimmermann, O., Brogi, A., 2020. Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS Softw.-Intensive Cyber-Phys. Syst. 35 (1), 3–15. http://dx.doi.org/10.1007/s00450-019-00407-8.

Newman, S., 2015. Building Microservices, first ed. O'Reilly, Newton, MA, United States.

Newman, S., 2016. Security and Microservices. Devoxx, https://youtu.be/ZXGaC3GR3zU.

Nkomo, P., Coetzee, M., 2019. Software development activities for secure microservices. In: Misra, S., Gervasi, O., Murgante, B., Stankova, E., Korkhov, V., Torre, C., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Tarantino, E. (Eds.), Computational Science and Its Applications – ICCSA 2019. Springer International Publishing, Cham, pp. 573–585.

O'Neill, L., 2020. Microservice Security - What You Need to Know. CrashTest Security, https://crashtest-security.com/microservice-security-what-you-need-to-know/.

Pahl, C., Jamshidi, P., 2016. Microservices: A systematic mapping study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2. In: CLOSER 2016, SciTePress, Setúbal, Portugal, pp. 137–146. http://dx.doi.org/10.5220/0005785501370146.

Parecki, A., 2019. OAuth: When Things Go Wrong. Okta Developer, https://www.youtube.com/watch?v=H6MxsFMAoP8.

Pereira-Vale, A., Fernandez, E.B., Monge, R., Astudillo, H., Márquez, G., 2021. Security in microservice-based systems: A multivocal literature review. Comput. Secur. 102200.

Perera, S., 2016. Walking the Wire: Mastering the Four Decisions in Microservices Architecture. Medium, https://medium.com/systems-architectures/walking-the-microservices-path-towards-loose-coupling-few-pitfalls-4067bf5e497a.

Pigazzini, I., Fontana, F.A., Lenarduzzi, V., Taibi, D., 2020. Towards microservice smells detection. In: Proceedings of the 3rd International Conference on Technical Debt. In: TechDebt '20, Association for Computing Machinery, New York, NY, USA, pp. 92–97. http://dx.doi.org/10.1145/3387906.3388625.

Ponce, F., 2021. Towards resolving security smells in microservice-based applications. In: Zirpins, C., et al. (Eds.), Advances in Service-Oriented and Cloud Computing. Springer International Publishing, Cham, pp. 133–139. http://dx.doi.org/10.1007/978-3-030-71906-7_11.

Radware, 2020. Microservice architectures challenge traditional security practices. https://blog.radware.com/security/2020/01/microservice-architectures-challenge-traditional-security-practices/.

Raible, M., 2020a. Security Patterns for Microservice Architectures. Okta Developer, https://developer.okta.com/blog/2020/03/23/microservice-security-patterns.

Raible, M., 2020b. 11 Patterns to Secure Microservice Architectures. DZone, https://dzone.com/articles/11-patterns-to-secure-microservice-architectures.

Rajasekharaiah, C., 2020. Cloud-Based Microservices: Techniques, Challenges, and Solutions, first ed. A Press, New York, NY, United States.

Richter, D., Neumann, T., Polze, A., 2018. Security considerations for microservice architectures. In: Proceedings of the 8th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,. SciTePress, Setúbal, Portugal, pp. 608–615. http://dx.doi.org/10.5220/0006791006080615.

Sahni, V., 2019. Best Practices for Building a Microservice Architecture. Vinay Sahni, https://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture.

Sass, R., 2017. Security in the World of Microservices. ITProPortal, https://www.itproportal.com/features/security-in-the-world-of-microservices/.

Sharma, S., 2019. Mastering Microservices with Java, third ed. Packt Publishing, Birmingham, United Kingdom.

Siriwardena, P., 2014. Mutual authentication with TLS. In: Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE. A Press, Berkeley, CA, pp. 47–58. http://dx.doi.org/10.1007/978-1-4302-6817-8_4.

Siriwardena, P., 2019. Microservices Security Landscape. WSO2 Integration Summit 2019, https://youtu.be/6jGePTpbgtI.

Siriwardena, P., 2020. Challenges of Securing Microservices. Medium, https://medium.facilelogin.com/challenges-of-securing-microservices-68b55877d154.

Siriwardena, P., Dias, N., 2020. Microservices Security in Action, first ed. Manning Publications, Shelter Island, NY, United States.

Smith, T., 2017. How do you Secure Microservices?. DZone, https://dzone.com/articles/how-do-you-secure-microservices.

Smith, T., 2019. How to Secure APIs. DZone, https://dzone.com/articles/how-to-secure-apis.

Soldani, J., Tamburri, D.A., Van Den Heuvel, W.-J., 2018. The pains and gains of microservices: A systematic grey literature review. J. Syst. Softw. 146, 215–232. http://dx.doi.org/10.1016/j.jss.2018.09.082.

SumoLogic, 2019. Improving security in your microservices architecture. https://www.sumologic.com/insight/microservices-architecture-security/.

Taibi, D., Lenarduzzi, V., 2018. On the definition of microservice bad smells. IEEE Softw. 35 (3), 56–62. http://dx.doi.org/10.1109/MS.2018.2141031.

Taibi, D., Lenarduzzi, V., Pahl, C., 2018. Architectural patterns for microservices: A systematic mapping study. In: Proc. of the 8th Int. Conf. on Cloud Computing and Services Science - Volume 1: CLOSER. SciTePress, Setúbal, Portugal, pp. 221–232. http://dx.doi.org/10.5220/0006798302210232.

Thönes, J., 2015. Microservices. IEEE Softw. 32 (1), 116. http://dx.doi.org/10.1109/MS.2015.11.

Troisi, M., 2017. 8 Best Practices for Microservices App Sec. TechBeacon, https://techbeacon.com/app-dev-testing/8-best-practices-microservices-app-sec.

Wallarm, 2019. Shift to microservices: Evolve your security practices & container security. https://lab.wallarm.com/shift-to-microservices-evolve-your-security-practices-container-security/.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Amsterdam, Netherlands.

Wolff, E., 2016. Microservices: Flexible Software Architecture, first ed. O'Reilly, Newton, MA, United States.

Yarygina, T., Bagge, A., 2018. Overcoming security challenges in microservice architectures. In: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE). IEEE Computer Society, United States, pp. 11–20. http://dx.doi.org/10.1109/SOSE.2018.00011.

Yu, D., Jin, Y., Zhang, Y., Zheng, X., 2019. A survey on security issues in services communication of Microservices-enabled fog applications. Concurr. Comput.: Pract. Exper. 31 (22), e4436. http://dx.doi.org/10.1002/cpe.4436.

Ziade, T., 2017. Python Microservices Development, first ed. Packt Publishing, Birmingham, United Kingdom.

Zimmermann, O., 2017. Microservices tenets. Comput. Sci. - Res. Dev. 32 (3), 301–310. http://dx.doi.org/10.1007/s00450-016-0337-0.

Francisco Ponce is a Ph.D. candidate at Universidad Técnica Federico Santa María, Chile. He got his Computer Engineer degree in 2016 from the Universidad de Valparaíso, Chile. His current research interests include microservices, security smells and refactorings, software architecture, and software engineering. Before becoming a Ph.D. student, he worked in the banking industry and as a Mobile developer. Contact him at francisco.ponceme@usm.cl

Jacopo Soldani is a tenure-track assistant professor at the Department of Computer Science of the University of Pisa. He received the Ph.D. degree in Computer Science in 2017 from the University of Pisa (Italy). His current research interests include services, cloud, and fog computing, with a particular focus on microservices, cloud-native applications, and faults and fault-resilience. He has been involved in multiple research projects on the orchestration of service-based applications on cloud and fog platforms. Contact him at jacopo.soldani@unipi.it

Hernán Astudillo is professor of software engineering at Universidad Técnica Federico Santa María, Chile since 2003. He got his Ph.D. in Computer Science and Information (Georgia Tech, 1995) and was Sr. Application Architect at MCI Systemhouse and later at Financial Systems Architects (NYC). He was founding president of ArquiTIC (Chilean association of architects of information technologies), and has represented Chilean universities to CLEI (Latin American Association of Informatics Departments) and also CLEI itself in IFIP TC2 (Software Engineering). Contact him at hernan@inf.utfsm.cl

Antonio Brogi is a full professor at the Department of Computer Science, University of Pisa (Italy) since 2004. He holds a Ph.D. in Computer Science (1993) from the University of Pisa. His research interests include service-oriented, cloud-based and fog computing, coordination and adaptation of software elements, and formal methods. He has published the results of his research in over 150 papers in international journals and conferences. Contact him at antonio.brogi@unipi.it