Computers & Security 127 (2023) 103119

Contents lists available at ScienceDirect

Computers & Security

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

SoK: Run-time security for cloud microservices. Are we there yet?☆

Francesco Minna[a,*], Fabio Massacci[a,b]

[a] Vrije Universiteit Amsterdam, Netherlands
[b] University of Trento, Italy

ARTICLE INFO

Article history:
Received 11 September 2022
Revised 19 December 2022
Accepted 22 January 2023
Available online 24 January 2023

Keywords:
Survey
Microservices
Containers
Security
Cloud

ABSTRACT

The adoption of microservice architecture is rapidly growing, involving industries of every size. Their ability to scale and reconstitute complex functionalities into small, cohesive, and interconnected components (the microservices), and their limited use of isolation contribute to this success. Unfortunately but unsurprisingly, these very factors enlarge the attack surface and increase the security risks of today's deployments. In this study, we performed a systematization of knowledge about the run-time security of microservices. Starting from a keyword search, we initially reviewed 807 papers available in digital libraries (e.g., Google Scholar and Scopus), which we filtered down to 48 by applying a number of selection criteria (e.g., the presence of a proof-of-concept implementation). We also considered over 30 industry tools that offer various security services for microservices. We categorized both papers and tools and highlighted areas where research is abundant, where it is lacking, and where it is misleading. We conclude that the run-time security of microservices is still in its infancy and we supplement our analyses with insights into addressing the key challenges.

1. Introduction

Microservices are becoming the "norm" for modern applications (Global Industry Analysts, 2019) as they enable resources to be scaled up to meet a surge in demand, counter failures, orchestrate routine maintenance, and continuously develop, integrate, and deploy applications as ensembles of small, well-tested, and easy-to-manage components. Yet, having multiple services communicating among each other over a network, usually through REST APIs and possibly developed with different technologies, enlarges the attack surface (Nkomo and Coetzee, 2019) and raises new security issues — from vulnerabilities in software packages used by container images to insecure configurations, unto common network attacks, such as misconfigured authentication or man-in-the-middle (MITM) attacks. In this paper we perform a systematization of knowledge investigating the run-time security of microservices, highlighting the strengths and weaknesses of both research approaches and industry tools. We discuss key challenges for future work and unexpected insights. Although this study is explicitly focused on microservices, the inferences may also apply to monolithic applications deployed on containers. In the latter case, the number and size of containers are irrelevant. A similar argument applies to cloud and SaaS platforms (e.g., AWS Lambda functions). Security-wise, such platforms are essentially similar to microservices running atop a container or VM, with the only difference being who is in charge of the security of such containers (i.e., the cloud platform or the company's DevOps). Also, we do not discuss low-level components (or architectural layers) of the cloud stack: they are beyond the scope of our study. This SoK study focuses on the following three research questions.

- **RQ1**: *Which architectural layer and which security properties are attacked or defended within microservice architectures?*
  We classify each contribution based on the entry-point to which it refers (e.g., host OS or container image) and the tested property (e.g., authentication and authorization).
- **RQ2**: *Do a proof-of-concept (PoC) actually exists?*
  We found out that several papers are just "evocative security architectures". When discussing security a key aspect we are interested in is the type of security services that are proposed (between defenses and attacks).
  RQ3: *What is the effectiveness of industrial tools?*
  The effectiveness of certain tools may be misleading. Good examples are tools for container image static analysis. To anybody familiar with software security, the terminology evokes a cer-

☆ This work has received funding from the European Union under the H2020 grant 952647 (AssureMOSS).
* Corresponding author.
E-mail addresses: f.minna@vu.nl (F. Minna), fabio.massacci@ieee.org (F. Massacci).

https://doi.org/10.1016/j.cose.2023.103119

tain level of "depth" of the analysis, and not just the result of a database join.

In answering these research questions, we make the following contributions.

- We first introduce microservices, containers, and the architectural layers used to classify microservice solutions (§2).
- We discuss previous surveys (§3) and detail the methodology used for this SoK study (§4).
- We answer each research question by categorizing the prior work that we found, discussing the advantages and limitations of current solutions, and highlighting new challenges for each area.

We conclude this study by discussing the limitations and threats to the validity of this work (§8) and presenting a brief roadmap for future work (§9).

2. Background

A *microservice* architecture splits an application into many cohesive components, each of which can be designed, implemented, and deployed independently (Fowler and Lewis, 2014; Newman, 2015). These components, or microservices, interact with one another typically through well-defined APIs. Microservices, therefore, significantly deviate from the traditional monolithic architecture: a change in one component of a monolithic application necessitates re-deploying all the components, while a microservice architecture allows only the changed component to be re-deployed. A microservice is typically executed in a *container*, which comprises a set of processes along with their dependencies. In contrast to virtual machines (VMs), which emulate the entire hardware and software (i.e., all the hardware – the hypervisor – and the kernel), containers are lightweight, emulating only the OS (and sharing the same kernel), which in turn could run either on a VM or directly on "bare-metal" servers (or physical hardware). An application typically consists of several microservices, and deploying, managing, scaling, and networking the containers hosting each of these microservices are non-trivial tasks, hence these tasks are usually automated using container orchestrators, such as Kubernetes. Along with orchestration software, other tools are used to ease the deployment of microservices, among which are the Container Network Interface (CNI) *plugins* and *service mesh* tools. Essentially, a network plugin sets up the virtual network (i.e., configuring the network interface and IP addresses) to which the containers instantiated by the container engine will be connected.[1] To provide network observability and enforce network policies, plugins often rely on *extended Berkeley Packet Filter (eBPF)-based* programs. Essentially, eBPF is a programmable virtual machine running in the Linux kernel that can execute user-defined code. It acts as a man-in-the-middle service between containers and the kernel, allowing the users the execute actions on system calls (e.g., opening a file or a network socket) made by containers. In such a way, eBPF provides observability, monitoring, security (e.g., policies), and networking functionalities on the host where the containers are running. A service mesh is an additional way to provide network configuration and functionalities such as discovering new services and managing, filtering (through network policies), monitoring (up to layer 7), load balancing, and encrypting the network traffic. Such functionalities are provided by a proxy or *sidecar*, which runs beside each service. All traffic received and sent by the service is routed through the sidecar, at the price of resource overhead.

[1] The CNI, a Cloud Native Computing Foundation (CNCF) project, is a template for writing network plugins aiming at providing connectivity between a container engine and a network. Various CNI plugins, discussed in §7, provide connectivity through different interfaces (e.g., bridge, ipvlan, and macvlan).

3. Related surveys

Security risks affecting the microservice architecture have been already investigated. Pereira-Vale et al. (2019) surveyed the security mechanisms proposed in prior work between 2015 and 2018, and they found that most mechanisms focus on stopping or mitigating attacks, and not on recovery. From a software development perspective, Nehme et al. (2019) analyzed the security threats relevant to microservices in different stages of the development lifecycle, while Waseem et al. (2021) performed an empirical study of over 1000 issues from five open-source microservice-based projects on Github. They identified common issues (e.g., security and configuration issues) and the corresponding causes (e.g., legacy version, compatibility, and dependency problem). In terms of deployment and operations, Yu et al. (2019) highlighted security issues in secure data exchanges between services, access-control mechanisms, and the network protocols used for facilitating communication between services. Hannousse and Yahiouche (2020) classified the threats, risks, and vulnerabilities as well as some defensive mechanisms for tackling them. Pereira-Vale et al. (2021) performed a literature review of the most used security mechanisms in microservice-based applications, reviewing both academic articles and, the so-called, "grey" literature sources (e.g., blog posts). They analyzed the articles' publication times, classifying the proposed solution (e.g., Access control or Secure communication) and scope (e.g., implementation security or security evaluation), and identifying missing gaps. Other surveys, such as Sultan et al. (2019), made strong assumptions on what the attacker has *already* been able to achieve before launching the attack (i.e., the "rogue" container model).

Key Takeaway. The techniques for (continuously) evaluating, mitigating, and recovering from run-time security threats in a microservice-based architecture still remain largely unchartered, and there are no maps for the concrete existence of proof of concepts and benchmarks for validating and verifying those techniques.

4. Methodology

We hereby present the different steps of the conducted survey to find and classify available tools and techniques for evaluating, mitigating, and recovering from run-time security issues affecting microservices, by following guidelines similar to those in Kitchenham and Charters (2007).

Classification Dimensions (Fig. 1)

We assess previous work along some independent dimensions to allow a uniform comparison of the contributions. In particular, the following are the four dimensions by which we classify prior contributions and answer our research questions.

- *Architectural Layers*: due to the many layers involved in the security of cloud deployments (e.g., infrastructures, clusters, containers, and code) we want to relate each contribution to the security layer it applies to Souppaya et al. (2017), Documentation (2021).
- *Security Properties*: besides the type of implemented prototype and the architectural layer it refers to, we also classify the security property (one or more) a contribution guarantees or compromises. Our classification is loosely based on NIST (2017).
- *Solutions*: we classify a contribution based on what kind of prototype was implemented or the PoC that was demonstrated.
- *Verification & Validation*: to qualify the quality of contribution and replicability of results, we also want to classify the verification and validation methods being used.

Similarly to academic paper contributions, we also assess industry tools using the previously defined dimensions based on the

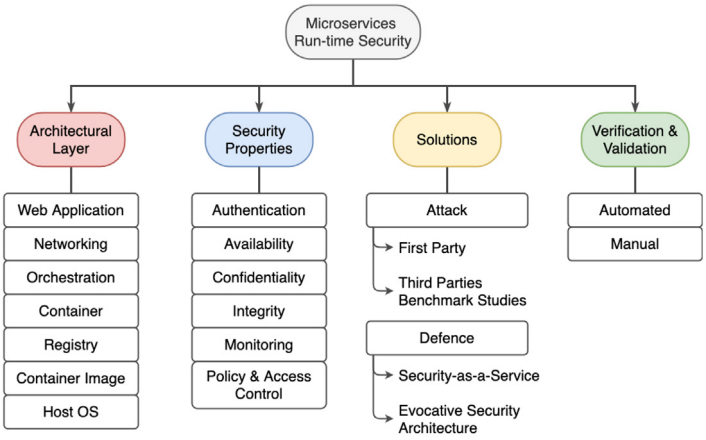

Fig. 1. The proposed multidimensional taxonomy to classify scientific papers and existing tools related to microservices run-time security.

specific functionalities they provide (e.g., analysis of configuration files or visualization of network traffic).

Regarding our classification, the first two dimensions are a natural classification well understood and, for sake of conciseness, we do not discuss them further. The *Solutions* dimension is specific to security to illustrate the typology of PoC contribution. To belong to a specific dimension, a contribution has to meet the following criteria:

- *Attack*: to qualify as a first party attack, a contribution must actually present (or, for tools, be able to launch) an attack in a microservice deployment while complying with our threat model (i.e., no prior access to the environment).
- *Defence*: contributions belonging to this sub-dimension present a defence mechanism that circumvents or detects one or more attacks and CVEs in a microservice deployment. For example, defence mechanisms that prevent CVEs, exploitation of kernel bugs to compromise containers, DoS attacks, or anomaly detectors, classify here.

We further categorized contributions belonging to the "Attack" category as *Benchmark Studies* if they analyze one or more *third party* microservice applications with conclusions applicable to general deployments. Similarly, we further categorized contributions belonging to the "Defence" category as *Security-as-a-Service*, and *Evocative Security Architecture*. *Security-as-a-Service* contributions must provide or guarantee one or more security properties (e.g., confidentiality, integrity, or intra-service network authentication) on top of existing deployments, and a PoC must exist. We also classify *monitoring* under this sub-dimension. The classification of monitoring as a security service is open to debate. We interpreted it as providing services akin to intrusion and anomaly detection and therefore included it within defence mechanisms. Instead, contributions belong to the *Evocative Security Architecture* sub-dimension when it is unclear whether a PoC actually exists, although they might discuss security issues and defence mechanisms (e.g., reporting run-time plots of simulations).

Regarding the "Verification & Validation", we classify contributions into two sub-dimensions, namely, automated and manual testing. We also consider a third sub-dimension, "Not Tested?", for papers where it was unclear whether they have been tested at all. We further categorized "manual testing" into three sub-dimensions: injection (e.g., manually injecting faults or vulnerabilities), generation of script or test cases, and simulation of attacker's presence. Similarly, we also categorized "automated testing" into three sub-dimensions: vulnerability management (e.g., vulnerability detection or fixing), performance simulation (i.e., without real traffic or application deployment), and performance measurement (e.g., using a synthetic dataset, real traces, or interacting with the application).

Scoping Criteria: Threat Model and Implementation

We define a list of criteria to select (*only*) relevant work and retain a paper only if it matches *all* criteria below. At first, we look at the threat models considered in the papers and for papers that match the following two assumptions on the adversaries.

TM1:ExternalAttacker. We assume that an attacker has *not* yet gained any initial access into the system. For example, we discard "container-escape" attacks, where an attacker already gained access to a container (e.g., through a reverse shell), or attacks that start from un-privileged accounts within the cloud environment; these are privilege escalation attacks in which the container is only the attack vector to compromise the host system.

TM2:Container. We do not consider web application, host OS or API gateways-related attacks because vulnerabilities affecting these layers are not specific to the microservice architecture. For example, a remote code execution (RCE) vulnerability in a web application could be exploited regardless of how the application is deployed — on containers, VMs, or on a monolithic server.

Therefore, we focus on attack vectors that do not make (initial access) assumptions or could be exploited regardless of the deployment architecture type; instead, we do consider attacks such as cloud misconfigurations as compliant with our threat model (from a report released in 2020 by the NSA (2020), misconfiguration is the most common vulnerability in cloud environments).

As a second set of criteria, we identify at first the type of security that is considered (run-time vs static time) and further whether a proof of concept exists. To be selected, papers must match both criteria below.

C1:RunTime. The contribution focuses on run-time security aspects of microservice deployments. In practice, this criterion filters papers that only propose new architectural solutions or design principles for microservice-based applications (as opposed to studying the security of the microservice itself).

C2:PoCExists. The work must include a PoC implementation. If it reports on testing an (already existing) prior work in an environment different from where it was originally evaluated, it must clearly specify the new environment where it was tested (or benchmarked) and include tangible test results.

With respect to our methodology, we propose a four-dimension taxonomy to classify relevant prior work, shown in Fig. 1. The dimensions are, namely, *Architectural Layers, Security Properties, Solutions*, and *Verification & Validation*.

Paper Search and Selection Criteria

Keywords-based search. We defined a list of keyword strings used to find previous relevant work on digital libraries. An example

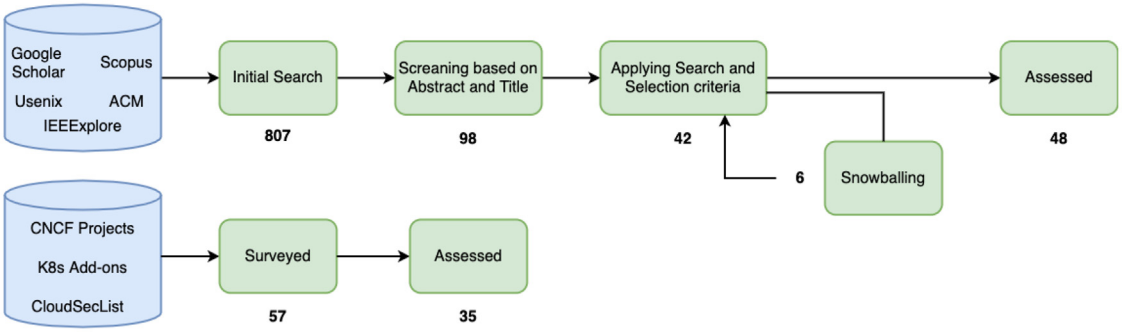

Fig. 2. The different steps of the survey methodology along with the number of publications selected at each step.

string is: ""microservice" AND ("run-time" OR "runtime" OR "dynamic") AND ("security" OR "metrics")". The complete list of keywords, including their impact on the discovery of relevant contributions, are in the appendix Appendix A.

Time constraint. We consider previous work only within the last eleven years, since the term "microservice" was introduced in 2011 (Dragoni et al., 2017).

Digital Libraries. As digital libraries, we consider Scopus, Google Scholar, ACM DL, and Usenix, using the keywords previously defined. As a control mechanism, we also recommend searching on IEEExplore. We searched as an anonymous user (i.e., without logging into an account) on Google Scholar, and sorted the search results based on "Relevance" on Scopus and ACM DL; also, we ignored contributions after a certain number of search results as they start to have no relevance or be out of scope (e.g., how to build a blockchain with microservices).

Citation Closure. To include all relevant contributions, we may expand the set of prior work by *snowballing* (Wohlin, 2014), reviewing references within each paper and repeating the procedure to decide their inclusion by applying the previously defined criteria.

Compendium of Security Tools

The methodology allows to also classify tools available on the market that perform either static or run-time security checks, scans, monitoring, or tests of containers and related artefacts — images, configuration files, instances, and orchestration environments, on which microservices are usually deployed. We consider a tool only if it matches *all* criteria below:

C3:Available. Open-source tools must be available to download from an online repository (e.g., Github), whereas proprietary tools must be available to download (perhaps after purchasing a license).

C4:Active. Open-source tools must have been updated after January 2022, while proprietary tools must be actively supported by the company (demonstrated via recent FAQs or official documentation).

We classify industrial tools in a separate section (§7) because the solutions (i.e., PoC and prototypes) from scientific papers are usually not usable in practice, if not to replicate the results of the paper itself. Using the same dimensions to classify the scientific paper solutions or PoC, namely, Attack, Defence, and Security-as-a-Service, we classify proprietary and open-source, static and dynamic, tools based on four additional subcategories, namely, *configuration analyzers, vulnerability scanner, network visualizers*, and *security monitors*. Tools that do not directly perform any security checks, but visually offer insights that can help in troubleshooting an issue can be classified as *network visualizers*. The remaining three dimensions represent tools that inspect one or more security aspects of a microservice deployment. *Security monitors* inspect, for instance, the security or networking policies of a deployment. *Configuration analyzers* and *vulnerability scanners* analyze the configuration files and settings or report vulnerabilities in software packages within container images.

Static-time tools (i.e., configuration analyzers and vulnerability scanners) belong to the Security-as-a-Service dimension (e.g., tools that highlight vulnerabilities in software packages or misconfigurations), whereas run-time tools (i.e., network visualizers and security monitors) may belong to all three dimensions (i.e., Attack, Defence, and Security-as-a-Service), depending on the service they offer (e.g., penetration testing or policy enforcement).

Tools Search and Selection Criteria

To look for microservice security-related tools, we mostly referred to the next three following sources:

- *Graduated and incubating CNCF projects*: list of cloud-related projects belonging to the CNCF.[2]
- *List of Kubernetes Addons*: list of addons to extend Kubernetes's functionalities.[3]
- *The Cloud Security Reading List*: weekly grey-literature newsletter where new tools, blog posts, and cloud platform features are published [4].

Following the same principle of snowballing for relevant contributions, we applied the same principle to extend the set of available tools by looking at each tool's alternatives, both using search engines and the tool's documentation.

Results

In Fig. 2, we illustrate the steps along with the corresponding impact on the discovery of relevant publications.

We discovered a substantial collection of 807 papers from digital libraries. Despite this very large number of papers, after reading the title and abstract, we retained only 98 papers that we later on pruned to a set of 42 by further reviewing them and applying the selection criteria. The main reason behind this sharp drop in papers is that most of them were out of the scope of this study (e.g., papers about how to transition from a monolith to a microservice architecture). The rest were mostly about (microservices) performance (e.g., network and scaling performance) or did not include a PoC of the proposed solution. We expanded the corpus with another 6 publications via snowballing. Of the 48 papers, 42 are primary studies (proposing attacks, defences, etc.) and 6 are secondary studies (surveys reporting other studies). In the following sections, we classified the found publications using the multidimensional taxonomy previously proposed.

Regarding industrial tools, we found 15 static-time related tools, in particular, 8 configuration analyzers and 7 vulnerability scanners, and 18 run-time related tools, also classified using the multidimensional taxonomy previously proposed.

[2] https://www.cncf.io/projects/
[3] https://kubernetes.io/docs/concepts/cluster-administration/addons/
[4] https://cloudseclist.com/

Table 1
Classification of prior work based on which architectural-layer and security properties issues they address. The majority of prior work focuses on networking services such as monitoring and access control (i.e., controlling access to network services). Orchestration and registry layers, in contrast, have no relevant contributions. Consequently, there are no studies on security issues concerning information flow among services. We omit the host OS layer since OS attacks require prior access by the attacker, which does not comply with our threat model (refer to §).

Layer	Authentication	Availability	Confidential.	Integrity	Monitoring	Policy & AC
Web Application	Baker and Nguyen (2019)	Baarzi et al. (2020); Pietrantuono et al. (2018, 2020); Torkura et al. (2019)	-	-	Yarygina and Otterstad (2018); Zhou et al. (2019)	Torkura et al. (2017, 2018)
Networking	Baker and Nguyen (2019), Preuveneers and Joosen (2017), George and Mahmoud (2017), Jander et al. (2018), Pahl and Donini (2018), Zaheer et al. (2019), Díaz-Sánchez et al. (2019), Yarygina and Bagge (2018), Ranjbar et al. (2017), Li et al. (2021), Walsh and Manferdelli (2017)	Pahl and Donini (2018), Meinke and Nycander (2015), Wu et al. (2018), Heorhiadi et al. (2016), Jin et al. (2021), Li et al. (2020), Torkura et al. (2019), Liu et al. (2021)	George and Mahmoud (2017), Osman et al. (2019), Gerking and Schubert (2019), Jander et al. (2018)	Gerking and Schubert (2019), Jander et al. (2018), Pahl and Donini (2018), Zaheer et al. (2019), Sun et al. (2016), Díaz-Sánchez et al. (2019)	Sun et al. (2016), Phipathananunth and Bunyakiati (2018), Suneja et al. (2019), Yarygina and Otterstad (2018), Wang et al. (2020), Guan et al. (2019), Meng et al. (2021), Chen et al. (2019), Pahl and Aubet (2018), Zhou et al. (2019), Ibrahim et al. (2019), Ma et al. (2021), Chen et al. (2021b), Cai et al. (2021)	Pahl et al. (2018), Baker and Nguyen (2019), George and Mahmoud (2017), Gerking and Schubert (2019), Sun et al. (2016), Preuveneers and Joosen (2017), Zaheer et al. (2019), Ranjbar et al. (2017), Li et al. (2019), Fadhel et al. (2018)
Orchestration	-	-	-	Melara and Bowman (2021)	Lim et al. (2021)	- *Container* - Jin et al. (2021), Meinke and Nycander (2015) - - Suneja et al. (2019), Zhou et al. (2019), Wang et al. (2020), Guan et al. (2019), Chen et al. (2021a) Zhu and Gehrmann (2021), Lim et al. (2021)
Container Image	-	-	-	-	Ibrahim et al. (2019)	Torkura et al. (2018)

5. RQ1: Architectural layers and security properties

We report in Table 1 the classification of prior contributions based on their target architectural layer along with the security properties issues they addressed.

Architectural Layers

Following the discussion of prior work based on architectural layers they referred to; identifying the architectural layer of prior contributions is somewhat easy.

Networking Most of the discovered contributions focus on the networking layer and the majority of them focus on defining and enforcing security policies (e.g., for users or among services). There exist several networking plugins (discussed later in §7) that can automatically generate networking policies by monitoring the traffic flows between microservices automating an otherwise tedious and error-prone process. Although the generated policies are simple (point-to-point), they require interaction with the application (tricky with many microservices) and is not clear what parameters are taken into account to generate such policies. Besides, no known tools and techniques exist to test the correctness of policies, their robustness against attacks, or their capabilities in containing rogue services.

Key Takeaway. Many techniques process and enforce security policies, albeit no prior work, clearly delineate what constitutes a "right" policy. Hence, the efficacy and correctness of security policies are an unmet challenge.

Registry & Orchestration There are several well-known attacks on exploiting misconfigurations or container-engine vulnerabilities. Per one estimate, more than 40000 container-hosting devices simply have the default configuration for their containers (Quist, 2019). Several CVEs[5] and vulnerabilities (Chen, 2020) associated with the registry layer have been discovered. These attack vectors can be easily exploited by attackers for jeopardizing microservice deployments. We found only one paper related to the Orchestration layer that allows defining security policies that are later on enforced by the orchestration software (e.g., on which node a service should be deployed).

Key Takeaway. Despite the issues and vulnerabilities related to the registry and orchestration layers, there exist very little prior work on investigating them nor robust design approaches to address their issues.

Container Image We found six papers belonging to this layer, although none present a real exploit. Even though typical

[5] https://nvd.nist.gov/vuln/detail/CVE-2019-16097

container-image scanning tools could detect such vulnerabilities, they often fall short in continuously monitoring the image of a running container evolving over time. We did not find any tool or technique to address the challenge of continuously monitoring at run-time.

Key Takeaway. Instead of treating a container image as static, tools and techniques for checking container-image vulnerabilities must take into account that the image may change (or evolve) over time. We need mechanisms to continuously monitor and detect vulnerabilities in a running container, and eventually, contain them within a sandbox.

Security Properties. Following, instead, the discussion on security properties prior contributions addressed. Here we just provide a bird's eye view of which are the most popular security properties for each paper; additional details are provided when discussing the type of solutions and the level of V&V in §6.

Authentication 10 papers belonging to this dimension: George and Mahmoud (2017), Ranjbar et al. (2017), Pahl and Donini (2018), Zaheer et al. (2019), Jander et al. (2018), Díaz-Sánchez et al. (2019), Baker and Nguyen (2019), Yarygina and Bagge (2018), Preuveneers and Joosen (2017), Walsh and Manferdelli (2017). Most container or microservice-based deployments (especially generic services) don't really authenticate/authorize at the "end-user identity" level; rather, they typically authenticate/authorize at the service identity level.

Availability 12 papers belonging to this dimension: Wu et al. (2018), Heorhiadi et al. (2016), Meinke and Nycander (2015), Zhou et al. (2019), Liu et al. (2021), Baarzi et al. (2020), Li et al. (2020), Pietrantuono et al. (2018), Pietrantuono et al. (2020), Jin et al. (2021), Torkura et al. (2019), Pahl and Donini (2018). Despite the rich literature on DoS mitigation techniques for the cloud (Bakr et al., 2019), there are only two techniques explicitly focused on microservices. No prior work has attempted to correlate scalability with the cost for cloud environments. We found only one paper (Martinez et al., 2017) proposing a methodology to constraint memory consumption of containers, container sub-processes, and to deallocate containers' memory after use.

Key Takeaway. We need DoS defense mechanisms for microservice deployments that cater to trade-offs between scalability and cost-efficiency.

Confidentiality & Integrity 11 papers belonging to these dimensions: George and Mahmoud (2017), Jander et al. (2018), Osman et al. (2019), Gerking and Schubert (2019) about Confidentiality and Jander et al. (2018), Zaheer et al. (2019), Díaz-Sánchez et al. (2019), Pahl and Donini (2018), Sun et al. (2016), Gerking and Schubert (2019), Melara and Bowman (2021) about Integrity. There is no work on remediation, as also noted in Pereira-Vale et al. (2019). Once an attacker gains entry into a container, defense mechanisms are wanting.

Key Takeaway. Remediation and recovery are open challenges, as well as providing integrity at the orchestration and registry layers.

Monitoring 16 papers belong to this dimension: Wang et al. (2020), Guan et al. (2019), Baarzi et al. (2020), Meng et al. (2021), Zhou et al. (2019), Ma et al. (2021), Chen et al. (2021b), Cai et al. (2021), Chen et al. (2019), Pahl and Aubet (2018), Sun et al. (2016), Phipathananunth and Bunyakiati (2018), Lim et al. (2021). Two papers belong to the *Penetration testing* sub-dimension (Ibrahim et al., 2019; Suneja et al., 2019), whereas one to the *Tracking* sub-dimension (Chen et al., 2021a).

Policy & Access Control 12 papers belonging to this dimension: Preuveneers and Joosen (2017), Gerking and Schubert (2019), Sun et al. (2016), Li et al. (2021), Torkura et al. (2017), Torkura et al. (2018), Fadhel et al. (2018), Zaheer et al. (2019), Ranjbar et al. (2017), Li et al. (2019), Zhu and Gehrmann (2021), Pahl et al. (2018). **Key Takeaway.** Automatic generation of networking policies and their subsequent testing, both at static and runtime, still remain largely unexplored.

Key Takeaway. Although there are several works testing defenses, the practice for researchers proposing new security services for microservice architectures seems to be focusing on the performance of those solutions, rather than on efficacy against attacks.

6. RQ2: Solutions and V&V methodologies

Within this section, we discuss implemented solutions and verification and validation methodologies used in prior work. In particular, hereby we only discuss contributions from *scientific papers*, whereas contributions from industrial tools are discussed in the next section (§7).

Solutions Following a discussion on implemented PoCs and prototypes.

Attack Although there is prior work that demonstrated or analyzed practical attacks on microservices (e.g. Gao et al., 2017; Otterstad and Yarygina, 2017), they all assume that the attacker already has access to one or more components of the deployment, which does not satisfy our threat model (refer to §4). Conceptually, they are identical to traditional privilege escalation attacks, and some have already been analyzed (Sultan et al., 2019). Belonging to the first party attack dimension, only the work by Ibrahim et al. (2019) investigated (theoretical) attacks on microservices by using vulnerabilities identified in a deployment to build an attack graph of all possible attacks. Whether these attacks are actually possible has not been tested.

Key Takeaway. Assuming prior access (e.g., a shell to deployed containers) simplifies the complexity of attack technologies but narrows the scope of most prior work. Attacks without such assumptions are common in practice but have not been widely studied in the literature.

Benchmark Studies We found only two contributions under this dimension. Baker and Nguyen (2019) tested some microservices, implemented using the Spring Security Framework, against three common OWASP top-10 attacks, namely cross-site request forgery (CSRF), cross-site scripting (XSS), and brute force attacks. Kumar and Trivedi (2021) provides a detailed performance (and not security) comparison of several CNI plugins in Kubernetes.

Key Takeaway. There are no security benchmarks for orchestration software, e.g., Kubernetes or Docker swarm, despite being the target of several real-world attacks.

Defence We found 19 contributions as defenses that we summarize in Table 2, by reporting (a) the corresponding contribution, (b) what assumptions, if any, they made, (c) what aspects were tested (e.g., vulnerability detection performance), (d) where, or in what kind of environment, were they tested (e.g., with how many microservices), and, finally, (e) whether they can be reproduced.[6] Unfortunately, many papers do not adequately specify the details of their evaluation environment.

Key Takeaway. Evaluation environments are seldom described in sufficient detail to be fully reproducible. Reproducibility is further hindered because only a few defense mechanisms implementations are available.

Security-as-a-Service We found 13 papers under this dimension; they usually present PoCs that prove the feasibility and efficiency of the proposed methodologies to guarantee or provide one or more security properties. Six solutions for policy & access control. Specifically, a tool to automatically generate access control policies between services (Li et al., 2019), a multi-language

[6] Table A.3 in the appendix provides the links to the open-source applications.

Table 2
Analysis of Defense Mechanisms.

Sol.	Assumption	V&V	Testbed	Repr.
Heorhiadi et al. (2016)	Developers know the type of traffic exchanged among the microservices.	Evaluated the learning curve for using the framework, the efficacy and the proxy overhead by injecting delay and abort faults into a WordPress plugin.	Tested on an enterprise application and on the ElasticPress WordPress plugin.	
Torkura et al. (2017)	Internal communications are secure, but an attacker can take over a microservice.	Time overhead, vulnerabilities detection (SQL injection, HTTP Parameter Pollution, and CSRF), and enforcement of security policies.	Spring Petclinic (4 + 1 services) application.	√
Torkura et al. (2018)	Unspecified	Vulnerability detection rate and security policy enforcement (by dynamically adding a policy).	Tested on 4 applications: Spring PetClinic, Movie recommendations, eShop, and PiggyMetrics.	√
Pahl and Aubet (2018)	Attacks considered are system scan, spying, malicious control and operations, DoS, and incorrect configs.	Tested performance, robustness, and detection rate against a dataset containing common IoT attacks.	Testbed with seven different services monitored over 24 h.	
Baker and Nguyen (2019)	Unspecified	Tested against three API attacks (CSRF, XSS, and brute force).	Implemented 2-factor authentication extending Security DaoAuthenticationProvider.	
Torkura et al. (2019)	Unspecified	Application attack-surface reduction with and without this solution using 696 container image vulnerabilities and 167 web application vulnerabilities.	A diversified Spring PetClinic application (4 services in total).	√
Zhou et al. (2019)	Unspecified	Tested effectiveness by manually injecting 32 faults (e.g., multi-instance, configuration, and asynchronous interaction faults) and 142 faults into two applications.	Tested on 2 applications: Sock Shop (8 services, 10 trace types) and TrainTicket (41 services, 86 trace types) deployed on 20 VMs on a private cloud.	√
Chen et al. (2019)	Unspecified	Evaluated the detection efficacy of malicious RPC traffic.	Tested on a dataset with 1000 records of RPC logs with malicious traffic retrieved from a Kubernetes deployment.	
Suneja et al. (2019)	Utility-microservices contain vulnerabilities.	Attack isolation and performance overhead using 18 attacks against core-containers.	Unknown	
Wang et al. (2020)	Unspecified	Anomaly detection rate and time efficiency (using computing resource exhaustion, network transmission delay, and abortion).	Evaluated on TrainTicket (41 microservices).	
Li et al. (2020)	Network traffic is constant, arrival rate follows the Poisson distribution, and service rate follows an exponential distribution.	Evaluated the CPU resource consumption and service rate of each container.	Tested on a Python application with a ReDoS vulnerability.	
Baarzi et al. (2020)	Did not consider previous attacks to detect new ones.	Performance evaluation of response time for a legitimate user's request in presence of an attack (sending malicious requests and exhausting CPU and memory).	Tested on an application-as-a-service deployed with four replicas (i.e., two pods on two worker nodes) with 12 users (11 normal users, 1 attacker).	
Meng et al. (2021)	Unspecified	Performance overhead (before and after faults injection – CPU and network congestion, memory leak) and anomaly detection rate.	Tested on 2 applications, Bench4Q and Social Network.	√
Ma et al. (2021)	No prior knowledge of the application.	Evaluated the impact graph construction (by simulating anomalies with Pymicro) and result accuracy against other algorithms.	Tested on both simulated (Pymicro) and production (IBM Bluemix incident dataset) environments.	
Liu et al. (2021)	Unspecified	Evaluated accuracy and efficiency of locating anomalies, and pruning efficiency with different correlation coefficients.	Tested on 75 availability issues from 28 Alibaba e-commerce subsystems, with an average of 265 services over 5 months.	
Zhu and Gehrmann (2021)	Unspecified	Evaluated against 11 container-related exploits.	3-service (backend, reverse-proxy, and database) application deployed on three containers.	
Chen et al. (2021b)	Unspecified	Fault localization effectiveness using two metrics (the highest K accuracy rate (PR@K) and the Mean Average Precision (MAP)).	Tested on the Sock Shop application (8 services, deployed with 20, 30, 40, and 50 containers.	
Chen et al. (2021a)	Linux audit is trusted, and attackers have no a priori privileges.	Vulnerability detection, along with (audit) run-time and storage overhead.	Tested against 3 CVEs, namely, CVE-2019-5736, CVE-2019-14271, and CVE-2018-15664	
Cai et al. (2021)	Unspecified	Root cause localization accuracy using the PR@K, MAP and AFP metrics.	Tested on a real dataset with 81 faults, each lasting 5 min.	

prototype for Kubernetes and Istio to automatically generate inter-service network policies (Li et al., 2021), and a solution based on the Host Identity Protocol (HIP) to provide secure and persistent communication for containers (Ranjbar et al., 2017). An RBAC mechanism for complex policies for microservices based on the GemRBAC+CTX model (Fadhel et al., 2018), an API-based tool for network monitoring and policy enforcement (Sun et al., 2016), and an eBPF-based tool for segmenting microservice networks (Zaheer et al., 2019). Two solutions for authentication. Specifically, A Java framework using SAML tokens for IoT microservices authentication (George and Mahmoud, 2017), and a Python tool for microservices authentication Yarygina and Bagge (2018). Finally, we found a framework deployed on Docker Swarm to evaluate the performance of Moving Target Defence (MTD) solutions (Jin et al., 2021), a microservice reliability estimation algorithm (Pietrantuono et al., 2020), a framework to deploy microservices on orchestration tools complying with security policy (Melara and Bowman, 2021), an eBPF-based solution to audit container security events (Lim et al., 2021), and a honeypot technique for live confinement of suspicious services (Osman et al., 2019).

Evocative Security Architecture We found 13 contributions belonging to this dimension. Two solutions to authenticate IoT microservices (Pahl et al., 2018; Pahl and Donini, 2018), a framework for the authentication and authorization of both services and users (Preuveneers and Joosen, 2017), a solution for monitoring user identity and session management (Phipathananunth and Bunyakiati, 2018), and a verification technique to check the security of microservices communications (Gerking and Schubert, 2019). A machine learning and model checking-based methodology for functional testing of microservices (Meinke and Nycander, 2015), an approach based on cryptographic primitives for microservices authentication (Jander et al., 2018), and an algorithm for on-demand reliability estimation of microservices (Pietrantuono et al., 2018). A mechanism for microservices mutual authentication (Walsh and Manferdelli, 2017), and an intrusion response system based on monitoring processes Yarygina and Otterstad (2018). Finally, a protocol that guarantees the integrity of the DNS data (Díaz-Sánchez et al., 2019), a fault tolerance testing framework (Wu et al., 2018), and an anomaly detection and root cause proposal based on the Microscope technology (Guan et al., 2019).

Key Takeaway. Simulations, emulations, and empirical measurements all have their place in evaluating security solutions, but the lack of accurate specifications hinders both scientific progress and adoptions by other parties.

V&V Methodologies

Finally, we discuss prior contributions that we found based on the verification and validation methodologies they used. Table 3 summarizes the used verification and validation methodology related to each implemented solution (e.g., attack, and defence).

We classified (Zhou et al., 2019) as manual testing, despite the fault injection process in that work being partially automated because developers must still oversee the injection process, and in some instances manually inject the faults. With respect to attackers' simulations, both Osman et al. (2019), Yarygina and Otterstad (2018) present solutions that are triggered by alerts from an intrusion detection system (IDS), which detects the presence of an attacker, and both aim at containing further exploitation. Options for recovering or limiting the damage, when an attacker gains entry into a component are never discussed in prior work. Unlike traditional monolithic applications, microservices may have advantages when it comes to limiting an attack, even after an attacker has gained entry. Although (Ibrahim et al., 2019) automatically generates attack graphs, the correctness of the model is *manually* verified by the authors. The performance of these defense mechanisms (Chen et al., 2019; Pahl and Aubet, 2018) were tested on two datasets; the authors of the second paper shared the dataset with the community. Five contributions (Chen et al., 2019; Meinke and Nycander, 2015; Pahl and Aubet, 2018; Wang et al., 2020; Zhou et al., 2019) out of the 37 primary studies relied on one or more machine learning algorithms while only two used actual experiments (e.g., within capture the flags exercises Di Tizio et al., 2020; Ruef et al., 2016). Also, we did not find any testbed similar to what Soldani and Brogi proposed (Soldani and Brogi, 2021) for testing the performance of fault resilience methodology but to test the resilience to attacks or simulate attackers' behaviour.

Key Takeaway. Solutions that simulate an attacker's behaviour, as well as (automated or manual) solutions for attack remediation are open challenges.

7. RQ3: Industrial tools effectiveness

We classify industrial tools with the dimensions and subcategories previously proposed in §3. In particular, we distinguish between static-time (i.e., configuration analyzers and vulnerability scanners) and run-time tools (i.e., network visualizers and security monitors). Static-time tools belong to the Security-as-a-Service dimension, whereas run-time tools may belong to all three dimensions (i.e., Attack, Defence, and Security-as-a-Service). In Table 4 we summarize tools belonging to the first two sub-categories (configuration analyzers and vulnerability scanners) and we show whether the tool is open source or not if it scans container images for known vulnerabilities if it checks for misconfigured configuration files or the latter compliance with industry standards, and eventually, if it allows defining custom check policies (e.g., to prioritize vulnerabilities); tools references can be found in Table A.3 in the appendix.

We opted not to use the terminology static analyzers (occasionally used by the tool themselves) because these tools are often launched at deployment time as opposed to development time. A second and most important reason is what they actually do as we explain in the sequel. So we clustered them into two broad classes.

Configuration Analyzers: these tools analyze configuration files and settings and check them against common or custom security best practices and policies (e.g., compliance with the CIS Benchmarks). Among the tools for benchmarking assessment, we did not include those that essentially provide only compliance tests.[7]

Key Takeaway. What a "static analyzer" does for the security of microservices is way more modest than what a tool with the same name is expected to do for software security (Chess and McGraw, 2004).

Vulnerability Scanners: Such tools are used to check container images or application configuration files (e.g., JSON files) before running a container instance. Typically, a container image scanner retrieves a software bill of material (SBOM) gathering all software packages used in an image or in configuration files and compares them against vulnerabilities repositories (e.g., NVD[8] or Debian Security Bug Tracker[9]) to check if the current version of a package in use is *reported to be* vulnerable.

Such tools often generate false alerts, because they do not test the deployed code. Hence a vulnerability might have not been patched in the OS version reported in the vulnerability database while the actual specific version used in the container is not vulnerable. This problem of overcautious vulnerability reporting is well known in secure software engineering for the analysis of

[7] For example, `terrascan` by Accurics, built on top of OPA, and `checkov`, which are open source, along with `CIS-CAT Pro`, and `Lacework` which are commercial.

[8] https://nvd.nist.gov/

[9] https://security-tracker.debian.org/tracker/

Table 3
Classification of Verification & Validation Methods.

Solution	Manual Testing			Automated Testing			Not Tested?
	Injection *Vulnerability Management*	*Script/Test cases* *Performance Simulation*	*Attacker Simulation* *Performance Measurement*				
Attack				Ibrahim et al. (2019)			
Defence	Wang et al. (2020), Meng et al. (2021), Baarzi et al. (2020), Zhou et al. (2019), Ma et al. (2021), Chen et al. (2021b), Cai et al. (2021)	Heorhiadi et al. (2016), Suneja et al. (2019), Zhu and Gehrmann (2021)	Chen et al. (2021a)		Pahl and Aubet (2018), Chen et al. (2019), Liu et al. (2021), Li et al. (2020)	Torkura et al. (2019), Torkura et al. (2017), Torkura et al. (2018), Baker and Nguyen (2019)	
Evocative Sec. Arch.	Meinke and Nycander (2015)			Jander et al. (2018)	Pietrantuono et al. (2018), Walsh and Manferdelli (2017)		Pahl et al. (2018), Phipathananunth and Bunyakiati (2018), Wu et al. (2018), Gerking and Schubert (2019), Yarygina and Otterstad (2018), Pahl and Donini (2018), Preuveneers and Joosen (2017), Díaz-Sánchez et al. (2019), Guan et al. (2019)
Security-as-a-Service			Osman et al. (2019), Li et al. (2021)	Li et al. (2019)	George and Mahmoud (2017), Zaheer et al. (2019), Sun et al. (2016), Ranjbar et al. (2017), Yarygina and Bagge (2018), Jin et al. (2021), Pietrantuono et al. (2020), Li et al. (2021), Fadhel et al. (2018)	Melara and Bowman (2021), Lim et al. (2021)	

Table 4
Classification of Configuration Analyzers and Vulnerability Scanners.

Tool	Company	Open Source	Images	Config. Files	Benchmark	Custom Policies
iSkan	Alcide	√	√			
sKan & Advisor	Alcide		√	√	√	√
Anchore Engine	Anchore	√	√			
kube-bench	Aqua	√		√	√	
Trivy	Aqua	√	√			
Chef InSpec	Chef	√		√	√	√
Docker scan	Docker	√	√			
Xray	JFrog		√			
Conftest	Open Policy Agent	√		√	√	√
Prisma Cloud	paloalto		⋆	√	√	√
Qualys CS Sensor	Qualys		√			
Clair	Red Hat	√	√			
kubeaudit	Shopify	√		√		
Snyk Container	Snyk		√	√	√	√
Container Security	Tenable		√	√	√	√

⋆ Twistlock. Tool's references are in the appendix, respectively, in Table A.3.

vulnerabilities in open source libraries (Dashevskyi et al., 2018; Pashchenko et al., 2020).

We believe that there is a major issue related to the use of the terminology. According to the NIST definition, vulnerability scanning is "a technique used to identify hosts attributes and associated vulnerabilitie" (NIST, 2021). From a legal perspective, all these tools can claim to be "vulnerability scanners". Yet they do not offer the outcome that a security expert would intuitively expect even from the classical `nmap` network scanner (Lyon, 2008). We eventually decided to keep the terminology (albeit this means propagating further the misinterpretation) because this is widely used within the community. **Key Takeaway.** The outcome of a microservices vulnerability "scanner" is essentially a database join between a software bill of material and a vulnerability reference database and thus only provides a shallow sense of (in)security than the word "scan" would intuitively suggest. In Table 5, instead, we summarized the tools belonging to the other two sub-categories (network visualizers and security monitors). In particular, we show whether the tool is open source or not, whether it is based on eBPF, if it allows defining network policies, visualizing network flow, and if it does anomaly detection, container-drift prevention, or penetration testing. Most of such tools provide more functional-

Table 5
Classification of network visualizers and security monitors.

Tool	Company	Open Source	eBPF	Network Policy	Network Visual.	Anomaly & Malware Detection	Container Drift Prevention	Penetration Testing
Alcide Runtime	Alcide		√		√	√	√	
Aqua Container Sec. Platf.	Aqua				√	√	√	
kube-hunter	Aqua	√						√
Linkerd	Bouyant	√		√	√			
kubesploit & KubeScan	Cyberark Lab	√						√
Datadog	Datadog	√	√	√	⋆	⋆	⋆	
Istio	IBM	√		√	⋆⋆			
Peirates	InGuardians	√						√
Cilium	Isovalent	√	√	√	⋆⋆⋆			
Qualys Cloud Platf.	Qualys				√	√	√	
StackRox Kub. Sec. Platf.	StackRox		√		√	√		
Falco	Sysdig	√	√			√		
Sysdig Platf.	Sysdig		√		√	√	√	
Calico	Tigera	√	√	√	⋆	⋆		
Antrea	VMware	√		√	√			
Weave Kub. Platf.	Weaveworks				√	√	√	
Weave Net	Weaveworks	√	√	√				
Weave Scope	Weaveworks	√	√		√			

⋆ `Enterprise feature`, ⋆⋆ `Kiali tool`, ⋆⋆⋆ `Hubble tool`. Tool's references are in the appendix, respectively, in Table A.3.

ities than the ones we listed in the table, however, we only showed the relevant ones; tools references can be found in Table A.3 in the appendix.

Network Visualizers support the visualization of microservices networks and provide additional services such as service discovery, performance monitoring, or load balancing. Of these, some tools that claim to offer (security) "monitoring services" are actually just network visualization tools. Hence we opted to report them as such. A famous example in this dimension, albeit not related to microservices, is the "SolarWinds" suite.

Security Monitors allow security monitoring *and* enforcement at run-time. We have tried to be as close as possible to James Anderson's classical concept of Reference Monitor (Anderson, 1972). A weaker version of this definition, even if complete mediation is not warranted as the Anderson report recommended, includes tools that are at least able to work as intrusion/anomaly detection systems.

All tools in this dimension allow the user to define security policies, prevent specific vulnerabilities to be exploited (e.g., container escaping techniques) or networking policies, providing networking segmentation and access control (e.g., avoiding lateral movement among containers). While there are comprehensive analyses of network security policies and the verification of their correctness being them firewall policies (Adao et al., 2016; Lyu and Lau, 2000) or low-level access control policies for cloud resources (Backes et al., 2019), no such studies exist for microservice-related network policies. To support users in this respect, some tools allow them to automatically generate policies and visualize the traffic. Yet the expressiveness of the policies generated by these visualization tools is unclear, in contrast to what happened for firewall policies (Bartal et al., 1999). **Key Takeaway.** There is yet no tool *and* no theory to automate the process of verifying or testing the correctness of policies in spite of their fragility.

At a first glance, another finding can also be identified. **Key Takeaway.** There is no open-source solution for container drift prevention, albeit preventing it is conceptually simple. Despite each service (or container) should be responsible to perform only one task, there is yet no publicly available solution to enforce this and how to do so.

Some tools are omitted as they piggyback on others. For example, `Hubble` is used by `Cilium` for providing network visualization, and `Istio` uses `Kiali` for the same purpose. Other tools, such as `Flannel`, `Kube-router`, and `Romana` exist but they have not been included because they only provide networking functionality without the possibility of defining policy or even visualizing the traffic. We also have not included tools tracing events within containers, e.g., `Tracee` from Aqua, `Zipkin` from Twitter, or `Jaeger` from Uber. It is unclear whether they could be considered "inline intrusion detection systems" in the spirit of inline reference monitors (Erlingsson, 2003) or are just profiling tools.

Finally, we included tools that allow performing penetration testing on microservices deployments albeit they should belong to a dimension of their own.

Key Takeaway. There is room for improving penetration testing tools for microservices by adding, for example, functionality to correlate exploitation steps to MITRE ATT&CK tactics and CVEs that might be exploited, or investigating the noise generated by such tools.

An ever-growing number of tools are based on eBPF technology. While eBPF seems the industry way forward for run-time security and policy enforcement, we found only two papers proposing eBPF-based tools (Lim et al., 2021; Zaheer et al., 2019). **Key Takeaway.** The security of eBPF must still be fully investigated. It is still unclear whether this technology is appropriate to capture container and network events and enforce complex security policies with reasonable overhead.

8. Limitations and threats to validity

A threat to construction validity that is not deletable in any systematic survey is the bias introduced by Google Scholar to determine relevance (O'Neil, 2016). To limit this bias we performed the search without logging in any Google-related websites, including institutional sites or IEEE (ieee.org email is provisioned by Google). We considered the criterion of focusing on "well-ranked venues" but deliberately avoided it. First, it is redundant (well-ranked venue papers should appear in the general search), and second, methodologies in other disciplines where metareviews are common (e.g., PRISMA) recommend using content to avoid source-based biases. Similarly, we avoided searching for "unofficial sources" (e.g., blog posts) because it is difficult to do it scientifically. How to systematically search for blogs, how many blogs should be investigated, and how to rate their reliability are questions that should be defined as community guidelines and that go beyond an SoK paper. Another threat to any keyword search-based survey paper's validity is the keywords used to retrieve the data.

As reported in the appendix, we tried to use different keywords to find all publications relevant to run-time security, testing, and detection; however, some publications and tools may not have been found with the used keywords, and thus not being present in the paper. We decided not to cover specific serverless platforms (e.g., AWS Lambda functions) because we preferred to be platform-independent, covering microservices security issues not bound to any cloud provider. Since a serverless system is just a system where the container is provided by a particular provider (e.g., AWS or Google) this would require a full additional analysis for each platform while providing limited research insights of general interest. In terms of conclusion validity, we proposed a comprehensive taxonomy in Fig. 1 by which we further categorized each paper that we found. However, a new paper or tool might require additional categories. Thus, we consider this categorization a starting taxonomy to be possibly extended. The misnomer "vulnerability scanner" is an example of such a possible gap. Finally, due to a large number of tools available [10], it is likely that some tools might be missing, other tools might provide additional functionalities than the ones shown in Table 5 and Table 4, or even completely misrepresent the functionalities that they actually offer.

9. Conclusion

We performed a systematization of knowledge on the run-time security of microservice-based applications by reviewing both scientific papers and tools that were available on the market. The microservice architecture has been in use for quite a while now, and it is surprising to discover that the topic of its run-time security has received only limited attention. Our findings indicate the potential avenues of research that are not covered or are poorly covered. Orchestration security and the verification of security policies, particularly in terms of information flow among microservices, is likely the greatest uncovered gap. Resilience against external and internal threats and recovery from attacks still remain open challenges. The key takeaway is that several solutions, both from academia and industry alike, do not actually security test the deployed systems as expected. We found some experimental testbeds openly available and we want to stress more the importance of reproducibility of further research on the microservice security topic. We also found relatively fewer studies on security and vulnerabilities "in the wild", compared to other traditional security fields (such as web or internet security). Based on our findings, we believe security experimentation with testbeds and security studies in the wild are the most promising candidates for future work.

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.

Data availability

Data will be made available on request.

Acknowledgments

We thank B. Chandrasekaran for comments on a preliminary version of this paper and A. Blaise, K. Tuma, and F. Rebecchi for useful discussions on cloud security. This work has received funding from the European Union under the H2020 grant 952647 (AssureMOSS).

Appendix A. Search keywords

Table A.1 shows how many papers were retrieved for each search string and from which library. The numbers of unique papers are the ones that we found only in the respective library with the corresponding search string (Table A.2).

[10] As shown in the CNCF Cloud Native Landscape https://landscape.cncf.io/.

Table A.1
The search keywords and corresponding results for each digital library.

Keywords	Result on Scopus	Papers Extracted	Result on Scholar	Papers Extracted	Result on ACM DL	Papers Extracted
"microservice" AND ("run-time" OR "runtime" OR "dcnamic") AND ("security" OR "metrics")	86	13 (unique 2)	8.980	17 (unique 3)	471	3 (unique 1)
"microservice" AND ("run-time" OR "runtime" OR "dynamic") AND ("exploits" OR "attacks")	17	5 (unique 1)	2.760	14 (unique 1)	131	3 (unique 2)
"microservice" AND ("run-time" OR "runtime" OR "dynamic") AND ("testing" OR "analysis")	149	16 (unique 3)	10.400	10 (unique 4)	556	1 (unique 1)
"microservice" AND ("vulnerability" OR "vulnerabilities") AND ("monitoring" OR "detection")	10	5 (unique 1)	1.950	11 (unique 2)	63	none

Table A.2
Link to Applications Website.

Application	Link
Spring PetClinic	https://github.com/spring-projects/spring-petclinic
Diversified Spring PetClinic Torkura et al. (2019)	https://github.com/maineffort/Swaggerized-Spring-PetClinic
Movie recommendations	https://github.com/kbastani/spring-cloud-microservice-example
eShop	https://github.com/ewolff/microservice
PiggyMetrics	https://github.com/sqshq/piggymetrics
Bench4Q	https://github.com/rzs840707/Bench4Q-TPCW
Social Network	https://github.com/delimitrou/DeathStarBench/tree/master/socialNetwork
TrainTicket	https://github.com/FudanSELab/train-ticket
Sock Shop	https://github.com/microservices-demo/microservices-demo

Table A.3
Link to Tools Website.

Tool	Link	Tool	Link
Anchore Engine	https://github.com/anchore/anchore-engine	kube-hunter	https://github.com/aquasecurity/kube-hunter
Antrea	https://antrea.io/	kube-router	https://github.com/cloudnativelabs/kube-router
Aqua Container Sec. Platf.	https://www.aquasec.com/aqua-cloud-native-security-platform/	kubeaudit	https://github.com/Shopify/kubeaudit
BridgeCrew	https://bridgecrew.io/	Linkerd	https://github.com/linkerd/linkerd2
Calico	https://github.com/projectcalico/calico	Peirates	https://github.com/inguardians/peirates
Chef InSpec	https://docs.chef.io/inspec/	Prisma Cloud	https://www.paloaltonetworks.com/prisma/cloud
Cilium	https://github.com/cilium/cilium	Qualys Cloud Platform	https://www.qualys.com/cloud-platform/
CIS-CAT Pro	https://www.cisecurity.org/cybersecurity-tools/cis-cat-pro	Qualys CS Sensor	https://www.qualys.com/apps/container-security/#container-sensor
Clair	https://github.com/quay/clair	Romana	https://github.com/romana/romana
Conftest	https://www.conftest.dev/	sKan & kAdvisor	https://github.com/alcideio/skan
Container Security	https://www.tenable.com/products/tenable-cs	Snyk Container	https://snyk.io/product/container-vulnerability-management/
Datadog	https://www.datadoghq.com/	StackRox Kub. Sec. Platf.	https://github.com/stackrox/stackrox
Docker scan	https://docs.docker.com/engine/scan/	Sysdig Platform	https://sysdig.com/products/secure/
Falco	https://github.com/falcosecurity/falco	terrascan	https://terrasolid.com/products/terrascan/
Flannel	https://github.com/flannel-io/flannel	Tracee	https://github.com/aquasecurity/tracee
Hubble	https://github.com/cilium/hubble	Trivy	https://github.com/aquasecurity/trivy
iSkan	https://github.com/alcideio/iskan	Twistlock	https://www.paloaltonetworks.com/prisma/cloud
Istio	https://github.com/istio/istio	Weave Kub. Platf.	https://www.weave.works/product/gitops/
Jaeger	https://github.com/jaegertracing/jaeger	Weave Net	https://github.com/weaveworks/weave
Kiali	https://github.com/kiali/kiali	Xray	https://jfrog.com/integration/xray-docker-security-scanning/
kube-bench	https://github.com/aquasecurity/kube-bench	Zipkin	https://github.com/openzipkin/zipkin

References

Adao, P., Focardi, R., Guttman, J.D., Luccio, F.L., 2016. Localizing firewall security policies. In: Proc. of IEEE CSF '16. IEEE, Lisbon, Portugal, pp. 194–209.

Anderson, J.P., 1972. Computer Security Technology Planning Study. Technical Report ESD-TR-73-51. J.P. Anderson and Company. Prepared for the US Air Force Electronic Systems Division.. https://apps.dtic.mil/sti/pdfs/AD0758206.pdf

Baarzi, A.F., Kesidis, G., Fleck, D., Stavrou, A., 2020. Microservices made attack-resilient using unsupervised service fissioning. In: Proc. of EuroSec '20. Association for Computing Machinery, New York, NY, USA, pp. 31–36. doi:10.1145/3380786.3391395.

Backes, J., Bolignano, P., Cook, B., Dodge, C., Gacek, A., Luckow, K., Rungta, N., Tkachuk, O., Varming, C., 2019. Semantic-based automated reasoning for AWS access policies using SMT. Proc. FMCAD '19 2018, 206–214. doi:10.23919/FMCAD.2018.8602994.

Baker, O., Nguyen, Q., 2019. A novel approach to secure microservice architecture from OWASP vulnerabilities. In: Proc. of CITRENZ '19. ITxNew Zealands Conference of IT, Nelson, NZ, pp. 54–59. https://www.citrenz.ac.nz/Conf.s/2019/pdf/2019CITRENZPACITN.pdf.

Bakr, A., Abd El-Aziz, A.A., Hefny, H.A., 2019. A survey on mitigation techniques against DDoS attacks on cloud computing architecture. Internat. J. AST 28 (12), 187–200.

Bartal, Y., Mayer, A., Nissim, K., Wool, A., 1999. Firmato: a novel firewall management toolkit. In: Proc. of IEEE Symposium on S&P '99. ACM New York, NY, USA, pp. 17–31. doi:10.1109/SECPRI.1999.766714.

Cai, Y., Han, B., Li, J., Zhao, N., Su, J., 2021. ModelCoder: a fault model based automatic root cause localization framework for microservice systems. In: 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), pp. 1–6. doi:10.1109/IWQOS52092.2021.9521318.

Chen, J., 2020. Unit 42 CTR: leaked code from docker registries. https://unit42.paloaltonetworks.com/leaked-docker-code/.

Chen, J., Huang, H., Chen, H., 2019. Informer: irregular traffic detection for containerized microservices RPC in the real world. In: Proc. of ACM/IEEE SEC '19. Association for Computing Machinery, New York, NY, USA, pp. 389–394. doi:10.1145/3318216.3363375.

Chen, X., Irshad, H., Chen, Y., Gehani, A., Yegneswaran, V., 2021. CLARION: sound and clear provenance tracking for microservice deployments. In: Proc. of USENIX Security '21. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/chen-xutong.

Chen, Y., Chen, N., Xu, W., Lian, L., Tu, H., 2021. MFRL-CA: microservice fault root cause location based on correlation analysis. In: 2021 8th International Conference on Dependable Systems and Their Applications (DSA), pp. 90–101. doi:10.1109/DSA52907.2021.00018.

Chess, B., McGraw, G., 2004. Static analysis for security. Proc. IEEE Secur. Privacy 2 (6), 76–79.

Dashevskyi, S., Brucker, A.D., Massacci, F., 2018. A screening test for disclosed vulnerabilities in FOSS components. Proc. IEEE TSE 45 (10), 945–966.

Di Tizio, G., Massacci, F., Allodi, L., Dashevskyi, S., Mirkovic, J., 2020. An experimental approach for estimating cyber risk: a proposal building upon cyber ranges and capture the flags. In: Proc. of IEEE EuroS PW '20. IEEE, Genoa, Italy, pp. 56–65. doi:10.1109/EuroSPW51379.2020.00016.

Díaz-Sánchez, D., Marín-Lopez, A., Mendoza, F.A., Cabarcos, P.A., 2019. DNS/DANE collision-based distributed and dynamic authentication for microservices in IoT. Sensors (Switzerland) 19 (15), 1–23. doi:10.3390/s19153292.

Documentation, K., 2021. Overview of cloud native security. (accessed: 30/08/2022) https://kubernetes.io/docs/concepts/security/overview/.

Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L., 2017. Microservices: Yesterday, Today, and Tomorrow. Springer International Publishing, Cham, pp. 195–216.

Erlingsson, U., 2003. The inlined reference monitor approach to security policy enforcement. Cornell University Ph.D. thesis.

Fadhel, A.B., Bianculli, D., Briand, L.C., 2018. Model-driven run-time enforcement of complex role-based access control policies. In: Proc. of IEEE/ACM ASE '18, pp. 248–258. doi:10.1145/3238147.3238167.

Fowler, M., Lewis, J., 2014. Microservices a definition of this new architectural term. (accessed: 30/08/2022).https://martinfowler.com/articles/microservices.html.

Gao, X., Gu, Z., Kayaalp, M., Pendarakis, D., Wang, H., 2017. ContainerLeaks: emerging security threats of information leakages in container clouds. In: Proc. of IEEE/IFIP DSN '17. IEEE, Denver, CO, USA, pp. 237–248. doi:10.1109/DSN.2017.49.

George, V.M., Mahmoud, Q.H., 2017. Claimsware: a claims-based middleware for securing IoT services. In: Proc. of IEEE COMPSAC '17, Vol. 1, pp. 649–654. doi:10.1109/COMPSAC.2017.85.

Gerking, C., Schubert, D., 2019. Component-based refinement and verification of information-flow security policies for cyber-physical microservice architectures. In: Proc. of IEEE ICSA '19. IEEE, Hamburg, Germany, pp. 61–70. doi:10.1109/ICSA.2019.00015.

Global Industry Analysts, I., 2019. Cloud microservices - global market trajectory & analytics. (accessed: 05/12/2022). https://www.marketresearch.com/Global-Industry-Analysts-v1039/Cloud-Microservices-32405615/.

Guan, Z., Lin, J., Chen, P., 2019. On anomaly detection and root cause analysis of microservice systems. In: ICSOC '18 Workshops. Springer International Publishing, Cham, pp. 465–469.

Hannousse, A., Yahiouche, S., 2020. Securing microservices and microservice architectures: a systematic mapping study.

Heorhiadi, V., Rajagopalan, S., Jamjoom, H., Reiter, M.K., Sekar, V., 2016. Gremlin: systematic resilience testing of microservices. In: Proc. of ICDCS '16, pp. 57–66. doi:10.1109/ICDCS.2016.11.

Ibrahim, A., Bozhinoski, S., Pretschner, A., 2019. Attack graph generation for microservice architecture. In: Proc. of ACM/SIGAPP '19. Association for Computing Machinery, New York, NY, USA, pp. 1235–1242. doi:10.1145/3297280.3297401.

Jander, K., Braubach, L., Pokahr, A., 2018. Defense-in-depth and role authentication for microservice systems. Procedia Comput. Sci. 130, 456–463. doi:10.1016/j.procs.2018.04.047.

Jin, H., Li, Z., Zou, D., Yuan, B., 2021. DSEOM: a framework for dynamic security evaluation and optimization of MTD in container-based cloud. IEEE TDSC '21 18 (3), 1125–1136. doi:10.1109/TDSC.2019.2916666.

Kitchenham, B., Charters, S., 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report. Software Engineering Group School of Computer Science and Mathematics Keele University Keele, Staffs ST5 5BG, UK.
Kumar, R., Trivedi, M.C., 2021. Networking analysis and performance comparison of Kubernetes CNI plugins. In: Advances in Computer, Communication and Computational Sciences. Springer Singapore, Singapore, pp. 99–109.
Li, X., Chen, Y., Lin, Z., 2019. Towards automated inter-service authorization for microservice applications. In: Proc. of ACM SIGCOMM '19 Conference Posters and Demos. Association for Computing Machinery, New York, NY, USA, pp. 3–5. doi:10.1145/3342280.3342288.
Li, X., Chen, Y., Lin, Z., Wang, X., Chen, J.H., 2021. Automatic policy generation for inter-service access control of microservices. In: Proc. of USENIX Security '21. USENIX Association. https://www.usenix.org/conference/usenixsecurity21/presentation/li-xing.
Li, Z., Jin, H., Zou, D., Yuan, B., 2020. Exploring new opportunities to defeat low-rate DDoS attack in container-based cloud environment. IEEE TPDS '20 31 (3), 695–706. doi:10.1109/TPDS.2019.2942591.
Lim, S.Y., Stelea, B., Han, X., Pasquier, T., 2021. Secure namespaced kernel audit for containers. In: Proceedings of the ACM Symposium on Cloud Computing. Association for Computing Machinery, New York, NY, USA, pp. 518–532. doi:10.1145/3472883.3486976.
Liu, D., He, C., Peng, X., Lin, F., Zhang, C., Gong, S., Li, Z., Ou, J., Wu, Z., 2021. MicroHECL: high-efficient root cause localization in large-scale microservice systems. In: Proc. of ICSE-SEIP '21, pp. 338–347. doi:10.1109/ICSE-SEIP52600.2021.00043.
Lyon, G.F., 2008. Nmap network scanning: the official Nmap project guide to network discovery and security scanning. Insecure. Com LLC (US), US.
Lyu, M.R., Lau, L.K., 2000. Firewall security: policies, testing and performance evaluation. In: Proc. of COMPSAC 2000. IEEE, Taipei, Taiwan, pp. 116–121.
Ma, M., Lin, W., Pan, D., Wang, P., 2021. ServiceRank: root cause identification of anomaly in large-scale microservice architecture. IEEE TDSC '21 1. doi:10.1109/TDSC.2021.3083671.
Martinez, R.G., Lopes, A., Rodrigues, L., 2017. Automated generation of policies to support elastic scaling in cloud environments. In: Proceedings of the Symposium on Applied Computing. Association for Computing Machinery, New York, NY, USA, pp. 450–455. doi:10.1145/3019612.3019658.
Meinke, K., Nycander, P., 2015. Learning-based testing of distributed microservice architectures: Correctness and fault injection. In: Proc. of SEFM '15. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 3–10.
Melara, M. S., Bowman, M., 2021. Enabling security-oriented orchestration of microservices. 2106.09841.
Meng, L., Ji, F., Sun, Y., Wang, T., 2021. Detecting anomalies in microservices with execution trace comparison. FGCS '21 116, 291–301. doi:10.1016/j.future.2020.10.040.
Nehme, A., Jesus, V., Mahbub, K., Abdallah, A., 2019. Securing microservices. IT Prof. 21 (1), 42–49. doi:10.1109/MITP.2018.2876987.
Newman, S., 2015. Building Microservices, first ed. O'Reilly Media, Inc., Sebastopol, CA.
NIST, 2017. An introduction to information security. (accessed: 30/08/2022).https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf.
NIST, 2021. Vulnerability scanning. (accessed: 30/08/2022) https://csrc.nist.gov/glossary/term/Vulnerability_Scanning.
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.
NSA, 2020. Mitigating cloud vulnerabilities. (accessed: 05/12/2022) https://media.defense.gov/2020/Jan/22/2002237484/-1/-1/0/CSI-MITIGATING-CLOUD-VULNERABILITIES_20200121.PDF.
O'Neil, C., 2016. Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy. Crown, Vikalpa.
Osman, A., Bruckner, P., Salah, H., Fitzek, F.H.P., Strufe, T., Fischer, M., 2019. Sandnet: towards high quality of deception in container-based microservice architectures. In: Proc. of ICC '19. IEEE, Shanghai, China, pp. 1–7. doi:10.1109/ICC.2019.8761171.
Otterstad, C., Yarygina, T., 2017. Low-level exploitation mitigation by diverse microservices. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (Eds.), Service-Oriented and Cloud Computing. Springer International Publishing, Cham, pp. 49–56.
Pahl, M.-O., Aubet, F.-X., 2018. All eyes on you: distributed multi-dimensional IoT microservice anomaly detection. In: Proc. of CNSM '18. IEEE, Rome, Italy, pp. 72–80.
Pahl, M.-O., Aubet, F.-X., Liebald, S., 2018. Graph-based IoT microservice security. In: Proc. of NOMS '18. IEEE, Taipei, Taiwan, pp. 1–3. doi:10.1109/NOMS.2018.8406118.
Pahl, M.-O., Donini, L., 2018. Securing IoT microservices with certificates. In: Proc. of NOMS '18. IEEE, Taipei, Taiwan, pp. 1–5. doi:10.1109/NOMS.2018.8406189.
Pashchenko, I., Plate, H., Ponta, S., Sabetta, A., Massacci, F., 2020. Vuln4Real: a methodology for counting actually vulnerable dependencies. Proc. of IEEE TSE SE-13 (01), 1. doi:10.1109/TSE.2020.3025443.
Pereira-Vale, A., Fernandez, E.B., Monge, R., Astudillo, H., Mrquez, G., 2021. Security in microservice-based systems: a multivocal literature review. Comput. Secur. 103, 102200. doi:10.1016/j.cose.2021.102200. https://www.sciencedirect.com/science/article/pii/S0167404821000249.
Pereira-Vale, A., Mrquez, G., Astudillo, H., Fernandez, E.B., 2019. Security mechanisms used in microservices-based systems: a systematic mapping. In: Proc. of CLEI '19. IEEE, Panama, Panama City, pp. 01–10. doi:10.1109/CLEI47609.2019.235060.
Phipathananunth, C., Bunyakiati, P., 2018. Synthetic runtime monitoring of microservices software architecture. In: Proc. of COMPSAC '18, Vol. 02, pp. 448–453. doi:10.1109/COMPSAC.2018.10274.
Pietrantuono, R., Russo, S., Guerriero, A., 2018. Run-time reliability estimation of microservice architectures. In: Proc. of ISSRE '18, pp. 25–35. doi:10.1109/ISSRE.2018.00014.
Pietrantuono, R., Russo, S., Guerriero, A., 2020. Testing microservice architectures for operational reliability. STVR '20 30 (2), 19–24. doi:10.1002/stvr.1725.
Preuveneers, D., Joosen, W., 2017. Access control with delegated authorization policy evaluation for data-driven microservice workflows. Future Internet 9 (4), 1–21. doi:10.3390/fi9040058. https://www.mdpi.com/1999-5903/9/4/58.
Quist, N., 2019. Misconfigured and exposed: container services. https://unit42.paloaltonetworks.com/misconfigured-and-exposed-container-services/.
Ranjbar, A., Komu, M., Salmela, P., Aura, T., 2017. Synaptic: secure and persistent connectivity for containers. In: Proc. of CCGRID '17. IEEE, Madrid, Spain, pp. 262–267. doi:10.1109/CCGRID.2017.62.
Ruef, A., Hicks, M., Parker, J., Levin, D., Mazurek, M.L., Mardziel, P., 2016. Build it, break it, fix it: Contesting secure development. In: Proc. of ACM SIGSAC '16. Association for Computing Machinery, New York, NY, USA, pp. 690–703. doi:10.1145/2976749.2978382.
Soldani, J., Brogi, A., 2021. Automated generation of configurable cloud-native chaos testbeds. In: Adler, R., Bennaceur, A., Burton, S., Di Salle, A., Nostro, N., Olsen, R.L., Saidi, S., Schleiss, P., Schneider, D., Schwefel, H.-P. (Eds.), Dependable Computing - EDCC 2021 Workshops. Springer International Publishing, Cham, pp. 101–108.
Souppaya, M., Morello, J., Scarfone, K., 2017. Application Container Security Guide. NIST Special Publication 800-190.
Sultan, S., Ahmad, I., Dimitriou, T., 2019. Container security: issues, challenges, and the road ahead. IEEE Access 7, 52976–52996.
Sun, Y., Nanda, S., Jaeger, T., 2016. Security-as-a-service for microservices-based cloud applications. In: Proc. of CloudCom '15. IEEE, Vancouver, BC, Canada, pp. 50–57. doi:10.1109/CloudCom.2015.93.
Suneja, S., Kanso, A., Isci, C., 2019. Can container fusion be securely achieved? In: Proc. of the 5th Internat. WOC '19. ACM, Davis CA USA, pp. 31–36. doi:10.1145/3366615.3368356.
Torkura, K.A., Sukmana, M.I., Kayem, A.V., Cheng, F., Meinel, C., 2019. A cyber risk based moving target defense mechanism for microservice architectures. In: Proc. of IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom '18. IEEE, Melbourne, VIC, Australia, pp. 932–939. doi:10.1109/BDCloud.2018.00137.
Torkura, K.A., Sukmana, M.I., Meinel, C., 2017. Integrating continuous security assessments in microservices and cloud native applications. In: Proc. of UCC '17 Companion. Association for Computing Machinery, New York, NY, USA, pp. 171–180. doi:10.1145/3147213.3147229.
Torkura, K.A., Sukmana, M.I.H., Cheng, F., Meinel, C., 2018. CAVAS: neutralizing application and container security vulnerabilities in the cloud native era. In: Security and Privacy in Communication Networks. Springer International Publishing, Cham, pp. 471–490.
Walsh, K., Manferdelli, J., 2017. Mechanisms for mutual attested microservice communication. In: Proc. of UCC '17 Companion. Association for Computing Machinery, New York, NY, USA, pp. 59–64. doi:10.1145/3147234.3148102.
Wang, L., Zhao, N., Chen, J., Li, P., Zhang, W., Sui, K., 2020. Root-cause metric location for microservice systems via log anomaly detection. In: Proc. of IEEE ICWS '20. IEEE, Beijing, China, pp. 142–150. doi:10.1109/ICWS49710.2020.00026.
Waseem, M., Liang, P., Shahin, M., Ahmad, A., Nassab, A.R., 2021. On the nature of issues in five open source microservices systems: an empirical study. In: Proc. of EASE '21. Association for Computing Machinery, New York, NY, USA, pp. 201–210. doi:10.1145/3463274.3463337.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of EASE '14. Association for Computing Machinery, New York, NY, USA doi:10.1145/2601248.2601268. https://doi-org.vu-nl.idm.oclc.org/10.1145/2601248.2601268.
Wu, N., Zuo, D., Zhang, Z., 2018. An extensible fault tolerance testing framework for microservice-based cloud applications. In: Proc. of ICCIP '18. Association for Computing Machinery, New York, NY, USA, pp. 38–42. doi:10.1145/3290420.3290476.
Yarygina, T., Bagge, A.H., 2018. Overcoming security challenges in microservice architectures. In: Proc. of SOSE '18. IEEE, Bamberg, Germany, pp. 11–20. doi:10.1109/SOSE.2018.00011.
Yarygina, T., Otterstad, C., 2018. A game of microservices: automated intrusion response. IFIP Internat. Conf. on DAIS '18 10853 (January), 169–177. doi:10.1007/978-3-319-93767-0_12.
Yu, D., Jin, Y., Zhang, Y., Zheng, X., 2019. A survey on security issues in services communication of microservices-enabled fog applications. Concurrency Comput. 31 (22), 1–19. doi:10.1002/cpe.4436.
Zaheer, Z., Chang, H., Mukherjee, S., Van der Merwe, J., 2019. eZTrust: network-independent zero-trust perimeterization for microservices. In: Proc. of SOSR '19. Association for Computing Machinery, New York, NY, USA, pp. 49–61. doi:10.1145/3314148.3314349.
Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q., He, C., 2019. Latent error prediction and fault localization for microservice applications by learning from system trace logs. In: Proc. of ESEC/FSE '19. ACM, Tallinn Estonia, pp. 683–694. doi:10.1145/3338906.3338961.
Zhu, H., Gehrmann, C., 2021. AppArmor profile generator as a cloud service. In: CLOSER, pp. 45–55.

Francesco Minna is a PhD candidate at Vrije Universiteit Amsterdam, 1081 HV, The Netherlands. His research interests include cloud security and dynamic risk analysis for free and open-source software. Minna received a double master's in cybersecurity from the University of Trento and the University of Rennes 1. Contact him at f.minna@vu.nl.

Fabio Massacci is a professor at the University of Trento, Trento, 38123, Italy, and Vrije Universiteit, Amsterdam, 1081 HV, The Netherlands. Massacci received a PhD in computing from the University of Rome "La Sapienza." He received the IEEE Requirements Engineering Conference Ten Year Most Influential Paper Award on security in sociotechnical systems. He participates in the FIRST special interest group on the Common Vulnerability Scoring System and the European pilot CyberSec4Europe on the governance of cyber-security. He coordinates the European AssureMOSS project. He is a Member of IEEE. Contact him at fabio.massacci@ieee.org.