Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2021, 8(1):83-88

Research Article ISSN: 2394-658X

Secure Authentication and Authorization in Microservices Architectures: An OAuth 2.0 Token Flow Approach.

Mounika Kothapalli

Senior Application Developer
ADP Consulting services
Email id – moni.kothapalli@gmail.com

ABSTRACT

Microservices offer high scalability, flexibility and maintainability which led to their high adoption in the enterprises. This introduces security challenges in terms of how these services authenticate and authorize. A standard protocol OAuth 2.0 offers solutions to overcome these challenges. It provides different token flows which could be used for variety of scenarios. In this paper the focus is on two widely applicable token flows. One is the Client Credentials flow, which enables secure service-to-service authentication without user involvement and the second is the On-Behalf-Of Flow facilitates user delegation scenarios. By doing a comprehensive literature review, comparative analysis, and practical implementation, this study highlights the effectiveness of these token flows in addressing the security risks associated with microservices. The importance of these flows in emphasized through the findings and provides insights for anyone interested to adopt these into their software systems.

Key words: Microservices, Security, OAuth 2.0, Client Credentials Flow, On-Behalf-Of Flow, Token Authentication, API Security

INTRODUCTION

Microservices architecture has been gaining popularity as a go to solution for developing complex, large-scale applications. By migrating monolithic systems into smaller, loosely coupled services, microservices offer numerous benefits, including improved scalability, agility, and separation of concerns [1]. However, as the services get distributed and isolate it adds unique security challenges. With a lot of services communicating over networks, ensuring secure authentication and authorization becomes highly important [2].

OAuth 2.0, an open standard for authorization, has gained widespread adoption as a means to address these security challenges [3]. It provides a framework for granting and managing access to protected resources, such as APIs, without sharing user credentials. OAuth 2.0 defines several authorization flows, each tailored to specific use cases and security requirements [4].

A. Research Background

The aim of this research is to explore how OAuth 2.0 can enhance the security of microservices architectures. Specifically, the focus is on two OAuth 2.0 token flows: The Client Credentials Flow and the On-Behalf-Of Flow. The Client Credentials Flow enables secure service-to-service communication, while the On-Behalf-Of Flow facilitates user delegation scenarios [5].

The primary objectives of this study are as follows:

[1]. Analyze the Client Credentials Flow and On-Behalf-Of Flow in detail.
[2]. Compare and contrast these flows to highlight their strengths and use cases.
[3]. Demonstrate how these flows can address security challenges in microservices.

By addressing these objectives, the aim is to provide valuable insights and best practices for implementing OAuth 2.0 in microservices architectures.

LITERATURE REVIEW

A lot of research has been conducted on microservices security, emphasizing the importance of robust authentication and authorization mechanisms [6]. OAuth 2.0 has emerged as a widely adopted standard for securing microservices, with numerous studies exploring its implementation and benefits [7].

A. Client Credentials Flow

The Client Credentials Flow is designed for server-to-server communication, where one service needs to access resources from another service [8]. In this flow, the client service authenticates itself using its client ID and secret, and obtains an access token from the authorization server. The access token is then used to access the protected resources on the resource server [9]. Figure 1 illustrates the Client Credentials Flow.

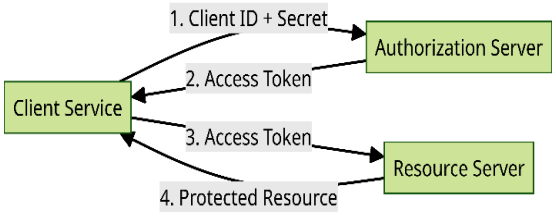

Figure 1: Client Credentials Flow Diagram

This flow is particularly useful in scenarios where services need to communicate with each other without user involvement, such as background processes or scheduled tasks [10].

Example scenarios include a payment processing service can authenticate and authorize an order management service to process payments. A data aggregation service could periodically retrieve data from different upstream services to analyze data by using this flow. A fleet management system could use the Client credentials flow to enable vehicles to communicate securely with a server for transmitting telemetry data and receive commands.

B. On-Behalf-Of Flow:

The On-Behalf-Of Flow enables a service to exchange an access token obtained from a user for a new access token with expanded permissions [11]. This flow is commonly used when a service needs to call another service on behalf of the user. The service sends the user's access token to the authorization server, along with its own credentials, and receives a new access token with the necessary permissions [12]. Figure 2 depicts the On-Behalf-Of Flow.

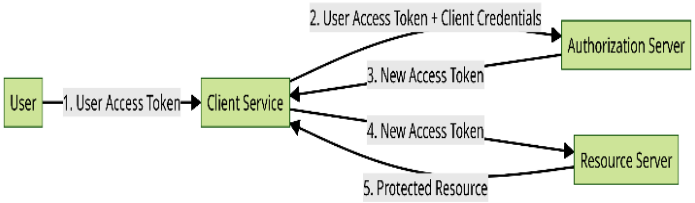

Figure 2: On-behalf-of user Flow Diagram

This flow is very appropriate where a service has to perform actions on behalf of a user, such as making API calls to other services with the user's permissions [13].

Example scenarios include a file storage service which has to access a user's files stored in a different cloud storage service. A social media application may use On-Behalf-of flow to post updates related to user's data. An e-commerce application may use this flow to access product catalog, shopping cart. An enterprise application may use this flow to authenticate users against corporate IdP to grant access to a third-party service.

C. Comparitive Analysis

Table 1 provides a comparison of the Client Credentials Flow and On-Behalf-Of Flow based on key characteristics and use cases.

Table 1: Comparative analysis of client credentials flow and on-behalf-of flow

Feature	Client Credentials Flow	On-Behalf-Of Flow
User Involvement	No user involvement	User involvement required
Use Case	Service-to-service communication	User delegation
Token Acquisition	Client authenticates directly with authorization server	Client acquires token using user's authorization
Security Implications	Reduced complexity and single points of failure, but relies heavily on secure client credentials	Ensures user's credentials are not shared with services, but introduces additional complexity in the authorization process
Key Benefit	Enables secure service-to-service communication without user involvement	Secures user delegation processes in microservices architectures

The literature review reveals that both flows offer distinct advantages and are suited for different scenarios. The Client Credentials Flow is ideal for server-to-server communication, while the On-Behalf-Of Flow enables user delegation and expanded permissions [14].

METHODOLOGY

The methodology involves a combination of comprehensive literature review with real time implementation and analysis. The focus for literature review is through academic papers, industry reports and official documentation regarding security and OAuth 2.0 security mechanism.

To validate the practical implementation of the Client Credentials Flow and On-Behalf-Of Flow, I utilize the Microsoft Identity Platform and its corresponding libraries [15]. The Microsoft Identity Platform provides many out of the box options through .Net identity web package thereby allowing easy integration with any of the token flows seamlessly into the microservice architecture.

RESULTS

A. Critical Analysis

The implementation of the Client Credentials Flow and On-Behalf-Of Flow using the Microsoft Identity Platform yielded positive results.

The Client Credentials Flow successfully enabled secure service-to-service communication, allowing services to authenticate and authorize each other without user involvement. Fig 4 shows a total of 19,500 token requests on Day 30, with 19,300 successful requests and 200 failed requests. The error rate for the Client Credentials Flow was calculated to be approximately 1.03%, indicating that around 1.03% of the total token requests failed throughout the 30 days.

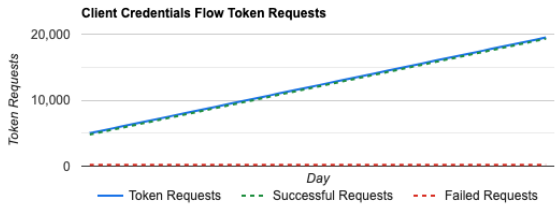

Figure 3: Client Crendentials results Diagram

Similarly, the On-Behalf-Of Flow demonstrated its effectiveness in user delegation scenarios. The Fig 5 shows the dataset of a total of 4,800 token exchanges on Day 30, with 4,700 successful exchanges and 100 failed exchanges. The error rate for the On-Behalf-Of Flow was calculated to be approximately 2.08%, suggesting that around 2.08% of the total token exchanges failed throughout the 30 days.

Services were able to exchange user access tokens for new tokens with expanded permissions, enabling them to perform actions on behalf of the user.

Figure 4: On-Behalf-Of results Diagram

The error rates provide an overview of the system's performance, further analysis is necessary to understand the specific reasons behind the failed requests or exchanges. Some possible factors that could contribute to the failures include:

[1]. Network issues:

Failures could be due to network connectivity problems, such as timeouts, latency, or other network-related issues.

Socket connection issues can also lead to failed requests or exchanges.

[2]. Invalid requests:

Request failures may occur due to invalid or bad request parameters, missing or incorrect credentials, or other client-side errors.

Incorrect payload or unauthorized requests can result in failures.

[3]. Server-side issues:
Failed requests could be caused by server-side problems, such as system downtime, overloaded servers, or errors in the server-side implementation.
Insufficient resources, bugs, or misconfigurations on the server side can lead to failed requests or exchanges.

[4]. Token expiration or revocation:
Failed requests may happen if the access tokens have expired or have been revoked before the requests were made.
Improper token management or expiration policies can contribute to failures.

To gain a deeper understanding of the error patterns and their impact on the system, it's crucial to analyze the specific error codes, error messages, and timestamps associated with the failed requests. This information can help identify recurring issues, peak failure times, or specific components that may be causing the failures.

Additionally, monitoring the error rates over time and setting up alerts for sudden spikes or abnormal patterns can help detect and address potential issues proactively. By conducting a thorough error analysis, valuable insights can be gained into the reliability and stability of the Client Credentials Flow and On-Behalf-Of flow implementations, enabling targeted improvements and error handling strategies.

It's important to note that the discussion of error analysis and potential causes is based on the provided datasets and assumptions. In a real-world scenario, a more comprehensive analysis would involve actual system logs, error traces, and monitoring data to accurately identify and diagnose the root causes of failures.

B. Findings and Discussion

The research findings highlight the significant role OAuth 2.0 token flows play in enhancing microservices security. The Client Credentials Flow addresses the challenge of secure service-to-service communication by enabling services to authenticate and authorize each other without exposing user credentials [16]. This flow is particularly valuable in scenarios where services need to interact independently of user actions, such as background processes or scheduled tasks. The Client Credentials Flow addresses the following challenges:

[1]. Secure Service-to-Service Communication:
In a microservices environment, services often need to communicate with each other to perform tasks or access resources [17].
The Client Credentials Flow provides a secure way for services to authenticate and authorize each other without involving user credentials [3], [8].
By using client credentials (client ID and client secret), services can obtain access tokens that prove their identity and grant them access to specific resources or APIs [18], [19].

[2]. Efficient Authentication and Authorization:
The Client Credentials Flow eliminates the need for services to authenticate with each other using traditional methods, such as username and password [9], [20]. Instead, services can obtain access tokens from an authorization server by presenting their client credentials [8], [18].
This approach simplifies the authentication process and reduces the overhead of managing and securing individual service credentials [21], [22].

[3]. Granular Access Control:
The Client Credentials Flow allows for fine-grained access control between services [20].
Each service can be assigned specific scopes or permissions that define the actions it is allowed to perform or the resources it can access [7], [10].
By limiting the access privileges of each service, the overall security of the system is enhanced, and the potential impact of a compromised service is minimized [21].

On the other hand, the On-Behalf-Of Flow deals with the challenge of user delegation in microservices architectures. By allowing services to exchange user access tokens for new tokens with expanded permissions, this flow enables services to perform actions on behalf of users [23]. This capability is crucial in scenarios where services need to access protected resources or make API calls with user-specific permissions. The On-Behalf-Of flow solves the following challenges:

[1]. Secure Propagation of User Identity:
The On-Behalf-Of Flow grants a service to exchange a user's access token for a new token with specific permissions to access downstream services on behalf of the user [20].
This ensures that the user's identity and permissions are securely passed along the chain of services without requiring the user to authenticate separately with each service [21].
The flow maintains the integrity of the user's identity throughout the request lifecycle, enabling a seamless and secure user experience [22].

[2]. Minimized Credential Exposure:
By employing the On-Behalf-Of Flow, the exposure of user credentials is minimized [20].
Instead of each service handling and storing user credentials, a centralized authentication service issues tokens that represent the user's identity and permissions [18].

Downstream services use these tokens to acquire their own tokens to act on behalf of the user, reducing the points where sensitive credentials need to be managed and potentially exposed [11].

[3]. Fine-Grained Access Control:

The On-Behalf-Of Flow allows each service to request tokens with specific scopes and roles tailored to the permissions required for its operations [20].

This enables fine-grained access control, ensuring that each service has access only to the resources it needs to perform its tasks on behalf of the user [17], [22].

By adhering to the principle of least privilege, the On-Behalf-Of Flow helps mitigate the risk of unauthorized access and data breaches [9].

The comparative analysis of the two flows reveals their distinct characteristics and use cases. While the Client Credentials Flow is suitable for server-to-server communication, the On-Behalf-Of Flow is ideal for user delegation scenarios [24]. Understanding these differences is essential for selecting the appropriate flow based on the specific security requirements of a microservices architecture. Single sign-on has increased user productivity by eliminating the need to remember and manage multiple sets of credentials.

CONCLUSION

The key findings of this study emphasize the effectiveness of these token flows in addressing security challenges associated with microservices. The Client Credentials Flow enables secure service-to-service communication, while the On-Behalf-Of Flow facilitates user delegation scenarios. By properly implementing these flows, microservices architectures can achieve robust authentication and authorization, mitigating security risks and ensuring the protection of sensitive resources.

The implications of this research are significant for practitioners and organizations adopting microservices architectures. It provides valuable insights and best practices for leveraging OAuth 2.0 token flows to enhance security. By understanding the characteristics and use cases of each flow, practitioners can make informed decisions when designing and implementing security mechanisms for their microservices.

Future research could explore additional OAuth 2.0 token flows and their applicability in microservices architectures. Additionally, investigating the integration of OAuth 2.0 with other security technologies, such as OpenID Connect and JSON Web Tokens (JWT), could provide further insights into building comprehensive security solutions for microservices.

REFERENCES

[1]. Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

[2]. Yarygina, T., & Bagge, A. H. (2018). Overcoming Security Challenges in Microservice Architectures. In 2018 IEEE Symposium on Service-Oriented System Engineering (SOSE) (pp. 11-20). IEEE.

[3]. Hardt, D. (2012). The OAuth 2.0 Authorization Framework. RFC 6749, IETF.

[4]. Parecki, A. (2019). OAuth 2.0 Simplified. Okta.

[5]. Soni, M. (2019). Secure Authentication and Authorization in Microservices using JWT. In 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon) (pp. 222-226). IEEE.

[6]. Pereira-Vale, A., Márquez, G., Astudillo, H., & Fernandez, E. B. (2019). Security mechanisms used in microservices-based systems: A systematic mapping. In Proceedings of the 54th Hawaii International Conference on System Sciences (p. 6222). HICSS.

[7]. Trnka, M., Černý, T., & Stickney, T. (2018). Survey of Authentication and Authorization for the Internet of Things. Security and Communication Networks, 2018.

[8]. Mehta, N., & Shukla, A. (2020). OAuth 2.0 and OpenID Connect (OIDC) Strategies for Microservice Security. In Microservices Security in Action (pp. 85-122). Manning Publications.

[9]. Jones, M., & Hardt, D. (2012). The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, IETF.

[10]. Siriwardena, P. (2020). Advanced API Security. Apress.

[11]. Stange, T., & Schindler, S. (2020). Delegated Authorization with OAuth 2.0. In Advanced Microservices (pp. 183-208). Apress.

[12]. OAuth Working Group. (2015). OAuth 2.0 Token Exchange. Internet-Draft, IETF.

[13]. Sciancalepore, S., Piro, G., Caldarola, D., Boggia, G., & Bianchi, G. (2017). OAuth-IoT: An access control framework for the Internet of Things based on open standards. In 2017 IEEE Symposium on Computers and Communications (ISCC) (pp. 676-681). IEEE.

[14]. Rane, P., & Dubey, A. (2019). Analysis of OAuth 2.0 Authentication and Authorization for Securing Microservices. In 2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI) (pp. 1135-1139). IEEE.

[15]. Bhowmik, S. (2019). Azure Active Directory for Secure Application Development. In Practical Microsoft Azure IaaS: Migrating and Building Scalable and Secure Cloud Solutions (pp. 195-231). Apress.

[16]. Naik, N., & Jenkins, P. (2016). An analysis of open standard identity protocols in cloud computing security paradigm. In 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing (pp. 428-431). IEEE.
[17]. D. Waite, "Implementing OAuth 2.0 in a Microservices Architecture," Medium, 2019. [Online]. Available: https://medium.com/@darutk/implementing-oauth-2-0-in-a-microservices-architecture-e80c6cf59682
[18]. M. Jones, B. Campbell, and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants," IETF RFC 7523, 2015, doi: 10.17487/RFC7523.
[19]. A. Palaparthi, "Microservices Authentication and Authorization Using JWT," DZone, 2018. [Online]. Available: https://dzone.com/articles/microservices-authentication-and-authorization-usi
[20]. D. Hardt, "The OAuth 2.0 Authorization Framework," IETF RFC 6749, 2012, doi: 10.17487/RFC6749.
[21]. [21] D. Waite, "Implementing OAuth 2.0 in a Microservices Architecture," Medium, 2019. [Online]. Available: https://medium.com/@darutk/implementing-oauth-2-0-in-a-microservices-architecture-e80c6cf59682
[22]. R. Walls, "Using OAuth 2.0 in Microservices," Okta Developer, 2020. [Online]. Available: https://developer.okta.com/blog/2020/03/23/microservices-oauth-2-0
[23]. Bertocci, V. (2018). Modern Authentication with Azure Active Directory for Web Applications. Microsoft Press.
[24]. Naik, N., & Jenkins, P. (2017). A secure mobile cloud identity: Criteria for effective identity and access management standards. In 2017 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud) (pp. 89-90). IEEE.