Towards a Security Benchmark for the Architectural Design of Microservice Applications

Anusha Bambhore Tukaram
anusha.bambhoretukaram@tuhh.de
Hamburg University of Technology
Hamburg, Germany

Simon Schneider
simon.schneider@tuhh.de
Hamburg University of Technology
Hamburg, Germany

Nicolas E. Diaz Ferreyra
nicolas.diaz-ferreyra@tuhh.de
Hamburg University of Technology
Hamburg, Germany

Georg Simhandl
georg.simhandl@univie.ac.at
University of Vienna
Vienna, Austria

Uwe Zdun
uwe.zdun@univie.ac.at
University of Vienna
Vienna, Austria

Riccardo Scandariato
scandariato@tuhh.de
Hamburg University of Technology
Hamburg, Germany

ABSTRACT

The microservice architecture presents many challenges from a security perspective, due to the large amount of services, leading to an increased attack surface and an unmanageble cognitive load for security analysts. Several benchmarks exist to guide the secure configuration of the deployment infrastructure for microservice applications, including containers (e.g., Docker), orchestration systems (e.g., Kubernetes), cloud platforms (e.g., AWS), and even operating systems (e.g., Linux). In this paper we approach the creation of a benchmark for the design of the microservice applications themselves. To this aim, we inventorize a number of relevant security rules for the architectural design of microservice applications and assess (in a preliminary way) how these rules could be checked automatically.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

microservices, security, architecture, rules, constraints

ACM Reference Format:
Anusha Bambhore Tukaram, Simon Schneider, Nicolas E. Diaz Ferreyra, Georg Simhandl, Uwe Zdun, and Riccardo Scandariato. 2022. Towards a Security Benchmark for the Architectural Design of Microservice Applications. In *The 17th International Conference on Availability, Reliability and Security (ARES 2022), August 23–26, 2022, Vienna, Austria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3538969.3543807

1 INTRODUCTION

The microservice architectural style organizes an application as a composition of services that have their own scoped responsibility and implement a self-contained business capability. The size of a microservice should be adequate for one team to develop and test it. Naturally, dependencies are unavoidable and microservice can rely on other services. However, the goal is to create loosely coupled services so that they can be developed and deployed independently. Loose coupling implies, for instance, that there is no database shared across microservices and that there are no code dependencies requiring code changes in lock-steps on multiple services. Also, the API of a service should be small, stable over time, and designed in a such a way that the internal workings of the service are not exposed. For instance, there should not be any expectation on behavior or state changes across multiple invocations of the same service.

The microservice architecture has become very popular as it reflects the needs of agile development teams working in a continuous integration and continuous delivery way. Further, services lend themselves to be containerized and deployed in the cloud, which is another major trend in the software industry.

On the flip side, microservice architecture presents many challenges from a security perspective, due to the large amount of services combined with their exposure to attacks over the Internet. These challenges have been described in several academic papers [? ? ?] and are abundantly mentioned in the gray literature from professionals [? ?]. In particular, commonly mentioned challenges refer to the problem of establishing trust between services via access control, the issue of an increased attack surface, and the problem of secret management.

The analysis of the above-mentioned literature reveals that a principled approach to securing microservice architectures from the ground up is still yet to come. Yarygina and Bagge [?] have identified 6 abstraction levels at which security measures need to be defined. These levels include (i) the lower levels of hardware, virtualization, and cloud, as well as (ii) the higher levels of communication (e.g., using mTLS, or JWT security), services (e.g., using a logging sidecar, or protecting data when stored), and orchestration (e.g., using secure service discovery). Concerning the lower levels, several guidelines have been developed (e.g., by CIS, the Center for Internet Security[1]) in the form of benchmarks. These benchmarks contain rules that need to be followed in order to avoid security flaws and be in conformance with the benchmark. Typically, these benchmarks contain rules that can be checked automatically via

ARES 2022, August 23–26, 2022, Vienna, Austria

ACM ISBN 978-1-4503-9670-7/22/08…$15.00
https://doi.org/10.1145/3538969.3543807

[1] https://learn.cisecurity.org/benchmarks

tools like Kics[2], Checkov[3], Terrascan [4], and so on. The goal of this paper is to focus on the higher levels and set the first steps towards the construction of a security benchmark that could be used to evaluate the architectural design of a microservice application. Such a benchmark, especially if backed by automated tools, would support a security-by-design approach to the construction of microservice architectures and would be beneficial for the certification of microservice applications.

As a first step towards establishing such as benchmark, this paper addresses the following two **research objectives**:

- *RQ1: What rules should be included in a security benchmark for microservice architectures?* In particular, we are interested in rules that set security constraints and define checks that are applicable at the level of architectural design.
- *RQ2: What tool-based approaches already exists that could be useful to check said rules?* As we are aware that specific tools for microservice architectures do not exist, here we are more interested in collecting approaches that could provide fruitful inspiration in the creation of a benchmark tool. In future work, we plan on thoroughly analysing the identified tools with respect to the microservice-specific security architecture rules.

Accordingly, this paper makes the following **contributions**:

- We analyze the relevant documentation, guidelines, and standards to distill a small yet comprehensive set of 18 security rules. In this respect the main challenges are related to (i) the overlaps we found in the relevant documents (e.g., the same concepts are mentioned with different name, or slight variations of the same concepts are presented), and (ii) the different levels of abstraction contained in the documents, which often mix architectural concepts with low level concepts (e.g., OS configuration) and even process guidelines.
- We inventory 11 model-based security analysis approaches that could be used to validate the architectural design against the 18 rules.
- We provide an initial discussion of the applicability of the 11 approaches as a way to automate the execution of the benchmark.

The rest of the paper is organized as follows. Section 2 discusses the related work and Section 3 presents the methodology we have applied for the elaboration of the rule set. Section 4 includes the results of our analysis thus answering RQ1. Section 5 elaborates on RQ2 by discussing suitable approaches for the automatic evaluation of microservice security rules. Finally, Section 6 summarizes the threats to validity of our study and Section 7 our concluding remarks and directions for future work.

2 RELATED WORK

A number of studies on securing microservice applications have been published in recent years, including grey and white literature. Pereira et al. [?] conducted a systematic literature review of 26 academic sources and collected a set of 18 security mechanisms for microservice applications. The prevailing topics in this set are authentication, authorization, and credentials. Another systematic mapping study of 46 academic sources done by Hannousse et al. [?] yielded an ontology of security threats and mechanisms for microservices. Similar to Pereira et al., the authors state the topics that are discussed the most to be access control, protecting sensitive data, and securing individual microservices. The two studies are a valuable contribution to the body of knowledge, however we see the need for a study that focuses on architectural constraints, which does not exist in these sets of general rules.

Some authors see the academic literature to be trailing behind experience already gained in industry and thus set forth to conduct grey literature studies aiming to close this gap. A first systematic grey literature mapping was done by Soldani et al. [?], where the authors acknowledge both disadvantages and advantages of the microservice architectural style as *pains* and *gains*. The mentioned *gains* correspond to the widely known and listed above benefits, while the topics access control, centralised support, CI/CD, endpoint proliferation, human errors, and size/complexity are identified as *pains* resulting from the inherent complexity introduced by the architecture. The paper's focus, however, is on microservices Application Programming Interfaces (APIs).

A third body of knowledge aside from academic literature and practitioner experience is formed by large organizations that produce guidelines, best-practices, and similar resources (see Table 1). Although they are no standards, these documents carry a comparable reputation in industry, as the publishing organizations often enjoy large trust by developers.

Our work is different from the above related work in two major regards: (i) we focus on architectural constraints that cover the higher levels of the classification done by Yarygina and Bagge [?] and (ii) we strive to create rules that can be checked automatically.

3 METHODOLOGY

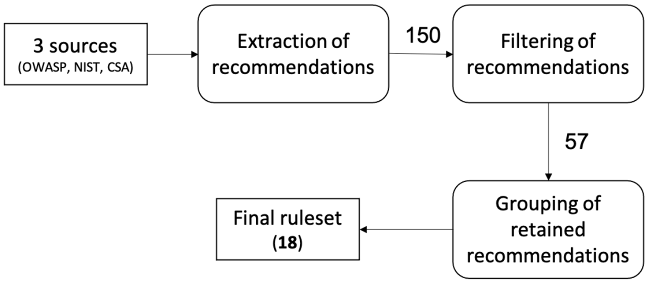
3 sources
(OWASP, NIST, CSA)
Extraction of recommendations
150
Filtering of recommendations
57
Grouping of retained recommendations
Final ruleset
(18)

Figure 1: Research methodology for the identification of the security rules.

Figure 1 shows the methodology we followed in order to identify relevant security rules. To arrive at the rules for microservices architectural security, we started from 3 sources that are contributed by well-known security organizations (OWASP, NIST, CSA) and are shown in Table 1. These sources have been identified by the authors as part of a literature survey on microservice security and by looking at the systematic literature studies mentioned in Section 2, thus concluding that the majority of the references point to these 3 sources.

[2] https://kics.io
[3] https://www.checkov.io
[4] https://runterrascan.io

Table 1: Identified sources of microservice security recommendations.

ID	Organization	Sources	Recommendations	Retained
S1	OWASP	Microservices Security – Cheat Sheet Series (https://cheatsheetseries.owasp.org/cheatsheets/Microservices_security.html)	27	20
S2	NIST	SP 800-204 – Security Strategies for Microservices-based application systems (https://csrc.nist.gov/publications/detail/sp/800-204/final)	54	21
S3	CSA	Best Practices in Implementing a Secure Microservices Architecture (https://cloudsecurityalliance.org/artifacts/best-practices-in-implementing-a-secure-microservices-architecture)	69	16
			150	57

Table 2: Filtering criteria.

Filtering Criteria	
Concrete	(i) Entails a (non) desired system behaviour or configuration **AND** (ii) lends itself to algorithmic enforcement.
In Scope	(i) The rule points towards architectural elements that can be clearly identified (e.g., modules, components, connectors, etc) **AND** (ii) should address security.

All of these sources elaborate on a set of *recommendations* (e.g., best practices, strategies, and guidelines) that address the security of microservice architectures. Hence, each source was initially analysed by one of our team members for the identification of security recommendations. We focused particularly on prescriptive statements within each source as these can be good candidates for the later elaboration of security rules.

After having identified a set of initial recommendations, all four researchers assessed each recommendation independently according to the criteria listed in Table 2. Particularly, we looked for recommendations that are architectural in nature (as opposed, e.g., to infrastructure or implementation) and are concrete enough so that the rule could be checked by either inspection or automated analysis. In case of disagreement among the individual assessments, we discussed the diverging opinions until consensus was achieved. Finally, we grouped the *retained recommendations* by considering their thematic area and their similarity. Such a final grouping was thoroughly discussed among a team of 3 researchers.

4 RESULTS

A total of 150 recommendations were identified within the examined sources: 27 from S1, 54 from S2, and 69 from S3. After applying the criteria of Table 2, 57 of these recommendations were retained: 20 from S1, 21 from S2, and 16 from S3 (as shown in Table 1). We grouped these retained recommendations into a set of 18 security rules (Table 3) and 3 technological suggestions (Table 4). From these 18 security rules, 6 correspond to *authentication/authorization*, 2 to *encryption*, 4 to *logging*, 3 to *availability*, 2 to *service registry*, and 1 to *secret management*:

(i) Rules for Authentication/Authorization (R1-R6) These rules highlight the importance of introducing API gateways for the authentication and authorization of external requests. The overall purpose of such gateways is to prevent external entities from accessing microservices in a direct way. For this, it is also important to keep both processes (i.e., authentication and authorization) decoupled from the rest of architecture and from each other to allow their reuse. Moreover, microservices should mutually authenticate and authorize each other to avoid any request that may have bypassed the API gateway. On the other hand, the representation of external entities (i.e., external access tokens) must be mapped into internal token representations in order to protect their actual identity. Finally, a limit of login attempts should be established to prevent credential abuse.

(ii) Rules for Encryption (R7-R8) Communication between services may entail the exchange of sensitive data or access permissions, which no other services in the system should hear. These rules recommend the use of encryption and secure communication protocols between external users, entities, and services to preserve the integrity and confidentiality of the information being exchanged. Thereby, the application context will be protected against tampering, and man-in-the-middle attacks.

(iii) Rules for Logging (R9-R12) A central logging subsystem with a monitoring dashboard should be implemented to detect security anomalous operations via log analysis. Such a dashboard could, for instance, display input validation failures and the status of network segments that would help identifying injection attack attempts. It is also recommended to implement local logging agents that are decoupled from the microservice but deployed under the same host. Such local agents will be responsible for collecting the log data from microservices, sanitizing such data (e.g., remove PII, passwords, and API keys) and write it to a local log file. This avoids the direct exchange of log messages between microservices and the central logging subsystem and mitigates the chances of data loss (e.g., in cases of logging service failure due to attacks). Furthermore, a message broker should be in charge of the communication between the central logging system and the local agents to enforce their mutual authentication and mitigate spoofing and traffic injection threats.

(iv) Rules for Availability (R13-R15) To avoid delayed responses or service crashed due to overload, the API gateway should perform a load balancing of the system. Additionally, a circuit breaker should be implemented to avoid cascading failures. Finally, service mesh deployments should define usage limits for their components in order to enhance the resiliency of the system.

(v) Rules for Service Registry (R16-R17) As a general rule, service registry services should (i) be deployed in dedicated services (or as part of a service mesh architecture), and (ii) implement validation checks to legitimate services. This is mainly to ensure that only legitimate services perform registrations, refresh operations, and database queries to identify microservices.

(vi) Rules for Secret Management (R18) Secrets such as API tokens, SSH keys, and passwords should be managed centrally following a Secret as Service principle. Particularly, database credentials for each application must be (i) created on-demand and (ii) revoked after a certain leasing time, in order to control their permissions.

Table 4 summarizes the technological suggestions we found in the analysed sources. Overall, these suggestions refer to secure communication/encryption (e.g., SSL/TLS, HTTP), authorization (e.g., mTLS, OAuth), and authentication (OpenID, API Keys). Both, the curated set of rules and the identified suggestions, should not be seen as final but as a first attempt towards the elaboration of a benchmark for the architectural design of microservice applications. Moreover, for the sake of replication and knowledge sharing, we provide the information gathered throughout this work as a supplementary material[5]. This includes a spreadsheet containing the assessments of each recommendation (i.e., according to the criteria defined in Table 2) and the provenance of each security rule added to the final set.

5 TOWARDS AN AUTOMATED SECURITY BENCHMARK

To answer RQ2, we conducted a preliminary assessment of state-of-the-art approaches for automatic security analysis. We have narrowed-down such an assessment to a set of 11 tool-supported methods that we have identified through an opportunistic screening of academic sources. Thereby, we elaborate suggestions for an automated benchmark of the security rules identified in the previous section.

5.1 Tools for automated benchmark

To identify the tool-supported approaches that could be used to check architecture-level security rules in microservice-based systems, we focused on the academic literature. We opted not to consider commercial tools as architectural security analysis is not yet at a maturity level that could suggest a widespread implementation of commercial solutions. Initially, we compiled a list of approaches based on our own experience as active members of the research community in this field. Next we performed a sanity check by conducting an opportunistic search on Google Scholar for academic sources referring to architectural security analysis and selected the most promising ones. Certainly, this strategy is not systematic and hence does not yield an exhaustive set of suitable approaches. Nonetheless, we believe that there are not major gaps in the identified literature, thus offering a good starting point for a preliminary analysis.

A list of eleven security analysis approaches was analysed regarding their ability to support the automatic checking of the rules presented in Section 4. Table 5 describes each of them in terms of the type of analysis they perform, whether they are tool-supported, their generated output, and modelling approach. We can observe that, in terms of modelling language, these methods often employ either a Unified Modelling Language (UML) representation of the system under analysis (A1, A2, A3), a Data Flow Diagram (DFD) (A4, A5, A6, A7), or an Architectural Description Language (ADL) specification of such system (A8 and A9). Except for A6, A9, and A10, all approaches are tool-supported and most of them employ either static analysis techniques (A1, A3, A4, A5, A6) or a combination of static and dynamic ones (A2, A10. A11). Conversely, only a few approaches (A8 and A9) rely on purely dynamic methods for security analysis. Regarding the generated output, five of these methods manage to localize security threat(s) (i.e., in code), three of them are capable to modify an architectural model (A2, A4 and A11), and two of providing suitable countermeasures (A7 and A11). Overall, this list offers a good overview of the different state-of-the-art techniques for the automatic analysis of microservice security, and can be used as a starting point for a rule-coverage assessment. That is, for identifying suitable methods that could check these rules and to identify areas of microservice security that may require further support in terms of automatic architectural analysis.

5.2 Preliminary assessment

When evaluating the approaches listed in Table 5 against the rules in Table 3, we consider the following cases:

- *Rule supported.* Here, we consider the cases where the rule can be checked by the approach either (i) out of the box, provided that the design model contains the appropriate annotations, or (ii) the approach provides the user with a language that allows the appropriate customization of the tool.
- *Support missing.* Here, either (iii) the approach is not amenable at all for this type of analysis or (iv) the approach contains enough semantics in the model that a checker could be written, but the tool would require a significant extension.

At this stage, we have not performed a full evaluation of the approaches listed in Table 5. However, we have gathered some initial remarks and observations on the analysed tools. First, we emphasise that no approach seems to support all the rules we have identified. Moreover, combining multiple tools for the benchmark (hence achieving a larger coverage of the rules) is not attainable, as each tool requires the user to prepare a specific model according to different notations.

We also noticed that some rules are completely unsupported by all the approaches. One example is rule R4, which requires the API gateway to transform external identity representations (tokens) to an internally used one. This rule is quite unique to microservices and refers to a specific microservice pattern. As the approaches in Table 5 are not designed with microservices in mind, it is quite natural that rules like this are not supported.

Finally, we noticed that the support for checking security rules on DFDs is noticeably inferior with respect to UML, where, for instance, approaches A1 and A2 already provide some coverage. Considering ADLs, approach A10 seems promising but a more thorough evaluation is necessary, also from a usability perspective.

[5] https://tinyurl.com/microservice-security-rules

Table 3: Security rules from the recommendations of OWASP, NIST and CSA.

ID	Security Rule
	Authentication / Authorization
R1	An API Gateway or similar facade should exist as a single entry point to the system and perform authorization and authentication of external requests to avoid external entities directly accessing services.
R2	Services should mutually authenticate and authorize requests from other services.
R3	Authorization and authentication processes should be decoupled from other services and should be implemented at platform level to enable reuse by different services.
R4	All the external entity identity representations should be transformed into an extendable internal identity representation. The internal identity representations should be secured with signatures and propagated but not exposed outside. They should be used for authentication and authorization at all levels.
R5	Authentication tokens should be validated.
R6	A limit for the maximum number of login attempts before preventive measures are taken should exist.
	Encryption
R7	All communication traffic from external users and entities should be encrypted using secure communication protocols.
R8	All communication between the services should be encrypted using secure communication protocols.
	Logging
R9	A central logging subsystem which includes a monitoring dashboard should exist.
R10	For all microservices, there should exist a local logging agent decoupled from the microservice but deployed on the same host. Log data from microservices should not be send to the central logging system directly, but collected by the logging agent, written to a local file, and eventually send to the central system by it.
R11	The local logging agent should sanitize the log data and remove any PII, passwords, API keys, etc.
R12	A message broker should be used to realize the communication between local logging agent and central logging system. These two should use mutual authentication and encrypt all transmitted data and availability should be ensured by providing periodic health and status data.
	Availability
R13	A circuit breaker should be used at the proxy.
R14	The API gateway should perform load balancing.
R15	Service mesh deployments should have configuration capabilities to specify resource usage limits for its components.
	Service Registry
R16	Service registry services should be deployed on dedicated servers or as part of a service mesh architecture.
R17	Service registry services should have validation checks to ensure that only legitimate services are performing the registration, refresh operations, and database queries to discover services.
	Secret Management
R18	Secrets should be managed centrally following the Secret as a Service principle.

Table 4: Security technologies that are recommended in support of the rules.

ID	Technology Suggestions
TS1	Secure communication / encryption: standard encryption protocols like SSL/TLS or HTTPS.
TS2	Authorization: mTLS, OAuth JWTs, OAuth 2.0, OIDC Tokens, API Tokens, TLS/SSL, Federation and authorization based on certificates and least privilege- RBAC. Identity propagation: Label-based identity, token-based identity, Oauth 2.0, OpenID Connect.
TS3	Authentication: mTLS/Mutual Authentication, Token based authentication, OpenID, SSL- or SASL-based authentication, API Keys, TLS/SSL, STS, Reverse STS.

In a more precise evaluation round, we plan to select a common case study and model it according to the different approaches. This will give us a more concrete basis to assess the fitness of each approach to the goal of adding the necessary annotations to the model and performing the security checks. This would also allow us to assess the intuitiveness and user-friendliness of each tool/method.

6 THREATS TO VALIDITY

Selection of sources (docs, approaches) is opportunistic, but this is exploratory work. Future work could be more systematic. We are

Table 5: Automated security analysis: approaches and tools.

ID	Approach	Tool support	Analysis	Output	Modelling Approach
A1	• Automated Software Architecture Security Risk Analysis using Formalized Signatures [?]	Yes	static	threat localization	UML
A2	• UMLSec [?] • Model-based privacy and security analysis with CARiSMA [?]	Yes	static + dynamic	modified model	UML
A3	• SecureUML: A UML-Based Modeling Language for Model-Driven Security [?] • Automated analysis of security-design models [?]	Yes	static		UML
A4	• Flaws in Flows: Unveiling Design Flaws via Information Flow Analysis [?]	Yes	static	modified model	DFD
A5	• SPARTA: Security and Privacy Architecture through Risk-driven Threat Assessment [?]	Yes	static	threat localization	DFD
A6	• Analyzing Security Architectures [?]	No			DFD
A7	• The Architectural Security Tool Suite — ARCHSEC [?] • Automatically Extracting Threats from Extended Data Flow Diagrams [?]	Yes	static	threat localization and countermeasures	DFD
A8	• Data-Driven Software Architecture for Analyzing Confidentiality [?]	Yes	dynamic	threat localization	ADL (Palladio) + DFD
A9	• A Secure Software Architecture Description Language [?]	No	dynamic		ADL (xADL)
A10	• Enforcing Architectural Security Decisions [?]	No	static + dynamic	threat localization	ADL
A11	• Architecture Modeling and Analysis of Security in Android Systems [?]	Yes	static + dynamic	modified model, threat localization, and countermeasures	ADL

very well aware of a possible selection bias, as only those sources were selected which are repeatedly referenced by the majority of academic and practitioners publications. A crucial selection criteria is the concrete applicability. While this maximizes external validity, abstract recommendations lacking actionable information were excluded. As these abstract recommendations contain valuable information for the synthesis of future security rules, these recommendations will serve as a basis for future work.

There might be a possible data extraction bias. i) The choices of variables to be extracted from these sources, ii) the quality assessment subjectivity and iii) data extraction inaccuracies were continuously discussed among the authors. All authors of this study were also involved in extraction and synthesis of security rules to overcome this bias.

It is noteworthy that the author team is well balanced. Software security researchers and software architecture researchers contribute broad knowledge and multiple perspectives to this study.

7 CONCLUSIONS AND FUTURE WORK

A benchmark for the architectural design of microservice applications should consist of rules that are unambiguously decidable and, preferably, automatically checkable. We have analyzed several referential documents that provide advise with respect to the design of this type of applications. Accordingly, we have observed that there are many overlaps and even contradictions across these documents, and no single source could be considered as complete. Further, the documents often contain a mix of recommendations that go well beyond the scope of architectural design and include rules that are already covered elsewhere (e.g., the CIS benchmarks). This might result in a frustrating experience for practitioners. Hopefully, this study helps in getting an abridged overview of the relevant security rules and provides a starting point for future research. In our own future work, we plan on providing a more precise description of the rules and on assessing the approaches more thoroughly.

ACKNOWLEDGMENTS

This work was partly funded by the European Union's Horizon 2020 programme under grant agreement No. 952647 (AssureMOSS).