International Journal of Advanced Science and Technology
Vol. 29, No. 12s, (2020), pp. 2781-2789

Password less Authentication Using Single Sign On (SSO)

1st Sachin Jadhav

Department of Information Technology, Pimpri Chinchwad College of Engineering, Pune
srjadhav02@rediffmail.com

2nd Siddharaj Mane

Department of Information Technology
Pimpri Chinchwad College of Engineering, Pune manesiddharaj8@gmail.com

3rd Shobhan Chandvale

Department of Information Technology, Pimpri Chinchwad College of Engineering, Pune
cshobhann1@gmail.com

4th Chaitraly Bhagywant

Department of Information Technology, Pimpri Chinchwad College of Engineering, Pune
chaitraly.bhagyawant@gmail.com

5th Dnyaneshwar Bidkar

Department of Information Technology Pimpri Chinchwad College of Engineering, Pune
dnyaneshwarbidkar17@gmail.com

Abstract

In today's world of where there is plethora of millions of apps that are created and published for the users and also there are millions of the people which use this apps. Therefore, the data that is being used by apps from the user the users sometimes don't have much control of it. We also came across various news that data of the users are compromised due the vulnerability in the process of handling the authentication or storing the user's data. Which violates the essential right of the user's privacy. So, we have come up with a solution that an infrastructure can be designed and implemented which is much secure and the user will have a controlled way of sharing the scope of the user's data. get a more secured password less authentication and more controlled way of data flow between the service providers and the identity providers. So each time the user can be authenticated without passwords in a much secured way and user has much precise monitoring and control over data that has been shared with the help of the carriers. So user is not dependent on social media accounts which user may not have trust on which data they store of them. Also we would provide the detailed analysis for the user about how and which data has been shared to the apps and also a more easy interface API's for the service providers to easily integrate this authentication method. So at the users get a more secured password less authentication and more controlled way of data flow between the service providers and the identity providers.

I. INTRODUCTION

Project is based upon the presentation of architecture for the authentication of the user using the SSO that is Single Sign On and also the user would have control on which of the data to be shared to the app that is SP (Service Provider). Also the user would able to maintain multiple apps without the hassle of remembering of the multiple passwords of the apps. As this way there is no vulnerability of the password and data that is shared to the SP app. The user base for the apps is increasing day by day due to which the user may have multiple types of apps installed which requires the user and password. So the solution for new authentication method for the app should be designed and implemented. So this can be taken an opportunity to design an new authentication method. In today's world of where there are plethora of millions of apps that are created and published for the users and

also there are millions of the people which use this apps. Therefore the data that is being used by apps from the user the users sometimes don't have much control of it. We also came across various news that data of the users are compromised due the vulnerability in the process of handling the authentication or storing the user's data. Which violates the essential right of the user's privacy. So we have come up with the way of new design and authentication method.

II. LITERATURE SURVEY

For the implementation of the Single Sign On (SSO), there are two methods for this implementation as follows:

SAML (Security Assertion Markup Language)

Security Assertion Markup Language is a publicly available protocol that enables the identity provider to transfer authorization credentials to service provider. It means that a set of credentials can be commonly shared between multiple websites. Handling one email and password credential each is easier thanhandling multiple account credentials made on multiple websites, customer relationship and management (CRM) software, Active Directories, etc. SAML transactions use Extensible Markup Language (XML) for making communications standardized between the identity providers and service providers. SAML is the link between the authentication identity of a user and the authorization use as a service.[5]

- *OIDC (Open ID Connect)*

The OpenID Connect protocol authenticates users to the Relying Parties by their existing account at an Identity Provider. (Generally, this uses an email account at the Identity Provider.) OIDC has been defined in the OpenID Foundation in a Core document as well as in the extension documents. Standardization was bought to the supporting technologies at the IETF. Central to OIDC is a document which is signed cryptographically and the id token. It is created by the user's Identity Provider and it serves as a one-time proof as a part of the user's identity to the Relying Party.

- Why to use OIDC:

- OIDC is one of the most widely used protocols single sign-on use cases. Even though same services like AWS Cognito support OIDC integration's they still provide options to federate it to a remote IDP with the help of SAML or ODIC.

- SAML generally has a high weight due to the size of the XML messages that are being transmitted between the Service Provider and Identity Provider, whereas OIDC is pretty light in weight.

- SAML specification does not flow according to the user consent even if it can be built in the begin flow. OIDC specification expects that the user should provide consent to share the profile details which meets all the updated privacy regulations and restrictions like GDPR.

- Also supports the modern API security scope. So, OIDC basically supports both authentication as well as authorization of the users.SAML token (Assertion) is technically not used for API security. In the case of OIDC, a SP gets an refresh token, access token along with the id_token. As OIDC is built on top of the OAuth protocol.

- Due to the increased demand in mobile apps and service-oriented architecture, OIDC is used in alarge scale for Authentication and Authorization due to its characteristics such as being lightweight and following the REST based architecture.

- OIDC and OAuth2 protocols can be combined used for various use cases like device-to-device

authentication, machine authentication etc. whereas SAML is not preferred in such use cases.

- *What is OAuth?*

OAuth can be defined as open standard authorization protocol or framework that defines how different servers and services can securely allow authenticated access to their assets without sharing the crendential, personal info, logon credential. Therefore OAuth is extensively used for authorization of the access.

- *Why is OAuth used?*

OAuth was created to respond the legacy authentication pattern. This method of authentication was made popular by HTTP basic authentication, when the user is invoked for providing credentials. Basic api's is used for authentication purpose. Therefore for server side application instead of transfer of user name and password to the server with per request, the user transfers the API Key ID and the secret. Before the OAuth, the sites would prompt us to enter the user name and the password for the authentication directly through the forms and if the login was successful then would redirect to the respective home page. This is called as password anti-pattern.
Similarly in the enterprises and the intranet scenarios where the users may required access for the multiple sites there was introduction of the federated identity which was created for single sign on (SSO). In this condition, an end user would interact to their respective identity provider, and the identity provider would generate a cryptographic signed token_id which it was send to application to authenticate the user. As long as the token was valid the user would able to access the resources until his federated identity was intact. [3]

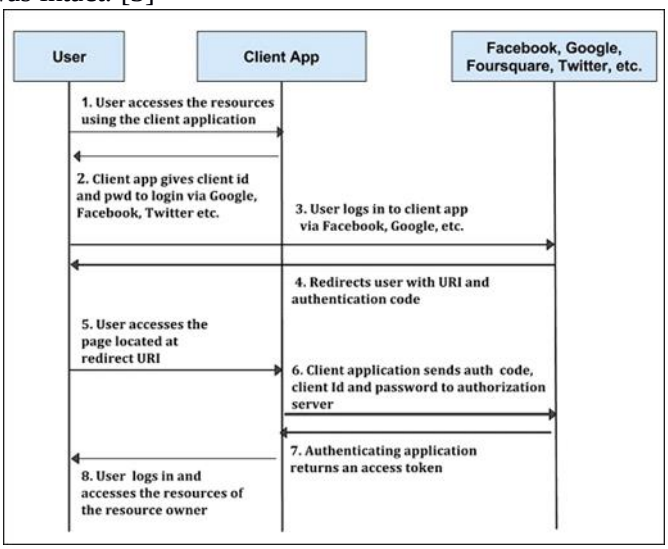

Fig. 2. OAuth architecture

- *Working of OAuth in web server Applications*

The Google OAuth 2.0 can be considered as the perfect example to study OAuth2.0. The Google supports various web platforms such as Java, PHP, Python, Ruby and Node.js and various other apps framework such as flutter, IOS and android. The process of authorization begins when the process redirects to the Google URL, the URL consists of query params which represents or indicates the type of access being requested. Google on other side handles all other user authentication, session management and the user data consent. After successful authentication is the authorization code, which application can use to obtain the access token or the refresh tokens for its need. The application can preserve the refresh token for further use and the access token to access the Google API. If the access token is expired the application can get the new token to access by the help of the refresh token. [3]

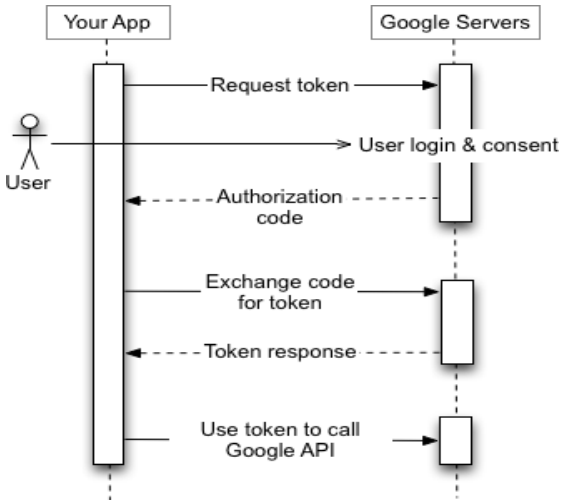

Fig. 3. Working of Oauth

III. RELATED WORK

- *Difference between SAML and OIDC [5]*

◦ SAML:

i. Authentication request:
a) Access the service provide request sp.com
b) create a SAML <AuthnRequest> and send a redirect to the browser.
c) get a request to Identity Provider with SAML <AuthnRequest>

ii. Authentication:
a) Display login page in browser.
b) User enters the credential.
c) Validate the credential and redirects to SP ACS URL with signed SAML<response>in html form.
iii. Response validation:
a) Post signed<response> to SAML ACS URL.
b) Validate the signed<Response> retrieve user attributes and return to the service provider.
iv. Signature:
a) It contains <ds:signature> and X509certificate used for signing the response.

- OIDC: [3] [4]

i. Authentication request
a) Access the service provide request sp.com
ii. Authentication:
a) Display login page in browser.
b) User enters the credential.
c) Show the user consent page with the attributes that are requested.
d) User accepts the consent.
e) The user is redirected with authorization code.

iii. Response validation
a) User gets the authorization code through redirect URI
b) Submits auth code and get valid id_token
c) validates the token and resources can be accessed.
iv. Signature:
a) It contains JWT ID token. The header contains the algorithms which used for signing.

b) The certificate can be obtained by jwks_uri URL.

IV. PROPOSED SOLUTION

- *Architecture Design and Implementation:*

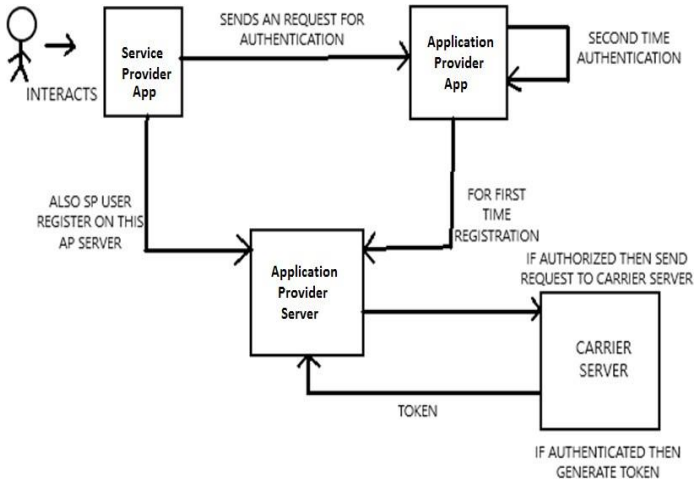

Fig. 1. Architecture of Password less Authentication using Single Sign On

- *Implementation / Working:*

- If Service provider application register for for first time then it required to register to application server.
- Once the request is authenticated from the carrier server then the token given by carrier is saved on application server. This token can be reused for further information for which user has consent granted..
- The that are used are to be used are JWT (Json Web Tokens).Therefore the user only requires to click one button to get registered from service provider application by using this password less authentication.
- *For sensitive data token passing POST request to to be used. But last thing the authentication is done by identifying the unique phone id(IMEI) and last 24 Hrs SIM status and the authenticated by carrier server.*

- *Open ID Connect for the implementation of the proposed solution*

The OPENID connect is used to authenticate tp Relying parties using their account credentials which may be an email or phone number to the Identity provider. The foundations of OIDC was dictated by OPENID Foundation in their core documents and also in extension. The supported tech were publicized by the IETF. We define OIDC as the crytographically signed document which is a identity token. The

token is created by users Identity provider and it proves an identity of the user to the relying party. The overall abstract overview of the OIDC is given by the figure 1 displayed below. At beginning the user has to be logged into Relying Parties. Then the Relying parties acquires the required information of the user maybe some URL's and at last it must be registered to the Identity provider. So when user initiates the flow the user is redirected to the Identity provider where he/she himself prove its identity using the credentials. The the Identity Provider itself register token to the user and

then give to the Relying party. (The method by which the identity provider gives token is having various methods or the modes, which is described further in the review paper.) The token consist of the fields such as the Issuer, An unique identifier with respect to the Identity Provider and its signed by Identity Provider. Then it helps Relying Party by the identifier and the users identifier to prove the identity of the user. At last if flow is successful the Relying party can set the browser cookie and start the session of the user. [3]

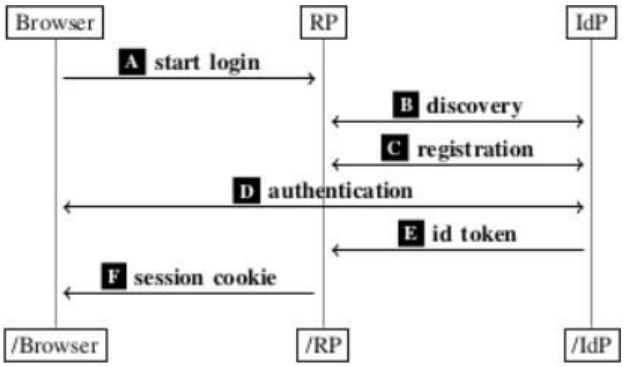

Fig. 4. OpenID Connect high level review

- *Modes in which OIDC works*

OIDC specifies three modes in which it works: 1) authorization code mode, 2) implicit mode and the last 3)Hybrid mode. In authorization code mode, the identity token is passed to the user by relying party through provided by identity provider by direct server-to-server communication that is from back-channel, the implicit mode work in the way by the identity token is provided by relying party from it receives from identity provider through the browser that is front channel. The another mode that can be defined is the hybrid mode which is the combination of the both that is by back channel and the front channel. [3]

- Authorization mode: In the following mode, The relying party transfers to the Identity provider. Where the user authenticates itself and get's the authorization code to the Relying party. By help of the authorization code now the Relying party can obtain the token_id from the identity provider. The figure 2 is used to illustrate the concept. [3]

- Implicit Mode: This mode shared same process as of authorization mode. But in this method instead of the authorization mode the identity provider directly provides the token_id to the relying party by the way of the browser that is when user authenticate itself to the identity provider. Hence the steps 1 to 13 of the authorization code mode [Figure 2] will be same. After following steps, the identity provider tranfers to the transfer endpoint at the RP, providing an token_id, the acess token can also be provided, the state value, and the issuer identifier. The following values are not provided as URL param but in the URL fragment instead. Hence browser not directly sends it to relying party at initial, Instead the relying party has to provide the JS that fetches the values from the fragment and sends to the relying party. If token_id is legal, the issuer is valid, and state should be asserted by provided Relying party, the relying consider it as correct and allows user to be logged in. [3]

- Hybrid Mode: This mode is combination of both authorization code and implicit

mode. Initially this mode function as the implicit mode, but when identity provider redirects the browser back to relying party, the identity provider provides an authorization code or else token_id or access token or both. The relying party retrieves this as the implicit mode and uses authorization code to obtain access token and token_id. [3]

- *Relationship to OAuth 2.0*

Technically, OIDC is dependent of OAuth2.0. It goes far beyond what was specified in OAuth2.0 and introduce many new concepts: OIDC states a method for authentication (while also supports for authorization) using new defined tokens, the token_id. Some messages and token_id can be cryptographically signed and encrypted while OAuth2.0 does not use these this methods. The new type of hybrid mode flow combines the features of the implicit mode and the authorization code mode. Basically oidc also includes adhoc discovery and dynamic registration which is out of the scope of the OAuth2.0 but OIDC automates this whole process. There OIDC brings a new way of authentication. [3] [4]

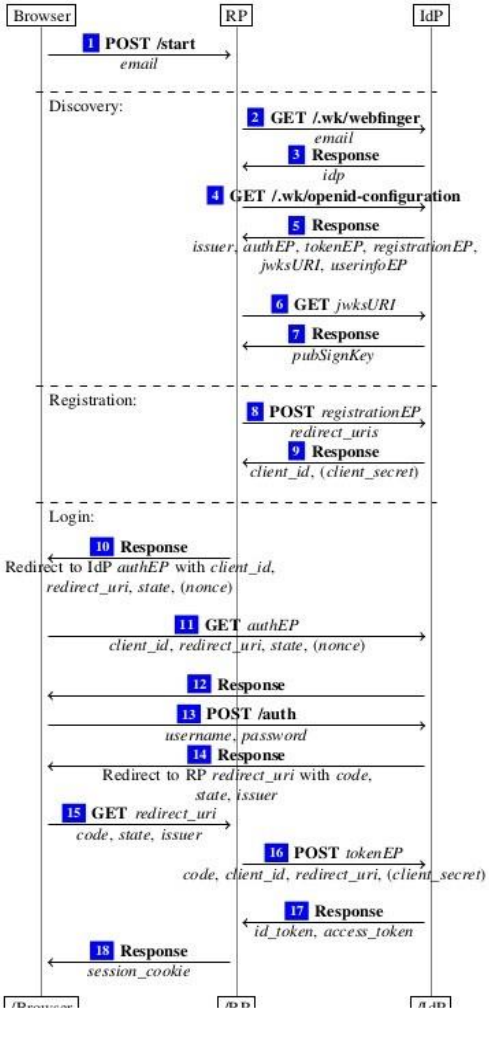

Fig. 5. OpenID Connect authorization code mode

- *JSON Web Token*

JSON Web Token is an open standard RFC 7519 method for representing claims securely between two parties.Its token structure is encoded in a compact JSON serialization format which consists

JSON Web Signature also known as JWS and JSON Web Encryption also known as JWE.Inside each JWT token, JSON Web Key and JSON Web Algorithm are embedded cryptographically. RFC 7519 standard tells us about the implementation of how JSON Web Tokens can be created,signed and encrypted. There are various types of the tokens; such as access tokens, identity tokens and refresh tokens. Access tokens are used to validate a user without making any calls to the database. This feature overcomes the service latency caused by OAuth2. For access token validation JWS and Public Key Cryptography are used. Access token use can be limited using expiration time. RFC 7519 standard tells us about the implementation of how JSON Web Tokens can be created,signed and encrypted. There are various types of the tokens; such as access tokens, identity tokens and refresh tokens. Access tokens are used to validate a user without making any calls to the database. This feature overcomes the service latency caused by OAuth2. For access token validation JWS and Public Key Cryptography are used. Access token use can be limited using expiration time.RFC 7519 standard tells us about the implementation of how JSON Web Tokens can be created,signed and encrypted. There are various types of the tokens; such as access tokens, identity tokens and refresh tokens. Access tokens are used to validate a user without making any calls to the database. This feature overcomes the service latency caused by OAuth2. For access token validation JWS and Public Key Cryptography are used. Access token use can be limited using expiration time. [1] [2]

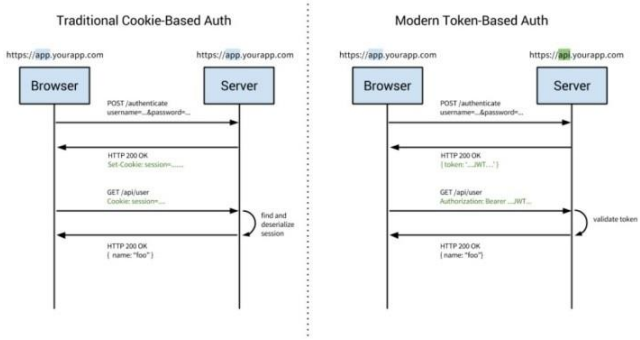

Fig. 6. JWT authentication

- JWT consists of three structures:

i. Header: JWT's header consists of the following data, first is type of the token,in this case it is JWT token and the next section is the name of the hashing algorithm used, it can be HMAC, RSA or SHA256.
ii. Payload: This is the second part of the token. It mainly consist of the meta data about the claim been made.
iii. Signature: In the signature part in order to make the signature, we have to encode the header and payload, then add a secret key to it by the algorithm specified in the header part and sign that token.

V .CONCLUSION

Therefore by referring and analysis from various resources and research paper we studied the protocols and mechanisms that are used for authentication of the users across various devices and platforms. We studied the OAuth protocol architecture and working, the underlying technologies behind the working of the system such as OpenID Connect, JSON Web Tokens. We conclude that password less authentication of users is possible using the proposed solution which will eliminate the disadvantages of the existing system as well as include the features of accessibility, flexibility, security and provide freedom of choice to the users.

VI. REFERENCES

[1] A JSON Token-Based Authentication and Access Management Schema for Cloud SaaS Applications. Published in: 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud) NSPEC Accession Number: 17373975

[2] An authentication based scheme for applications using JSON web token Published in: 2019 22nd International Multitopic Conference (INMIC) INSPEC Accession Number: 19434889

[3] SoK: Single Sign-On Security – An Evaluation of OpenID Connect Published in: 2017 IEEE European Symposium on Security and Privacy Christian Mainka, Vladislav Mladenov, Jörg Schwenk Horst Görtz Institute for IT Security Chair for Network and Data Security Ruhr University Bochum.

[4] The Web SSO Standard OpenID Connect: In-Depth Formal Security Analysis and Security Guidelines Published in: 2017 IEEE 30th Computer Security Foundations Symposium Daniel Fett, Ralf Küsters, and Guido Schmitz University of Stuttgart, Germany

[5] Web single sign on authentication using SAML IJCSI International Journal of Computer Science Issues, Vol. 2, 2009