Implementation of Role-Based Access Control on OAuth 2.0 as Authentication and Authorization System

Zehan Triartono
School of Electrical Engineering
Telkom University
Bandung, Indonesia
zehantr@telkomuniversity.ac.id

Ridha Muldina Negara
School of Electrical Engineering
Telkom University
Bandung, Indonesia
ridhanegara@telkomuniversity.ac.id

Sussi
School of Electrical Engineering
Telkom University
Bandung, Indonesia
sussiss@telkomuniversity.ac.id

***Abstract*— As today's technology transition from monolithic towards microservices architecture, the authentication and authorization system also becomes a new concern because of the difference between monolithic and microservices pattern. Monolithic mostly uses role-based access control while microservices uses scope with OAuth 2.0. With this in mind, there is a need for a model that can integrate OAuth 2.0 with role-based access control. With role-based access control implemented on OAuth 2.0, we expect a simpler authorization process and a more secure authentication and authorization system for microservices backend architecture. This paper proposes a model to implement role-based access control on OAuth 2.0 using Laravel framework, we also test the performance of the system following by response time, data transferred and throughput. From the performance test, this approach has a good performance and can handle certain requests with simulated users even with limited resources.**

Keywords—Microservices, OAuth 2.0, Authentication, Authorization, Scope, Role-Based Access Control

I. INTRODUCTION AND BACKGROUND

As today's technology begins to make a transition from monolithic to microservices [1], many problems emerge as the latter face many change. One of them is the authentication and authorization system on microservices itself. The authentication and authorization system on microservices is different from monolithic [2]. In monolithic architecture, we only need one time authentication and authorization, this is because the application process in monolithic is not separated and becomes one process with the architecture itself. Meanwhile, in microservices each process and service can be handled differently according to the service itself, including handling authentication and authorization to verify each user for accessing the service [2].The microservices architecture itself is different from the monolithic architecture, where microservices is believed to be more dynamic than monolithic.

The microservices architecture is an application architecture made of a collection of services, each service can be independently deployed and loosely coupled [3]. These architectural patterns make microservices becomes harder to handle, especially on backend system where the backend organize relation between each service. Based on the microservices architecture, the need of a service that can manage the relation between each service is necessary. This service also needs to authenticate and authorize user that can be used for communication between each service.

The need for service that can control the relation between each service, authenticate and authorize every user so the user can access the service source, this service also need to manage whole microservices architecture on backend level. With this concept in mind, the need for a protocol that can implement authorization on scope level for service protection is also necessary. This protection is needed because each service will have different levels of permission for each relation. The protocol must also supple so we can modify it according to our system needs.

OAuth 2.0 protocol is usually used for authorization by the third party to get the permission for accessing source from a service [4]. OAuth 2.0 uses scope mechanism to limit the information access to source from the application [4]. This scope mechanism works when the application is requesting access token via authentication and then inform the user what scope is needed to access the source, although sometimes the application does not inform what scope is needed for accessing the source. Then, the scope will be placed on a generated token when the authentication is valid, from this point the source will only check the token and the scope embedded in it, to check if the user is allowed to get the necessary source information.

OAuth 2.0 also uses several grant types for several use case. There are five grant types that are commonly used [5]. The grant types commonly used are Client Credentials grant, Authorization Code grant, Device Code grant, Implicit grant and Password grant [4]. As for this implementation, we will implement the Password grant. This type of grant work is like a normal authentication where the user authenticate themselves to the source and get token to access their own data or other related data [5].

From the explanation above, we can assume that all requirements are already fulfilled. The missing one is how we can integrate RBAC (role-based access control) with OAuth 2.0 Password grant. With this concept in mind, we also need a framework that can implement OAuth 2.0 along with backend framework that can manage every service in microservices backend system.

Laravel is a free open-source framework that implements model-view-controller architecture [6]. Laravel also has a library called Laravel Passport, this library is fully implement OAuth 2.0 protocol to Laravel framework.

With Laravel framework and Laravel Passport it safe to assume that every requirement already fulfilled. The next step is to integrate between role-based access control and scope.

Role-based access control or RBAC is type of user assigning permission based on the user role within certain organization. This type of permission have many benefit,

some of them is this permission is easily audit every user based on their role, easy implementation to add or change role for user and third-party integration.

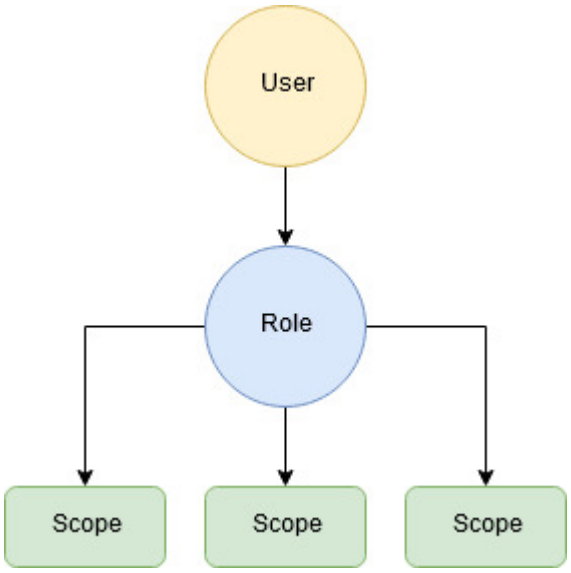

Fig. 1. Proposed Role-Based Access Control with Scope.

As shown in Fig 1. We proposed a role-based access control integrate with scope. Therefore, one user will have one role but with many scopes. This approach is expected to make the process for authorization in microservices architecture becomes simpler and at least has similar performance with scope-based only permission.

This approach is not new, many resources from internet already try this approach, but there is no actual implementation with performance measure on the implemented system. In this paper the author tries to measure the server performance, so we can get the actual result of role-based access control integrated with scope permission on microservices backend system.

The role-based access control integrated with scope is also needed for transition between monolithic and microservices because of many monolithic application implement role-based access control and want change into microservices architecture.

This paper implement OAuth 2.0 protocol on backend system using Laravel framework with role-based access control and measure the system performance from the API using response time, data transferred and throughput. Siege performance test tools is used to measure the performance of the system. Laravel framework is chosen because this framework already implements OAuth 2.0 protocol with Laravel Passport package and is also flexible as well as easily modified.

The performance is tested using 50, 100, 150, 200 and 250 simulated users using siege. Each simulated users will be recorded 30 times and the average result will be taken for response time, data transferred and throughput. The result shows that this approach can simplify the authentication and authorization process on microservices backend system especially on management side with the similar performance as scope-based permission.

II. RELATED WORKS

There are several approaches having been used to implement OAuth 2.0 with access control, such as access control with application scope for IoT used by Federico Fernandez [7]. Thus, the architectural concept enables access control on IoT. The architectural concept also makes it possible to implement access control by modifying the scope mechanism of OAuth 2.0. This paper presents the model that enables a service authorization with OAuth 2.0, this model can be used as a base to create another model to implement another access control, with this paper as a basis. The author tries to implement OAuth 2.0 with role-based access control, also in this paper it is just a proposal without performance testing, while the author implements and measures the performance of the system.

For other related works there is Se-Ra Oh focused on interoperable of existing OAuth 2.0 framework [8]. This paper proposes an additional authorization layer for OAuth 2.0, this paper can be used as a base to separate between authorization and service in microservices architecture.

As for the authentication principle with role-based access control, it has nearly the same principle as Nazmul Hossain OAuth-SSO framework [9]. Based on the paper, we can implement authentication for OAuth 2.0, so we will also use this to implement it on the system. This paper discuss about only authentication and security of OAuth 2.0 and propose a model to increase security of OAuth 2.0 framework and not measure the performance of the system itself.

Performance of OAuth 2.0 protocol is also presented by Marwah Darwish [10]. It evaluates OAuth 2.0 protocol on web server with limited resource. In this paper, it measures the VM memory load, VM load and VM storage load. This paper only measures the VM and does not measure the output performance. While in the author's paper, the author presents the output performance of the implemented system.

There is also a research by Dhiraj Ray implementing OAuth 2.0 with role-based authorization [11]. In this article the author implements OAuth 2.0 using role-based access control and integrates it with scope using spring framework. This article is interesting and already implements OAuth 2.0 with RBAC, but it does not have performance measurement test, also in this article the proposed framework does not explain the role and scope mapping clearly.

All the existing approach focusing on modified OAuth 2.0 and its mechanism. Thus, it makes the OAuth 2.0 can easily be implemented and modified with custom system. In this paper, the author implements the system on one of the private universities in Indonesia to replace the monolithic academic information system into microservices architecture.

III. IMPLEMENTATION OAUTH 2.0 WITH ROLE-BASED ACCESS CONTROL USING LARAVEL FRAMEWORK

This section describe the proposed OAuth 2.0 with role-based access control using Laravel framework for microservices backend system.

A. Proposed Access Control Flow

The new proposed OAuth 2.0 with role-based access control is shown in Fig. 2.

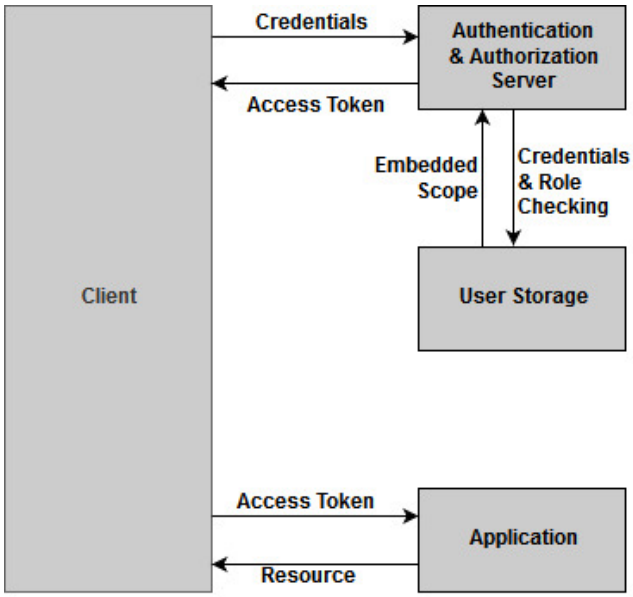

Fig. 2. Proposed OAuth 2.0 Authorization Flow.

Firstly in the proposed flow in Fig. 2, the client or third party application must authenticate every user who wants to access application resource by providing their credentials to the authentication and authorization server. To do this, client application will call API from the authentication and authorization server so the user can authenticate themselves from the client application frontend.

After the client provides credentials to the authentication and authorization server, the server will check if the credentials are valid or not by checking it to the user storage server. If the user is valid and registered at the user storage, it will then check the role of the user and get all scope for the user scope. The scope is then implanted to the user scope and then is returned to authentication and authorization server to be generated into access token.

After the authentication and authorization server generates and returns the access token to the client, the client can use it to access resource on application. The application itself also check the embedded scope inside the access token role to check the access control granted for the user.

These approaches are expected to simplify the authorization process and make every scope becomes exclusive for their own resource access.

In addition, this approach has great benefits where every user can have more than one role but have different access control to the source. For example, an administrator having an administrator role as well as user role and employee role. Each role normally will have different access control, but sometimes one role will have higher access control than other role. With this approach, the access control can be divided into scope and then embedded into role, this makes it not necessary to embed the scope into the user, which is more complicated and hard to do.

B. Implementation OAuth 2.0 with Role-Based Access Control using Laravel

As mentioned in first section, we will implement the proposed system using Laravel framework and Laravel Passport library.

This section will cover how to implement role-based access control into Laravel Passport library on Laravel framework.

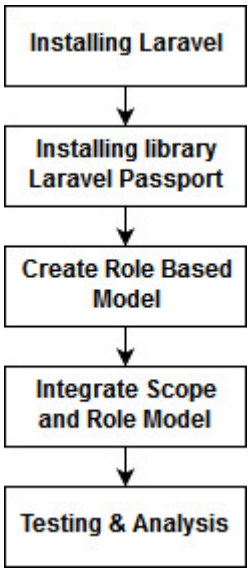

Fig. 3. Implementation Step of OAuth 2.0 with RBAC.

From Fig. 3, the first thing to start is installing Laravel framework, because its bundled with Laravel Passport, it safe to also installing the Laravel Passport.

The next step after the Laravel framework ready is create role-based model on database and integrate it with scope, because Laravel Passport is come default as library, its better if our model the one adapted to the library. So the process becomes simpler.

As for the role-based model itself, we try to make it as simpler as possible as shown in Fig. 4.

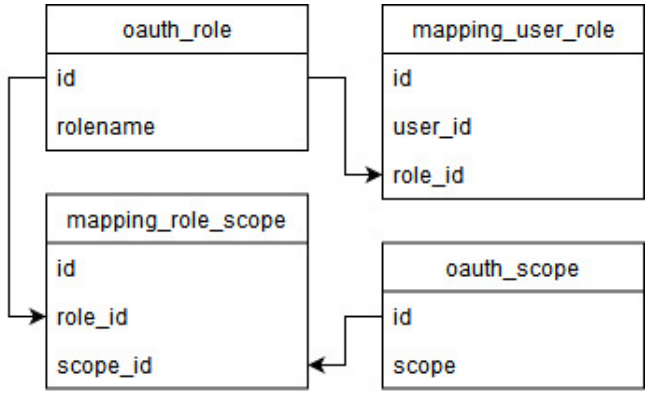

Fig. 4. Proposed Role Model for RBAC.

As shown on Fig. 4, every scope will be mapped to the role where relation between role and scope is one has many. It means one role will have many scope. The same relation is implement to the user and role as well, where every user will have more than one role.

After the implementation, the last step is measuring performance of this proposed role-based access control on OAuth 2.0.

C. OAuth 2.0 with Role-Based Access Control Performance

This section will cover the performance testing from the proposed system to see if the proposed system is have good

performance as the original system or at least nearly identical.

For performance testing, we will test it using siege testing tools. The parameters that will measured is response time, data transferred and throughput. The performance test will test a single post login REST API from the system and is measured by siege with simulated users.

Siege testing tools will get several measurement parameter from accessing the REST API. For this case author only measure response time, data transferred and throughput. Each parameter is tested 30 times to get the average and desired result.

The response time performance of the system is shown in Fig. 5 below.

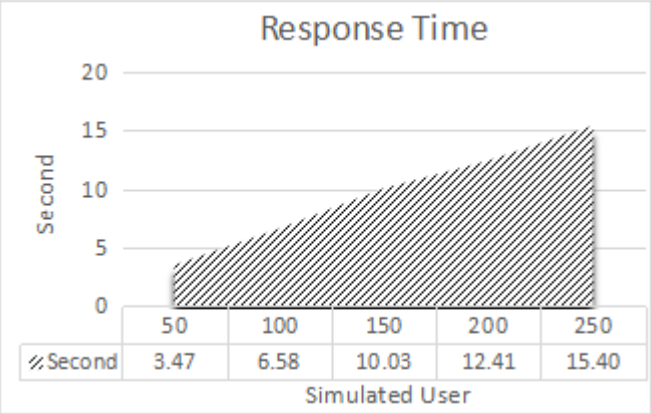

Fig. 5. Response Time Performance OAuth 2.0 with RBAC.

As shown in Fig. 5, the response time of the performance test we get average of 9.58 second. This average result obtained from the average of sum 5 simulation test.

If we take the average time and the average users, where we will get 9.58 second for average response time and 150 for average simulated users, we will get each transaction response time is around 0.063 second. This result is considered very good, because it needs less than 1 second to complete one transaction for one user.

The data transferred performance of the system is shown in Fig. 6 below.

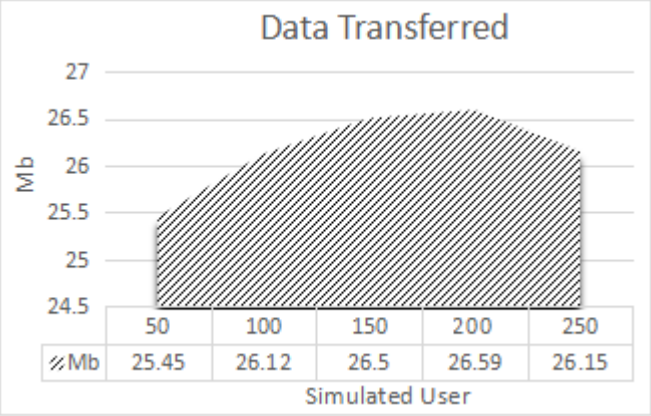

Fig. 6. Data Transferred Performance OAuth 2.0 with RBAC.

As shown in Fig. 6, performance of the data transferred parameter test we get average of data transferred is 26.16 Mb. This result is obtained from average sum of 5 simulation tests.

If we also take the average data transferred which is 26.16 Mb and the average 150 users we will get each transaction data transferred or data transferred to one user is around 0.17 Mb per user.

In this performance test, we also can see that the system performance is slowing down when it hit 250 user where the curve is going down and there are some failed connections that inflict data loss. We guess that because of the limited source where we implement the system.

The throughput performance of the system is shown in Fig. 7 below.

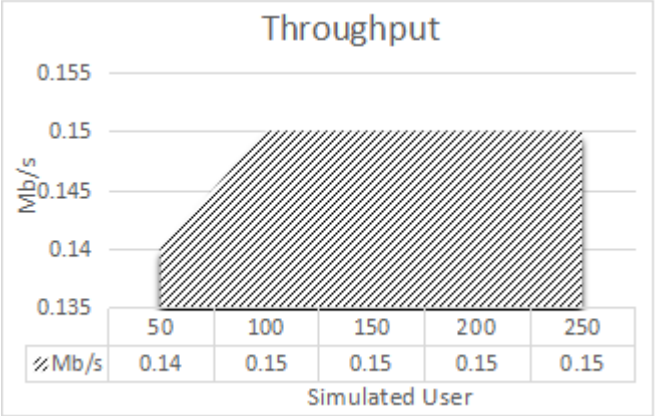

Fig. 7. Throughput Performance OAuth 2.0 with RBAC.

As shown in Fig. 7. the throughput performance test is quite stable and reliable. This performance is considering good for the system.

From the test performance shown by Fig. 5, Fig. 6 and Fig. 7. The performance of the system is quite well and reliable despite slow down performance and data loss starting from the 250 simulated users on data transferred test. This slow down performance is caused by connection failed that inflict data loss, we assume it is because the limited resource of the implemented system.

IV. CONCLUSION

This paper proposed the role-based access control approach for OAuth 2.0 protocol on microservices backend system. The benefit of this system is to simplify authorization process on backend side of microservices architecture.

The OAuth 2.0 with role-based access control approach in the author paper is implemented using Laravel Framework and Laravel Passport library using author proposed model. This implementation is also used in one private university in Indonesia to transition the information academic system from monolithic to microservices architecture.

Also from the performance testing of the implemented OAuth 2.0 using role-based access control we can conclude that the system is reliable despite the slowing down performance when the system accessed by certain simulated users.

In the near future, we plan to enhance this approach and make some kind of package so this approach can be easily implemented like any other library.

[1] Kharenko, A., "Monolith vs Microservices Architectures", https://articles.microservices.com/monolithic-vs-microservices-architecture- 5c4848858f59, accessed at May 2019.

[2] Ayoub, M., "Microservices Authentication and Authorization Solutions", https://medium.com/tech-tajawal/microservice-authentication-and-authorization-solutions-e0e5e74b248a, accessed at July 2019.

[3] Microservices, https://microservices.io, accessed at July 2019.

[4] D. Hardt, Ed, "The OAuth 2.0 Authorization Framework", IETF, 2012.

[5] Bilbie, A., "A Guide To OAuth 2.0 Grants", https://alexbilbie.com/guide-to-oauth-2-grants/, accessed at July 2019.

[6] Laravel, https://laravel.com, accessed at July 2019.

[7] Federico Fernandez, Alvaro Alonso, Lourdes Marco, Joaquin Salvachua, "A Model to Enable Application-scoped Access Control as a Service for IoT Using OAuth 2.0" in 20th Conference on Innovations in CLouds, Internet and Networks (ICIN). 2015.

[8] Oh, Se-Ra, Kim, Young-Gab, "Interoperable OAuth 2.0 Framework" in 2019 International Conference on Platform Technology and Service (PlatCon).

[9] Hossain, N., Hossain, M. A., Hossain, M. Z., Sohag, M. H. I., Rahman, S., "OAuth-SSo: A Framework to Secure the OAuth-Based SSO Service for Packaged Web Applications" 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE).

[10] Darwish, M., Ouda, A, "Evaluation of an OAuth 2.0 protocol implementation for web server applications" 2015 International Conference and Workshop on Computing and Communication (IEMCON).

[11] Dhiraj, Ray., "Spring Boot OAUTH2 Role-Based", https://www.devglan.com/spring-security/spring-oauth2-role-based-authorization, 27 December 2018, accessed at 12 August 2019.