Flexible Method for Supporting OAuth 2.0 Based Security Profiles in Keycloak

Takashi Norimatsu[1,2], Yuichi Nakamura[1] and Toshihiro Yamauchi[3]

Abstract: Keycloak is identity and access control open-source software. When used for open banking, where many OAuth 2.0 clients need to be managed and a different OAuth 2.0-based security profile needs to be applied to each type of API, the problem of increasing managerial costs by the Keycloak administrator occurs because Keycloak's security profile logic depends on the client settings, and the logic cannot be changed for each client's request. This paper proposes its solution by separating the security profile logic from the client settings, and by changing the security profile for each client's request based on the content of the request, and actual security profiles Financial-grade API (FAPI) are implemented to Keycloak. The paper calculates managerial costs in both the existing and proposed methods in scenarios managing FAPI, and compares the results. The comparison shows that using the proposed method reduces costs. Our implementations are contributed to Keycloak.

Keywords: OAuth 2.0, Security Profile, FAPI, Open Source, Keycloak, Open Banking

1 Introduction

OAuth 2.0 [Ha21] is a widely used web-based authorization protocol. It is defined as a framework, so it can be used flexibly in a wide range of use cases. This flexibility, however, might introduce security holes if it is used incorrectly or inappropriately. To prevent such problems, detailed methods for using OAuth 2.0 securely have been developed. These are called *security profiles*.

Some organizations have standardized and published security profiles. Examples of such security profiles are Financial-grade API Security Profile 1.0 Baseline (FAPI1-baseline) [Fi21a] and Advanced (FAPI1-advanced) [Fi21b] by the OpenID Foundation (OID-F).

The security profiles of several in-service open banking systems are based on FAPI1-advanced. Examples of such security profiles are Open Banking Security Profiles [Op21a] in the UK, Consumer Data Right (CDR) security profile [Co21] in Australia, and Open Banking Brasil Financial-grade API Security Profile 1.0 [Op21b] in Brazil.

[1] Hitachi, Ltd., 6-6, Marunouchi 1-chome, Chiyoda-ku, Tokyo, 100-8280 Japan
[2] Graduate School of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530 Japan
[3] Faculty of Natural Science and Technology, Okayama University, 3-1-1 Tsushima-naka, Kita-ku, Okayama 700-8530 Japan, https://orcid.org/0000-0001-6226-5715

cbe doi:18.18420/OID2022-07

In systems supporting security profiles based on OAuth 2.0, the authorization server is a key component because it plays a central role in OAuth 2.0. A variety of proprietary and open-source identity and access management (IAM) software support the functionality of authorization server. Keycloak[4] is one example of such IAM open-source software. It is written in Java and is used widely for authentication and authorization purposes [NK20]. Keycloak can be used free-of-charge but there are some charged services[5].

Keycloak manages several kinds of entities. Figure 1 shows some of the relationships among them. The figure only depicts the entities and relationships that relate to the topic of this paper.

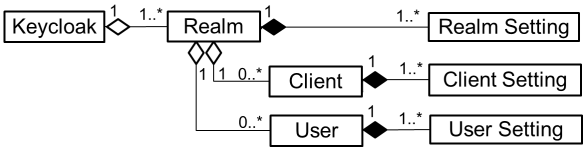


Fig. 1: Some of the relationships among entities managed by Keycloak

Keycloak creates realms that are separated and not accessible to each other. Keycloak manages clients, users and other entities within a realm. Each realm has its own settings.

Keycloak manages a client application ("Client" in Figure 1 and generally referred to as just "client"), which provides some services to end users, and treats this client as an OAuth 2.0 client. A client has several kinds of settings. Some are defined by OAuth 2.0 and called Client Metadata [Jo21] while others are defined by Keycloak. A client can send several kinds of requests to Keycloak, and these requests (for example, authorization requests and token requests) are defined by OAuth 2.0. When a client sends a request to Keycloak, Keycloak processes the request based on the client settings in Keycloak.

Keycloak manages each end user ("User" in Figure 1) of a client. For example, in an authorization code flow of OAuth 2.0, Keycloak needs to authenticate a user and get consent from the user to allow the client to access the user's own resources.

Applying a security profile by Keycloak means that Keycloak processes a request from a client and judges whether the request satisfies the requirements of the security profile. If the requirements are satisfied, Keycloak returns a normal response. If they are not satisfied, Keycloak returns an error response.

Existing open banking systems need to manage many clients because the systems are applied nation-wide. Open banking systems also need to support multiple types of APIs that require different security levels because the systems need to apply a different security profile for each type of API.

[4] https://www.keycloak.org, accessed: 06/07/2021.
[5] https://access.redhat.com/products/red-hat-single-sign-on, accessed: 16/12/2021.

When Keycloak is used as an authorization server in open banking, two problems increase the managerial costs of a Keycloak administrator. The first problem is that the Keycloak administrator needs to manage many client settings to apply security profiles. The second problem is that the Keycloak administrator needs to create a realm for each security profile and needs to manage the realm and entities included in the realm. The cause of these problems is that Keycloak's security profile logic depends on the client settings and cannot be changed for each client's request.

To resolve these problems, we designed a flexible method for supporting several security profiles and implemented it with Keycloak. To resolve the first problem, the method can separate the logic related to security profiles from the client settings in Keycloak, so that the client settings of all clients do not need to be set up in Keycloak to apply a security profile. This separation reduces the costs for managing clients. To resolve the second problem, the method can change the security profile dynamically for each client request based on the content of the client request, so that a realm does not need to be created for each security profile. Making such changes dynamically reduces the costs for managing realms and entities included in the realms.

To show that the proposed method can resolve the problems, we calculated the managerial costs needed to apply several security profiles for many clients, and compared the resulting costs with costs from before implementing the proposed method. The comparison shows that the proposed method reduces managerial costs.

The implementations were contributed to Keycloak, reviewed by Keycloak maintainers, and successfully merged into Keycloak’s main branch.

The rest of this paper is structured as follows: Section 2 describes, in detail, the problems that occur when Keycloak is used for open banking. Section 3 gives details on the proposed method for resolving the problems. Section 4 shows the evaluation of the proposed method to show that it can resolve the problems. Section 5 describes the results of submitting the implementation to the upstream Keycloak repository. Section 6 gives works related to the proposed method. Finally, Section 7 gives a conclusion.

2 Problems related to applying security profiles

In open banking, an appropriate security profile should be applied to protect APIs. As Figure 2 shows, the straightforward method (the *client settings-based method*) of applying a security profile with Keycloak is to find the values of the client settings (in Keycloak) relevant to the security profile, set such values to the client settings, and process the client’s request by following the values. If existing client settings cannot cover a security profile, adding new client settings and logic becomes necessary.

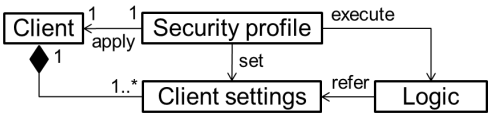


Fig. 2: Applying a security profile to a client by the client settings (in Keycloak)

Requirements for deploying an authorization server in open banking are the following:

- Requirement 1: An authorization server needs to manage many clients and end users, because open banking is applied nation-wide. For example, 319 clients have been registered for Open Banking in the UK[6].
- Requirement 2: In open banking, an authorization server needs to manage several security profiles to protect several types of APIs that require different security levels. For example, one type of API (a read API) is used to retrieve an end user's bank account's balance and transaction history. Another type of API (a write API) is used to initiate a payment service on behalf of the end user. Protecting each type of API requires different security levels. In these examples, FAPI1-baseline is intended to be used to protect a read API while FAPI1-advanced is for a write API.

However, applying security profiles by the client settings-based method for open banking is difficult due to the following two problems.

- Problem 1: The client settings of every client need to be set up appropriately for a security profile. The amount of managerial operation for managing clients increases when there are many clients.
- Problem 2: Only one security profile can be applied to one client in a realm. If multiple security profiles need to be managed, one realm needs to be created for each security profile. The amount of managerial operations for managing realms and entities included in the realms also increases when there are multiple security profiles.

These problems cause difficulties. Increasing the required operations increases costs. In addition, the increase in operations increases the risk of operational mistakes, which often cause security incidents.

3 Policy-based method for applying security profiles

To resolve the problems described in section 2, a flexible method (the *policy-based method*) for applying a security profile was designed. Its design principles are as follows.

[6] https://www.openbanking.org.uk/fintechs/, accessed: 09/01/2022.

- Design principle 1: To resolve problem 1, the logic related to security profiles is separated from client settings so that the client settings of every client do not need to be configured to be appropriate for a security profile.
- Design principle 2: To resolve problem 2, a security profile applied to a request from the same client in a realm can be changed dynamically so that another realm does not need to be created for each security profile.

Figure 3 shows a schematic diagram of the policy-based method. "Request Context" indicates a request's content and context, such as the HTTP header's value. "Security policy" determines which security profile is applied to a client's request based on the request context and client setting. To apply a security profile, the policy-based method executes its logic to judge whether the request satisfies a security profile's requirements regardless of the client settings.

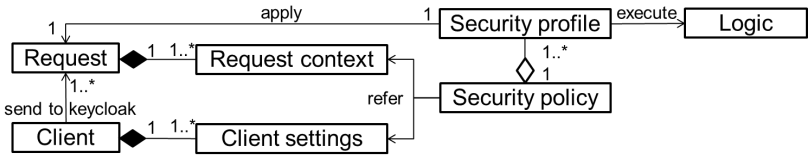


Fig. 3: Applying a security profile to a request from a client regardless of the client settings

By following the above design principles, *client policies*, the framework for processing a request from a client, are implemented as an implementation of the policy-based method.

Figure 4 shows logical components of the client policies. As shown below, four types of components are defined for client policies: executor, profile, condition. and policy.

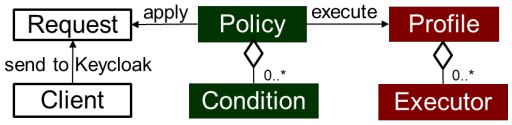


Fig. 4: Logical components of client policies

An *executor* is a component that includes the logic needed to apply part of a security profile. By following design principle 1, the logic does not depend on client settings.

A profile is a component that includes all the logic needed to apply a security profile itself. A profile consists of several executors. FAPI1-baseline, FAPI1-advanced, and FAPI-CIBA security profile (FAPI-CIBA) are implemented as executors and profiles.

A *condition* is a component that includes the logic determining whether a profile is to be applied to a client's request. By following design principle 2, the condition can use the context data of a client's request (for example, parameters including a request and the HTTP context) to determine whether a profile is to be applied to a client's request. Therefore, based on the content of the request, Keycloak can change the security profile.

A *policy* is a component that includes all the logic needed to determine whether a profile is to be applied to a client's request. A policy consists of several conditions. If all evaluation results of the conditions are positive, the policy applies a security profile.

Figure 5 shows how client policies work. The example has two policies, the FAPI1-baseline policy and the FAPI1-advanced policy. There are also two profiles, the FAPI1-baseline profile and the FAPI1-advanced profile. The FAPI1-baseline policy decides to apply the FAPI1-baseline profile if a client's request includes the "read_account" scope value of OAuth 2.0. The FAPI1-advanced policy decides to apply the FAPI1-advanced profile if the client's request includes the "bank_transfer" scope value. Then:

- When a client sends a token request with a scope including "read_account", the FAPI1-baseline policy decides to apply the FAPI1-baseline profile to the request, but the FAPI1-advanced policy decides not to apply the FAPI1-advanced profile.
- When a client sends a token request with a scope including "bank_transfer", the FAPI1-baseline policy decides not to apply the FAPI1-baseline profile to the request, but the FAPI1-advanced policy decides to apply the FAPI1-advanced profile to the request.

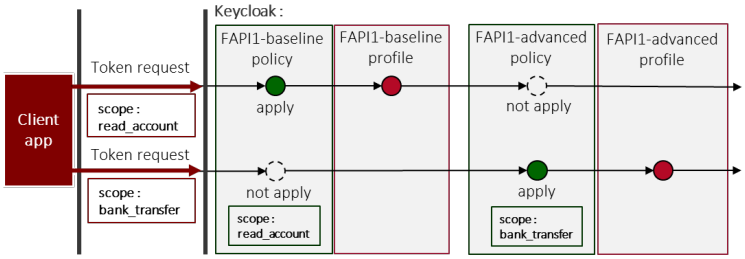


Fig. 5: An example of how client policies work

4 Evaluation

The policy-based method was evaluated to confirm that it can resolve the problems described in section 2. Using Keycloak 15.0.2, we derived the theoretical costs for managing security profiles by a Keycloak administrator in both the client settings-based method and the policy-based method. We calculated the managerial costs in scenarios managing FAPI for both the client settings-based method and the policy-based method and compared the costs.

4.1 Assumptions

To focus the discussion on managing security profiles, only entities related to security profiles (namely, realms, clients, and client policies) are considered. To simplify the

discussion, it is assumed that the managerial cost for manually specifying any setting for one entity is the same and is considered to be 1.

As described in section 2, it is assumed that a security profile can be applied by setting valid values for the client settings (in Keycloak) of a client managed by Keycloak. However, to apply FAPI1-baseline, FAPI1-advanced, and FAPI-CIBA used afterwards in the examples, some logic in Keycloak's body code becomes necessary. Such additional coding is ignored in the evaluation.

4.2 Theoretical Costs

The calculations consider the following three task patterns for managing security profiles. In each pattern, the theoretical cost is derived from the number of settings that need to be managed.

1. Initializing the Keycloak environment that supports security profiles.
2. Adding new security profiles to existing the Keycloak environment.
3. Modifying security profiles of existing the Keycloak environment.

The following variables are defined.

N_{SREL} ≡ The number of realm settings

N_{CLI} ≡ The number of clients

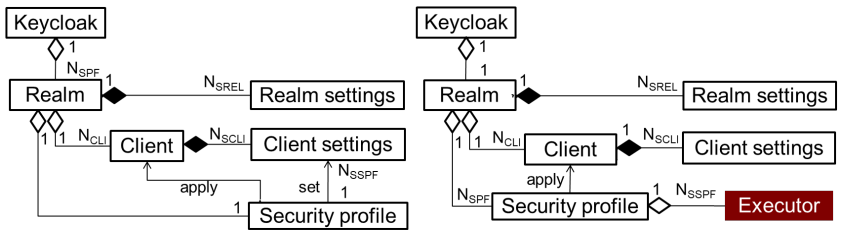
N_{SCLI} ≡ The number of client settings

N_{SPF} ≡ The number of security profiles applied to the same client

N_{SSPF} ≡ The number of settings for a security profile

N ≡ The number of settings for managing security profiles

Figure 6 shows which entities the variable corresponds to in both the client settings-based method and the policy-based method.



1. Settings-based method 2. Policy-based method

Fig. 6: Numerical relationships among security profile related entities

In the client settings-based method, N_{SSPF} is the number of client settings relating to a security profile; however, in the policy-based method, N_{SSPF} is the number of executors for a security profile.

Table 1 shows the managerial cost for each task pattern.

Pattern	N (client settings-based)	N (policy-based)
Initializing Keycloak for security profiles	$N_{SPF} \times (N_{SREL} + N_{CLI} \times (N_{SCLI} + N_{SSPF}))$ $\propto N_{SPF} \times N_{CLI}$	$N_{SREL} + N_{CLI} \times N_{SCLI} + N_{SPF} \times N_{SSPF}$ $\square N_{SPF} + N_{CLI}$
Adding new security profiles	$N_{SPF} \times (N_{SREL} + N_{CLI} \times (N_{SCLI} + N_{SSPF}))$ $\square N_{SPF} \times N_{CLI}$	$N_{SPF} \times N_{SSPF}$ $\square N_{SPF}$
Modifying security profiles	$N_{SPF} \times N_{CLI} \times N_{SSPF}$ $\square N_{SPF} \times N_{CLI}$	$N_{SPF} \times N_{SSPF}$ $\square N_{SPF}$

Tab. 1: Theoretically derived managerial costs for security profiles

In pattern 1, if the client settings-based method is used, the Keycloak administrator needs to create a realm for each security profile, and create clients and set up their client settings for the security profile for each realm. If the policy-based method is used, the Keycloak administrator needs to create only one realm, create clients in the realm and set up executors for each security profile.

In pattern 2, if the client settings-based method is used, the Keycloak administrator needs to perform the same tasks as in pattern 1. If the policy-based method is used, the Keycloak administrator needs to only set up executors for each added security profile.

In pattern 3, if the client settings-based method is used, the Keycloak administrator needs to set up client settings for all clients for a modified security profile. If the policy-based method is used, the Keycloak administrator needs to only set up executors for each modified security profile.

4.3 Calculating the Costs in the Examples

Three examples for the above task patterns are suggested: initializing the Keycloak supporting FAPI1-baseline, adding FAPI1-advanced, and modifying FAPI1-advanced to FAPI-CIBA. Keycloak has 117 realm settings (N_{SREL} = 117) and 110 client settings[7] (N_{SCLI} = 110), and it is assumed that this security profile is applied to requests from 1000 clients (N_{CLI} = 1000).

[7] https://www.keycloak.org/docs/15.0/server_admin/index.html#_oidc_clients, accessed: 09/01/2022.

Using the client settings-based method, 13 client settings (Proof Key for Code Exchange Code Challenge Method, Client Authenticator, Root URL, Admin URL, Base URL, Web Origins, Backchannel Logout URL, Valid Redirect URIs, JWKS URL, Valid Request URIs, CIBA Client Notification Endpoint URL, Consent Required, and Full Scope Allowed) need to be set up properly ($N_{SSPF} = 13$) to support FAPI1-baseline.

Using the policy-based method, 6 executors[8] (Secure Session Enforce, PKCE Enforcer, Secure Client Authenticator, Secure Client URIs, Consent Required, and Full Scope Disabled) need to be set up ($N_{SSPF} = 6$) to support FAPI1-baseline.

Using the client settings-based method, 21 client settings (Access Type: bearer, Access Type: public, Client Authenticator, Root URL, Admin URL, Base URL, Web Origins, Backchannel Logout URL, Valid Redirect URIs, JWKS URL, Valid Request URIs, CIBA Client Notification Endpoint URL, Use ID Token as a Detached Signature, User Info Signed Response Algorithm, ID Token Signature Algorithm, Access Token Signature Algorithm, Request Object Signature Algorithm, Signature Algorithm, Consent Required, Full Scope Allowed, OAuth 2.0 Mutual TLS Certificate Bound Access Tokens Enabled) need to be set up ($N_{SSPF} = 21$) to support FAPI1-advanced.

Using the policy-based method, 11 executors (Secure Session Enforce, Confidential Client Accept, Secure Client Authenticator, Secure Client URIs, Secure Request Object, Secure Response Type, Secure Signing Algorithm, Secure Signing Algorithm for Signed JWT, Consent Required, Full Scope Disabled, and Holder of Key Enforcer) need to be set up ($N_{SSPF} = 11$) to support FAPI1-advanced.

Using the client settings-based method, 22 client settings (21 of them are the same for FAPI1-advanced, and CIBA Backchannel Authentication Request Signature Algorithm) need to be set up properly ($N_{SSPF} = 22$) to support FAPI-CIBA.

Using the policy-based method, 14 executors (11 are the same for FAPI1-advanced, Secure CIBA Authentication Request Signing Algorithm, Secure CIBA Session Enforce, and Secure CIBA Signed Authentication Request) need to be set up ($N_{SSPF} = 14$) to support FAPI-CIBA.

Table 2 shows managerial costs for managing security profiles. In the example, “client” means the client settings-based method and “policy” means the policy-based method.

Example	N_{SSPF} (client)	N_{SSPF} (policy)	N (client)	N (policy)
Initializing Keycloak for FAPI1-baseline	13	6	123117	110123
Adding FAPI1-advanced	21	11	131117	11
Modifying FAPI1-advanced to FAPI-CIBA	22	14	22000	14

Tab. 2: Calculated managerial costs for security profiles based on the examples

[8] https://www.keycloak.org/docs/15.0/server_admin/index.html#_client_policies, accessed: 09/01/2022.

4.4 Discussion

Considering the theoretically derived managerial costs and their examples shown in Table 1 and Table 2, we can conclude that using the policy-based method is a good option from the perspective of managerial costs for the following usage situations while the client settings-based method is an acceptable option in other cases.

- When different security profiles need to be applied to requests from the same client
- When many clients need to be supported
- When security profiles need to be added or modified frequently

5 Contribution to the Keycloak Upstream Repository

To contribute the implementation of client policies and security profiles (FAPI1-baseline, FAPI1-advanced and FAPI-CIBA) to the Keycloak upstream repository, we needed to confirm that the implementation for security profiles complies with the FAPI specifications. To validate the implementation, we used OpenID Conformance Suite provided by OID-F[9] and confirmed that Keycloak code including the implementation could pass the conformance tests for both FAPI1-advanced and FAPI-CIBA. After this validation, the implementation was contributed to the Keycloak project, reviewed by Keycloak maintainers, and merged into Keycloak's main branch in Keycloak 15[10].

6 Related work

The policy-based method introduces the idea of Attribute Based Access Control (ABAC) to change the security profile applied to a client's request based on the content of the request. [EY05] established ABAC's formal model and proved that this model can represent the Mandatory Access Control (MAC) and Discretionary Access Control (DAC) model. Generally, ABAC's access control part can be achieved by the policy-based method. [PRR16] categorized the access control part into a logic-based policy and enumerated policy. [PRR16] proposed Label Based Access Control for enumerated policies and established its formal model. [XRR12] derived the minimum requirements of ABAC for representing MAC, DAC, and Role Based Access Control (RBAC) and formalized the minimum ABAC model satisfying these requirements.

OAuth 2.0 itself is so flexible that it has been used in a wide range of use cases for web-based authorization. [Fe17] applied OAuth 2.0 to accessing data collected by IoT devices. They modified the OAuth 2.0 protocol by considering a situation where IoT-device

[9] https://openid.net/certification/about-conformance-suite/, accessed: 06/07/2021.
[10] https://www.keycloak.org/docs/latest/release_notes/index.html#financial-grade-api-fapi-improvements-fapi-ciba-and-open-banking-brasil, accessed: 09/12/2021.

resources are limited and constrained. [Fe17] used RBAC to access such data. [ZRS19] applied OAuth 2.0 to access services provided by microservices in a container-orchestrated platform. [AAM19] applied and modified OAuth 2.0 for user centric identity management. All these studies use OAuth 2.0 to control access to resources provided by resource servers via APIs in OAuth 2.0's context by using ABAC. Unlike these studies, the proposed policy-based method in this paper utilizes the idea of ABAC for access control at OAuth 2.0's endpoints to determine whether an authorization server like Keycloak accepts a request from a client at these endpoints.

7 Conclusion

For Keycloak to support security profiles based on OAuth 2.0 and meet large-scale use cases such as open banking, the problem of increasing Keycloak managerial costs must be resolved. To resolve the problem, a policy-based method was proposed and implemented as client policies to enable flexible management of security profiles. Actual security profiles like FAPI1-baseline, FAPI1-advanced, and FAPI-CIBA have been implemented using client policies.

To confirm that the proposal resolves the problem, three scenarios of initializing, adding, and modifying security profiles were considered. Costs were calculated for these scenarios, and the results show that managerial costs of the Keycloak administrator decrease compared with the existing client settings-based method.

To validate the implementation of FAPI security profiles, we proved that it complies with the FAPI security-profile specifications by passing FAPI conformance tests. All implementations were contributed to Keycloak and merged into Keycloak's main branch. In the future, other security profiles like FAPI 2.0 [Fe21], the next major version of FAPI 1.0, will be implemented and contributed to Keycloak.

Acknowledgement

We would like to express our thanks to Stian Thorgersen (Keycloak project lead), Marek Posolda (Keycloak development team), FAPI-SIG's members, and the Keycloak community. They encouraged us to hold further discussions and provided us with advice about our implementation of client policies.

Bibliography

[AAM19] Abubakar-Sadiq, Shehu; António, Pinto; Manuel, E. Correia: Privacy Preservation and Mandate Representation in Identity Management Systems. In (Álvaro Rocha, et.al. Eds.): Proc. the 2019 14th Iberian Conference on Information Systems and

Technologies (CISTI), 2019.

[Co21] Consumer Data Right Security Profile, https://consumerdatastandardsaustralia.github.io/standards/#security-profile, accessed: 06/07/2021.

[EY05] Eric, Yuan; Jin, Tong: Attributed based access control (ABAC) for web services. In (Randall Bilof, ed.): Proceedings of the IEEE International Conference on Web Services (ICWS), vol. 856, pp. 561-569, 2005.

[Fe17] Federico, Fernández et.al.: A model to enable application-scoped access control as a service for IoT using OAuth 2.0. In (Noel, Crespi et.al. Eds.): Proc. the 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), pp.322-324, 2017.

[Fe21] Daniel, Fett: FAPI 2.0: A High-Security Profile for OAuth and OpenID Connect. In (Heiko Roßnagel, Christian H. Schunck, Sebastian Mödersheim Eds.): Proc. Open Identity Summit 2021, pp. 71-81, 2021.

[Fi21a] Financial-grade API Security Profile 1.0 - Part 1: Baseline, https://openid.net/specs/openid-financial-api-part-1-1_0-final.html, accessed: 06/07/2021.

[Fi21b] Financial-grade API Security Profile 1.0 - Part 2: Advanced, https://openid.net/specs/openid-financial-api-part-2-1_0.html, accessed: 06/07/2021.

[Ha21] Dick, Hardt et.al.: The OAuth 2.0 Authorization Framework, https://datatracker.ietf.org/doc/html/ rfc6749, accessed: 06/07/2021.

[Jo21] Michael, Jones et.al.: OAuth 2.0 Dynamic Client Registration Protocol, https://datatracker.ietf.org/doc/html/rfc7591, accessed: 06/12/2021.

[NK20] Yuichi, Nakamura; Kazufumi, Enomoto: Authentication and Authorization Based on OSS for Secure System Interoperation, https://www.hitachi.com/rev/archive/2020/r2020_05/05a04/index.html, accessed: 16/12/2021.

[Op21a] Open Banking Security Profiles, https://standards.openbanking.org.uk/security-profiles, accessed: 06/07/2021.

[Op21b] Open Banking Brasil Financial-grade API Security Profile 1.0 Implementers Draft 3, https://openbanking-brasil.github.io/specs-seguranca/open-banking-brasil-financial-api-1_ID3.html, accessed: 04/12/2021.

[PRR16] Prosunjit, Biswas; Ravi, Sandhu; Ram, Krishnan: Label-based access control: An ABAC model with enumerated authorization policy. In (Elisa Bertino, et.al. Eds.): Proc. the 2016 ACM International Workshop on Attribute Based Access Control, pp. 1-12, 2016.

[XRR12] Xin, Jin; Ram, Krishnan; Ravi, Sandhu: A unified attribute-based access control model covering DAC, MAC and RBAC. In (David Sadek, et.al. Eds.): DBSec'12: Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy, pp. 41-55, 2012.

[ZRS19] Zehan, Triartono; Ridha, Muldina Negara; Sussi: Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System. In (Hendri Irawan , et.al. Eds.): Proc. the 2019 6th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), pp. 259-263, 2019.