Authentication and Authorization in Microservice-Based Applications

Niklas Sänger[1], Sebastian Abeck[2]

Abstract: The development of microservice-based applications adds challenges when using different cloud services. One such challenge is the integration of authentication and authorization among different systems. In this publication, we describe the development of a software as a service solution with the focus on the integration of authentication and authorization. For the development of the business logic, the integration platform as a service MuleSoft is used. The identity and access management as a service solution Okta is used to provide the necessary means for authentication. To perform authorization decisions, JSON Web Tokens and API proxies are used.

Keywords: Authentication; Authorization; Microservices; Engineering; Okta; MuleSoft

1 Introduction

Compared to the development of a classical monolithic application, the goal of a modern cloud application using a microservice architecture is to reuse existing software components or services by integrating them through well-defined application programming interfaces (APIs) [Ne15]. This publication describes the integration of authentication and authorization in the development of a Software as a Service (SaaS) using the products MuleSoft and Okta.

As a demonstrator, the case study of the development of a connected car application, called PredictiveCarMaintenance (PCM), is used. Fig. 1 provides an overview of the groups accessing as well as the products and protocols used to build the application PCM which provides vehicle data (e.g., state, maintenance intervals) for vehicle users and garages. For the implementation of the application, the well-established products Okta and MuleSoft are used. Okta provides an IAM as a Service (IAMaaS) solution to, among others, register and authenticate users [In22b]. MuleSoft calls itself an Integration Platform as a Service (iPaaS), which can be used to develop tailored APIs by integrating existing systems [Mu22b]. In order to meet the IAM requirements, the two products must be strictly aligned with each other. The established protocols OpenID Connect (OIDC) and OAuth 2.0 are used for this purpose. To allow a user to login and display the requested data, the User Interface (UI) is developed using the TypeScript-based framework Angular.

[1] Karlsruhe Institute of Technology, Research Group Cooperation & Management, Zirkel 2, 76131 Karlsruhe, Germany, niklas.saenger@kit.edu

[2] Karlsruhe Institute of Technology, Research Group Cooperation & Management, Zirkel 2, 76131 Karlsruhe, Germany, sebastian.abeck@kit.edu

doi:10.18420/inf2022_19

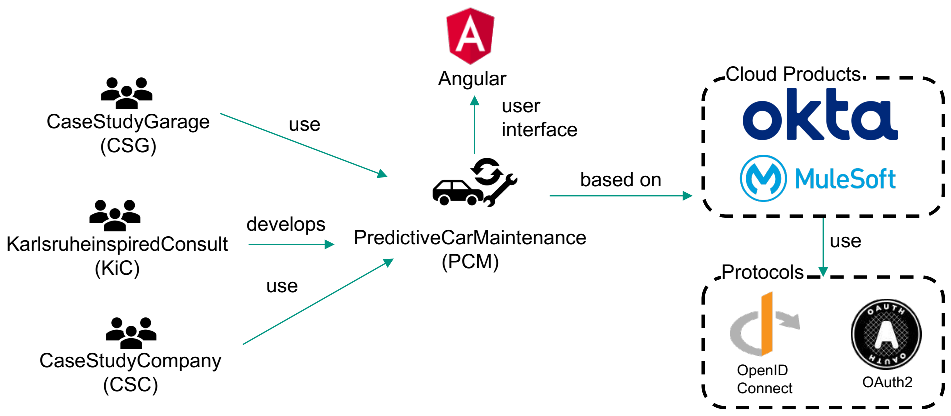

Fig. 1: Overview of the Application PredictiveCarMaintenance

Following the principle of least privilege, the users of the application PCM should only have the privileges to perform their necessary tasks [SS75]. Those tasks differ for each group and should be defined by a detailed requirements analysis. In the context of this publication, the groups, their tasks, and the requirements are described informally in the following.

PCM is developed by the company KarlsruheinspiredConsult (KiC). The CaseStudyCompany (CSC) is a customer of KiC which has a large vehicle fleet and uses the application PCM to manage their fleet. Finally, the company CaseStudyGarage (CSG) is a garage and uses the application PCM to organize vehicle repairs. Since employees from KiC are the developers of the application PCM, they must also perform administrative tasks as well as support for their customers. Therefore, KiC employees require access to all data that is available in the system. To protect these comprehensive permissions, a second factor is required for authentication of KiC employees. This factor must be verified by the IAMaaS Okta. Employees of CSC use the application PCM to see current and past data of the vehicles they are assigned to. An employee is assigned to a vehicle by the company CSC. While an employee can schedule a maintenance appointment with the CSG through the application PCM, the CSC employee cannot access the advanced diagnostics data and maintenance information. Vice versa, employees of the CSG must only access the diagnostics data and maintenance information of a vehicle they have been assigned to by a supervisor.

The present article is structured as follows: Sect. 2 presents the related work. The design of the frontend architecture and the implementation of the authentication based on the previously described requirements is described in Sect. 3. The design and the implementation of the authorization are described in Sect. 4. Finally, the results and future work are discussed in Sect. 5.

2 Related Work

In their systematic review from 2012, Iankoulova and Davena analyze security requirements for cloud computing and find that access control, as the degree to which a system limits access to its resources, is a key requirement [ID12]. This is also a problem in a microservice architecture as de Almeida and Canedo [AC22] point out in their systematic literature review on authentication and authorization in microservice architectures. They find that challenges are trust between microservices compromised by unauthorized access, individual concern for each microservice, and microservice access control. Furthermore, the authors find the most used security mechanisms are OAuth 2.0, JSON Web Tokens (JWT), and API Gateways. Other important mechanisms are Single-Sign On, OpenID Connect (OIDC), as well as Role Based Access Control (RBAC) and Attribute Based Access Control (ABAC).

The Open Web Application Security Project (OWASP) further strengthens the security issues in their report of the top 10 security risks in web applications [Fo21]. The 2021 report includes broken access control (A01:2021) as the most serious web application security risk.

He and Yang [HY17] describe the realization of authentication and authorization of an end-user in a microservice architecture. They propose an authorization solution using JWTs provided by an authentication server and an API gateway to validate the JWT. Zhao et al. [ZJJ18] also describe an authorization mechanism using an API gateway and JWT issued by an OAuth 2.0 authorization server. Xu et al. introduce the development of a microservice security agent in an edge computing environment [XJK19]. The authors propose a security agent consisting of an API gateway, a client server including a user interface, and a security agent. The client server is used to configure the API gateway. The API gateway then performs authorization decisions based on a JWT which are issued by the security agent. Similar to the described approaches, this publication performs authentication and authorization decisions based on an API gateway as well as JWTs. However, this publication sets the focus on the realization based on the integration of the cloud solutions MuleSoft and Okta.

Nehme et al. describe fine-grained authorization for a microservice architecture based on OAuth 2.0 and eXtensible Access Control Markup Language (XACML) [Ne19]. They develop a gateway solution which applies XACML policies to resource requests, thus decoupling the security aspects from functional requirements of a microservice. Furthermore, the authors describe the shortcomings of the coarse-grained nature of OAuth 2.0 which is used in this publication.

MuleSoft offers the API-led connectivity approach, which aims at connecting data to applications through reusable APIs [Mu22a]. To do so, MuleSoft uses a three-layered architecture with each layer having its own API type. First, the System layer is the lowest layer and provides System APIs integrating the data from external systems. The middle layer is the Process layer which performs business logic and provides the results through Process APIs. Finally, the top layer is the Experience layer providing the processed data in the required format for an arbitrary client through an Experience API. This layer can be compared to a Backend for Frontend (BFF), which does not perform business logic but rather supports the frontend [Ne15].

3 Authentication

This chapter describes the systematic realization of the authentication requirements for the application PCM described in Sect. 1. To restrict access to the application PCM for authenticated users, the architecture of the application PCM must take authentication into account during the design phase. Next, the implementation phase consists of two steps. First, the configuration of the IAM system Okta to implement the required authentication mechanisms (i.e., OIDC Client, multifactor authentication). Second, the implementation of the frontend using the results of the Okta configuration.

3.1 Design of the Architecture

The UML-based diagram displayed in Fig. 2 presents an excerpt of the architecture of the application PCM with a focus on the frontend. It is a combination of a UML deployment diagram and a UML component diagram [OM17]. It allows to further enrich a physical node from a deployment diagram by applying software components from a UML component diagram running on the subsystems. It also presents a more detailed view on which and how components inside physical nodes communicate with each other (i.e., protocol).

Based on the case study requirements depicted in Sect. 1, the diagram in Fig. 2 contains three nodes for the involved systems «iamaas system» Okta and «ipaas system» MuleSoft as well as «browser» PCMFrontend for the required Angular frontend. While the architecture of the frontend is generally independent from the implementation, the components are enriched by Angular-specific stereotypes «ng component» and «ng service» to differentiate views from reusable functions or tasks (e.g., authentication).

To allow a user to log into the PCMFrontend, the «ng component» Login provides a web page with fields for a username and password as well as a login button. The «ng service» Authentication, initiates the authentication by executing the OIDC flow against the OAuth «authorization server» Default provided by Okta. The «ng service» Authentication uses the «oidc client» PredictiveCarMaintenanceOpenidconnectClient (PCMOC) to authorize itself to the «authorization server» Default. After a user is successfully authenticated, they are redirected to the «ng component» LandingPage which provides references to the pages a user is allowed to access. The design of the components required for authentication and authorization is inspired by the OIDC flow and allows for separation of concerns in the frontend architecture.

Furthermore, for each user group (i.e., VehicleUser, Garage), a dedicated «ng component» is created to display the requested information. To retrieve the information, an experience

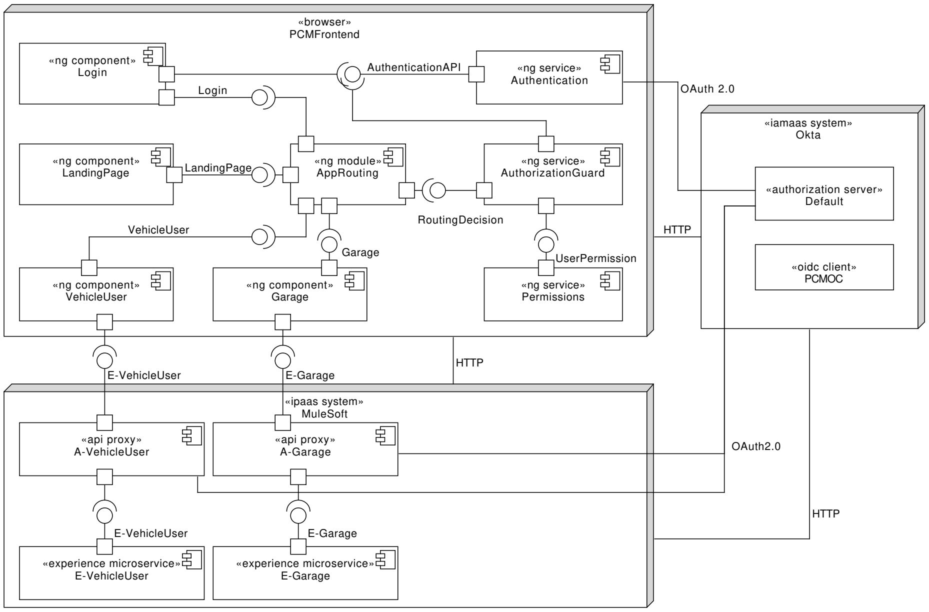

Fig. 2: Excerpt of the PCMFrontend with Focus on Authentication and Authorization

microservice is developed for each user group. This allows to only provide the data required for a user group and further restrict the access to the experience microservice with the use of an API proxy. This is also inspired by the MuleSoft API-led connectivity approach [Mu22a]. The authorization decisions in the «browser» PCMFrontend and the experience microservices are further elaborated in Sect. 4.

3.2 Configuration of the IAMaaS Okta

Before users can authenticate themselves in the PCMFrontend and access sensitive data, the IAM system Okta must be configured accordingly. Therefore, the first thing that must be decided is where and how the users will be stored inside Okta. In this example, the users are created manually inside Okta. Okta offers a universal directory which provides a single view on users and groups and provides integration of further directories (e.g., LDAP, Active Directory). To be able to distinguish between the affiliation and the function of a user (e.g., a mechanic or vehicle user), each user is assigned to a corresponding group which must first be created in Okta. For PCM, the groups CaseStudyCompany, CaseStudyGarage, and KarlsruheinspiredConsult are created for the employees of the respective companies being part of the case study.

The OIDC protocol is based on OAuth 2.0 and allows authenticating users to an identity provider. In this example, the IAM system Okta acts as the identity provider and provides an OIDC client to the PCMFrontend. Depending on the requirements, there are several options to configure and use an OIDC client (e.g., client types, grant types). Okta guides the developer through the creation of an OIDC client by asking for the application which has implications on the basic configuration. In case of PCMOC, the SPA application type is used. This forces the developer to use a public OIDC client since the client ID and secret can be extracted from the deployed frontend otherwise. Furthermore, the developer has to use Proof Key for Code Exchange (PKCE) which is also recommended by the Internet Engineering Task Force (IETF) for public OAuth 2.0 clients [SBA15]. To be able to use the OIDC client and authenticate the users to the PCMFrontend, valid redirect URLs must be configured (e.g., *https://localhost:4200/home*). The redirect URLs are validated during the authentication to ensure that a request is coming from the PCMFrontend.

So far, the users of the group KarlsruheinspiredConsult can access the PCMFrontend through OIDC client PCMOC with their username and password. However, since the KiC employees have vast access to the application PCM, the authentication is further strengthened by introducing a second factor. This process is called Multi-factor Authentication (MFA). Okta can enable MFA per OIDC client or per Okta organization. It is also possible to enable the MFA for selected users or groups. The configuration of MFA is done through an authentication policy. For the application PCM, time-based one-time-passwords (TOTP) are selected as a second authentication factor [Vi11].

To enforce the second factor, a new multifactor authentication policy is created and enrolled for the group KarlsruheinspiredConsult. Enabling the MFA policy changes the authentication process displayed in Fig. 3. After the user is prompted the authentication form and enters the credentials, the authorization server verifies the credentials and checks if there is an MFA policy enabled for the user. If the user is from KiC, the user is asked to enter a TOTP. After the user enters the correct one-time password, the authentication process is continued and the user is successfully authenticated using MFA. Otherwise, the authentication process fails, and the user is not authenticated.

3.3 Implementation of the Authentication

For the implementation of the authentication, Okta allows to either sign in users through a redirect to their page or to use an embedded widget which performs the authentication in the background. The PCMFrontend uses the customized embedded Okta sign-in widget to provide users a seamless interface throughout the authentication process [In22a].

The authentication process for a user is further described in the UML sequence diagram in Fig. 3. The goal of the authentication process is to obtain an ID token as well as an access token. After a user opens the «ng component» Login, the «ng service» Authentication is called, which uses the Okta sign-in widget to initiate the OIDC flow with PKCE. Therefore, the «ng service» Authentication first has to generate a verifier as well as a code challenge. The verifier is a random string with a length between 43 and 128 characters. The code challenge is the SHA256 hashed value of the verifier [SBA15]. The code challenge is then included in an HTTP GET request to the endpoint */authorize* of the OAuth 2.0 «authorization server» Default. Afterwards, the «ng component» Login presents the embedded Okta sign-in widget. The user can then enter their credentials (i.e., username and password) and press the login button. The credentials are then parsed to the «authorization server» Default and are subsequently verified. Otherwise, an authorization code is returned to the «ng service» Authentication. The authorization code can then be exchanged by the «ng service» Authentication to obtain the requested tokens. Therefore, the «ng service» Authentication has to send an HTTP POST request which includes the authorization code as well as the code verifier to the */token* endpoint of the «authorization server» Default. The «authorization server» Default can then verify the authorization code. To verify the code verifier, the SHA256 hash function must first be applied to the code verifier and the result compared to the code challenge. If the authorization code and code verifier can be verified, the ID token and the access token are returned to the «ng service» Authentication. Finally, the user is redirected to the «ng component» LandingPage.

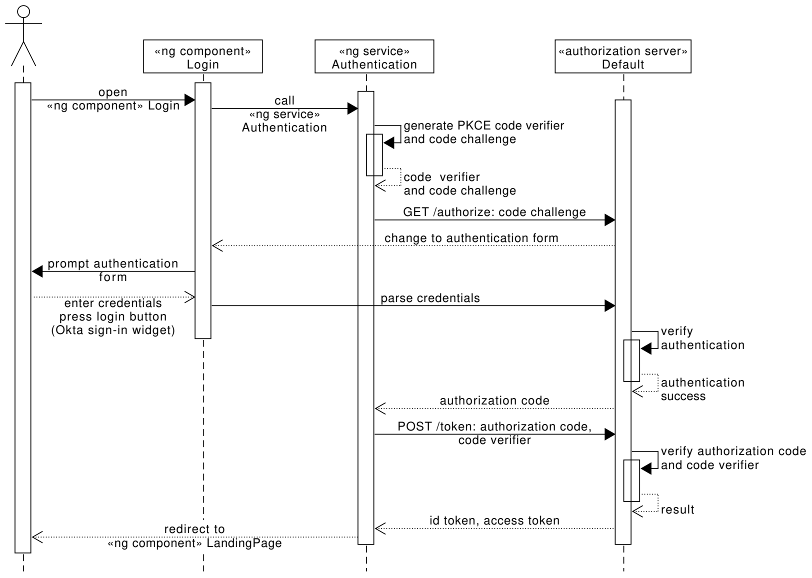
«ng component» Login
«ng service» Authentication
«authorization server» Default
open «ng component» Login
call «ng service» Authentication
generate PKCE code verifier and code challenge
code verifier and code challenge
GET /authorize: code challenge
change to authentication form
prompt authentication form
enter credentials press login button (Okta sign-in widget)
parse credentials
verify authentication
authentication success
authorization code
POST /token: authorization code, code verifier
verify authorization code and code verifier
result
id token, access token
redirect to «ng component» LandingPage

Fig. 3: Sequence Diagram of the Authentication Flow

4 Authorization

The realization of authorization is required to protect the user data of the application PCM. Similar to the authentication, the authorization is an essential aspect of the design phase and consequently realized in the implementation phase. To perform authorization decisions, this section will describe the design of the access control mechanisms in the frontend and backend. For the application PCM, the access control decision has to be implemented in two places. First, the PCMFrontend has to decide what view a user can access based on the group membership of a user (i.e., Attribute Based Access Control). Second, the iPaaS system MuleSoft must decide what entities a user can access on an API-level.

4.1 PCMFrontend-level Authorization

After the successful authentication, the access token is returned from the authorization server Default to the PCMFrontend. Listing 1 presents an excerpt of an access token provided by Okta. The access token contains requested information in the JWT format and is used by the PCMFrontend to decide which «ng component» a user can access [JBS15].

In the architecture of the PCMFrontend presented in Fig. 2, the «ng service» Authorization Guard and the «ng service» Permissions are used to perform the authorization. The «ng service» Permissions contains a mapping from group names to paths (i.e., «ng component») that a user is allowed to access. The AuthorizationGuard examines the access token and extracts the claim groups. In the example access token of Listing 1, the user is in the groups Everyone and CaseStudyCompany (line 5). Based on the extracted groups, the AuthorizationGuard decides if the user can access a path or not. If a user only has the group CaseStudyCompany in their access token, the access to the «ng component» VehicleUser is allowed, while the access to the «ng component» Garage is denied.

```
1.    "exp": 1643886904,
2.    "cid": "0oa3ouku57Q89YCEc0x7",
3.    "scp": [ "openid", "email", "profile" ],
4.    "sub": "alex.twin@csc.com",
5.    "groups": [ "Everyone", "CaseStudyCompany" ]
```

List. 1: Excerpt of an Access Token Provided by Okta

4.2 API-Level Authorization

Protecting the access to an «ng component» is not enough since the «ng component» obtain their data from a RESTful API which could otherwise be directly accessed. Thus, the access must also be protected on an API-level. There are two challenges that generally must be overcome: First, coarse-grained authorization requires to protect the API itself (i.e., who can access a specific API endpoint). Second, fine-grained authorization decides what resources a user can access and which operations they can perform. A common approach to solve coarse-grained authorization, which is also supported by MuleSoft, is the use of API gateways as a mediator between a client and a service. An API gateway allows to create API proxies and apply and enforce policies to such a proxy. As depicted in Fig. 2, the application PCM has the «experience microservice» E-VehicleUser and the «experience microservice» E-Garage which provide the data for the «ng component» VehicleUser and the «ng component» Garage respectively. A «experience microservice» provides tailored API endpoints for each user group. To manage the traffic to an «experience microservice», API proxies provided by MuleSoft are used. In Fig. 2, the «api proxy» A-VehicleUser and an «api proxy» A-Garage for the respective «experience microservice» are shown.

To solve the coarse-grained authorization, the JWT can be inspected by the «api proxy» similar to the PCMFrontend. The corresponding policy is called JWT validation. Before an «api proxy» can introspect the JWT, the PCMFrontend has to include the access token which it previously received from Okta in every HTTP request that is sent to the «api proxy». To include a JWT in an HTTP request, the HTTP authorization header must be set with the JWT set as bearer [FR14; JH12]. If the «api proxy» receives a request without a JWT, the request is immediately returned as unauthorized. Otherwise, the JWT validation policy will perform three checks:

First, it must be validated that the JWT itself has not been tempered with and is not yet expired. A JWT consists of a header, a payload, and a signature [JBS15]. The signature is encrypted by the token issuer (i.e., Okta). To verify the contents of a token, the signature of the token has to be decrypted using a public key. The OAuth 2.0 authorization server can provide the keys to a third party through a JSON Web Key Set (JWKS) endpoint [Jo15]. In this case, the authorization server Default offers a JWKS endpoint which is referenced by the «api proxy» A-VehicleUser and the «api proxy» A-Garage to verify the origin of the JWT. Second, Okta adds the custom claim cid to its JWT which represents the id of the OIDC client that performed the authentication request. An example can be found in line 2 of the JWT displayed in Listing 1. In this case, the value $0oa3ouku57Q89YCEc0x7$ represents the client id of the «oidc client» PCMOC which was used to issue the access token. With this verification, the «api proxy» can verify that the token was requested by the OIDC client of the application PCM. Third, based on the group assigned to the user, the access to an «experience microservice» should either be allowed or denied. Hence, the claim group from the JWT must be evaluated. In the example access token in Listing 1, the user has the groups EveryOne and CaseStudyCompany. Thus, the user should only be allowed to access the «experience microservice» E-VehicleUser. To access the «experience microservice» E-Garage, the group CaseStudyGarage must be present in the JWT.

An exemplary request to receive vehicle information for a specific Vehicle Identification Number (VIN) is presented in Fig. 4. Here, the interaction between involved systems can be seen. The user of the PCMFrontend initiates an HTTP GET request to an endpoint /vin/id

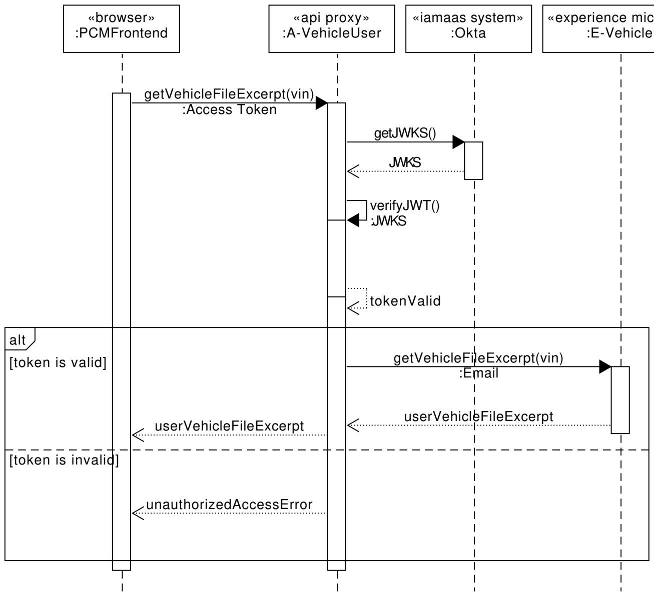

Fig. 4: Excerpt of the Authorization Sequence

which is intercepted by the «api proxy» A-VehicleUser. Before the proxy can apply the JWT validation policy, the JWKS is fetched. Subsequently, the JWT is validated, and the custom claims are checked. If the request is performed by an employee of CSC, the request is forwarded to the «experience microservice» E-VehicleUser and the result is returned to the PCMFrontend. Otherwise, an unauthorized error message is returned to the PCMFrontend.

At this point, the «experience microservice» E-VehicleUser is protected against requests by users outside the CSC and KiC. This leaves the question on how to perform fine-grained authorization, e.g., how to decide which vehicle a user is allowed to access and how to enforce this access. Generally, deciding the access to a resource on a fine-grained level can either be performed inside the application (i.e., business logic) or externalized (i.e., by an external policy enforcement point).

In case of the «experience microservice» E-Vehicle and E-Garage, the authorization decision is performed inside the microservice based on the the email address of the requesting user which is available in the JWT. The experience microservices will perform an internal lookup to check if the email address is allowed to access the VIN. For example, if Alex Twin performs a request for a vehicle file using the access token from Listing 1, the «experience microservice» E-VehicleUser will use the email address alex.twin@csc.com from the token to lookup if the email address is allowed to access the VIN. If so, the request is allowed and the vehicle file is returned to the «browser» PCMFrontend. Otherwise, an unauthorized error is returned.

5 Conclusion

In this publication, we described the development of a software as a service solution using the cloud products MuleSoft and Okta. The focus is set on securing the application by integrating authentication and authorization mechanisms using the protocols OIDC and OAuth 2.0. The integration of authentication and authorization requires the coordination between systems. The IAMaaS Okta stores the user information and groups and provides the OIDC client which is used by the PCMFrontend. The Angular frontend uses the configured OIDC client to initiate the authentication flow. The PCMFrontend decides which views can be accessed based on the groups of a user. To decide which experience microservices can be accessed by a user, API proxies with a JWT validation policy are used. The fine-grained authorization decisions are performed inside the experience microservices.

Authenticating users in a single page application using OIDC and Angular is a well established topic. However, performing authorization decisions, especially fine-grained authorization decisions, in a microservice architecture must be further researched. In this publication, we performed coarse-grained authorization decisions inside an API proxy by only allowing access to an experience microservice for a certain group. Currently, the decision of who can access a specific vehicle is performed in the logic of the experience microservice. While this is a simple solution, if the requirements for the decision changes, the experience microservice must also be changed due to the tight coupling. Thus, authorization decisions should rather be performed outside the microservice. This also leaves the microservice with its core functionalities. However, externalizing the authorization decision requires additional infrastructure (i.e., decision points) as well as the definitions of access policies. Future research will investigate the structured development, the systematic storage, and the enforcement of authorization policies in applications using the microservice architecture.

References

[AC22] de Almeida, M. G.; Canedo, E. D.: Authentication and Authorization in Microservices Architecture: A Systematic Literature Review. en, Applied Sciences 12/6, p. 3023, Mar. 2022, visited on: 03/31/2022.

[Fo21] Foundation, O.: OWASP Top 10:2021, Publication, OWASP Foundation, 2021.

[FR14] Fielding, R. T.; Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Authentication, RFC 7235, June 2014.

[HY17] He, X.; Yang, X.: Authentication and Authorization of End User in Microservice Architecture. en, Journal of Physics: Conference Series 910/, p. 012060, Oct. 2017, visited on: 03/31/2022.

[ID12] Iankoulova, I.; Daneva, M.: Cloud computing security requirements: A systematic review. In: 2012 Sixth International Conference on Research Challenges in Information Science (RCIS). IEEE, pp. 1–7, 2012.

[In22a] Inc., O.: Okta Documentation: Embedded Okta Sign-In Widget fundamentals, tech. rep., [retrieved 05/04/2022], 2022.

[In22b] Inc., O.: The World's #1 Identity Platform | Okta, tech. rep., [retrieved 05/04/2022], Okta Inc., 2022.

[JBS15] Jones, M.; Bradley, J.; Sakimura, N.: JSON Web Token (JWT), RFC 7519, May 2015.

[JH12] Jones, M.; Hardt, D.: The OAuth 2.0 Authorization Framework: Bearer Token Usage, RFC 6750, Oct. 2012.

[Jo15] Jones, M.: JSON Web Key (JWK), RFC 7517, May 2015.

[Mu22a] MuleSoft: API-led connectivity - The next step in the evolution of SOA, `https://www.mulesoft.com/lp/whitepaper/api/api-led-connectivity`, 2022, visited on: 03/31/2022.

[Mu22b] MuleSoft: MuleSoft | Integration Platform for Connecting SaaS and Enterprise Applications, `https://mulesoft.com/`, 2022, visited on: 03/31/2022.

[Ne15] Newman, S.: Building Microservices: Designing Fine-grained Systems. O'Reilly Media, Inc., 2015.

[Ne19] Nehme, A.; Jesus, V.; Mahbub, K.; Abdallah, A.: Fine-Grained Access Control for Microservices. In (Zincir-Heywood, N.; Bonfante, G.; Debbabi, M.; Garcia-Alfaro, J., eds.): Foundations and Practice of Security. Vol. 11358, Series Title: Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 285–300, 2019, ISBN: 978-3-030-18419-3, visited on: 04/20/2022.

[OM17] OMG: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1, tech. rep., 2017, visited on: 08/02/2021.

[SBA15] Sakimura, N.; Bradley, J.; Agarwal, N.: Proof Key for Code Exchange by OAuth Public Clients, RFC 7636, Sept. 2015.

[SS75] Saltzer, J.; Schroeder, M.: The protection of information in computer systems. Proceedings of the IEEE 63/9, pp. 1278–1308, 1975.

[Vi11] View, M.; Rydell, J.; Pei, M.; Machani, S.: TOTP: Time-Based One-Time Password Algorithm, RFC 6238, May 2011.

[XJK19] Xu, R.; Jin, W.; Kim, D.: Microservice Security Agent Based On API Gateway in Edge Computing. en, Sensors 19/22, p. 4905, Nov. 2019, visited on: 04/20/2022.

[ZJJ18] Zhao, J. T.; Jing, S. Y.; Jiang, L. Z.: Management of API Gateway Based on Micro-service Architecture. en, Journal of Physics: Conference Series 1087/, p. 032032, Sept. 2018, visited on: 04/20/2022.