OPEN ACCESS
J. OF ICT, VOL. 5, NO. 2, PP.186-195, DES 2023

ISSN: 26861089 (ONLINE)
DOI: 10.52661

Journal of Informatics and Communications Technology (JICT)

IMPLEMENTATION OF OAUTH 2.0 BASED ON LARAVEL FRAMEWORK IN A CASE STUDY OF CLIENT INFORMATION MANAGEMENT SYSTEM

Arthur Oliviana Zabka, Asep Id Hadiana, Herdi Ashaury
Informatika, Universitas Jenderal Achmad Yani Cimahi
Jl. Terusan Jend. Sudirman, Cibeber, Kec. Cimahi Sel., Kota Cimahi, Jawa Barat 40531
arthur.oliviana@student.unjani.ac.id

Received on 03-11-2023, revised on 23-12-2023, accepted on 29-12-12023

Abstract
The swift growth of the internet and its utilization by businesses for operational purposes, such as developing information systems and utilizing cloud-based data storage, has been remarkable. The discussed client data recording system is designed to facilitate the recording of wedding photography bookings, easing the workload for employees. However, the rapid expansion of the internet has also introduced security concerns, particularly regarding unauthorized access due to weak website authorization and authentication. Consequently, ensuring and effectively managing access rights to information systems becomes crucial. This study aims to implement secure website login authorization using the OAuth 2.0 method with Laravel Passport in the client data recording information system. Post-authentication, the authorization in the context of OAuth2, used within Laravel Passport, provides users with access tokens to reach the primary interface. This process involves an API that both furnishes and safeguards the intended resources. Upon authentication and receipt of a valid access token from the OAuth2 system, users can utilize the token to access the API. The research outcomes enhance the security of information system access rights, aiming to reduce unauthorized breaches in websites storing vital data, thus ensuring the safety and protection of stored client data. Testing results using SQL Injection yielded 4418 messages sent and 2209 task IDs, with a current fuzzer count of 0, signifying that the system remained secure and impervious to SQL Injection attacks.

Keywords: *authorization, authentication, website, OAuth2, API, access token, Laravel Passport, information system*.

I. INTRODUCTION

The rapid development of the internet has made obtaining and managing data very convenient. Currently, companies utilize the internet for operational purposes such as establishing information systems, storing cloud-based data, among others. The client information management system is a recording system for booking wedding photography, encompassing client data entry, management of work orders, invoicing, and more, all within a single website. This setup streamlines employee tasks in efficiently recording the booking process. However, due to the internet's accessibility to anyone, anytime, and anywhere, the risk of cybercrimes such as hacking into login access and other breaches exists.[1][2] This might be due to weaknesses in the authentication and authorization processes on the website. Authorization, also known as 'Authorization,' is a part of setting user access rights within a service, microservice, or API..[3], [4] The ease with which users other than administrators and employees gain access rights to enter the interface of the information system is a concern. Consequently, access rights to the information system must be protected and there should be well-managed configurations in place.

Laravel is a PHP framework built on the model-view-controller (MVC) architectural pattern used for web development. This framework was created by Taylor Otwell in April 2011. Laravel Passport is a package developed by Laravel to provide API authentication using an OAuth2 server. It enables developers to easily add API authentication to Laravel applications, allowing users to authenticate using API tokens.[5], [6][7]

OAuth2 is employed in authorization security to regulate access to resources held by the server. The process begins with the user clicking the "Login" link on the application they wish to access, then being directed to the OAuth2 server for authentication and validation of credentials. If the credentials are valid, the OAuth2 server issues an authorization code that is sent back to the client application via a redirect URL. The client application then uses this authorization code to obtain an access token and access the necessary resources from the OAuth2 server. The OAuth2 server verifies the access token and validates its authenticity, controlling the access token's duration and revoking it if necessary. This enhances application security, ensuring reliability in managing access to the server's resources.[2], [8]

Besides OAuth2.0, there are other methods used in securing authorization, such as Auth Token[9], OAuth[10], JSON Web Token (JWT)[11]. The aforementioned methods each have their specific advantages in their respective domains. For instance, Auth Token leans more towards server-to-server communication by generating secret codes used by users for authentication. Meanwhile, OAuth, JWT, and Spring Security are more oriented towards the interaction between users and servers[10]. Therefore, in this research, it is highly suitable to use OAuth due to the client information management system's user-server interactions. Comparing OAuth 1.0 and OAuth 2.0, it's evident that 2.0 is considered the latest version of OAuth. OAuth 1.0 only caters to web-based application authorization and cannot be utilized for mobile or native applications. Its process becomes complicated due to the need for signing each request. Simplifying the authorization standard development allows accommodation across various scenarios[12]. Another reason supporting my use of OAuth 2.0 is its ability to protect microservice-based information systems from CSRF (Cross-Site Request Forgery) and XSS (Cross-Site Scripting) attacks[13].

II. LITERATURE REVIEW

In previous research, it was found that using Auth Token with REST API fulfilled the security requirements for integrated information systems and effectively addressed security vulnerabilities associated with session-based authentication systems and cookies[9]. In previous OAuth research, it has been demonstrated that the OAuth protocol can mitigate potential attacks, such as MITM (Man-In-The-Middle) attacks, hijack attacks, and damaging counterattacks, by employing noncedan attributes and timestamps implemented by the OAuth resource provider[10]. Previous research delved into the design and implementation of Spring Security. However, these studies were conducted separately without any confirming research on the effectiveness of the Spring Framework (SF), Spring Security Framework (SSF), and OAuth 2.0 (OAuth2) when applied to authenticate and authorize endpoints within a Microservice Architecture API[14].
The subsequent research involves securing Oauth2.0 using the Spring Security Framework. The findings indicate that the utilization of these two elements is highly effective, having been tested against various types of attacks such as CSRF, XSS, and Brute Force attacks[13].
Another research project extended the SSO technology to the faculty system and curriculum information system by leveraging Laravel Socialite for integration purposes. The methodology employed in this study was prototyping. The evaluation was conducted using a black box approach, concluding that the authentication system functions properly.[5] The research conducted by Afwan Ghiffari and Purwono Hendra focused on SSO developed using the OAuth2.0 protocol within Laravel Socialite. They also utilized REST architecture involving Backend and Frontend in the Moca and KRS systems, which ran smoothly, simplifying the user access process to the system.[6] There is a previous study on SSO. Explore SSO using the OAuth protocol from Facebook. This study implements single sign-on using the OAuth protocol and leverages Facebook login API and PHP Native. [15]

Based on the arising issues, this research introduces an additional feature, which involves integrating the OAuth2.0 method into Laravel Passport. This allows for a login process after successful authentication. In the context of OAuth2 used by Laravel Passport, upon user authentication, they will receive an access token granting them permission to access the main interface or dashboard of the client data management system protected by an API.

III. RESEARCH METHOD

A. *Single Sign On*

Single Sign-On (SSO) is a process where a user logs into one application and is then automatically logged into other applications, regardless of the platform, technology, or domain the user is using. As its feature name suggests (single sign-on), users only need to log in once.[3], [16]

For instance, if you sign in to a Google service such as Gmail, you are automatically authenticated to YouTube, AdSense, Google Analytics, and other Google applications. Similarly, if you log out of Gmail or any other Google application, you will be automatically logged out of all these applications, which is referred to as single logout.[16]

B. Application Programming Interface (API)

Application Programming Interface (API) refers to the concept of application interface functions that enable access and use of applications by other parties without altering the core code structure or system's database. Its purpose is to facilitate communication between systems, even if they operate on different platforms. A Web Service is a type of API that provides user access to retrieve data. Operating on the Representational State Transfer (ReST) architecture atop the Hypertext Transfer Protocol (HTTP), this API delivers information in the form of Javascript Object Notation (JSON) files to users when accessed.[17]

C. Laravel Passport

Laravel is an open-source PHP framework following the MVC (Model-View-Controller) pattern, utilized for web application development. It was first created on February 22, 2012, by Taylor Otwell. Laravel boasts security features such as Laravel Passport and Laravel Socialite. These features are implemented through a module known as League OAuth2 Server, developed by Alex Bilbie in his Github repository. Laravel Passport provides a comprehensive OAuth2 server implementation for Laravel applications.[5], [6]

Laravel Passport, an offering developed by the Laravel team, aims to simplify API authentication by utilizing the OAuth2 server. This package's development facilitates seamless integration of API authentication into Laravel applications, allowing users to authenticate using API tokens. The primary feature introduced in Laravel version 5.3 is the inclusion of Laravel Passport, which is an implementation of the module known as League OAuth2 Server, developed by Alex Bilbie on his Github account. With Laravel Passport, a complete implementation of the OAuth2 Server is made available for Laravel applications.[7], [18]
Here are the main features of Laravel Passport:

1. OAuth2 Server: Passport offers a full implementation of an OAuth2 server, enabling applications to handle authentication and authorization using API tokens.
2. Token-based Authentication: It utilizes tokens for user authentication, whether through personal access tokens or via OAuth2.
3. Token System: Passport manages the creation and administration of API tokens, including generating access tokens, token storage, refresh tokens, and more.

D. Open Authorization (OAuth 2.0)

Open Authorization (OAuth) is an authorization method that involves a third party in its operation, granting access rights to users to access desired or provided data.[1] With secure authority ensured due to standardized and straightforward methods for web, desktop, and mobile applications. OAuth accesses this crucial data and restricts it with designated access through the HTTP protocol[19][20]. And specifically designed to address the classical authentication issues in client-server systems.[8] And OAuth serves as the resource server, which means it has the authority to manage all existing credentials. The resource server component also functions as an authorization server, authorizing client requests to microservices by exchanging authorization codes using tokens[21].

E. Authorization

Authorization is a set of rules that guide users in determining what activities or actions are permitted and which are not within a system.[3][22][23] It is a component of the 3A (Authentication, Authorization, and Accounting).[24] Within authorization, there are two stages: policy management and access control.

a) Policy management

involves defining what needs protection and allowing access only to specific users

b) Access control.

is a system that restricts access closest to the resource being traversed[23][25][4].

Research methodology is an approach applied to complete conducted research. It involves several stages such as problem identification, literature review, needs analysis, design, prototype development, testing, and implementation. For a clearer understanding, please refer to Figure 1.

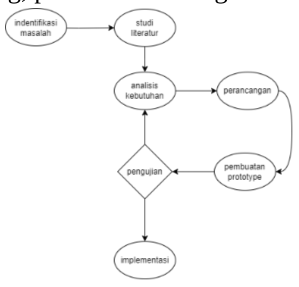


Figure 1. Research methodology

A. Problem Identification

At this stage, the problem identification is carried out based on the requirements needed in the Single Sign-On (SSO). There are two main users in this system: supervisors and employees. Supervisors hold the role of administrators in the client data registration application. The problem identification focuses on the login page that utilizes the OAuth2.0 protocol, based on the development foundation of Laravel Passport, and connecting the main interface with the CodeIgniter API framework. The goal of this research is to enhance the login interface through the implementation of OAuth2.0 based on Laravel Passport.

B. Literature Review

This stage is conducted to find necessary information sources to delve deeper into concepts related to SSO, Laravel Passport, OAuth2.0, API, and authorization, as outlined in the introduction regarding relevant previous research.

C. Needs Analysis

This stage involves analysis conducted after complete data collection. It encompasses supporting data and problem-solving approaches that will be applied in building the system. Some analyses performed include:

- Web Service

  SSO uses the Client-Server model where the Client is a program that connects to the Server to send requests. The Server, as another program, receives connections and provides responses to these requests. This system is based on an API architecture that facilitates the implementation of web services to manage state transitions.
- Laravel Passport

  Laravel Passport is a feature of Laravel that simplifies API application development. It allows for the creation of OAuth2 tokens, streamlining authentication and authorization in Laravel API applications. Passport offers a hassle-free solution for authenticating API applications, ensuring data access only for legitimate users.
- OAuth2.0

  OAuth 2.0 is a popular SSO protocol due to its ease of implementation and widespread support from ASP. This protocol enables third-party applications to access resources on the Server without User credentials. In OAuth 2.0, there are several key roles: Resource Owner, Client, Authorization Server, and Resource Server. Implementing SSO using OAuth 2.0 requires one application as the OAuth Server and another application as the OAuth Client. The OAuth Server requests user credentials and provides OAuth tokens for Client OAuth data access. This allows multi-application access without repeated authentication. The *roles of OAuth are as follows:*

  A) Client: A third-party application accessing data from the Resource Server.
  B) Resource Owner: Users who own resources, accessing data through third-party applications.
  C) Resource Server: Protects and provides data access through the Web API, validates tokens from the Client.
  D) Authorization Server: Component that provides login pages, information approval, and token management.

D. Design

This stage involves designing the SSO based on the literature review and needs analysis in the client data registration system.

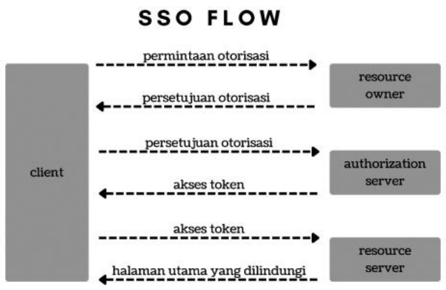
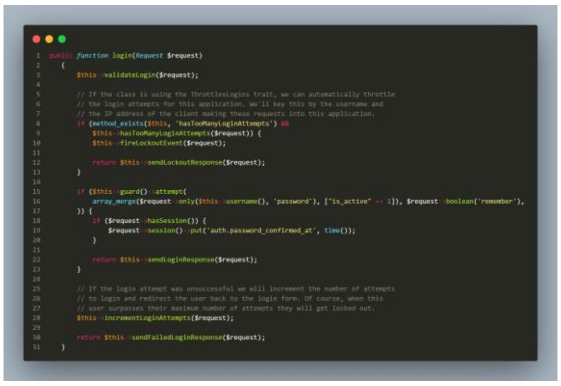


Figure 2. SSO Flow

The steps are as follows:

a) The user accesses the client.
b) The client redirects the user to the Authorization Server for the login process.

c) The user enters credential information to log in.
d) The Authorization Server verifies the user's credentials and requests an authorization code to generate an access token.
e) The user grants permission, and the Authorization Server returns the authorization code to the client.
f) The client uses the authorization code to request a token from the resource server to verify its validity.
g) The resource server stores the token and uses it to access services or the client (main page)..

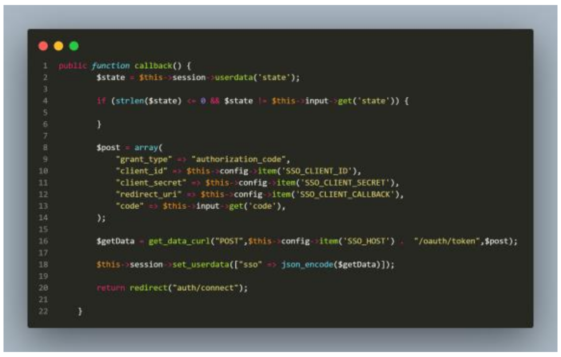


```php
public function login(Request $request)
    {
        $this->validateLogin($request);

        // If the class is using the ThrottlesLogins trait, we can automatically throttle
        // the login attempts for this application. We'll key this by the username and
        // the IP address of the client making these requests into this application.
        if (method_exists($this, 'hasTooManyLoginAttempts') &&
            $this->hasTooManyLoginAttempts($request)) {
            $this->fireLockoutEvent($request);

            return $this->sendLockoutResponse($request);
        }

        if ($this->guard()->attempt(
            array_merge($request->only($this->username(), 'password'), ["is_active" => 1]), $request->boolean('remember'),
        )) {
            if ($request->hasSession()) {
                $request->session()->put('auth.password_confirmed_at', time());
            }

            return $this->sendLoginResponse($request);
        }

        // If the login attempt was unsuccessful we will increment the number of attempts
        // to login and redirect the user back to the login form. Of course, when this
        // user surpasses their maximum number of attempts they will get locked out.
        $this->incrementLoginAttempts($request);

        return $this->sendFailedLoginResponse($request);
    }
```

Figure 3. Auth login

The image above, labeled as Figure 3, depicts the method that handles login requests. The sequence of events here is as follows:

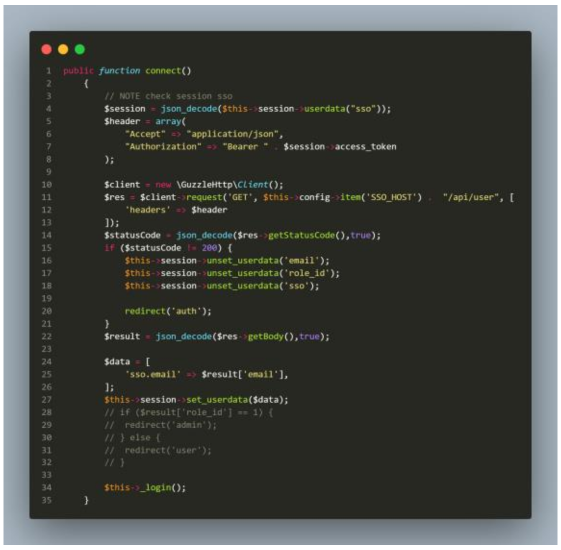
- Validation of the login request using the validateLogin() method.
- Checking for excessive login attempts using hasTooManyLoginAttempts() (if the ThrottlesLogins Trait is used).
- If the login attempt is successful (attempt()), the user will be authenticated and directed to the specified page.
- In case of a failed login attempt, the login attempt count will be incremented, and the user will be redirected back to the login form..



```php
public function callback() {
        $state = $this->session->userdata('state');

        if (strlen($state) <= 0 && $state != $this->input->get('state')) {

        }

        $post = array(
            "grant_type" => "authorization_code",
            "client_id" => $this->config->item('SSO_CLIENT_ID'),
            "client_secret" => $this->config->item('SSO_CLIENT_SECRET'),
            "redirect_uri" => $this->config->item('SSO_CLIENT_CALLBACK'),
            "code" => $this->input->get('code'),
        );

        $getData = get_data_curl("POST",$this->config->item('SSO_HOST') . "/oauth/token",$post);

        $this->session->set_userdata(["sso" => json_encode($getData)]);

        return redirect("auth/connect");

    }
```

Figure 4. Token exchange

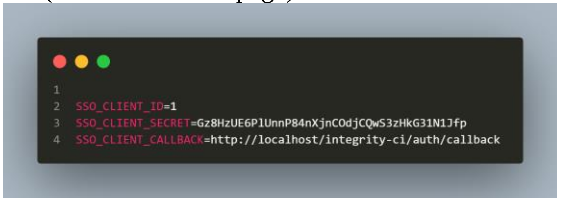
This function receives a callback from the Laravel SSO server after the successful authentication of the user. In this context, an HTTP POST request is used to obtain a token from the SSO server..



```
public function connect()
    {
        // NOTE check session sso
        $session = json_decode($this->session->userdata("sso"));
        $header = array(
            "Accept" => "application/json",
            "Authorization" => "Bearer " . $session->access_token
        );

        $client = new \GuzzleHttp\Client();
        $res = $client->request('GET', $this->config->item('SSO_HOST') .  "/api/user", [
            'headers' => $header
        ]);
        $statusCode = json_decode($res->getStatusCode(),true);
        if ($statusCode != 200) {
            $this->session->unset_userdata('email');
            $this->session->unset_userdata('role_id');
            $this->session->unset_userdata('sso');

            redirect('auth');
        }
        $result = json_decode($res->getBody(),true);

        $data = [
            'sso.email' => $result['email'],
        ];
        $this->session->set_userdata($data);
        // if ($result['role_id'] == 1) {
        //  redirect('admin');
        // } else {
        //  redirect('user');
        // }

        $this->_login();
    }
```

Figure 5. Token Authorization

Upon receiving a response from the SSO server, this function initiates further API requests to obtain user data from the SSO server using the access token obtained after a successful authentication process. With this token, the application can make requests to the SSO server to acquire additional information about the user or access resources protected by that server (such as the main page).



```

SSO_CLIENT_ID=1
SSO_CLIENT_SECRET=Gz8HzUE6PlUnnP84nXjnCOdjCQwS3zHkG31N1Jfp
SSO_CLIENT_CALLBACK=http://localhost/integrity-ci/auth/callback
```

Picture 6. Creating a client

The image above demonstrates the process of creating a client, where the user inputs the callback redirect location and generates the client ID, SSO client. This command will be executed when the author enters the command in the terminal using `php artisan passport:client`.

IV. RESULTS AND DISCUSSION

A. *Main page login SSO*

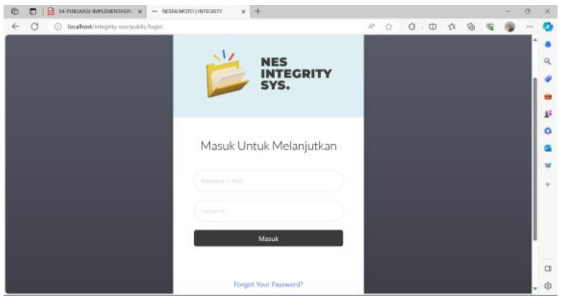
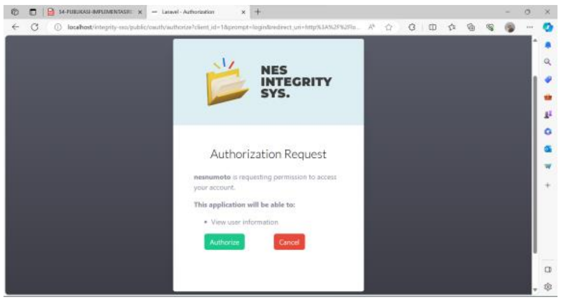


Figure 7. Main login interface

This is the main interface displayed when users first access the client data management application.

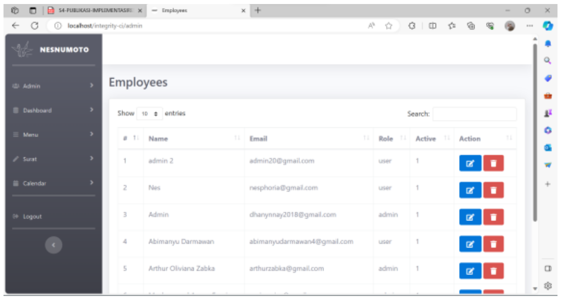
B. *Authorization permission interface*


NES INTEGRITY SYS.
Authorization Request
nesnumoto is requesting permission to access your account.
This application will be able to:
• View user information
Authorize
Cancel

Picture 8. Authorization Permission

In the image above, after the user enters their email and password, the system will verify the credentials to check if the email is active and correct within the database. If the credentials are accurate, the authentication will display the authorization permission view as shown above.

C. *Dashboard Display*

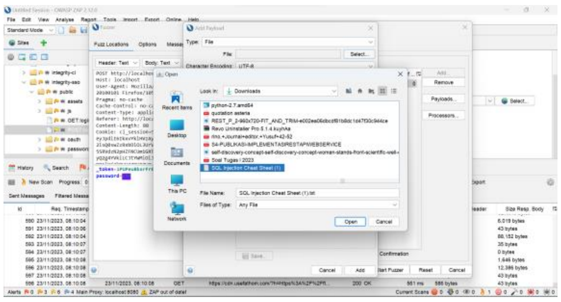


#	Name	Email	Role	Active	Action
1	admin 2	admin20@gmail.com	user	1	
2	Nes	nesphoria@gmail.com	user	1	
3	Admin	dhanynnay2018@gmail.com	admin	1	
4	Abimanyu Darmawan	abimanyudarmawan4@gmail.com	user	1	
5	Arthur Oliviana Zabka	arthurzabka@gmail.com	admin	1	

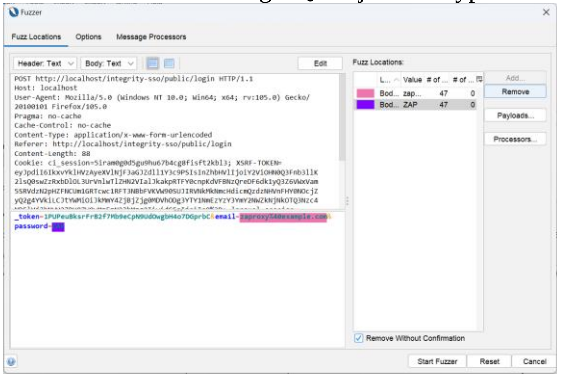
Figure 9. Dashboard display

If the authorization process runs smoothly and the token will be automatically provided by Laravel, users will access the main display or dashboard according to their stored role, whether it's an admin or an employee.

C. *SQL Injection Testing*



Gambar 10. inserting SQL Injection Bypass

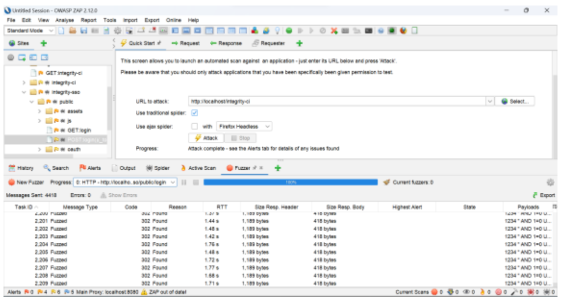


```
POST http://localhost/integrity-sso/public/login HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/
20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Content-Type: application/x-www-form-urlencoded
Referer: http://localhost/integrity-sso/public/login
Content-Length: 88
Cookie: ci_session=5iram0g0d5gu9hu67b4cg8fisft2kbl3; XSRF-TOKEN=
eyJpdiI6IkxvYklHVzAyeXVlNjF3aGJZdll1Y3c9PSIsInZhbHVlIjoiY2V1OHN0Q3Fnb3llK
2lsQ0swZzRxbDlOL3UrVnlwTlZHN2VIalJkakpRTFY0cnpKdVFBNzQreDF6dk1yQ3Z6VWxVam
5SRVdzN2pHZFNCUm1GRTcwc1RFT3NBbFVKVW90SUJIRVNkMkNmcHdicmQzdzNHVnFHY0NOcjZ
yQ2g4YVkiLCJtYWMiOiJkMmY4ZjBjZjg0MDVhODg3YTY1NmEzYzY3YmY2NWZkNjNkOTQ3Nzc4

_token=1PUPeuBksrFrB2f7Mb9eCpN9UdOwgbH4o7DGprbC&email=zaproxy%40example.com&
password=ZAP
```

L...	Value	# of ...	# of ...
🟪 Bod...	zap...	47	0
🟪 Bod...	ZAP	47	0

Figure 11. Inserting the Same Bypass

And the two images above represent one of the security testing methods. The author conducted SQL Injection testing using the OWASP ZAP application, in which I inserted the directory link to the login page and configured it to test each email and password by inputting SQL commands one by one to determine if a breach would occur or not.



This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://localhost/integrity-ci
Use traditional spider: ✓
Use ajax spider: with Firefox Headless
Progress: Attack complete - see the Alerts tab for details of any issues found

Progress: 0: HTTP - http://localho..so/public/login 100% Current fuzzers: 0
Messages Sent: 4418 Errors: 0 Show Errors

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
2,201	Fuzzed	302	Found	1.44 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,202	Fuzzed	302	Found	1.48 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,203	Fuzzed	302	Found	1.42 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,204	Fuzzed	302	Found	1.76 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,205	Fuzzed	302	Found	1.48 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,206	Fuzzed	302	Found	1.72 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,207	Fuzzed	302	Found	1.77 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,208	Fuzzed	302	Found	1.68 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...
2,209	Fuzzed	302	Found	1.71 s	1,189 bytes	418 bytes			1234 " AND 1=0 U...

Alerts 0 4 6 5 Main Proxy: localhost:8080 ZAP out of date! Current Scans 0 0 0 0 0 0 0 0

Figure 12. SQL Injection Test Results

As seen in the image above, the test result using SQL Injection shows 4418 messages sent and 2209 task IDs with the current number of fuzzers at 0. In other words, it is safe from SQL Injection breaches.

V. Conclusion

The conclusion of this research demonstrates that implementing SSO with OAuth2.0 based on the Laravel framework has proven to be secure when tested for SQL Injection, yielding specific results: a total of 4418 messages sent and 2209 task IDs with the current number of fuzzers at 0. The author suggests that future evaluations should include multiple types of testing, as this examination solely focused on SQL Injection. Additionally, to enhance the usage of SSO, it's recommended to expand beyond connecting only one system. Implementing the SSO to link more than one or two systems within a single login would truly exemplify the essence of SSO..

REFERENCES

[1] P. Sistem and P. Parkir, "IMPLEMENTASI RESTFUL WEB SERVICES DENGAN OTORISASI OAUTH 2 . 0," no. April, 2019, doi: 10.24176/simet.v10i1.3026.

[2] R. Kurniawan, "Perancangan dan Implementasi Sistem Otentikasi OAuth 2 . 0 dan PKCE Berbasis Extreme Programming (XP) Universitas Mercubuana Yogyakarta , Indonesia Design and Implementation of Authentication System OAuth 2 . 0 and PKCE Based on Extreme Programming (XP," vol. 2, no. 2, pp. 81–91, 2022.

[3] M. P. Oauth, "Jurnal JARKOM Vol . 5 No . 2 Desember 2017 E- ISSN : 2338-6304 PERANCANGAN DAN IMPLEMENTASI SSO (SINGLE SIGN ON) Jurnal JARKOM Vol . 5 No . 2 Desember 2017 E-ISSN : 2338-6304," vol. 5, no. 2, pp. 102–108, 2017.

[4] S. Syofian and R. I. Setya S, "IMPLEMENTASI MANAGEMENT AKSES USER UNTUK ROUTER CISCO MENGGUNAKAN METODE AAA (AUTHENTICATION, AUTHORIZATION, ACCOUNTING) Studi Kasus PT. PROXIS SAHABAT INDONESIA," *Jurnal Sains & Teknologi* , vol. 8, no. 1, pp. 33–40, 2018.

[5] H. Ajie, M. Insan Rizky, and M. F. Duskarnaen, "Pengembangan Teknologi Single Sign On Pada Sistem Informasi Dosen dan Sistem Informasi Kurikulum di Universitas Negeri Jakarta," *PINTER : Jurnal Pendidikan Teknik Informatika dan Komputer*, vol. 3, no. 1, pp. 32–37, Jun. 2019, doi: 10.21009/pinter.3.1.6.

[6] A. Ghiffari and P. Hendradi, "Implementasi Single sign on (SSO) Menggunakan Representational State Transfer (REST) dan Open Authorization (OAuth 2.0) (Studi kasus: Universitas Muhammadiyah Magelang)."

[7] H. Ajie, M. Insan Rizky, and M. F. Duskarnaen, "Pengembangan Teknologi Single Sign On Pada Sistem Informasi Dosen dan Sistem Informasi Kurikulum di Universitas Negeri Jakarta," *PINTER : Jurnal Pendidikan Teknik Informatika dan Komputer*, vol. 3, no. 1, pp. 32–37, Jun. 2019, doi: 10.21009/pinter.3.1.6.

[8] Y. Fatman and R. Octaviawati, "Implementasi Metode Open Authorization (OAuth2) Untuk Pengelolaan Data Dosen di Universitas Islam Nusantara," vol. 2, no. 1, pp. 10–18, 2020.

[9] I. G. Anugrah and M. A. R. I. Fakhruddin, "Development Authentication and Authorization Systems of Multi Information Systems Based REst API and Auth Token," *Innovation Research Journal*, vol. 1, no. 2, p. 127, 2020, doi: 10.30587/innovation.v1i2.1927.

[10] K. Saputra and K. Farhan, "Implementasi Protokol OAuth 1 . 0 Sebagai Autentikasi pada Aplikasi SMS Blast Berbasis Android," *Journal of Electrical Technology*, vol. 2, no. 3, pp. 27–30, 2017.

[11] A. Rahmatulloh, H. Sulastri, and R. Nugroho, "Keamanan RESTful Web Service Menggunakan JSON Web Token (JWT) HMAC SHA-512," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)*, vol. 7, no. 2, 2018, doi: 10.22146/jnteti.v7i2.417.

[12] "Apa itu OAuth 2." Accessed: Dec. 07, 2022. [Online]. Available: https://www.huzefril.com/posts/security/oauth2-apaitu/#apa-saja-yang-didefinisikan-dalam-standarnya-ini-

[13] Q. Nguyen and O. Baker, "Applying Spring Security Framework and OAuth2 To Protect Microservice Architecture API," *Journal of Software*, pp. 257–264, Jun. 2019, doi: 10.17706/jsw.14.6.257-264.

[14] L. Xie, M. H. Li, L. Han, and X. L. Dong, "Design and implement of spring security-based T-RBAC," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Oct. 2017, pp. 183–188. doi: 10.1145/3180496.3180629.

[15] M. Elsera and A. Di, "IMPLEMENTASI SINGLE SIGN ON PADA WEB MENGGUNAKAN PROTOCOL OAUTH FACEBOOK," Online.

[16] "Single Sign-On." Accessed: Jan. 14, 2023. [Online]. Available: https://auth0.com/docs/authenticate/single-sign-on

[17] B. A. Pranata *et al.*, "PERANCANGAN APPLICATION PROGRAMMING INTERFACE (API) BERBASIS WEB MENGGUNAKAN GAYA ARSITEKTUR REPRESENTATIONAL STATE TRANSFER (REST) UNTUK PENGEMBANGAN SISTEM INFORMASI ADMINISTRASI PASIEN KLINIK PERAWATAN KULIT," 2018.

[18] "Laravel Socialite - Laravel - The PHP Framework For Web Artisans." Accessed: Jul. 12, 2023. [Online]. Available: https://laravel.com/docs/10.x/socialite

[19] A. Alonso *et al.*, "An Identity Framework for Providing Access to FIWARE OAuth 2.0-Based Services According to the eIDAS European Regulation," *IEEE Access*, vol. 7, pp. 88435–88449, 2019, doi: 10.1109/ACCESS.2019.2926556.

[20] Z. Musliyana, A. G. Satira, M. Dwipayana, and A. Helinda, "Integrated Email Management System Based Google Application Programming Interface Using OAuth 2.0 Authorization Protocol," *Elkawnie*, vol. 6, no. 1, p. 109, 2020, doi: 10.22373/ekw.v6i1.5545.

[21] and F. M. Baozhong Gao, Fangai Liu, Shouyan Du, “An OAuth2.0-Based Unified Authentication System for Secure Services in the Smart Campus Environment,” *Springer Nature 2018*, vol. 1, pp. 350–357, 2018, doi: 10.1007/978-3-319-93713-7.
[22] A. M. Taufik, “Pembangunan Network Access Control Untuk Autentikasi dan Security dengan Menggunakan 802 . 1X Authentication Jurnal Ilmiah Komputer dan Informatika (KOMPUTA),” *Umum*, vol. 1, pp. 1–7, 2014.
[23] A. S. Sembiring, “Penerapan Model Protokol Aaa (Authentication, Authorization, Accounting) Pada Keamanan Jaringan Komunikasi Wan (Wide Area Network),” *Jurnal Multimedia dan Teknologi Informasi (Jatilima)*, vol. 2, no. 1, pp. 19–29, 2022, doi: 10.54209/jatilima.v2i1.140.
[24] A. Fauzi, J. Dedy Irawan, and N. Vendyansyah, “Rancang Bangun Sistem Manajemen User Aaa (Authentication, Authorization, Accounting) Dan Monitoring Jaringan Hotspot Berbasis Web,” *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 4, no. 1, pp. 176–183, 2020, doi: 10.36040/jati.v4i1.2328.
[25] M. A.-J. P. T. I. dan Ilmu, “Penerapan Authentication, Authorization, and Accounting untuk Pengamanan Jaringan Small Office/Home Office,” *J-Ptiik.Ub.Ac.Id*, vol. 6, no. 1, pp. 394–401, 2022, [Online]. Available: http://j-ptiik.ub.ac.id/index.php/j-ptiik/article/download/10522/4650