Design and Implementation of Passwordless Single Sign On Authentication Mechanism

Fatima Hussain*, Rasheed Hussain†, Damir Samatov†, Andrey Bogatyrev†and Salah Sharieh*,
†Innopolis University, Innopolis, Russia, Email: r.hussain@innopolis.ru
*Royal Bank of Canada, Toronto, Canada, Email: fatima.hussain@rbc.com

***Abstract*—Single Sign-On (SSO) is an access control mechanism that enables a user to get authenticated only once through an authenticated server, and get access to all other available services (related to authentication server) without providing credential again. Passwords are considered as the most popular method for user authentication. However, password selection and management is a challenging task. In this paper, we design and implement a *password less* authentication mechanism and also present the SSO implementation with magic-links technique. In essence, we design two password less SSO scenarios. In the first scenario of the proposed SSO technique, we create global and local sessions based on JSON Web Token (JWT) tokens and then grant access to services (based on JavaScript). In the second scenario, the open-source identity server framework is modified in a way to create a session key (token) distributed among the connected services and users can be authorized by using protocols, such as OAuth with OpenID Connect. The proposed mechanism addresses the problem of limitations with the passwords and further scales the SSO techniques across different services.**

***Index Terms*—*SSO(Single Sign-On)*, *Passwordless*, *Keycloack*, *OAuth*, *OpenID Connect*, *Identity Server*, *Magic-Link*, *Authentication*, *Authorization*.**

I. Introduction

In the wake of increased security risks and sophisticated cyber attacks, mutual authentication among clients, as well as servers, is of utmost importance for validating legitimate users and services. Modern authentication and authorization are usually based on a *username* and *password*. Since passwords are the easily manageable option for authentication, most of the users prefer using the same password for different Internet services. If a critical service requires the users to change their passwords frequently or has a restrictive password policy, users usually change the password(s) to obvious and guessable words and often repeat the same passwords. As a result, even if an unrelated application gets hacked or the application data is compromised, it puts users' account at risk because of the frequent use of same and related passwords. This way, the dictionary attacks on passwords are feasible and cheaper means of hacking passwords. In order to overcome these challenges, password less approaches are considered as revolutionary approaches to improve the existing username and password mechanisms. As an alternative, the users can just provide their usernames, and the system generates a one time code, and delivers it to the users via email, dedicated application or an SMS. Afterwards, users provide this code back to the system and the system verifies the credibility of the code (code is issued by the system). Benign users are authenticated if the verification process is successful.

Similarly, Single Sign-On (SSO) approach enables the consolidation of user identity and management. In our daily lives, we use plethora of different applications every day, such as email, issue tracker, file hosting, Customer Relationship Management (CRM) software, and so on. If unique set of credentials are required for each of these applications, it leads to a very fragmented identity system. To date, various software solutions are available for SSO integration [1]; however, the existing solutions are expensive and complex.

In this regard, it is important to decide whether a Software-as-a-Service (SaaS) subscription should be purchased that provides a ready-to-use SSO solutions that can be integrated into our products or is it better to build our own solutions. To answer these questions and put light on the existing SSO solutions, in this paper, we discuss different approaches of SSO implementation, and then design our own home-grown implementation of a password less SSO with a discussion on selecting the best of both approaches, i.e., SSO and passwordless authentication to one system. The rest of the paper is organized as follows. In Section II, we discuss the existing work on SSO and then outline our research goals and methodology in Section III. Passwordless SSO is discussed in Section IV and our proposed SSO model along with its implementation is discussed in Section V. In Section VI, we conclude the paper.

II. Related Work

SSO authentication allows users to get an access to different domains without the requirement to enter their credentials repeatedly. This allows users to maintain only one account instead of many. Some of the benefits of SSO include, but not limited to, boost to the user experience, improved security, and reduced cost in terms of creation and storage of passwords for different services. However, it also introduces various challenges to the authentication systems, such as difficult implementation, password security, dictionary attacks on the passwords, password management, and above all, the smooth integration into traditional password-based login systems. Furthermore, some of the most pressing issues are listed below:

- Reduced security as users create weak passwords and hackers usually use smart tools. Also, if access to the identity provider is compromised, all the connected services are also compromised.

 ISBN: 978-1-61208-770-2

- Degraded Quality of Service (QoS) experience for customers, employees, and users already forced to create and manage many passwords. According to the existing researches, the users do not like passwords and do not manage the passwords efficiently [2].

There are several password alternatives:

- Email-based authentication (also called *magic-link*). It is a unique link sent to the email with permission to authenticate only once. It becomes invalid automatically when the user is logged into account. It eliminates the password requirement entirely and makes use of the email address to validate an identity. It is worth mentioning that email is one way to send the magic-link. We could use another means such as messaging application and so on. Nevertheless, the security of the magic link is dependent on the secure use of the email. If the attackers can compromise the email service, then the authentication will be jeopardized as well.
- Social media sign-on or oAuth: A third-party application requests access to the identity provider (Google, Facebook, GitHub etc.) to gain information about the requested profile.
- Certificate-based: Users get a secure access to a server by exchanging a digital certificate. It is suitable in most cases and used only for the internal authentication in a company.
- Biometric technologies: The idea is "You are your key". This includes fingerprint, facial, eye, speech recognition and so on.

To date, there are commercially available solutions, frameworks, and protocols that allow us to provide SSO-based authentication as given below.

- OAuth 2.0 protocol is used for authorization and OpenID Connect on top of it, is used to verify the identity of the end-user.
- Keycloak is another open-source solution to allow SSO with identity and access management.
- IdentityServer4 framework is built on ASP.NET Core and it implements OAuth and OpenID protocols.

Furthermore, in [3], the authors proposed Loxin universal security framework for passwordless login. It supports two-factor and multifactor-authentication and consists of modular architecture. The architecture includes application, server, Certificate Authority and Push Message Service (PMS) that can resist the main security attacks, such as Man-In-The-Middle (MITM) and replay attacks. However, Loxin does not provide a wide list of authentication mechanisms and single sign-on abilities. Moreover, in [4], the authors describes a way of using CAPTCHA on the mobile phones when the server performs verification of the user's response according to the sender's IMEI code. In another work [5], the authors performed extensive experiments on FIDO-based passwordless authentication along with Shibboleth single sign-on technique.

Despite all afore-mentioned work, there is lack of research and implementations of combining benefits of password less approaches with SSO that will not only improve the efficiency, but also increase the security. This research intends to combine the SSO and passwordless techniques to improve security and user experience as well as increase efficiency and ease-of-use.

III. Research goals and methodology

In this section, we explain the goals of this research work. Furthermore, we also devise the methodology for our proposed SSO mechanisms. The research goals are summarized below.

A. Goals

- Build an authentication mechanism in which users are authenticated to multiple services without passwords. In other words, devise a passwordless SSO mechanism.
- Investigate the existing SSO techniques that (after necessary tweaking and modification) can be used for our proposed authentication system. Afterwards, we aim at implementing the SSO framework to provide access to various services through a single entry point.
- Combining SSO with the already developed or outsourced passwordless authentication is a daunting challenge since the existing SSO still needs password. Our goals is to integrate SSO with passwordless authentication.

B. Methodology

We divide the methodology into the following four stages.

- The first stage is mainly focused on preparation and field research. We thoroughly studied and analyzed the existing works related to SSO and authentication methods. Furthermore, we considered most popular methods for SSO implementation (and built from scratch, Keycloak, oAuth-based, OpenID Connect).
- In the second stage, the most suitable techniques for password less authentication are determined based on exploring the existing solutions.
- The third stage is focused on combining SSO and password less authentication approaches and develop the integrated system.
- In the fourth stage, we conduct the experiments to verify the proof-of-concept.

IV. Passwordless SSO

In this section, we discuss the passwordless SSO techniques.

A. OAuth 2.0 and OpenID Connect

OAuth 2.0 is a protocol that allows a user to provide limited access to their resources on one application, to another application without having to expose their credentials. The typical work-flow of the protocol is shown in Figure 1. To get access to the protected resources, OAuth 2.0 uses access tokens. An `access token` represents the granted permissions. Typically, access tokens are in JSON Web Token (JWT) format. JWTs contain three parts: a metadata about the type of token and the algorithms used to encrypt its contents, a set of statements about the permissions that should be allowed, and a signature to validate that the token can be trusted. The

 ISBN: 978-1-61208-770-2

permissions represented by the access token are known as `scopes`. The application specifies the scopes it wants when authenticates. If scopes are authorized by the end-user, then the access token will involve these authorized scopes.

The following roles are allowed in OAuth:

- Client: it is the application that requests access to a protected resource on behalf of the Resource Owner.
- Resource Owner is the end-user who has the credentials.
- Resource Server is the resource or API server. The resource server handles authenticated requests after the application has obtained an access token.
- Authorization Server: the server that authenticates the resource owner, and generates access tokens after getting proper authorization.

The OAuth 2.0 protocol specification defines different flows in which access token can be accessed. These flows are called grant types. Grant types are decided on the basis of individual cases, i.e., type of the application. Following are the common grant types.

- Authorization Code: is used by web applications with back-end server. This also can be used by mobile apps, using the proof key for code exchange technique.
- Implicit: is used by Single Page Applications executing only in browser without any back-end. There is no extra step of exchanging authorization code to access token.
- Resource Owner Password Credentials: In this grant type, the username and password are exchanged directly for an access token.
- Client Credentials: is mostly used for machine to machine communication.

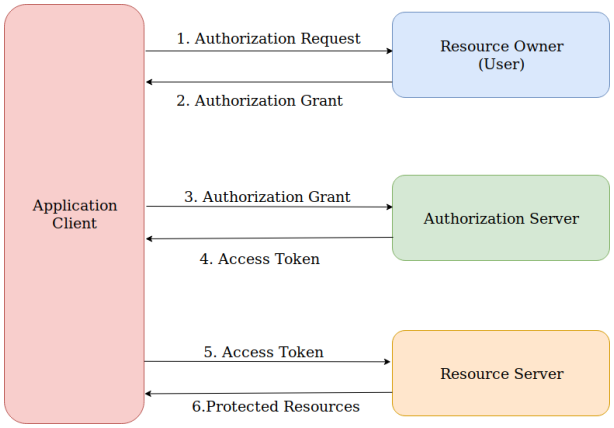

Figure 1: Protocol flow

OAuth and OpenID Connect are used together and compliment each other. OAuth 2.0 is used for resource access and sharing, while OpenID Connect is used for the user authentication. It is applied on top of OAuth protocol as an extra layer. It uses simple JSON Web Tokens (JWT), which are obtained using flows conforming to the OAuth 2.0 specifications. OpenID Connect provide one login for multiple sites, and whenever a user log into a website using OIDC, he is redirected to OpenID site for authentication and taken back to the original website.

OpenID Connect allows applications to verify the identity of the user based on the authentication performed by an authorization server, as well as to get basic profile information about the user by requesting an ID token. Various tokens and terminology used with OpenID Connect is explained as under.

- Access Tokens: are credentials used by an application to access any API. Access Tokens can be an opaque string, JWT, or non-JWT token. Access token informs the API that the owner of this token has been granted delegated access to the API and is in position to request specific actions.
- Identity Token: is a JSON Web Token (JWT) that contains identity data. Application use this to get user information such as , name, email etc. (typically used for UI display). ID Tokens contain three parts: a header, a body and a signature.
- Claims: JWT Tokens contain claims, which are statements (such as name or email address) about an entity or an user and some additional metadata. Set of standard claims are obtained through OpenID Connect specification, which include name, email, gender, birth date, etc. Custom claims can also be created and is added to token, if the information needed about a specific user isn't in a standard claim.

Now we explain basic functionality of OpenID Connect by using use case of, logging into using OAuth and OpenID Connect (by employing Google account.

1) When a client sign into OAuth using his Google account, OAuth sends an Authorization Request to Google.
2) Google authenticates client credentials and asks client to login if he is not already signed in. It also ask for authorization (lists all the permissions that OAuth wants, for example read permissions for email address, and asks client if he is ok with that).
3) Once client authenticate and authorize the sign in, Google sends an Access Token, and (if requested) an ID Token, back to OAuth.
4) OAuth0 can retrieve client information from the ID Token or use the Access Token to invoke a Google API.

V. Proposed SSO Model

We propose and develop SSO solution customized according to our specifications and requirements. This will not only reduce the cost and dependencies on the external vendors, but also give us a complete control over all of the data (storage, processing etc.). However, specific expertise and experience is required for developing solutions for identity management.

A. *Keycloak SSO*

Since Keycloak is the most popular Open Source Identity and Access Management application [6], we choose it as main tool for SSO implementation. To work with Keycloack, we performed following steps to enable SSO. We succinctly

 ISBN: 978-1-61208-770-2

enumerate the steps that will enable the SSO feature of Keycloack.

1) Login to Keycloak on localhost:8080 with default credentials admin:admin.
2) Create new realm.
3) Create new client applications (as many as we want), in the menu Configure $->$ Clients. Choose `Client Protocol` as `openid-connect` and `Access Type` as `confidential`, also put `Implicit Flow Enabled` in `ON` position.
4) Create a new user, which will be our test client, in the menu Manage $->$ Users
5) Copy client application credentials. And then go through the following path: Configure $->$ Clients $->$ `Your app name` $->$ Installation. Choose `Format Option` as `Keycloak OIDC JSON` and copy all provided configurations.
6) Download and launch the test NodeJS app from repository [7], which will use authentication methods provided by the Keycloak.

After performing these steps, if we are able to login in first launch, we will automatically login to second as well.

B. Identity Server 4 Passwordless SSO

Identity Server4 is an OpenID Connect and OAuth 2.0 framework for ASP.NET Core platform [8]. It enables the feature of Single Sign-on and Sign-out for our applications.

Identity Server is used as an authorization service in our case. In order to configure it for our needs, we developed the architecture as shown in Figure 2.

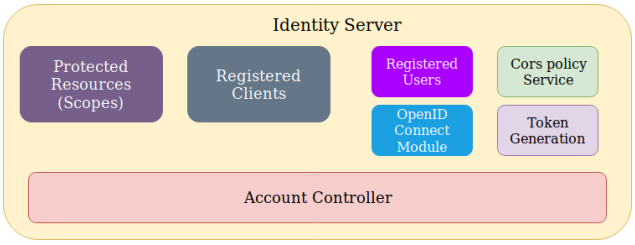

Figure 2: Identity Server Architecture

Client is an application or software that requests tokens from the Identity Server - either for authenticating a user or for accessing a resource. An application must be registered with Identity Server before it can request the tokens. We want to protect the resources, such as APIs, identity data for our users. Each resource has a distinctive name, and applications use these name to specify to which resources it need to induce access to. A user is a person that uses a registered client to access certain resources. We add CORS service to avoid problems with this policy. Moreover, we add support for OpenID Connect Identity Scopes. This is in contrast to OAuth, scopes in OpenID Connect do not represent resources, but represent identity data of user, such as ID, name or email address. At the end, we configure `AccountController`. We also created a method to check if the user's email exist or not, and if it does, we will generate a token for this user. Then we create a link which should be sent to the user's email with the generated token attached. By following this link, the user is then redirected back to the authorization server and it checks if the attached token is valid or not. If it successfully passes this checking the consent screen which requests the access to user's data for the application is generated. Users choose which kind of scopes they want to give to the application and then redirected back to the service itself. Developed solution can be found in the repository [9].

C. Magic link implementation

In traditional authentication scenario, a user is required to provide a username and password, while in passwordless authentication, users only provide their username. With this username, the system issues a one-time passcode and delivers it to the user via email. The user then provides this code back to the system and the system verifies that the provided code is legitimate (correct, not expired and never used). If the code is legitimate, the user is authenticated. Basic authentication flow of magic links implemented on NodeJS is described in Figure 3.

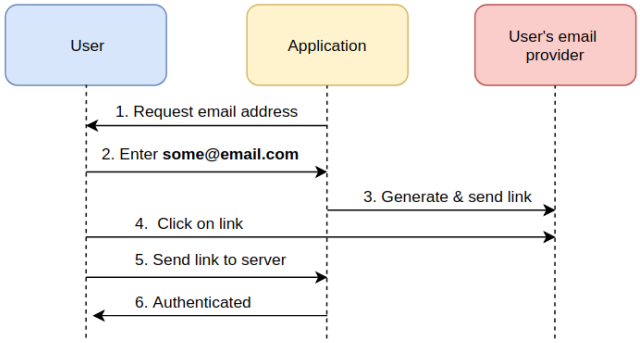

Figure 3: Magic link flow

D. SSO implementation

As proof of concept, we perform basic implementation and we create an SSO provider and clients on NodeJS as shown in Figure 4. Details description of the logical flow and sequence of authentication process is described as follows.

1) The user tries to access resource of system domain1 which is under the protection. domain1 detects that the user is not authenticated and jumps to the sso-server, using its own address as one of the parameters.
2) The SSO authentication server also detects that the user is not logged in and redirects the user to the generated login page.
3) User enters credentials and the SSO authentication server then verifies information that the user provided.
4) The session between the user and the SSO authentication server is established. This is called a global session. Then the server sends authentication token to "domain1". And the session cookies are stored in the browser cookie storage.

 ISBN: 978-1-61208-770-2

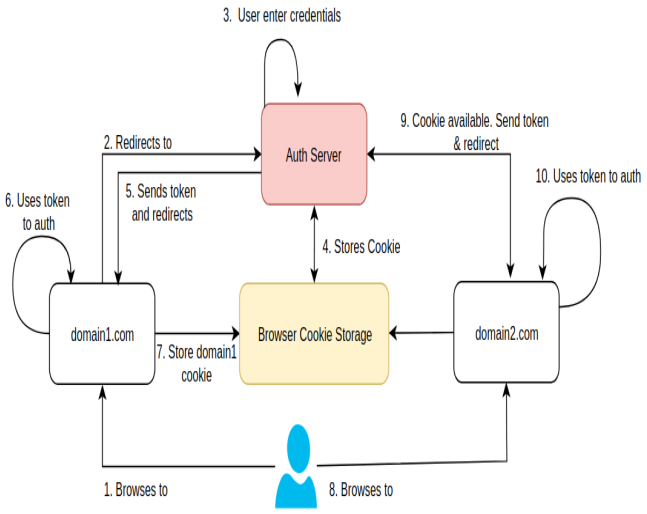

Figure 4: SSO workflow

5) The SSO authentication server takes the authorization token and jumps to the initial request address (system domain1).
6) The domain1 uses this token to create and establish a connection with the user. In our case we will return a signed JWT with user profile information if validation was successful.
7) Application under ”domain1” then creates local session based on JSON Web Tokens payload. Furthermore, domain1 cookie is also stored in the browser.
8) Now the user can reach any other websites connected as consumers without entering any credential because there are the global session stored by the SSO authentication server that will redirect the user to ”domain2” already with user profile info.

E. Testing environment

After implementation of our proposed SSO model, we tested it in practical environment. Therefore, after launching the server and clients, we use web-browser to validate the implementation of our SSO mechanism. When a client application is opened in the browser, it is automatically redirected to SSO server and it also provides a prompt for a valid email address. An authentication link is sent to the email provided and the user has to click on the link sent to the email. The link serves as an authentication token and after clicking the link, the user is authenticated with the SSO-provider. After authentication, the user can request any client sites and the access is granted without providing the user credentials again.

VI. Conclusion and Future work

In this work, we provide some alternative password techniques for protecting user credentials and identity for improving user experience and security. We presented methods of creating SSO based open-source solutions followed by their successful implementation and testing. Also, we implemented authentication system by combining SSO and passwordless approaches.

For future work, we aim to design and implement new passwordless techniques. It will also enable flexible adjusting of the users authentication flow. The development of the mobile application related to the SSO authentication server can be one of the possible solutions to grant access to the resources by confirming the request for authentication. Moreover, the ways of communication between the browser and physical device, such as USB stick can be developed to get the stored keys or certificates which will authenticate the user without prompt to enter a password.

References

[1] N. Heijmink, “Secure single sign-on a comparison of protocols.” 2015. [Online]. Available: https://www.coursehero.com/file/21561672/z-researchpaper-sso-final-nick-heijmink-s4250559/
[2] S. Wise, “Is memorizing passwords the easiest way to manage them?” 2015. [Online]. Available: https://www.passwordboss.com/password-habits-survey-part-1/
[3] B. Zhu, X. Fan, and G. Gong, “Loxin. a solution to password-less universal login,” 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6849280
[4] M. S. Shahreza and S. S. Shahreza, “Passwordless login system for mobile phones using captcha,” 2007. [Online]. Available: https://ieeexplore.ieee.org/document/4418840
[5] M. Morii, H. Tanioka, and K. Ohira, “Research on integrated authentication using passwordless authentication method,” 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8029677
[6] T. Darimont, “Awesome keycloak.” [Online]. Available: https://github.com/thomasdarimont/awesome-keycloak
[7] A. Bogatyrev and D. Samatov., “So-server & consumer implementation, keycloak test client.” 2017. [Online]. Available: https://bitbucket.org/bogatyr285/sso/src
[8] S. Brady, “Getting started with identityserver 4,” 2016. [Online]. Available: https://www.scottbrady91.com/Identity-Server/Getting-Started-with-IdentityServer-4
[9] A. Bogatyrev and D. Samatov., “Identity server with client and apis.” 2019. [Online]. Available: https://bitbucket.org/demmy_art/passwordless-sso

 ISBN: 978-1-61208-770-2