Microservices Security Challenges and Approaches

Amr S. Abdelfattah
Computer Science - Baylor University
Waco, TX 76798 USA *amr_elsayed1@baylor.edu*

Tomas Cerny
Computer Science - Baylor University
Waco, TX 76798 USA *tomas_cerny@baylor.edu*

Abstract

The fast-paced development cycles of microservices applications increase the probability of insufficient security tests in the development pipelines and consequent deployment of vulnerable microservices. The distribution and ephemeral of microservices create a discoverability challenge for traditional security assessment techniques, especially for microservices being dynamically launched and de-registered. To address this in applications and networks, continuous security assessments are used for vulnerability detection. Detected vulnerabilities are thereafter patched, essentially reducing the chances for security attacks. This paper illustrates the microservices architecture and its components from the security perspective. It investigates, summarizes, and highlights the microservices security-related challenges and the suggested approaches and proposals for facing them. It addresses the security impact on the different microservice architectural perspectives.

Keywords: Microservices Security, Cloud-native Security, OAuth, Policy Enforcement Point

1. Introduction

Cloud-native systems take full advantage of distributed computing offered by the cloud delivery model. These systems embrace microservices, a popular architectural style for constructing modern applications. In this architectural style, the system is broken down into reusable, lightweight, and granular services that interact with one another [15]. Microservices are independently deployed, are loosely coupled, and are self-contained. This means that services can be developed and tested by different development teams, dramatically lowering development time. It is expected that microservices become easier to maintain due to their smaller size and scope [15]. There is a recognized need for applying security control policies in a consistent way across all microservices belonging to the same system [13].

The lack of usage of a generic threat model for microservice security is influenced by the fact that the majority of research in this area comes from the software side of the field (engineering, languages) rather than from the side of security, which advocates for a security-by-design approach [13]. While there are attacks that specifically pertain to microservices, such as those that leverage the scalability of microservice architectures to cause a denial of service, there are no dedicated threat models to help developers become aware of those particular threats [13].

This paper addresses the most common challenges and security vulnerabilities that are raised with the evolution of microservices architecture. Therefore, the paper highlights different approaches that employ the microservices architecture components to support the security-related perspective in the entire architecture. The rest of the article is organized as follows: Section 2 illustrates the background required for the security approaches. Section 3 states the security-related challenges for microservices architecture. Section 4 depicts the approaches and directions shown for overcoming the challenges. Finally, Section 5 concludes the article.

2. Background

Securing a microservices architecture is a complex task that involves the application, infrastructure, and network layers. The *application layer* protection is to ensure that the proper user controls are in place so that we can authenticate and authorize the users. While the *infrastructure layer* keeps the service running, patched, and up to date to minimize the risk of vulnerabilities. Finally, the *network layer* helps in protecting network access controls so that a service is only accessible through well-defined ports and only to a small number of authorized servers.

This paper focuses on the application layer related to securing microservices based applications. These applications architecture has distributed components that communicate with each other through REpresentational State Transfer (REST) and messaging streaming [16]. The REST architecture defines a set of constraints for how the architecture emphasizes the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems.

Applying a single rule to all these distributed components could lead to many duplications and inconsistencies over the entire application. Therefore, the API gateway plays a role as a proxy and a policy enforcement point while it separates the interactions between the internal and external components. The API gateway has responsibilities to provide the client application with API, perform request routing, provide authentication, load balancing, monitoring, composition, and protocol translation. When a client makes a request, the request transmits to the API gateway [1, 16], and further, it routes to appropriate microservices. It helps to stop exposing internal concerns to external clients. The API gateway communicates with service discovery [8, 16] which includes a central server that maintains a global view of addresses and provides a dynamic mechanism for locating the microservices routes.

Securing these distributed components requires propagating the user authentication and authorization identity during the full communication cycle. While OAuth2 is a token-based security framework that describes patterns for granting authorization but does not define how to perform authentication, it can integrate with an Identity Provider (IdP), which allows users to authenticate themselves with a third-party authentication service [16]. That means an application can take action or access resources from a server on behalf of the user without them having to share their credentials. The JSON Web Token (JWT) [5] is a standard token which defines a compact and self-contained way for securely transmitting information as a JSON object.

This paper highlights these background concepts and techniques as they are employed in different approaches and proposals to challenge the security problems that impact the nature of microservices-based applications.

3. Security-based Microservices Challenges

Microservices architecture introduces new concepts related to the communication layers and technologies. The microservices development process lacks the adoption of security-by-design [13], such that security in microservices frequently comes as an afterthought. In contrast, it should be one of the main concerns for their engineering. Microservices use diverse sets of technology stacks, each characterized by peculiar exploits. To secure microservice architectures effectively, implementors need dedicated technological references to avoid known threats.

Moreover, the authors in [13] analyzed the leakage in microservices systems. They stated that issues raised with the technology transfer from academia to industry on microservices security; lack of guidelines for adopting security by design in microservices; lack of appropriate threat models; lack of guidelines for addressing the attack surface given by technology heterogeneity; and security issues when migrating systems to microservices. Furthermore, Lwakatare, Lucy Ellen, et al. [21] showed that the migration is one of the main challenges faced in this con-

text, such that migrating applications introduces important security concerns that are difficult to track due to the lack of appropriate devices (both organizational and linguistic) to elicit them from the source codebase and make sure they hold in the migrated one.

While the microservices architecture utilizes the usage of multiple components and concepts such as Distributed Computing, REST services, Cloud Computing, and the Diversity of Development languages, unfortunately, they face the challenges related to each part of them as well. More research investigated the microservices distributed-nature-related challenges. Therefore, most of them are highlighted in [24] as follows:

- *Distributed Communication*: It places an overhead on security tasks, especially for security assessments which are traditionally configured for static network resources, hosts, and applications. Therefore, traditional security assessment techniques are challenged with discoverability problems, i.e., the capacity to constantly locate deployed microservices.
- *Ephemeral Nature of Resources*: It means that an ephemeral microservice can be destroyed at any time and then resurrected to its last known state immediately.
- *Trust Among Inter-Communicating Microservices*: All inter-communicating service instances are assumed to operate within a common security trust domain. However, this could introduce security issues. For example, an attacker who gains control over a microservice could propagate an attack against other microservices.

Since microservices components employ the REST in their implementation, the microservices architecture is influenced by the REST-related challenges, such as the automated discovery of entry and exit points. Moreover, the responses produced by RESTful applications could be dynamically generated at request time, unlike web applications whose responses are predictable [24]. Therefore, traditional security assessment techniques fail to explore REST web services, which are the core of microservice implementation [24].

Cloud-specific vulnerabilities are most related to the different kinds of deployment strategies and the way to secure them, such as the Infrastructure as a Service (IaaS) platforms and the Containers as a service (Caas). However, this kind of challenge is most delegated to application owners (cloud users); they are responsible for securing rented Virtual Machines (VMs) or containers through approaches such as configuring firewalls and security groups. It is clear that microservices still need to mature at different levels such as the lack or unavailability of specialized elements (such as firewalls) that are aware of their specificity [22]. Moreover, the independence deployment could lead to deploying the services over different cloud providers, which adds additional challenges. For example, service providers offer services that need to share this data across multiple external services as part of an offered service plan without end-users being aware of their existence [17]. Furthermore, utility-microservices [23] is the category of microservices that implement middleware functionalities to be consumed through the core-microservices. The utility-microservices container may include vulnerabilities that threaten the entire application. They often need intrusive access and capabilities in the core-microservice container. Such access can be granted via injecting the utility-microservice into the same container as the core-microservice. That causes the leaking of sensitive information (such as security keys and credentials), causing security concerns.

On the other hand, the microservices approach motivates choosing the best development languages and tools for specific problems, which leads to having multiple languages in the same application. This means applying vulnerability detection techniques in this application requires different configurations for each microservice based on the development language. Besides, developers integrate several open source components while these components could be laden with vulnerabilities [24].

Finally, Security becomes a challenging aspect since the small and independent teams need to know many aspects of security [20] and those DevOps criteria for testing, building, and de-

ployment automation. Thus, another major challenge is the coordination between development teams in the context of privacy-handling issues [19].

4. Security Approaches

Prior research efforts are focused on security measures such as authentication and authorization in microservices, e.g., using token based-authorization like JWT [5]. Other works consider effective monitoring approaches such as integrating network security monitoring with cloud Software Defined Networks (SDN) [14] that enables the network to be intelligently and centrally controlled by separating the control plane from the data plane. Some works investigate injecting security in microservices Continuous Development and Continuous Integration (CD/CI) pipeline [24].

As depicted in the challenges above, security is multidimensional in the sense that it needs to be present at multiple layers of an application and at all stages of its development. For microservices, the security models focus on the following dimensions [22]:

- *Microservice Components*: Authentication and authorization are essentially considered at this layer toward securing services. The usage of OpenID Connect [7] is a common approach for these concerns. It is built on top of the OAuth 2.0 protocol to use JWT as an ID token to standardize areas that OAuth 2.0 leaves up to choices, such as scopes and endpoint discovery.
- *Application Architecture*: The potential need for specific security components or elements, such as instrumentation and detection. The gateway and the coordination logic should be updated in order to action failover mechanisms.
- *Infrastructure*: The underlying infrastructure, such as the operating systems and the network. A good approach is to use the Docker Security Scanning add-on prior to using images. Another good practice is to plan security roles within containers rather than running root users.
- *External Interfaces*: The external interfaces used for interdomain communication, where multiple third parties may need to cooperate, each with its own security controls.

Each of the above dimensions has different requirements and different concerns in order to approach the security in its layer. *Threat models* are the ones that need to be addressed for the microservices dimensions. While there are attacks that specifically pertain to microservices, such as those that leverage the scalability of microservice architectures to cause a denial of service, there are no dedicated threat models to help developers become aware of those particular threats. Threat modeling should be performed early in the development cycle when potential issues can be caught early and remedied, preventing a much costlier fix down the line. Using threat modeling to think about security requirements can lead to proactive architectural decisions that help reduce threats from the start [10].

Many threat-modeling methods have been developed such that they can be combined to create a more robust and well-rounded view of potential threats. Not all of them are comprehensive; some are abstract, and others are human-centric. There are 12 threat-modeling methods summarized in [10]. However, STRIDE Threat Modeling [9] is the one highlighted in the context of microservices and its distributed architecture [13]. It is a model of threats that can be used as a framework for ensuring secure application design. It considers Spoofing Identity, Tampering With Data, Repudiation Threats, Information Disclosure, Denial of Service, and Elevation of Privileges. And also, the quantity of spoofing, tampering, and repudiation attacks highlights the need to address the general problem of data provenance in microservices [13]. However, the microservices' development diversity makes this modeling methodology more challenging because it must be covered through these different development languages.

Some research proposed their models, such as in [17], the authors focused on building a

model for data sharing security. They applied distributed tracing to collect tracing information for building a repository of service endpoints, service traces, and network addresses. After that, they depended on the service's attributes, such as successability, reliability, compliance, etc., to measure the risk per microservice. They used QWS Dataset [11] to test their model with data flow between microservices.

The data provenance is highlighted in [25], such that the authors demonstrated how to define the privacy requirements formally and to enable the analysis of privacy policy preservation. Their approach supports traceability between the high-level privacy description – the user's consent to disclose particular data to certain organizations – and the formal definition of the privacy-preserving constraints. They used graphical modeling to represent system architecture and the data flow. After that, the diagrams are translated into a formal modeling framework Event-B [4], to verify the impact of privacy violations and security attacks on the behavior of the system. Event-B [4] is a formal method for system-level modeling and analysis which depends on the set theory as a modeling notation.

The distributed nature of microservices introduces the need for technologies that provide global yet decentralized observability and control. For instance, tools that aid in the enforcement of security policies over a whole architecture without single points of failure [13]. Clearly, having a centralized controller that manages the orchestration of microservices helps this process and is closer in spirit to the monolithic workflow. However, the advent of multi-cloud and microservices solutions has led to a decentralization of control, making new decentralized or hybrid solutions emerge for certificate-based authentication. The authors in [12] proposed a decentralized high-fidelity city-scale emulation to verify the scalability of the authorization tier.

Security Enforcement Points are commonly used to enforce security policies at runtime. The Service Discovery is one of the components that are able to provide this capability. It provides reactive mechanisms for the detection of security issues for the reason of being in the service registration procedures that include data for performing the preventive analysis of the composition [13]. Furthermore, The API Gateway is the other component that could work as Security Enforcement Point forcing a security assessment through the whole architecture.
The authors in [24] proposed a security gateway, though similar to the microservices API gateway pattern while it differs operationally. In addition, they introduced two additional concepts: dynamic document store and security health endpoints. The dynamic document store helps the security gateway to overcoming the challenge of exploring the REST web services. This approach uses OpenAPI 2 (formerly Swagger) for generating this document. However, the static analysis and microservices reconstruction approach should work perfectly, in this case, to automate this critical part of the process as illustrated in [15]. The security health endpoint effectively affords security observability by providing security health information for every deployed microservice instance. Health Endpoint Monitoring Pattern provides for periodic health checks against applications via heartbeat checks. This health endpoint to situational awareness fulfills the observability tenet of microservices. Moreover, this approach highlighted two important concerns for addressing important challenges, first is the Security gateway should not be discoverable by other microservices except the core services, e.g., service registry and discovery and API Gateway. Secondly, the solution should be able to automatically identify the development technology and test it accordingly.

Moreover, Security-as-a-Service (SecaaS) [14] is a model that provides security services like data loss prevention, antivirus management, and intrusion detection. Thus, the API gateway may be configured to grant access to such services after due authentication and authorization. Following this direction, Sun et al. [14] proposed a design for SecaaS for microservices-based cloud applications. They added a new API primitive FlowTap for the network hypervisor to flexible monitor and enforce policies for network traffic to secure cloud applications. This approach followed and leveraged SDN network technology such that it provides the ability to

scan through network packets at every forwarding element and control the forwarding as per the application requirements. Moreover, it boosts the decision separation about where to place security monitors from the network flows themselves. FlowTap establishes monitoring relationships between microservices and security monitors, allowing them to enforce policies over the network traffic seen by the microservices. Some of the enforcing policies could be *Connection Policy* which decides whether or not a microservice can have a direct connection to another microservice, *Request-specific Policy* that defines what kind of request a microservice can make to another microservice, and *Request Integrity Policy* that analyzes the body of requests to ensure that the same user is referred to throughout the processing of the user request by multiple microservices. FlowTap caused about 6% throughput drop for the web server, which makes it a promising practical approach, however deploying it in VMs requires architectural support from cloud infrastructure in order to deliver relevant network events to corresponding security VMs.

Addressing the different modes for operating the security approach, Torkura et al. [24] summarized the following modes for the security gateway:

- *Strict Mode*: The secure mode implements a strict policy enforcement strategy. Microservice instances are not registered with the service discovery server if they do not satisfy the security policy being implemented.
- *Permissive Mode*: In the permissive mode, the policy could specify a set of rules which could be combined to determine the action to be taken.

There are currently no policy languages for RESTful web service security. However, these policies should aid in keeping production environments healthy by defining risk levels and appropriate actions when policies are breached. The authors in [24] continued to propose and introduce the following policies:

- *Global Policy*: They are applied to all the microservices in an application.
- *Microservice-Specific Policy*: These policies are defined based on the implementation details of each microservice.
- *VM and Container Policy*: This policy acts as a security control for automating security testing of images and containers to detect vulnerabilities. Given that over 30 percent of official images in the docker hub contain high-priority vulnerabilities [18].

On the other hand, the security systems come in two different styles Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) [25]. IDS is a monitoring system that detects suspicious activities and generates alerts when they are detected. While IPS could use the IDS rules and policies, it is a network security program that continuously monitors a network for malicious activity and takes action to prevent it.

Finally, DevSecOps [2] is a methodology that tackles security issues in the DevOps process. It is an approach to culture, automation, and platform design that integrates security as a shared responsibility throughout the entire IT lifecycle. Therefore, the security policies are enforced in the development pipeline as an optimization effort to reduce the duration of pre-registration tests and strategize security testing for different versions of a microservice. Besides this, DevSecOps involves securing the system and its deployment model from the infrastructure and containers' perspectives. The authors in [23] promote the principle of least privilege to address the utility-microservices container challenge. They proposed an approach to diffuse certain boundaries between the containers of utility-microservices and core-microservices. They grant the utility-microservice the capabilities and permission it needs to perform its functions while protecting the core-microservice by limiting its visibility and accessibility. This approach challenges the balancing between affording accessibility and constraints, such that it concerns with preventing the utility-microservices container from *impacting the core container execution*, *communicating with the outside world*, and *leaking information to host-local accomplices (e.g. other malicious containers)*. Therefore, achieving this approach requires various aspects to be controlled such

as *namespace isolation* that starts with full isolation between these containers and then shares a subset of the namespaces in the next steps, *de-privileging* this container to make it an unprivileged entity by mapping its user ID inside its container to a non-root, *resource isolation* that prevents it from indirectly impacting the core-container and the host, *access to disk and memory state* that restricts it from accessing the core-container disk-level system state and the system process in the memory, *access to network state* that controls the core-container network connections, and *access to resource stats*. They accommodated Kubernetes [6] coupled with Docker [3] to demonstrate that this approach achieves the fusion without compromising the security of the target container. This approach suggests its future work to support automatic examination to figure out the needed capabilities and privileges.

5. Conclusion

Security challenges have been raised with the evolution of the microservices architectures. This architecture highlights the characteristics of distributed communication, ephemeral resources, and trust among inter-communicating microservices, besides the different approaches for deployment introduced specific Vulnerabilities. These multidimensional security challenges encourage designing different approaches to be present at multiple layers of an application and at all stages of its development. This paper focused on the application layer approaches that support securing the microservices based applications. Therefore, many approaches addressed the API gateway and service discovery components as Security Enforcement Point to enforce security policies at runtime. Furthermore, many threat-modeling methods have been developed, such that they can be combined together and use standard protocols such as OAuth 2.0 to create a more robust and well-rounded view of potential threats, such as STRIDE Threat Modeling is the one that is mentioned as the most suitable for microservices architecture nature. Security standards are a staple element of industries and organizations, which pushes future work.

Employing static and dynamic analysis techniques with these kinds of approaches is a proper future work to enhance the security approaches. Moreover, there is not enough research related to a better understanding of the relationships between vulnerabilities in intercommunicating microservices. Finally, it is clear that microservice architecture still needs to mature at different levels, such as to address the lack or unavailability of specialized elements (i.e., firewalls) and the data provenance challenges.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and a grant from Red Hat Research `https://research.redhat.com`.

References

1. Api gateway in microservices architecture (Jun 2022), `https://marutitech.com/api-gateway-in-microservices-architecture/`
2. Devsecops manifesto (Jun 2022), `https://www.devsecops.org`
3. Docker - build, ship, and run any app, anywhere (Jun 2022), `https://www.docker.com/`
4. Event-b and the rodin platform (Jun 2022), `http://www.event-b.org/index.html`
5. Introduction to json web tokens (Jun 2022), `https://jwt.io/introduction`
6. Kubernetes: Production-grade container orchestration (Jun 2022), `http://kubernetes.io`
7. Oauth vs open id (Jun 2022), `https://www.okta.com/identity-101/whats-the-difference-between-oauth-openid-connect-and-saml/`

8. Service discovery (Jun 2022), https://avinetworks.com/glossary/service-discovery/
9. Stride threat modeling (Jun 2022), https://www.softwaresecured.com/stride-threat-modeling/
10. Threat modeling: 12 available methods (Jun 2022), https://insights.sei.cmu.edu/blog/threat-modeling-12-available-methods/
11. Al-Masri, E., Mahmoud, Q.H.: Qos-based discovery and ranking of web services. In: 2007 16th international conference on computer communications and networks. pp. 529–534. IEEE (2007)
12. Andersen, M.P., Kolb, J., Chen, K., Fierro, G., Culler, D.E., Katz, R.: Democratizing authority in the built environment. ACM Transactions on Sensor Networks (TOSN) **14**(3-4), 1–26 (2018)
13. Berardi, D., Giallorenzo, S., Mauro, J., Melis, A., Montesi, F., Prandini, M.: Microservice security: a systematic literature review. PeerJ Computer Science **7**, e779 (2022)
14. Blakeley, B., Cooney, C., Dehghantanha, A., Aspin, R.: Cloud storage forensic: hubic as a case-study. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 536–541. IEEE (2015)
15. Bushong, V., Abdelfattah, A.S., Maruf, A.A., Das, D., Lehman, A., Jaroszewski, E., Coffey, M., Cerny, T., Frajtak, K., Tisnovsky, P., Bures, M.: On microservice analysis and architecture evolution: A systematic mapping study. Applied Sciences **11**(17) (2021). https://doi.org/10.3390/app11177856, https://www.mdpi.com/2076-3417/11/17/7856
16. Carnell, J., Sánchez, I.H.: Spring microservices in action. Simon and Schuster (2021)
17. Gorige, D., Al-Masri, E., Kanzhelev, S., Fattah, H.: Privacy-risk detection in microservices composition using distributed tracing. In: 2020 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE). pp. 250–253. IEEE (2020)
18. Gummaraju, J., Desikan, T., Turner, Y.: Over 30% of official images in docker hub contain high priority security vulnerabilities. Technical Report (2015)
19. Gupta, R.K., Venkatachalapathy, M., Jeberla, F.K.: Challenges in adopting continuous delivery and devops in a globally distributed product team: a case study of a healthcare organization. In: 2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE). pp. 30–34. IEEE (2019)
20. Leite, L., Rocha, C., Kon, F., Milojicic, D., Meirelles, P.: A survey of devops concepts and challenges. ACM Computing Surveys (CSUR) **52**(6), 1–35 (2019)
21. Lwakatare, L.E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., Lassenius, C.: Devops in practice: A multiple case study of five companies. Information and Software Technology **114**, 217–230 (2019)
22. Nehme, A., Jesus, V., Mahbub, K., Abdallah, A.: Securing microservices. IT Professional **21**(1), 42–49 (2019)
23. Suneja, S., Kanso, A., Isci, C.: Can container fusion be securely achieved? In: Proceedings of the 5th International Workshop on Container Technologies and Container Clouds. pp. 31–36 (2019)
24. Torkura, K.A., Sukmana, M.I., Meinel, C.: Integrating continuous security assessments in microservices and cloud native applications. In: Proceedings of the10th International Conference on Utility and Cloud Computing. pp. 171–180 (2017)
25. Vistbakka, I., Troubitsyna, E.: Analysing privacy-preserving constraints in microservices architecture. In: 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC). pp. 1089–1090. IEEE (2020)