Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 239 (2024) 2260–2269

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS – International Conference on ENTERprise Information Systems / ProjMAN – International Conference on Project MANagement / HCist – International Conference on Health and Social Care Information Systems and Technologies 2023

Enhancing Effectiveness and Security in Microservices Architecture

Michael Matias[a], Ernesto Ferreira[b,f], Nuno Mateus-Coelho[c], Luís Ferreira[d,e,*]

[a]*Mercedes-Benz.io, 1990-096 Lisboa, Portugal*
[b]*Checkmarx, 4710-011 Braga, Portugal*
[c]*Lusofona University, COPELABS, Porto, Portugal*
[d]*2Ai—Applied Artificial Intelligence Lab, School of Technology, IPCA, 4750-810 Barcelos, Portugal*
[e]*LASI – Associate Laboratory of Intelligent Systems, Guimarães, Portugal*
[f]*LAPI2S –Laboratory of Privacy and Information Systems Security, Porto, Portugal*

Abstract

Microservices have allowed the decomposition of large monoliths and introduced the simple responsibility principle into its essence. It allowed complex systems to be more cost efficient and easily scalable. With this new architecture, new vulnerabilities were introduced, such as the large surface that started to get exposed to the internet. Instead of having one monolithic surface exposed, we started to have several small services exposed. New patterns were introduced to try to diminish these threats and among them the “API Gateway" pattern was introduced. The purpose of this article is to further explore the API Gateway pattern, and to further enhance the microservices communication security as part of that pattern. This research proposes a hybrid approach, using HTTPS and JWT only when strictly necessary, towards overcoming those recognized limitations of security and performance. Furthermore, proof of concept on top of NB-IoT communication protocol was developed. It intends to explore the transmission of a large quantity of low-frequency data from devices to the cloud, while the communication maintains effective and with low latency. The created supporting microservices API followed the proposed architecture and was tested against security attacks towards mitigating its impacts.

Keywords: Microservices, Security, API Gateway, Microservices Architecture, Effectiveness, NB-IoT.

* Corresponding author.
E-mail address: lufer@ipca.pt

10.1016/j.procs.2024.06.417

1. Introduction

We can see the internet as a cluster of systems and devices that share and deliver resources to other devices. Users can request resources on demand through sessions and these resources are provided by services that can also be shared between other services. The Services Oriented Architecture (SOA) represented a shift in the traditional client-server-based systems architectures, exactly to be applied in such cluster of devices and systems. With SOA, clients and servers are seen as merely services, having their composition and interoperability as the innovative behavioral contributions [1], [2] . The microservice architectural pattern instead, represents a step forward on SOA, towards higher flexibility, maintainability, and scalability on software components. Developing these kinds of services, that are significantly smaller and independent, allow bigger systems to be decomposed into smaller systems [3]. Nevertheless, this multiplicity of processes can represent more opportunities for an attack to breach the security of each of those systems [4]. We are talking about web applications and the increase in security vulnerabilities on those type of applications remains one of the most crucial problems of any information technology organization [5]. Furthermore, building large or complex microservice-based applications with multiple client apps, represent an increased attack surface area, since being autonomous, each microservice must care with its own security [6]. The pattern API Gateway [7], [8]represents a way to deal with this, where a particular service provides a single-entry port for all involved services.
The focus of the proposed solution intends to improve security effectiveness over the communication layer. This is a layer where the famous *man in the middle* attack (MITM) [9] usually happens [10], [8].
A different approach that has also been observed is using *JSON Web Tokens* (JWT), where each microservice uses the information contained in the web token for authorization purposes. Eventually this token would get passed between microservices and can create a flow problem known as “confused deputy problem” [11].
The current document describes a proposal to address efficiency concerns of microservices that use HTTPS and JWT to secure messages, beyond the API gateway pattern. The goal is to gain both performance and time of implementing such patterns.

2. Microservices Security

Microservices represents a new trend of architecture for the common Software Oriented Architecture (SOA). According to the specifications, this new architecture allows services to be more autonomous, scalable, and deployable in containers that can be managed by a cloud platform.

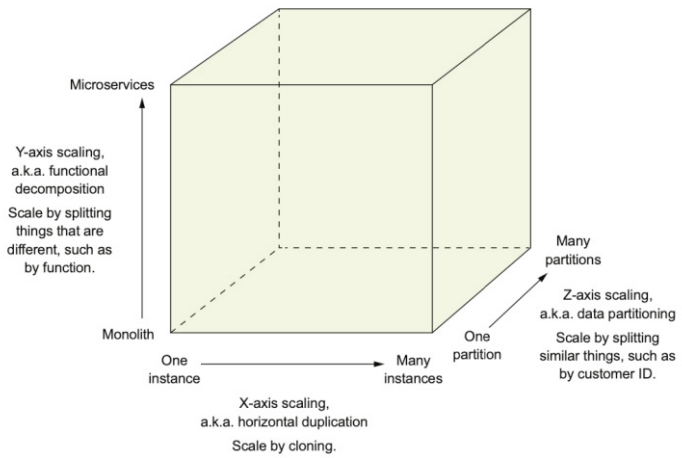

Fig. 1. Microservices scalability [6].

The scalability is inherent to the required change from the previous monolithic solutions to the new ones that are distributed and technically agnostically hosted (Fig. 1): many instances, many technologies, available somewhere. This new context exposes these services to several emergent security vulnerabilities [6]. Taking over one microservice could lead to, for instance, a denial of service (DoS) on other microservices, that can imply flooding a service or computer with a huge number of requests so that it cannot operate correctly [12]. Thus, while adopting a microservices architecture to accelerate development and simplify maintenance, each microservice demands new security attention.

2.1. Attack Surface

When it comes to security, there is a main difference between SOA and Microservices architectures. A traditional SOA usually has very few expositions to the internet. Instead, a microservice architecture approach, while delivering scalable and easily maintainable applications, can offer a higher number of services exposed. Because of that, the security risk of compromising one microservice which could compromise the rest is much higher. This is what is defined as a higher attack surface [2], [4]. Each microservice must be a simple part of code that executes a restricted set of functionalities. Indeed, each service represents a smaller attack surface. However, the large set of services represents a large area that may be attacked. Nevertheless, a smaller attack surface can make simpler to develop security patches.

2.2. Implementation Patterns

The API Gateway Pattern is a recommended pattern for microservices that can significantly reduce the attack surface [13]. This pattern consists in having a single-entry point into the system. This entry usually routes the different requests to their respective microservice [1]. Furthermore, not using this pattern in microservices is considered to be an anti-pattern, since it increases the system complexity that complicates their maintenance [14].Figure 2 shows the architecture of the API Gateway pattern. All requests are made to a single point (API Gateway) that gets delegated to other microservices that are never exposed to the exterior.
Another relevant concern that the API Gateway pattern mitigates is the cross-cutting concern [15]. A concern can in software engineering can be defined as something that affects computer software. There might be different types of concerns and different concerns, but for this article we will focus on the security functional concerns [16]. For each published microservice, authorization and secure socket layer (SSL), security must be handled. With the API Gateway pattern, the authorization can be centralized and located at the API Gateway level [17].

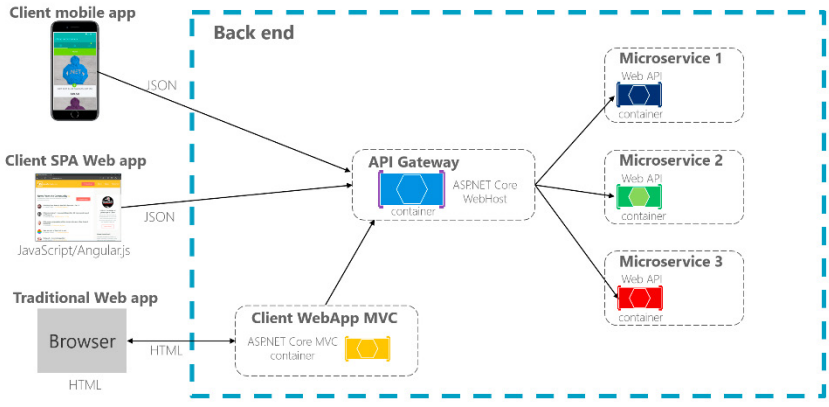

Fig. 2. API Gateway service [17].

When implementing the API Gateway service pattern, there are other patterns that should also be considered, since they might have to also be part of the solution, to have the best implementation.
In order to forward the request from the API Gateway to the microservices one of two approaches is usually used: the *Client-Side Discovery pattern (CSD)* or the *Server-Side Discovery pattern* (SSD) [18]. While the Client-Side

Discovery pattern relies on querying a service registry which knows the location of the microservices, the Server-Side Discovery pattern relies on a router that makes that query and then forwards the request. The second approach can usually be found as being part of the cloud environment, while the first one has to be programmed locally [19].
Another pattern that is often used in conjunction with the API Gateway pattern is the *Access Token pattern*. The way to communicate to the requested services the information about the user identification, authentication, and authorization, that makes the request, is usually made with a JSON token, but represents a critical challenge [20].
There are advantages and there are also some drawbacks when implementing the API Gateway pattern. One problem is related with the so called, a *single point of failure*. Indeed, if that API Gateway is not working properly, the availability of the entire services is compromised. Besides the single point of failure, if the API Gateway is not scaled properly, it can become a bottleneck. The bottleneck can be both on the development of the microservice or on the usage of the service. Another disadvantage is that the API gateway will also require additional development and future maintenance if it contains custom logic [17].
There are several approaches and pitfalls when it comes to find the best pattern for microservices. For that reason, a study has been previously made to compare several approaches and their problems [1]. [1] summarizes perceived harmfulness on the more common microservices anti-patterns. it should be emphasized that API Gateway pattern is considered a bad practice [1].

Table 1 - The microservices anti-patterns identified in a survey [1].

Microservices Anti-Pattern	Perceived Harmfulness (0-10)
Hardcoded Endpoints	8
Wrong Cuts	8
Cyclic Dependency	7
API Versioning	6.05
Shared Persistence	6.05
ESB Usage	6
Legacy Organization	6
Local Logging	6
Megaservice	6
Inappropriate Service Intimacy	5
Lack of Monitoring	5
No API-Gateway	5
Shared Libraries	4
Too Many Technologies	4
Lack of Microservice Skeleton	3.05
Microservice Greedy	3
Focus on Latest Technologies	2.05
Common Ownership	2
No DevOps Tools	2
Non-homogeneous adoption	2

2.3. Enhancing Microservices Communication Security

Several strategies, protocols and patterns exist to secure the information that “goes” from one point (service) to the other. Two of the most common technologies used for this effect are Hypertext Transfer Protocol Secure (HTTPS) and JSON Web Token (JWT) [21].

HTTPS is composed of an infrastructure of Transport Layer Security (TLS) encryption, a more secure version of SSL, and a public key infrastructure (PKI) [22]. TLS operates before the application layer and provides end to end encryption at the transport protocol layer. The public key infrastructure is composed by certification authorities (CAs) which are trusted organizations that issue and validate certificates [22]. This secure protocol prevents the Man In The Middle (MITM) attack, a commonly known security vulnerability. This kind of attack tries to take advantage of a vulnerable connection that could either have HTTPS or an unverified HTTPS certificate. When that happens, the attacker intercepts the traffic from point A to point B, accessing and modifying the message contents passing between the two points [23].

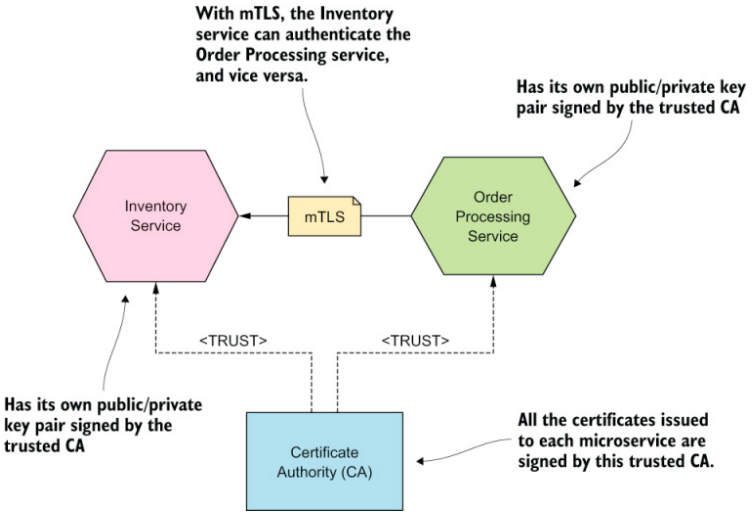

Fig. 3. mTLS between microservices [24].

TLS is also known as a one-way TLS because it helps the client to identify the server that it is communicating with. However, in a microservice paradigm, you often find out that mutual TLS (mTLS) is implemented, and this helps both the server and the client recognize each other. In the following example the server could be interpreted as the “Order Processing Service” and the client as the “Inventory Service”. Which leads the client to authenticate through the certificate authority, and vice versa, given that both have their own public and private key pair signed by the trusted Certificate Authority.

JWT is often used to develop stateless and secure RESTful web services. A stateless server needs to meet several requirements such as i) initial empty set of constraints; ii) client and server separation; iii) each communication to the server contains all necessary information to fulfill that request, i.e., no prior information is required; iv) the response to a request should be cacheable or not-cacheable; v) the information from the server should be standardized; vi) the architecture behind the system should be layered; and vii) the system should be able to send code to be executed by the client [25]. It is stated that JWT is a form of stateless authentication that encodes data in a JSON format and digitally signs it using for example a Hash-Based Message Authentication Code (HMAC) algorithm. The token is stored in the client and is checked by the server on each request to validate that the user is authorized to request a particular data [25].

Following excerpt of data represents an example of JWT.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

In this example of JWT (Base64Url encoded), the *header* “*eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9*” encodes to {"alg":"HS256","typ":"JWT"}. The *payload* “*eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ SI6IkpvaG4gRG9lIi*

wiYWRtaW4iOnRydWV9" decodes to {"sub":"1234567890","name":"John Doe","admin":true }. The verification signature is the last part, "*TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ*".

There are a few ways to share JTW between microservices. One approach is the *Security Token Service*, that receives a JWT from the API Gateway, validates it, and then issues a new JWT token to send it to the appropriate microservice. Figure 4 shows examples of this on steps 2 and 3 and steps 4 and 5.

Considering Microservices security or securing services-to-services communications among microservices, tokens (JWT) and certificates (HTTPS) are, indeed, the common basilar supporting tools.

We can conclude that several JWT's will be validated and created during a single API request. On top of that, all this communication is being done over HTTPS which is an encrypted communication. This could eventually lead to performance and latency issues which we will discuss in the following section.

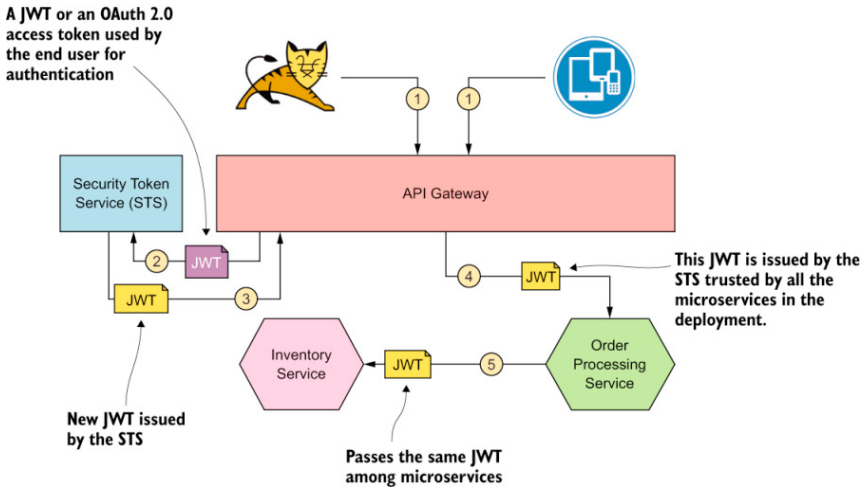

Fig. 4. JWT sharing between microservices [24].

2.4. Performance and Latency

Despite the decreasing cost implication of the infrastructure associated with the use of HTTPS over the years, it can still add direct and considerable performance costs that directly affect latency. Since the connection is encrypted, it is not possible to cache upstream data. That leads to 2TB of "uncached" data and around 30% extra energy consumption. These values will of course be lesser in the microservice context, however JWT also proves to have an impact on performance, especially when the size of it increases beyond 20KB [26].

Considering IoT contexts, low latency communications will surely compromise systems efficiency, namely those that deal with real-time analysis and decisions. Furthermore, securing low latency communications was proved to be difficult, although there exist strategies to effectively deal with it [27].

2.5. Secure communication

Service-to-service communication is an inherent requirement of a microservices architecture and is susceptible to attacks such as espionage and man-in-the-middle attacks [28]. HTTPS is preferred over HTTP basic authentication because it not only encrypts user credentials but also verifies that the client is communicating with the intended server. However, the use of HTTPS complicates the management of SSL certificates, especially in multi-machine environments. This includes managing hazardous issuing processes and revoking certain certificates, including self-signed ones [6].

Alternatives to this method include single-sign-on implementations such as SAML and OpenID, which are not only effective for authenticating and authorizing users, but also for service-to-service authentication. The use of client

certificates over TLS to confirm identities is an additional option, despite the operational difficulties associated with their administration [6].

There are also third-party options, such as Hash-based Message Authentication Code (HMAC) over HTTP. HMAC consists of hashing queries with a private key and sending the resulting hash along with the request. If the server's private key successfully recreates the hash, the request is granted [6].

API keys are an additional method for microservice-to-microservice communication. Some systems utilize a shared API key, whereas others utilize a pair of public and private keys. API keys are frequently favored due to their simplicity, robustness, and convenience of use when combined with potent methods. In addition, they are more straightforward than a Security Assertion Markup Language (SAML) communication. [6].

3. More effective and secure communication

The main problems that continue relevant to overcome are:

- *Latency*: Utilizing HTTPS and JWT for secure communication between microservices may result in performance and latency issues. This is especially important in IoT contexts, where low-latency communications are essential for real-time analysis and decision-making. The hybrid approach proposed seeks to circumvent these limitations by employing HTTPS and JWT only when strictly required.
- *Security*: The transition from monolithic to microservice architectures has increased the attack surface and introduced new vulnerabilities. While patterns such as the API Gateway can help mitigate these risks, they also introduce new difficulties, such as the possibility of a single point of failure and the need for additional development and maintenance if the gateway contains custom logic. The proposed architecture seeks to increase security by verifying and decrypting encrypted JWT at the API Gateway prior to allowing the request to proceed.
- *Efficiency*: The proposed architecture seeks to increase the efficiency of microservices communication by reducing development complexity and boosting performance. However, the effectiveness of this strategy will depend on a number of variables, such as the classification level of the data, who has access to the system network, and the security measures implemented to ensure a secure environment exists within the system network after the API Gateway.

During this research and considered these handicaps, a proof of concept that involved NB-IOT was explored. A huge amount of low frequency data being sent into the cloud, while keeping low latency, efficient communication, as well as the exposure of the specific API to the attack surface, continuously monitored.

Fig. 5 represents the proposed architecture. It tries to mitigate the security risks as well as improving performance to the microservice ecosystem. It is a mix between the use of HTTPS and JWT, instead of using any of it alone.

This architecture suggests to not use either HTTPS, or encrypted JWT beyond the API gateway. The following points justify and explain what is intended with that:

1. We can assume that if we are correctly implementing the API Gateway pattern (see Figure 2), the private network beyond that point is secure. MITM attack should not be a problem beyond that point, and the communication should be significantly faster given that it is not encrypted.
2. We must assure that the encrypted JWT is verified and decrypted at the API Gateway before allowing the request to go forward. This would also improve performance because we are not creating new tokens and not validating existing ones. A few possible solutions to keep the context present across microservices are:
    a. Have a microservice append the unwrapped JWT to the request and forward that request to the respective microservice after verifying that the token has access to the requested service. This would work as *Security Token Service*, as seen in Figure 2, with the exception that we would not be generating new JWT's.
    b. The second approach is simpler and is to append the token to the request and forward the request to its destination.

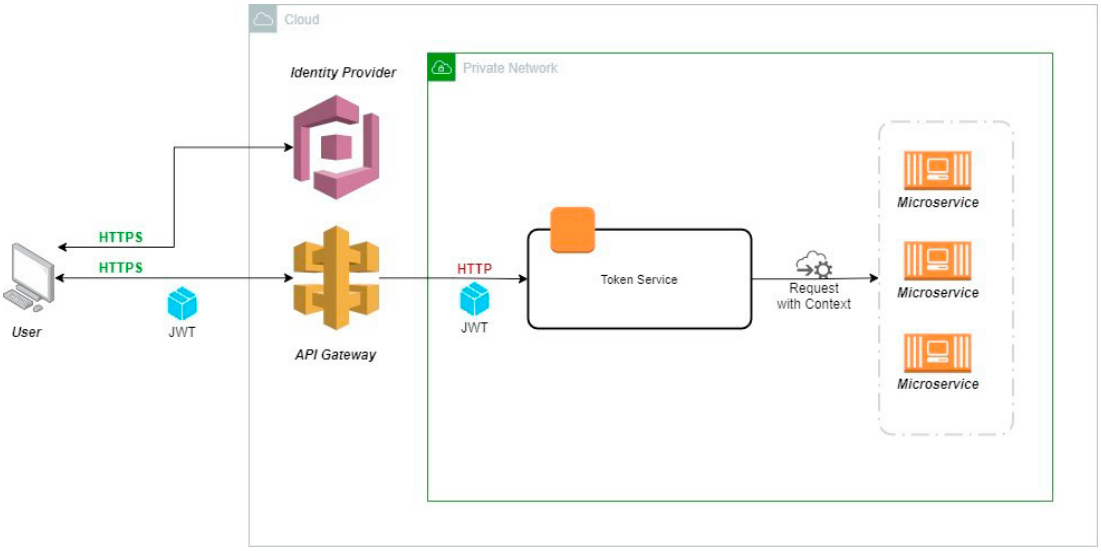

Fig. 5. Proposal of Hybrid approach.

3. At the API gateway it is possible to choose to use HMAC as an additional layer of security to guarantee the integrity of data. The HMAC process includes the following steps:
    a. Message Hashing: Your API Gateway can generate an HMAC for the JWT after receiving the request. This involves applying a hash function to the JWT and a secret key known only to the API Gateway and the JWT service.
    b. The API Gateway then transmits the HMAC and JWT to the JWT service. Importantly, HTTPS is no longer being used for this transmission.
    c. The JWT service receives the JWT and the HMAC for verification. It then generates its own HMAC of the received JWT using its copy of the secret key.
    d. The JWT service compared the HMAC it generated with the HMAC it received from the API Gateway. If they are identical, transmission of the JWT was not tampered with. If they do not match, the JWT service can reject the request because it indicates the JWT has been tampered with.

The proposed architecture will i) significantly downsize the complexity of development of microservices given that you will not need to manage any certificates for each internal microservice; ii) increase its performance and as a result since no more encryption and decryption are in place through HTTPS; iii) decrease the computation value because of not encrypting all internal network communication.

4. Conclusions and Future Work

In this investigation, we delved into the complexities of secure communication within a microservices architecture, focusing on the challenges of latency, security, and efficiency when utilizing HTTPS and JWT. Based on our investigation, we have proposed a hybrid strategy that selectively employs HTTPS and JWT. This strategy is predicated on the premise that a properly implemented API Gateway pattern guarantees a secure private network beyond the gateway, thereby reducing the need for encrypted communication and token validation within this secure zone.

Our findings indicate that the proposed architecture could substantially simplify the development of microservices, improve performance, and reduce computational costs. However, the effectiveness of this strategy depends on several variables. These include the classification level of the data, access permissions to the system network, and the implementation of measures to maintain a secure environment beyond the API Gateway within the system network.

It should also be noted that in most cases, if a MITM attack were to take place, there would be much higher concerns than that attack, given that malicious actors are already in our internal network.

Even though our proposed architecture provides a plausible solution to the challenges of secure communication in microservices, additional work is required to validate its efficacy. Future research could include benchmarking this solution and supplying empirical evidence to support the advantages of the proposed architecture.

Acknowledgements

This paper was funded by national funds (PIDDAC), through the FCT – Fundação para a Ciência e a Tecnologia and FCT/MCTES under the scope of the projects UIDB/05549/2020 and UIDP/05549/2020

References

[1] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," *CLOSER 2018 - Proc. 8th Int. Conf. Cloud Comput. Serv. Sci.*, vol. 2018-Janua, no. Closer 2018, pp. 221–232, 2018, doi: 10.5220/0006798302210232.

[2] D. Jorge, N. Mateus-coelho, and H. S. Mamede, "ScienceDirect Methodology for Predictive Cyber Security Risk Assessment (PCSRA)," *Procedia Comput. Sci.*, vol. 219, no. 2022, pp. 1555–1563, 2023, doi: 10.1016/j.procs.2023.01.447.

[3] N. Dragoni *et al.*, "Microservices : Yesterday , Today , and Tomorrow," pp. 195–216, 2017.

[4] C. A. Chen, "With great abstraction comes great responsibility: Sealing the microservices attack surface," *Proc. - 2019 IEEE Secur. Dev. SecDev 2019*, p. 144, 2019, doi: 10.1109/SecDev.2019.00027.

[5] F. Ö. Sönmez, "Security qualitative metrics for open web application security project compliance," *Procedia Comput. Sci.*, vol. 151, no. 2018, pp. 998–1003, 2019, doi: 10.1016/j.procs.2019.04.140.

[6] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira, "Security in microservices architectures," *Procedia Comput. Sci.*, vol. 181, no. 2019, pp. 1225–1236, 2021, doi: 10.1016/j.procs.2021.01.320.

[7] M. G. de Almeida and E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," *Appl. Sci.*, vol. 12, no. 6, 2022, doi: 10.3390/app12063023.

[8] F. Alves, N. Mateus-coelho, and M. Cruz-cunha, "ScienceDirect ScienceDirect ChevroCrypto – Blockchain Cryptographic File System Prototype," *Procedia Comput. Sci.*, vol. 219, no. 2022, pp. 1546–1554, 2023, doi: 10.1016/j.procs.2023.01.446.

[9] A. Mallik, A. Ahsan, M. M. Z. Shahadat, and J. C. Tsou, "Man-in-the-middle-attack: Understanding in simple words," *Int. J. Data Netw. Sci.*, vol. 3, no. 2, pp. 77–92, 2019, doi: 10.5267/j.ijdns.2019.1.001.

[10] T. Yarygina and A. H. Bagge, "Overcoming Security Challenges in Microservice Architectures," *Proc. - 12th IEEE Int. Symp. Serv. Syst. Eng. SOSE 2018 9th Int. Work. Jt. Cloud Comput. JCC 2018*, pp. 11–20, 2018, doi: 10.1109/SOSE.2018.00011.

[11] Y. Wang, Y. Hu, X. Xiao, and D. Gu, "iService: Detecting and Evaluating the Impact of Confused Deputy Problem in AppleOS," *ACM Int. Conf. Proceeding Ser.*, pp. 964–977, Dec. 2022, doi: 10.1145/3564625.3568001.

[12] U. S. Ci, "5106 Network," vol. 1, 2019.

[13] K. A. Torkura, M. I. H. Sukmana, A. V. D. M. Kayem, F. Cheng, and C. Meinel, "A cyber risk based moving target defense mechanism for microservice architectures," *Proc. - 16th IEEE Int. Symp. Parallel Distrib. Process. with Appl. 17th IEEE Int. Conf. Ubiquitous Comput. Commun. 8th IEEE Int. Conf. Big Data Cloud Comput. 11t*, no. September, pp. 932–939, 2019, doi: 10.1109/BDCloud.2018.00137.

[14] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices Anti-Patterns : A Taxonomy," vol. 1073, no. 2019, 2019.

[15] S. Cusimano, "Microservices and Cross-Cutting," 2022. [Online]. Available: https://www.baeldung.com/cs/microservices-cross-cutting-concerns.

[16] P. Beukema, "10 Frequently Occurring Cross-Cutting Concerns," 2021. [Online]. Available: https://peterbeukema.medium.com/top-10-cross-cutting-concerns-4cf30f7ab7fa.

[17] Microsoft Docs, "The API gateway pattern versus the Direct client-to-microservice communication," *https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern#what-is-the-api-gateway-pattern*, 2022. .

[18] C. Richardson, "Pattern: Server-side service discovery," *https://microservices.io/patterns/server-side-discovery.html*, 2019. .

[19] C. Richardson, "Pattern: Client-side service discovery," *https://microservices.io/patterns/client-side-discovery.html*, 2019. .

[20] C. Richardson, "Pattern: Access token," *https://microservices.io/patterns/security/access-token.html*, 2019. .

[21] S. Gadge, P. Architect, V. Kotwani, and S. Engineer, "Microservice Architecture : API Gateway Considerations," p. 13, 2017.

[22] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," *Proc. ACM SIGCOMM Internet Meas. Conf. IMC*, pp. 291–303, 2013, doi: 10.1145/2504730.2504755.

[23] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the HTTPS protocol," *IEEE Secur. Priv.*, vol. 7, no. 1, pp. 78–81, 2009, doi: 10.1109/MSP.2009.12.

[24] Prabath Siriwardena and Nuwan Dias, "Microservices Security in Action," *https://livebook.manning.com/book/microservices-security-in-action/microservices-security-in-action*, 2020. [Online]. Available: https://livebook.manning.com/book/microservices-security-in-action/microservices-security-in-action.

[25] S. I. Adam, J. H. Moedjahedy, and J. Maramis, "RESTful Web Service Implementation on Unklab Information System Using JSON Web Token (JWT)," *2020 2nd Int. Conf. Cybern. Intell. Syst. ICORIS 2020*, 2020, doi: 10.1109/ICORIS50180.2020.9320801.

[26] V. Viso, "Comparison of JWT and OAuth 2 . 0 authorisation and authentication techniques in REST services Primerjava tehnik JWT in oAuth 2 . 0 za avtorizacijo in avtentikacijo pri uporabi storitev REST," 2018.

[27] J. Hiller, M. Henze, M. Serror, E. Wagner, J. N. Richter, and K. Wehrle, "Secure Low Latency Communication for Constrained Industrial IoT Scenarios," *Proc. - Conf. Local Comput. Networks, LCN*, vol. 2018-Octob, pp. 614–622, 2019, doi: 10.1109/LCN.2018.8638027.

[28] N. Mateus-Coelho and M. Saraiva, "ScienceDirect CyberSoc CyberSoc Framework Framework a a Systematic Systematic Review Review of of the the State-of-Art," *Procedia Comput. Sci.*, vol. 204, pp. 961–972, 2022, doi: 10.1016/j.procs.2022.08.117.