On the Vulnerability of Modern Microservice Frameworks

Tamara Vukadinović[1], Anđela Grujić[1], Marko Gorišek[1], Jovana Jović[1,*], Milan Tančić[2] and Nemanja Zdravković[1]

[1]*Faculty of Information Technology, Belgrade Metropolitan University, Tadeuša Košćuška 63, 11000 Belgrade, Serbia*
[2]*Academy of Technical and Vocational Vocational Studies - Pirot Section, 18300, Pirot, Serbia*

Abstract

Software architecture based on microservices has emerged as a leading approach to building scalable, flexible, and independently deployable software systems. Its modular design allows organizations to enhance agility, support continuous deployment, and respond swiftly to evolving business requirements. However, despite these advantages, microservices introduce a unique set of security vulnerabilities due to their distributed and decentralized nature, complex inter-service communications, and dynamic orchestration environments.

This paper presents a comprehensive review of peer-reviewed studies addressing security vulnerabilities in microservice-based systems. Drawing from academic and empirical studies, we employed a rigorous filtering process to exclude non-English publications, duplicate entries, and studies lacking direct relevance to microservice security. The selected works were then analyzed and synthesized, in order to uncover the most critical vulnerabilities across six key domains: network and communication, data consistency and integrity, authentication and authorization, deployment and orchestration, heterogeneous features, and DevOps practices.

In addition to summarizing the state of the art, we extend the current literature by correlating these vulnerabilities with empirical insights from real-world migration practices toward microservices, based on industrial surveys and case studies. Our analysis identifies gaps between theoretical vulnerability taxonomies and the challenges faced by practitioners during system decomposition, data migration, and secure deployment.

Finally, we propose a set of mitigation techniques grounded in both academic research and industrial best practices. These recommendations aim to assist software architects, developers, and security professionals in designing and maintaining secure microservice-based systems, ultimately contributing to more robust and resilient software architectures.

Keywords

Microservice, cybersecurity, vulnerability analysis

1. Introduction

Microservice-based architectures have emerged as a modern software development paradigm, primarily designed to address the limitations of various types of monolithic architectures. By decomposing complex applications into loosely coupled and independently deployable services, microservices enhance scalability, fault isolation, and deployment flexibility. This design approach supports continuous integration and delivery (CI and CD, respectively), promotes autonomous team development, and facilitates rapid response to changing business requirements [1, 2].

However, even with the mentioned benefits, the shift from monolithic to microservice-based systems introduces a wide range of new challenges, especially regarding system complexity, data management, and perhaps most important - security. The distributed nature of microservices leads to increased surface area for attacks, more intricate communication patterns, and added dependencies on orchestration platforms, such as Kubernetes and Docker Swarm [1, 3]. With services running in isolated containers or virtualized environments, where communication is achieved over public or internal APIs, microservice

BISEC'2024: 15th International Conference on Business Information Security, November 28-29, 2024, Niš, Serbia

*Corresponding author.

tamara.vukadinovic@metropolitan.ac.rs (T. Vukadinović); andjela.grujic@metropolitan.ac.rs (A. Grujić); marko.gorisek.4310@metropolitan.ac.rs (M. Gorišek); jovana.jovic@metropolitan.ac.rs (J. Jović); milan.tancic@akademijanis.edu.rs (M. Tančić); nemanja.zdravkovic@metropolitan.ac.rs (N. Zdravković)
0009-0001-4147-390X (T. Vukadinović); 0009-0004-5713-6248 (A. Grujić); 0009-0003-9765-5045 (M. Gorišek); 0000-0002-4204-0233 (J. Jović); 0009-0002-5184-1558 (M. Tančić); 0000-0002-2631-6308 (N. Zdravković)

systems are susceptible to configuration errors, insecure interfaces, broken authentication, and service-specific vulnerabilities [4].

Moreover, the adoption of DevOps practices and CD pipelines, which may be essential for microservice agility, inadvertently introduce risks if security checks and governance mechanisms are insufficiently enforced [1]. Security issues may also arise from inconsistencies in access control, lack of centralized monitoring, improper handling of service-to-service authentication, or shared infrastructure vulnerabilities [5].

While considerable research has been conducted on microservice design and adoption, the topic of microservice vulnerabilities has only recently gained focused academic attention. A literature given in [1] synthesized insights from 62 peer-reviewed studies, identifying 126 distinct vulnerabilities categorized across six core domains. Their study combined a systematic literature review with empirical analysis of benchmark systems to derive both a conceptual taxonomy and real-world vulnerability examples.

To complement this taxonomy with industry perspectives, this paper also draws from the empirical survey conducted by the authors of [2], who interviewed and surveyed practitioners involved in the migration from monolithic and SOA-based systems to microservices. Their findings highlight practical security and engineering challenges, such as tightly coupled legacy code, difficulty in defining service boundaries, data migration issues, and insufficient initial infrastructure for secure deployment. These issues, while architectural in nature, often intersect directly with security vulnerabilities observed in practice.

In this paper, we conduct a targeted literature review of 32 selected peer-reviewed studies on microservice vulnerabilities. Our inclusion criteria focused on papers published in English, with duplicate and non-relevant sources excluded through manual filtering. Based on this review, we identify key vulnerability types, provide a synthesized overview of current mitigation efforts, and propose practical security strategies. Our goal is to bridge the gap between theoretical taxonomies and real-world deployment challenges, offering actionable guidance for developers, architects, and researchers concerned with securing microservice-based systems.

The rest of the paper is organized as follows. Section 2 gives a brief overview on microservices, their advantages and disadvantages over other types of software architectures, mainly comparing with the monolithic architecture. In Section 3, we present four methodologies and methods of obtaining and filtering papers for the review. Based on the identified vulnerabilities, Section 4 proposes some mitigation techniques. Finally, Section 5 concludes the paper.

2. Microservice Architecture: Advantages, vulnerabilities and Trade-offs

Microservice architecture design presents a paradigm shift from traditional monolithic software systems. Unlike monoliths, where all components are tightly integrated into a single deployable unit, microservices divide a system into a set of small, usually fully independently deployable services, where each service is responsible for a distinct business capability [1, 2]. This decentralized approach supports scalability and flexibility; However, it introduces a range of operational, architectural, and, for the purpose of this paper, security-related challenges that must be addressed holistically.

In monolithic systems, application components such as business logic, user interface, and data access layers are bundled and deployed as a single executable unit. While this model simplifies development and deployment in early stages, it often becomes a bottleneck as systems grow in complexity. Common issues include tight coupling between modules, difficulties in scaling individual components, limited fault tolerance, and long release cycles due to integration dependencies [3, 6, 7].

Microservices, by contrast, break down applications into loosely coupled services, each running in its own process and communicating through lightweight protocols such as HTTP/REST or messaging queues. These services are organized around business domains and can be developed, deployed, and scaled independently [1, 4]. This model allows teams to work in parallel, adopt polyglot technology stacks, and rapidly respond to changes in user demands or business strategy.

Table 1 and diagram summarize key differences between monolithic and microservice architectures:

Table 1
Comparison of Monolithic and Microservice Architectures

Characteristic	Monolithic Architecture	Microservice Architecture
Architecture Style	Single-tiered application	Decentralized services
Deployment	Deployed as one unit	Each service deployable independently
Scalability	Scale as a whole	Individually scalable
Fault Isolation	Harder to isolate faults	High fault isolation
Technology Stack	Usually one stack	Polyglot programming supported
Development Teams	One large team	Smaller, independent teams
Data Management	Centralized database	Distributed databases
Release Cycle	Slower, coordinated releases	Frequent, independent releases
Complexity	Simple to develop initially	Complex to manage
Security Surface	Smaller surface area	Larger, distributed surface

2.1. Advantages of using Microservices

One of the most notable advantages of using microservice-based architecture in software development is scalability. Since services are deployed independently, only the components that experience increased load need to be scaled, resulting in the optimization of resource usage and cost. For example, an e-commerce platform can scale its payment and checkout services during peak shopping hours without affecting the product catalog or recommendation engine [8]. In addition, each microservice can be deployed on its own timeline, allowing for CD and faster release cycles. This agility in service deployment is essential in dynamic business environments and supports A/B testing, feature toggling, and blue-green deployments with minimal risk [5]. Furthermore, by isolating individual services, microservice-based architectures increases overall system resilience. A failure in one service does not necessarily bring down the entire application. Circuit breakers, retries, and timeout mechanisms further enhance reliability, especially when combined with service mesh technologies like [9, 10].

Software development teams are free to choose the most appropriate language, framework, or database for a given service, based on the requirements and team expertise. This polyglot nature encourages innovation but requires well-defined API contracts and strong observability to maintain coherence [11]. Finally, microservices align well with DevOps practices. Independent teams can own the lifecycle of their services—from development and testing to deployment and monitoring. This fosters accountability, autonomy, and faster issue resolution, reducing communication overhead and dependency bottlenecks [12, 13, 14].

2.2. Challenges and Disadvantages of using Microservices

Despite the given benefits, the adoption of microservices also has significant challenges, particularly in complexity management, but also in security. This approach is not a one-size-fits-all solution. It is best suited for large-scale applications with multiple teams, independent domain boundaries, and requirements for high availability and continuous delivery. For small-scale applications or teams without prior experience in distributed systems, the overhead may outweigh the benefits.

For instance, a system which is composed of dozens or even hundreds of microservices is by default much more complex than a monolith-based system. Issues such as service discovery, distributed tracing, configuration management, and version compatibility arise in these systems, which must be addressed using infrastructure tools that are often unfamiliar to development teams [1, 15].

Differently from monolith-based systems, that use a single centralized database, microservices highlight decentralized data ownership; However, this introduces the challenge of maintaining consistency across those services without violating the principles of autonomy. Distributed transactions are discouraged, and eventual consistency models require careful design of compensating actions and idempotent operations [16].

Microservices, due to their distributed nature, expose a significantly larger surface area for security threats. Each service exposes interfaces that must be properly secured against unauthorized access, injection attacks, and denial-of-service (DoS) threats. Authentication and authorization must be consistently enforced across services, often using tokens like JWT and identity providers like OAuth2, OpenID Connect [17, 18]. Additionally, misconfigurations in container orchestration platforms like Kubernetes may introduce vulnerabilities that are hard to detect in traditional testing [19].

When it some to testing these systems, testing microservice-based ones is considerably more difficult due to their asynchronous interactions and interdependencies. Unit testing is insufficient; integration and end-to-end testing must be coordinated across multiple services. Debugging failures requires sophisticated distributed tracing and log aggregation tools like Jaeger, Zipkin, or ELK stacks [20, 21, 22].

Finally, managing the lifecycle of microservices requires advanced deployment strategies such as canary releases, service meshes, and configuration injection. Real-time observability becomes crucial to identify performance bottlenecks and failures. Tools such as Prometheus, Grafana, and Fluentd are often essential in production setups [15].

3. Review methodology

This Section outlines the methodology used to conduct a structured literature review on security vulnerabilities in microservice architectures. The objective of the review was to identify and analyze peer-reviewed academic works that provide insights into the types, causes, and mitigations of vulnerabilities within microservice systems. Our approach was grounded in established guidelines for systematic literature reviews in software engineering, particularly those proposed in [23]. The methodology involved four key steps – paper collection, inclusion and exclusion filtering, classification and taxonomy alignment, and finally synthesis of vulnerability domains.

The initial corpus of literature was collected from reputable scientific databases including IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and Scopus. The search was conducted using a combination of keywords such as “microservices”, “vulnerabilities”, “security”, “threats”, “architecture”, and “cybersecurity”. Boolean operators were used to refine the search queries (e.g., “microservices AND security vulnerabilities”). The search was restricted to publications written in English and published between 2015 and 2024, reflecting the growing maturity of microservice adoption in both industry and academia during this period. The initial search yielded a total of 212 articles. Duplicates were automatically removed based on DOI and title matching.

The following inclusion criteria were applied to filter the papers:

- The study must focus on microservice architecture or systems explicitly using MSA principles.
- The study must address security vulnerabilities, threats, or defense mechanisms.
- The study must be peer-reviewed (conference papers, journal articles, or book chapters).
- The study must provide either empirical evidence, conceptual frameworks, or case studies.

Exclusion criteria included:

- Non-English publications.
- Non-peer-reviewed materials (e.g., blogs, whitepapers, editorials).
- Studies focusing solely on performance, deployment, or development without security context.
- Papers dealing exclusively with general cloud security or SOA without clear microservice relevance.

After applying the above filters, a refined set of 62 papers was obtained. This set includes theoretical analyses, vulnerability taxonomies, tool evaluations, and empirical case studies.

To organize the selected studies, we adopted the vulnerability taxonomy proposed by Jayalath et al. [1], which categorizes vulnerabilities into six key domains:

1. Network and Communication
2. Data Consistency and Integrity
3. Authentication and Authorization
4. Deployment and Orchestration
5. Heterogeneous Features
6. DevOps and CI/CD Practices

Each paper was manually analyzed and mapped to one or more of these categories based on the primary security concern it addressed. Where necessary, additional tags were added for cross-cutting concerns such as service mesh integration, API gateways, or container-specific threats.

This classification helped structure the analysis in a consistent and reproducible manner, ensuring that findings from disparate studies could be compared and synthesized across similar vulnerability domains. In the final phase, the selected papers were subjected to qualitative synthesis. Papers were grouped according to their mapped taxonomy categories, and their findings were aggregated to identify recurring vulnerability patterns, critical risks, and proposed mitigations.

Special attention was given to studies that included empirical data—such as benchmarks, case studies, or vulnerability scans on real systems—as these provide practical grounding to otherwise theoretical models. Additionally, insights from the industrial survey by Di Francesco et al. [2] were integrated to validate whether the theoretical vulnerabilities align with practical challenges encountered during microservice migration and operation. The synthesis phase also involved identifying gaps in the literature, such as underexplored areas. e.g., heterogeneous deployment environments, dynamic service composition, or lack of validated mitigation strategies for certain vulnerability types. These gaps informed the recommendations and future work directions discussed in subsequent sections.

3.1. Network and Communication

Microservices rely heavily on network communication between decentralized services. This interconnectivity increases the system's attack surface, especially when services communicate over public networks or poorly secured channels. Improper encryption, weak segmentation, and vulnerable service discovery mechanisms contribute significantly to this category. This domain can be further broken down to several key sub-domains:

- Insecure Communication Protocols: Use of unencrypted HTTP or outdated TLS versions [1, 24].
- Man-in-the-Middle Attacks: Exploited due to weak certificate validation or missing TLS [25].
- Improper Input Validation in APIs: APIs exposed to injection and parsing attacks [26].
- Traffic Hijacking: Weak routing or DNS configurations allow interception [27, 28].

3.2. Data Consistency and Integrity

Data consistency is challenging in microservice systems that use distributed databases or asynchronous messaging. Lack of strong synchronization, improper data validation, and transactional boundaries across services often introduce subtle but critical vulnerabilities. Sub-domain include the following:

- Race Conditions and Concurrency Issues: Concurrent writes and updates lead to inconsistent data states [29, 30].
- Inadequate Transaction Handling: Missing rollback and commit guarantees in distributed workflows [31, 32, 33].
- Weak Encryption/Hashing: Use of deprecated or default configurations for data-at-rest security [34, 35, 36].
- Missing Integrity Checks: Absence of hashing or checksums for critical data [37, 38].

3.3. Authentication and Authorization

Ensuring robust authentication and access control is central to protecting microservices. Due to the large number of endpoints and services, poorly configured policies or weak identity federation mechanisms can introduce severe risks. The sub-domains for authentication and authorization are:

- Token Leakage/Reusability: JWT or OAuth2 tokens stored insecurely or logged accidentally [39, 40, 41].
- Lack of Multi-Factor Authentication (MFA): Single-factor setups are still common in practice [42, 43].
- Hardcoded Credentials: Credentials embedded in code repositories or container images [36, 44].
- Excessive Privilege and RBAC Flaws: Overly permissive roles and undelegated access controls [45, 46].

3.4. Deployment and Orchestration

This domain involves vulnerabilities originating from the deployment process, use of containers, orchestration platforms like Kubernetes, and CI/CD pipelines. Misconfigured infrastructure often serves as an attack vector. Sub-domains include:

- Insecure Container Images: Base images with known vulnerabilities or outdated libraries [47, 48].
- Privilege Escalation in Pods: Containers allowed to run as root or with host-level privileges [49].
- Misconfigured Orchestration Dashboards: Exposed Kubernetes dashboards without access controls [50, 51].
- Insecure CI/CD Pipelines: Secrets exposed in pipeline logs or insecure runners [52, 53].

3.5. Heterogeneous Features

Microservices are often developed using diverse languages, platforms, and libraries. While this allows agility, it can introduce inconsistencies in how security is implemented and monitored across services, with sub-domains:

- Inconsistent Security Practices: Varying standards for authentication, encryption, and logging [26, 54].
- Outdated/Unsupported Libraries: Use of unmaintained packages with known CVEs [55].
- Platform Misconfigurations: Divergent configurations in cloud environments, containers, or APIs [56].
- Patch Management Challenges: Difficulty synchronizing patches across decentralized services [57].

3.6. DevOps and CI/CD Practices

DevOps pipelines are often the backbone of microservice lifecycle management. If not securely configured, these pipelines can introduce vulnerabilities during build, test, or deployment phases. Sub-domains are:

- Lack of Security Scanning (SAST/DAST): Absence of automated vulnerability checks in CI/CD [58].
- Improper Secrets Management: Credentials hardcoded in pipelines or stored unencrypted [59].
- Uncontrolled Rollback or Recovery: Lack of safe rollbacks for failed or compromised builds [60].

The summary of the security vulnerability domains and sub-domains are shown in Table 2.

Table 2
Microservice Vulnerability Domains with Subcategories and References

Category	Subcategory	References
Network and Communication	Insecure Communication Protocols	[1, 24]
	Man-in-the-Middle Attacks	[25]
	Improper Input Validation in APIs	[26]
	Traffic Hijacking	[27, 28]
Data Consistency and Integrity	Race Conditions and Concurrency Issues	[29, 30]
	Inadequate Transaction Handling	[31, 32, 33]
	Weak Encryption or Hashing	[34, 35, 36]
	Missing Integrity Checks	[37, 38]
Authentication and Authorization	Token Leakage/Reusability	[39, 40, 41]
	Lack of Multi-Factor Authentication (MFA)	[42, 43]
	Hardcoded Credentials	[36, 44]
	Excessive Privilege and RBAC Flaws	[45, 46]
Deployment and Orchestration	Insecure Container Images	[47, 48]
	Privilege Escalation in Pods	[49]
	Misconfigured Orchestration Dashboards	[50, 51]
	Insecure CI/CD Pipelines	[52, 53]
Heterogeneous Features	Inconsistent Security Practices	[26, 54]
	Outdated/Unsupported Libraries	[55]
	Platform Misconfigurations	[56]
	Patch Management Challenges	[57]
DevOps and CI/CD Practices	Lack of Security Scanning (SAST/DAST)	[58]
	Improper Secrets Management	[59]
	Uncontrolled Rollback or Recovery	[60]

4. Mitigation Techniques

Securing microservice-based systems requires a multilayered approach that integrates security practices at the architectural, operational, and infrastructural levels. Given the taxonomy of vulnerabilities previously outlined, this section discusses mitigation strategies that align with each domain, drawing from peer-reviewed studies, industry best practices, and real-world deployment experiences.

In the domain of network and communication, protecting inter-service communication is critical. All communication—whether external or internal—must be encrypted using up-to-date protocols such as TLS 1.2 or higher. The adoption of mutual TLS (mTLS) within service meshes like Istio or Linkerd has emerged as a preferred approach to authenticate services and ensure message confidentiality and integrity. API gateways serve as a central enforcement point and should be configured to validate input schemas, enforce rate limits, and restrict access using IP whitelisting or geofencing mechanisms. In parallel, service discovery and internal routing must be hardened to prevent hijacking and spoofing attacks. This can be achieved by using secure service registries, encrypting DNS traffic, and limiting the exposure of service endpoints through internal-only networking configurations.

Data consistency and integrity pose another critical concern in distributed microservice systems. To mitigate issues arising from eventual consistency and transactional fragmentation, developers increasingly rely on design patterns such as the saga pattern, outbox pattern, and compensating transactions. These approaches allow for logical rollback and coordination across independently managed services. Sensitive data stored or transmitted between services must be protected using robust cryptographic mechanisms—AES-256 encryption for data at rest and TLS 1.3 for data in transit are generally recommended. Furthermore, services should implement hashing and digital signature mechanisms for critical data flows, especially in contexts such as financial transactions or audit logs. Concurrency-related vulnerabilities like race conditions can be addressed by applying version-based optimistic locking or database-level isolation controls to ensure integrity during concurrent operations.

Authentication and authorization in microservices require the adoption of centralized and federated identity solutions. Modern systems typically integrate with identity providers such as Keycloak, Auth0, or Azure Active Directory, which offer support for protocols like OAuth2 and OpenID Connect. Security tokens, such as JWTs, should be short-lived and securely rotated to minimize the risk of token leakage. Refresh tokens must be protected using secure storage practices, and administrative access should always require multi-factor authentication (MFA). In environments with numerous services, enforcing consistent role-based or attribute-based access control (RBAC/ABAC) policies is essential. These access controls should be configured at both the API gateway level and within service logic, ensuring defense in depth and limiting the scope of privilege escalation opportunities.

In deployment and orchestration, the security of containerized environments and orchestrators like Kubernetes is of paramount importance. Containers should adhere to the principle of least privilege, avoiding the use of root access and restricting kernel capabilities using Linux primitives such as Seccomp, AppArmor, or SELinux. Container images must be built from minimal, trusted base images and should be signed using tools such as Cosign or Docker Notary to ensure authenticity. Kubernetes clusters should be hardened by disabling insecure default features, restricting dashboard access, enabling audit logging, and enforcing network segmentation via NetworkPolicies. Secrets and credentials must not be hardcoded in manifests or stored in plaintext environment variables; instead, they should be managed using dedicated secret management platforms like HashiCorp Vault, AWS Secrets Manager, or Kubernetes sealed secrets.

The heterogeneity of microservice implementations—often involving multiple programming languages, frameworks, and deployment environments—poses unique challenges. Organizations must define and enforce security baselines for each technology stack in use, including standardized practices for input validation, encryption, and logging. Logging and monitoring should be centralized and uniform across services using stacks such as ELK, Prometheus, or Grafana, enabling correlation of events and early detection of anomalies. Additionally, all service dependencies should be scanned continuously for known vulnerabilities using tools like Snyk, Trivy, or Dependabot, with automated patching where feasible. Security as Code practices, such as linting configuration files, scanning infrastructure-as-code manifests, and applying policy-as-code enforcement (e.g., Open Policy Agent), can help maintain consistency and prevent drift across diverse environments.

Finally, the DevOps and CI/CD lifecycle represents a frequent source of vulnerabilities when security is not embedded from the outset. Static and dynamic application security testing (SAST/DAST) should be integrated into the CI/CD pipeline, using tools such as SonarQube, OWASP ZAP, or GitHub Advanced Security to catch vulnerabilities early in the development process. Infrastructure as Code (IaC) configurations must be version-controlled and scanned for misconfigurations using tools like Checkov or Terrascan. Secrets management must also be tightly controlled; credentials should never be stored in plaintext, and pipelines must be instrumented to detect accidental exposure via logs or commits. Deployment strategies such as canary releases or blue/green deployments enable safe rollouts and fast recovery, providing resilience in the event of a vulnerability or performance regression in newly deployed services.

In summary, mitigation of vulnerabilities in microservice-based systems requires comprehensive, domain-specific techniques that span communication, data, identity, deployment, heterogeneity, and operations. These controls must not only be technically sound but also automated, auditable, and consistently applied across the entire software delivery lifecycle to ensure sustainable, scalable security.

5. Conclusion

Microservices have revolutionized modern software engineering by offering scalability, modularity, and rapid deployment capabilities that far surpass the monolithic model. However, as this paper has shown, the distributed and decentralized nature of microservice-based systems also introduces a broad spectrum of security vulnerabilities across multiple architectural and operational domains. Through a systematic literature review of 32 selected peer-reviewed studies, we identified critical vulnerabilities grouped into six key domains: network and communication, data consistency and integrity, authentication and authorization, deployment and orchestration, heterogeneous features, and DevOps/CI/CD practices.

We have examined each of these domains in depth, classifying subcategories and highlighting risks that emerge from insecure APIs, inconsistent encryption, improper access control, misconfigured orchestration environments, and insecure automation pipelines. To address these challenges, we proposed a set of targeted mitigation strategies derived from both academic research and industrial best practices. These include the use of mutual TLS, centralized identity management, secrets management platforms, container hardening, policy-as-code enforcement, and the integration of security testing throughout the CI/CD lifecycle.

Our findings indicate that securing microservices is not a matter of point fixes but requires a holistic, continuous, and proactive security strategy. This involves collaboration across development, operations, and security teams to implement consistent and automated safeguards. Moreover, many vulnerabilities are not the result of obscure technical failures but stem from misconfigurations, inconsistent governance, and inadequate tooling—problems that can be effectively mitigated with proper processes and security-by-design thinking.

Future research should focus on extending empirical evaluations of microservice vulnerabilities in real-world deployments, developing automated and intelligent detection mechanisms, and refining security frameworks that can adapt to evolving architectures such as serverless computing and edge-based microservices. As microservices continue to be adopted at scale, a shared understanding between academia and industry will be essential for building resilient, trustworthy systems.

Acknowledgment

This paper was supported by the Blockchain Technology Laboratory at Belgrade Metropolitan University, Belgrade, Serbia.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] R. K. Jayalath, H. Ahmad, D. Goel, M. S. Syed, F. Ullah, Microservice vulnerability analysis: A literature review with empirical insights, IEEE Access (2024).

[2] P. Di Francesco, P. Lago, I. Malavolta, Migrating towards microservice architectures: an industrial survey, in: 2018 IEEE international conference on software architecture (ICSA), IEEE, 2018, pp. 29–2909.

[3] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, Present and ulterior software engineering (2017) 195–216.

[4] S. Newman, Building microservices: designing fine-grained systems, " O'Reilly Media, Inc.", 2021.

[5] D. Taibi, V. Lenarduzzi, C. Pahl, Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation, IEEE Cloud Computing 4 (2017) 22–32.

[6] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term, MartinFowler. com 25 (2014) 12.

[7] V. K. Thatikonda, H. R. V. Mudunuri, Microservices vs. monoliths: Choosing the right architecture for your project, International Journal of Software Computing and Testing 10 (2024) 31–38p.

[8] C. Richardson, Microservices patterns: with examples in Java, Simon and Schuster, 2018.

[9] R. Chandramouli, Z. Butcher, et al., Building secure microservices-based applications using service-mesh architecture, NIST Special Publication 800 (2020) 204A.

[10] A. Khatri, V. Khatri, Mastering Service Mesh: Enhance, secure, and observe cloud-native applications with Istio, Linkerd, and Consul, Packt Publishing Ltd, 2020.

[11] A. Cockcroft, Migrating to Cloud-Native Architecture - Netflix TechBlog, 2016.
[12] N. Forsgren, J. Humble, The role of continuous delivery in it and organizational performance, Forsgren, N., J. Humble (2016)." The Role of Continuous Delivery in IT and Organizational Performance." In the Proceedings of the Western Decision Sciences Institute (WDSI) (2016).
[13] L. Chen, Microservices: architecting for continuous delivery and devops, in: 2018 IEEE International conference on software architecture (ICSA), IEEE, 2018, pp. 39–397.
[14] R. Heinrich, A. Van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, J. Wettinger, Performance engineering for microservices: research challenges and directions, in: Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion, 2017, pp. 223–226.
[15] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, J. Wilkes, Borg, omega, and kubernetes, Communications of the ACM 59 (2016) 50–57.
[16] P. Helland, Life beyond distributed transactions: an apostate's opinion, Queue 14 (2016) 69–98.
[17] R. F. Cardoso, A performance comparison of authentication and authorization patterns for microservices applications (2024).
[18] N. Dimitrijević, N. Zdravković, M. Bogdanović, A. Mesterovic, Advanced Security Mechanisms in the Spring Framework: JWT, OAuth, LDAP and Keycloak, in: Proceedings of the 14th International Conference on Business Information Security (BISEC 2023), 2024, pp. 64–70.
[19] P. Raj, S. Vanga, A. Chaudhary, Cloud-Native Computing: How to design, develop, and secure microservices and event-driven applications, John Wiley & Sons, 2022.
[20] G. Hohpe, B. Woolf, Enterprise integration patterns, in: 9th conference on pattern language of programs, Citeseer, 2002, pp. 1–9.
[21] M. Waseem, P. Liang, G. Márquez, A. Di Salle, Testing microservices architecture-based applications: A systematic mapping study, in: 2020 27th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2020, pp. 119–128.
[22] M. Waseem, P. Liang, M. Shahin, A. Di Salle, G. Márquez, Design, monitoring, and testing of microservices systems: The practitioners' perspective, Journal of Systems and Software 182 (2021) 111061.
[23] S. Keele, et al., Guidelines for performing systematic literature reviews in software engineering, Technical Report, Technical report, ver. 2.3 ebse technical report. ebse, 2007.
[24] E. Paul, O. Callistus, O. Somtobe, T. Esther, K. Somto, O. Clement, I. Ejimofor, Cybersecurity strategies for safeguarding customer's data and preventing financial fraud in the united states financial sectors, International Journal on Soft Computing 14 (2023) 01–16.
[25] B. Ünver, R. Britto, Automatic detection of security deficiencies and refactoring advises for microservices, in: 2023 IEEE/ACM International Conference on Software and System Processes (ICSSP), IEEE, 2023, pp. 25–34.
[26] A. R. Nasab, M. Shahin, S. A. H. Raviz, P. Liang, A. Mashmool, V. Lenarduzzi, An empirical study of security practices for microservices systems, Journal of Systems and Software 198 (2023) 111563.
[27] K. R. Gade, Cloud-native architecture: Security challenges and best practices in cloud-native environments, Journal of Computing and Information Technology 2 (2022).
[28] T. Theodoropoulos, L. Rosa, C. Benzaid, P. Gray, E. Marin, A. Makris, L. Cordeiro, F. Diego, P. Sorokin, M. D. Girolamo, et al., Security in cloud-native services: A survey, Journal of Cybersecurity and Privacy 3 (2023) 758–793.
[29] R. Lu, Detecting race conditions in distributed concurrent systems (2000).
[30] J. C. Pereira, N. Machado, J. Sousa Pinto, Testing for race conditions in distributed systems via smt solving, in: International Conference on Tests and Proofs, Springer, 2020, pp. 122–140.
[31] A. De Iasio, E. Zimeo, Avoiding faults due to dangling dependencies by synchronization in microservices applications, in: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2019, pp. 169–176.
[32] A. De Iasio, E. Zimeo, A framework for microservices synchronization, Software: Practice and Experience 51 (2021) 25–45.
[33] X. Gu, Q. Wang, Q. Yan, J. Liu, C. Pu, Sync-millibottleneck attack on microservices cloud archi-

tecture, in: Proceedings of the 19th ACM ASIA Conference on Computer and Communications Security, 2024, pp. 799–813.

[34] C. S. S. Team, Cryptographic storage cheat sheet, Cryptographic Storage. url: https://cheatsheetseries. owasp. org/cheatsheets/Cryptographic _ Storage _ Cheat_Sheet. html (visited on 11/07/2024) (2023).

[35] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, A. Barros, Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency, Ieee Software 35 (2017) 63–72.

[36] U. Zdun, P.-J. Queval, G. Simhandl, R. Scandariato, S. Chakravarty, M. Jelic, A. Jovanovic, Microservice security metrics for secure communication, identity management, and observability, ACM transactions on software engineering and methodology 32 (2023) 1–34.

[37] T. Yarygina, A. H. Bagge, Overcoming security challenges in microservice architectures, in: 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE), IEEE, 2018, pp. 11–20.

[38] B. Barua, M. S. Kaiser, Blockchain-based trust and transparency in airline reservation systems using microservices architecture, arXiv preprint arXiv:2410.14518 (2024).

[39] A. Bánáti, E. Kail, K. Karóczkai, M. Kozlovszky, Authentication and authorization orchestrator for microservice-based software architectures, in: 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, 2018, pp. 1180–1184.

[40] D. Yu, Y. Jin, Y. Zhang, X. Zheng, A survey on security issues in services communication of microservices-enabled fog applications, Concurrency and Computation: Practice and Experience 31 (2019) e4436.

[41] A. Chatterjee, M. W. Gerdes, P. Khatiwada, A. Prinz, Sftsdh: Applying spring security framework with tsd-based oauth2 to protect microservice architecture apis, IEEE Access 10 (2022) 41914–41934.

[42] D. Wang, X. Zhang, Z. Zhang, P. Wang, Understanding security failures of multi-factor authentication schemes for multi-server environments, Computers & Security 88 (2020) 101619.

[43] A. M. Mostafa, M. Ezz, M. K. Elbashir, M. Alruily, E. Hamouda, M. Alsarhani, W. Said, Strengthening cloud security: an innovative multi-factor multi-layer authentication framework for cloud user authentication, Applied Sciences 13 (2023) 10871.

[44] C. K. Rudrabhatla, Security design patterns in distributed microservice architecture, arXiv preprint arXiv:2008.03395 (2020).

[45] R. A. Al-Wadi, A. A. Maaita, Authentication and role-based authorization in microservice architecture: A generic performance-centric design, Journal of Advances in Information Technology 14 (2023) 758–768.

[46] P. Thakur, N. S. Talwandi, S. Khare, Securing microservice architecture: Load balancing and role-based access control, in: 2024 IEEE Third International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), IEEE, 2024, pp. 1113–1117.

[47] S. Sultan, I. Ahmad, T. Dimitriou, Container security: Issues, challenges, and the road ahead, IEEE access 7 (2019) 52976–52996.

[48] G. Dell'Immagine, J. Soldani, A. Brogi, Kubehound: Detecting microservices' security smells in kubernetes deployments, Future Internet 15 (2023) 228.

[49] A. Sharma, Eliminating Misconfiguration and Privilege Escalation in Docker Images, Ph.D. thesis, Dublin, National College of Ireland, 2021.

[50] Z. Wen, T. Lin, R. Yang, S. Ji, R. Ranjan, A. Romanovsky, C. Lin, J. Xu, Ga-par: Dependable microservice orchestration framework for geo-distributed clouds, IEEE Transactions on Parallel and Distributed Systems 31 (2019) 129–143.

[51] V. Mahavaishnavi, R. Saminathan, R. Prithviraj, Secure container orchestration: A framework for detecting and mitigating orchestrator-level vulnerabilities, Multimedia Tools and Applications (2024) 1–21.

[52] P. Bajpai, A. Lewis, Secure development workflows in ci/cd pipelines, in: 2022 IEEE Secure Development Conference (SecDev), IEEE, 2022, pp. 65–66.

[53] A. K. Bhardwaj, P. Dutta, P. Chintale, Securing container images through automated vulnerability

detection in shift-left ci/cd pipelines, Babylonian Journal of Networking 2024 (2024) 162–170.

[54] F. Ponce, J. Soldani, H. Astudillo, A. Brogi, Microservices security: Bad vs. good practices, in: European Conference on Software Architecture, Springer, 2022, pp. 337–352.

[55] S. Pfleger, Data Security in Microservice-Systems, Ph.D. thesis, University of Applied Sciences, 2024.

[56] M. S. I. Shamim, Mitigation of Security Misconfigurations in Kubernetes-based Container Orchestration: A Techno-educational Approach, Ph.D. thesis, Auburn University, 2024.

[57] K. A. Torkura, M. I. Sukmana, C. Meinel, Integrating continuous security assessments in microservices and cloud native applications, in: Proceedings of The10th International Conference on Utility and Cloud Computing, 2017, p. 171–180.

[58] K. Oha, Advancements In Microservice Architectures: Tackling Data Communication, Scalability, And CI/CD Automation Challenges, Ph.D. thesis, Hochschule Rhein-Waal, 2024.

[59] M. Ahmadvand, A. Pretschner, K. Ball, D. Eyring, Integrity protection against insiders in microservice-based infrastructures: From threats to a security framework, in: Software Technologies: Applications and Foundations: STAF 2018 Collocated Workshops, Toulouse, France, June 25-29, 2018, Revised Selected Papers, Springer, 2018, pp. 573–588.

[60] D. R. Matos, M. L. Pardal, A. R. Silva, M. Correia, μverum: Intrusion recovery for microservice applications, IEEE Access 11 (2023) 78457–78470.