Access control approach in microservices architecture*

Ernestas Serkovas[1]

[1] *Kaunas University of Technology, K. Donelaičio St. 73, 44249 Kaunas, Lithuania*

Abstract

Modern and contemporary systems, the Internet of Things, are often large-scale and have a complex structure. To increase the reliability and availability of systems, together with the development of containerization technologies, the use of microservices architecture began. The autonomy and isolation of the services of this architecture made it possible to optimize the management and development of the fragmented system. As the number of systems based on microservices architecture grows, so do challenges related to the security of services and devices. Access control is one of the main security issues faced when designing and developing a system based on microservices architecture. Services are designed to trust requests that come from communicating services. This trust can be exploited to disrupt other services or devices if access control in one of the components of the microservices architecture is compromised. As the Internet of Things technology evolves, it increasingly uses small, constrained devices that also require adequate security and access control. For this reason, the goal of the paper is to propose a method that solves the current problems of access management in microservices architecture. After analyzing the microservices architecture and access management problems related to it, an access management method was proposed and designed, which solves the access management problem in an environment of limited resources. According to the requirements set out in the paper, a prototype was created and tested, and a study of the resource use and speed of the proposed access control method in the infrastructure was carried out.

Keywords

access control, microservices, microservices architecture, internet of things

1. Introduction

Modern and contemporary systems, the Internet of Things, are often large-scale and have a complex structure. To increase the reliability and availability of systems, together with the development of containerization technologies, the use of microservices architecture began. The autonomy and isolation of the services of this architecture made it possible to optimize the management and development of the fragmented system. As the number of systems based on microservices architecture grows, so do challenges related to the security of services and devices. Access control is one of the main security issues faced when designing and developing a system based on microservices architecture. Services are designed to trust requests that come from communicating services. This trust can be exploited to disrupt other services or devices if access control in one of the components of the microservices architecture is compromised. As the Internet of Things technology evolves, it increasingly uses small, constrained devices that also require adequate security and access control. For this reason, the goal of the paper is to propose a method that solves the current problems of access management in microservices architecture. After analyzing the microservices architecture and access management problems related to it, an access management method was proposed and designed, which solves the access management problem in an environment of limited resources. According to the requirements set out in the paper, a prototype was created and tested, and a study of the resource use and speed of the proposed access control method in the infrastructure was carried out.

*IVUS2025: Information Society and University Studies 2024, May 15, Kaunas, Lithuania

ernestas.serkovas@ktu.edu (E. Serkovas)

2. State-of-the-art study of access control in microservices architecture

2.1. Microservices architecture in the Internet of Things

The Internet of Things is an increasingly common technology that consists of devices connected to networks. These devices can range from common household items to mechanisms and sensors. These devices collect and transfer data to the fog layer, in which the servers perform decision-making and the received data processing. Fog computing works best with systems that require fast answers to requests, small delays, and real-time data processing. With the improvement of the Internet of Things technology, end devices are getting more compact, and this means that resources such as random-access memory and processors getting more limited. For communication between fog layer services, small lightweight, less complex protocols are used to shorten the delay for request processing. Even though modern end devices can support certain cryptographic operations, they take too long to process. For data collection and processing in the fog layer, devices with more resources are used which then take the processed data and transfer it to further outer layers [1].

Decreasing resources in devices poses a few challenges when trying to ensure security from a plethora of dangers. Even though standard wide-spread security methods could be used in the Internet of Things devices with a larger amount of resources, end devices require a different method which would allow them to achieve an acceptable security level with fewer resources. Due to the limited resource environment, ensuring confidentiality, integrity, and device access control becomes a pressing issue. Special, lightweight communication protocols, such as Constrained Application Protocol (CoAP), paired with different effective resource-saving security algorithms can help counter these issues [1].

For communication between the Internet of Things devices, the CoAP protocol is used [2]. It can also be used for communication between microservices. CoAP works with the User Datagram Protocol (UDP) transport protocol, so it is necessary to ensure security.

For CoAP security, a Datagram Transport Layer Security (DTLS) protocol is used. However, as showed in the table in Figure 1, a lot of extra actions introduced into protocol. For example: handshake, certificate data exchange and alerts. These actions require additional resources from a device, which is usually limited [2].

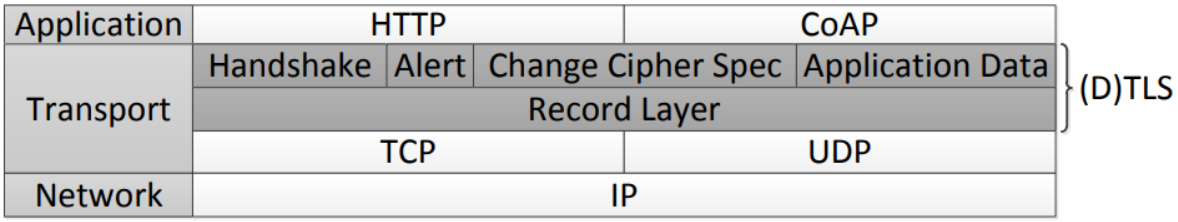

Figure 1: DTLS protocol structure [2]

Due to additional processes in the DTLS protocol and limited device resources, a simpler access control method for microservices architecture is needed which requires less resources from the end device and could be an alternative to the CoAP protocol.

2.2. Microservices architecture and access control issues

Microservices architecture is based on the distribution of functions into individual, small-scale services that operate in their separate processes and communicate with each other in lightweight protocols [3]. Microservices are autonomous and isolated from each other with the use of containerization technologies such as „Docker“ [4]. Figure 2 shows a diagram of a microservices architecture-based application showing three services serving an e-shop. Each service performs a specific function, for example – inventory administration. For communication between services

Representational State Transfer (REST) Application Programming Interface (API) is used, thanks to which services can change data regardless of the technology they are based on. Furthermore, this way of communication allows the application to be used on different devices.

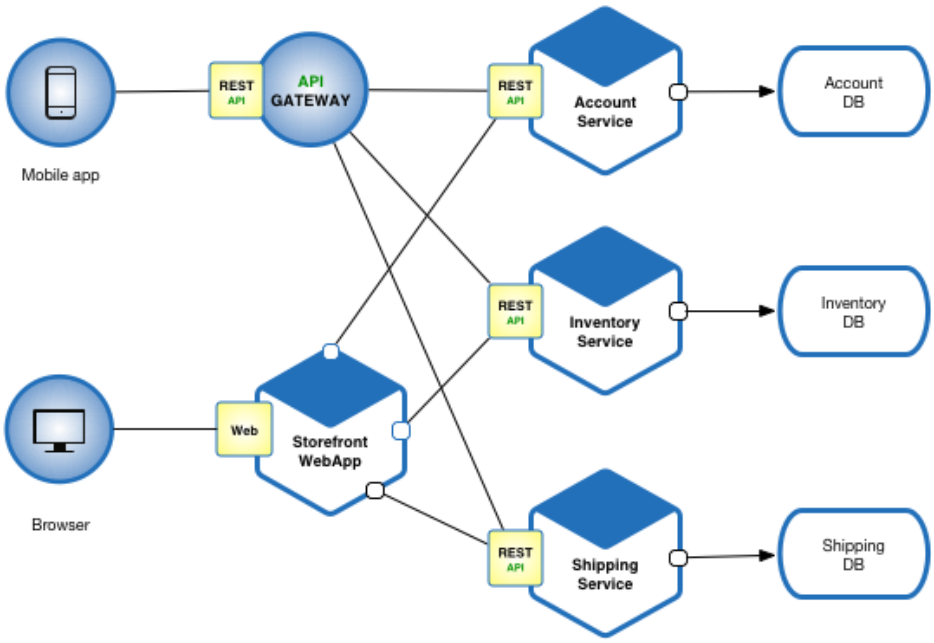

Figure 2: Application based on the microservices architecture [5]

Using microservices architecture, large-scale, complex applications are broken down into smaller parts, thanks to which it is possible to achieve greater availability and reliability. Additionally, we can achieve better optimization for their management and scalability [3]. In addition to these benefits, microservices architecture services can be independently managed, deployed, and scaled. Furthermore, due to the isolation of services from each other, each of them can use different technologies that can be adapted specifically to the service function that needs to be performed. Microservices architecture service isolation contributes to the application's reliability, because if an error occurs in a service, the problem will affect only the functions of this service, and all other parts of the application will continue to run their processes successfully [6].

Although microservices architecture offers advantages, it also brings challenges related to services protection. The main problems arise from the fact that security issues need to be addressed for each service individually [3]. One of the main security questions in microservices architecture is access control. Services share data, so it is important to ensure the security of the communication channels through which this data reaches other services. For communication between services, a Hypertext Transfer Protocol (HTTP) protocol is used. Microservices architecture services are usually designed in a way that in most cases, they are supposed to trust requests coming from intercommunicating services. If the access control mechanism of one of the services is breached, trust can be lost and be used to exploit other services. Additionally, all communication between the microservices architecture services takes place in a network in contrast to a monolithic architecture application. For this reason, it is necessary to secure each service individually, because of the increased opportunities to exploit network vulnerabilities [3].

2.3. Access control methods for microservices architecture services

The mutual Transport Layer Security (mTLS) method is based on the Transport Layer Security (TLS) protocol. It is a two-way authentication method in which the communicating services confirm their identity with each other using a combination of public and private key pairs with TLS certificates [7]. Communication between the internal services of microservices architecture takes place on a

client-server basis. For example, the client's service connects to the service from which it wants to receive data (in this case a server). The server's service then provides its certificate, which must be validated by the client's service. When the certificate is validated, the client's service transfers its certificate to the server's service, so that it can also perform its certificate validation. If both certificates are successfully validated, the server's service grants access to the requesting service and fulfills the request. The data between services is sent over a network that is encrypted, ensuring that the information being sent over is confidential and that it keeps its integrity [8].

Token-based access control relies on cryptographic tokens that can be used to securely transfer data between servers. One of the standards used in a microservices architecture is a JavaScript Object Notation (JSON) Web Token (JWT) [7]. Data is transferred in a JSON token that is signed with a digital signature. The signature could be made up of an encrypted tag or a pair of public and private keys. Information about the signature's algorithm and token type is stored in the header. The payload is where data is inserted. To create a signature, an encrypted header is used with an encrypted payload as well as an encrypted tag, and everything is signed with a signature algorithm [9]. Every single one of the microservices architecture services can contact the token issuer service with its data so that it could receive a token that is allocated for that service so that later it could be added to each request sent from the service [7]. Then the token is used to check if the contacting service could receive data from the service that it is contacting through the information stored in the token. Cryptographic tokens are simple and user-friendly [10], thus being able to be used in different situations where you would need to minimize the usage of an application's resources.

2.4. Service access control methods

A single sign-on (SSO) system can be used in microservices architecture both to provide access to users and to perform access control between individual services. However, when applying single sign-on access control between services, challenges may arise with system operating speeds [11]. Each service, wanting to check the status of an access token, must first check with an SSO server, before accepting a request. If there is a large quantity of services or queries, a considerable number of requests could be made. This can negatively impact the response time of requests both between the SSO server and services, eventually slowing down the whole system.

OAuth 2.0 protocol can also be used in a microservices architecture. However, if there is a larger number of servers, challenges may arise where each server will have to individually check request tokens, which can overload the services causing poor performance. For this reason, it is recommended to perform authorizations in the external layers of microservices architecture [11].

3. Approach of access control in microservices architecture

3.1. Microservices architecture approach

The access control method is being designed in microservices architecture, which is made up of three layers, which are – API gateway, fog layer and end devices layer. Figure 3 depicts a scheme of microservices architecture from twelve services and devices.

The user interface with API gateway is based on the REST style, which uses HTTP requests such as GET, POST, PUT. This user interface can be used in programs and internet or mobile applications. Due to this reason, proposed access control method is independent and can be used with any type of device. The API gateway works in a server, which has the data of all users that can use the services in microservices architecture and manages both request and access control. It also secures servers and devices in fog and end devices layers from outside threats because all requests are made through a gateway, and other layers are hidden from outside network access. The API gateway, fog layer servers and end layer devices will use HTTP requests to communicate between themselves according to REST style.

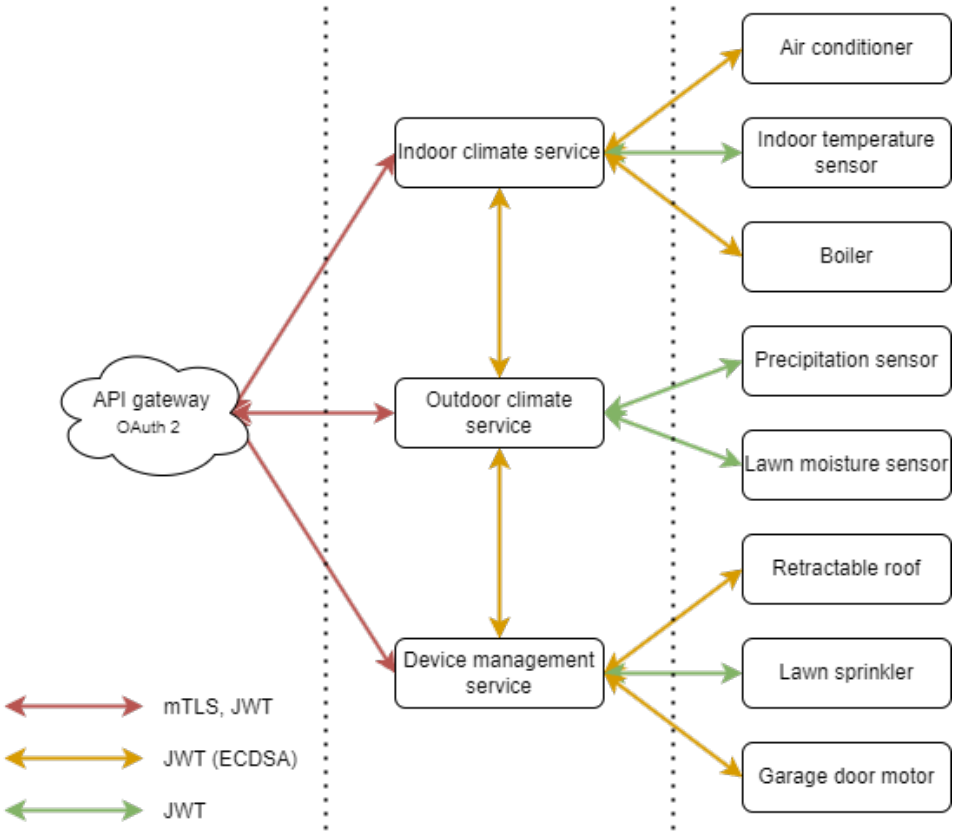

Figure 3: Microservices architecture prototype (source: created by author)

3.2. Access control method

For access control in the API gateway, an OAuth 2.0 protocol will be used. Users can authorize themselves in the API gateway by requesting an access token with OAuth 2.0 which is based on the JWT. Because the user is given a JWT access token it will be compatible to be used for further actions in microservices architecture layers. OAuth 2.0 protocol access tokens allow to submit scope settings that define access permissions. This functionality is used in order to implement role-based access control (RBAC) and so that devices could communicate amongst themselves and could determine whether to grant the requesting device access to its requested resource. As and additional advantage, third-party users can be authorized to gain access to services in microservices architecture from the platforms and services that they use.

API gateway and fog layer servers will use the mTLS method for access control, which is based on the TLS protocol. This method ensures security for API gateway communication between fog layer servers and will use a pair of private and public keys with TLS certificates so that both devices could verify their identity to each other and so that access could be granted just to those devices. Once a safe communication channel is established, further JWT tokens are used. The mTLS method is chosen because it ensures confidentiality of the data that is transferred and because the API gateway and fog layers have enough resources to support this method.

Fog layer servers will use JWT tokens amongst themselves, which are signed by the Elliptic Curve Digital Signature Algorithm (ECDSA) public and private key pairs. This decision allows to ensure the confidentiality and integrity of the data in tokens as well as access control with the key pair, since the tokens information can only be accessed by the device which holds the public key of the token sender. Signing a JWT token with a key requires additional resources, so this method is only used in devices that have a larger resource reserve and can perform key signing. JWT access token method with the ECDSA signing algorithm keys can also be used for communication between fog layer servers as well as some end layer devices which have a larger resource amount and will have the ability to use this method. Fog layer servers will choose access control method based on available

data about the end device which they will be establishing communication with and will be receiving or giving data to. This data includes end device resource restrictions and security requirements.

Most of the devices in the end layer of devices do not have many resources and capabilities to support higher security methods. Due to this reason, end devices will use simple, lightweight JWT tokens. Devices that have very low resources can use static access tokens which are generated in advance, thus not needing to use additional resources. If a device can generate JWT tokens with signing using the Hash-based Message Authentication Code (HMAC) algorithm, it can use the before mentioned method. Both static and generated JWT access tokens provide data security by ensuring their integrity and allows them to control the devices access rights.

All of the chosen methods will allow to manage access from the user to the end device while securing the data security through confidentiality and integrity. In addition, methods are selected by reflecting on device capabilities and resource limitations to adapt the proposed access control method to a wider range of devices and increase their security.

4. Prototype of the access control method in microservices architecture

The prototype uses a machine with the “Linux” “Ubuntu 20” operating system, “Nginx” web server, “PHP” programming language and “MariaDB” relational database management system. The access control method prototype is made up of two layers – internal and external, which have an API gateway, three simulated fog layer services, and eight simulated end devices implemented. In the prototype the OAuth 2.0 protocol is applied, as well as the JWT access control tokens with three signing algorithms and mTLS access control method.

For the implementation of the API gateway, the “PHP” programming language “Laravel” framework is used. At the time of programming the prototype, the newest version of the framework was chosen which was version 11.3.1. The “Laravel” framework was chosen because it allows the implementation of the OAuth 2.0 protocol for access control. In the “Laravel" framework is the “Laravel Passport” module, which grants full use of the OAuth 2.0 protocol in projects based on this framework.

Users and their roles are stored in a relational database which the API gateway can access and edit. The client wanting to access the API gateway services has to log in with their assigned credentials. When logged into the API gateway, the user roles are received, which are used for access control in the internal layers. After the client submits a request, the API gateway creates a JWT token, which paired with the request is sent to the corresponding fog layer service.

In the fog layer, there are three simulated services programmed with the “PHP” programming language. Services are deployed on containers with restricted resources. These services receive requests from the API gateway and contact the appropriate end devices to receive or transmit data. Each service performs role-based access control according to individual configuration.

The end devices layer consists of eight simulated devices programmed in the “PHP” programming language and deployed on “Docker” containers with restricted resources. The end devices receive requests from the fog layer services, and based on the configuration, perform access control, as well as provide data to the service or execute commands. The device configuration has a list of access rights to features based on the client's role. A secret key is also stored if the device does not have a lot of resources or a pair of ECDSA keys if the amount of resources is higher. A list of fog layer services, which have access rights to the end device, unique numbers are also stored in the configuration. If the requesting service is not on the list, the end device answers with a forbidden access message. Each end device has its own unique number, supported security algorithm, and specific data, which can be submitted to the fog layer services.

5. Experimental research of the access control prototype

The implemented access control method in a microservices architecture is researched with the use of tests and software for analyzing resource usage. When researching the prototype, the aim is to

figure out if the chosen access control methods work as intended and if they are effective. In addition, it is desired to evaluate the resource usage of the proposed access control method (e.g. processor, random-access memory) and speed. Resource usage and speed will then be compared to an identical infrastructure prototype without the proposed access control method.

The research for the access control method prototype resource usage was done by sending a single request to the indoor climate service approximately every second. In total, there were ten requests sent to the service access point and the same amount of resource usage measurements were taken. The indoor climate service was chosen specifically because this service addresses two end devices, the air conditioner and the temperature sensor. Both devices use different token algorithms and require different security levels. The research was conducted with two scenarios – without the proposed access control method and with it. During the time of measurements, information was collected about the containers' processor and random-access memory usage during the moment of measurement.

Table 1
Resource usage comparison

Simulated device	Method usage	Average processor usage	Average random-access memory usage
Air conditioner	Without proposed method	0,40 %	78,87 megabytes
	With proposed method	0,88 %	80,41 megabytes
Temperature sensor	Without proposed method	0,40 %	79,94 megabytes
	With proposed method	0,38 %	78,98 megabytes

During the research, the speed of the access control method prototype in the infrastructure was also evaluated. Measurements were collected by sending requests to the API gateway to three endpoints: indoor climate, outdoor climate, and device management. In total, there were thirty requests sent, ten for each endpoint. Measurements were made with two scenarios – without the proposed method and with the proposed method.

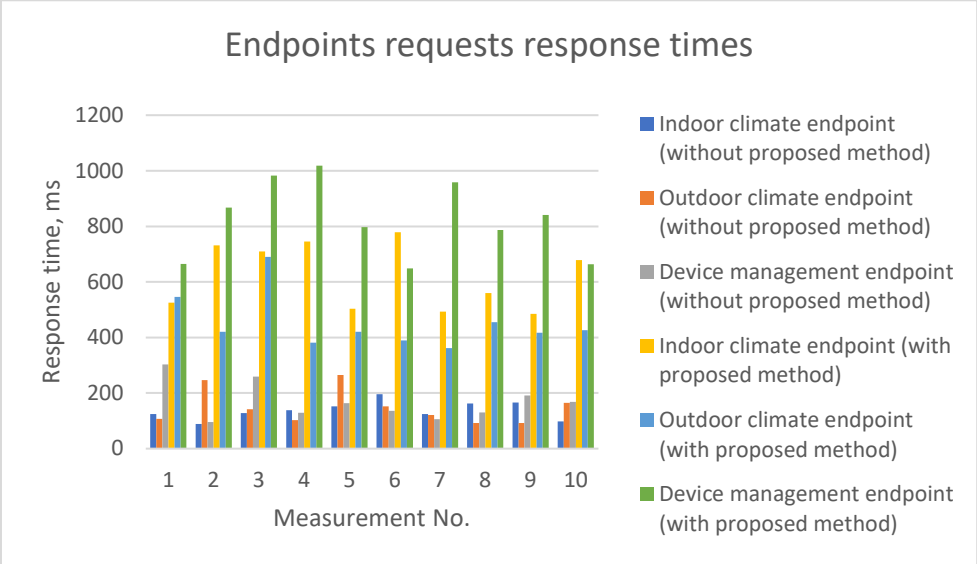

Figure 4: Endpoints response times comparison diagram (source: created by author)

After testing the prototype's components and functionalities it can be concluded that the prototype was implemented successfully and works correctly. Furthermore, the experimental research on the prototype allowed to compare the proposed method prototype with identical infrastructure without the method and evaluate the chosen methods' suitability, efficiency, resource

consumption, and speed. After analyzing the results of testing and experimental research, it can be concluded, that:

1. The prototype is implemented according to the requirements and works correctly;
2. The proposed access control method's increase in processor usage is insignificant, compared to the base measurements, and can be used in devices with limited resources;
3. The proposed method's increase in random-access memory usage is also insignificant which provides the possibility of using the method in devices with limited resources;
4. The difference in resource consumption between the method with ECDSA algorithm tokens and the method with HMAC security algorithm tokens is not high. This is why it is appropriate to use the ECDSA algorithm if the devices have the ability to perform cryptographic operations;
5. The speed of the proposed access control method is a lot slower, but the optimization of the method could possibly lower the negative impact on the requests' response time.

6. Conclusions

1. When analyzing literature sources, it was noticed that the main problems in the microservices architecture are access control in services, communication channel security between different services, the need to assess and design security for each service individually, and limited resources in end devices;
2. The analysis of existing access control methods showed, that widespread methods for various architectures and systems are used in a microservices architecture. After examining the application of methods in microservices architecture, it can be concluded that the secure implementation of access control can be achieved with already available methods;
3. When the analysis and comparison of existing access control methods in microservices architecture was done, it was noticed that in order to achieve a better security level and expand system functionality, it is appropriate to use a few different access control methods. This decision allows to combine the strengths of different methods and solves the problems that would arise when using those methods separately;
4. The proposed method in this work, using the OAuth 2.0 protocol, mTLS method, and JWT tokens would solve the problems of access control in microservices architecture-based Internet of Things devices with limited resources such as processor and random-access memory;
5. After the research on the implemented prototype of the access control method in microservices architecture, it was found that the proposed method's increase in resource usage is not high compared to the infrastructure without the method. It was also found that the proposed method reduces the speed, but it is hypothesized that the optimization of the source code and libraries used would allow to achieve higher speed. In summary, it can be concluded that the proposed method solves the problem posed in work.

Declaration on Generative AI

The author have not employed any Generative AI tools.

References

[1] A. Venčkauskas, N. Morkevičius, V. Jukavičius, R. Damaševičius, J. Toldinas, and Š. Grigaliūnas. "An Edge-Fog Secure Self-Authenticable Data Transfer Protocol." Sensors 19.16 (2019): 3612. doi:10.3390/s19163612.

[2] M. Brachmann, O. Garcia-Morchon, and M. Kirsche. "Security for Practical CoAP Applications: Issues and Solution Approaches." (2015). URL:

https://www.researchgate.net/publication/265973615_Security_for_Practical_CoAP_Applications_Issues_and_Solution_Approaches.
[3] M. G. De Almeida and E. D. Canedo. "Authentication and authorization in microservices architecture: A systematic literature review." Applied Sciences 12.6 (2022): 3023. doi:10.3390/app12063023.
[4] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo. "Microservices in Practice: A Survey Study." (2018). doi:10.48550/arXiv.1808.04836.
[5] C. Richardson. "What are microservices?" (n.d.). URL: https://microservices.io/.
[6] C. Richardson. “Microservices patterns: With examples in Java.” Simon and Schuster, 2018. URL: https://books.google.lt/books?id=QTgzEAAAQBAJ.
[7] A. Barabanov and D. Makrushin. "Authentication and authorization in Microservice-based systems: Survey of Architecture Patterns." Voprosy kiberbezopasnosti 4.38 (2020): 32–43. doi:10.21681/2311-3456-2020-04-32-43.
[8] "What is mutual TLS (mTLS)?” What is MTLS? Cloudflare (n.d.). URL: https://www.cloudflare.com/learning/access-management/what-is-mutual-tls/.
[9] "Introduction to JSON Web Tokens." JSON Web Token Introduction - jwt.io (n.d.). URL: https://jwt.io/introduction.
[10] N. Mateus-Coelho, M. Cruz-Cunha, and L. G. Ferreira. "Security in microservices architectures." Procedia Computer Science 181 (2021): 1225–1236. doi:10.1016/j.procs.2021.01.320.
[11] A. Rezaei Nasab, M. Shahin, S. A. Hoseyni Raviz, P. Liang, A. Mashmool, and V. Lenarduzzi. "An empirical study of security practices for microservices systems." Journal of Systems and Software 198 (2023): 111563. doi:10.1016/j.jss.2022.111563.