ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

A Survey on SSO (Single Sign-On) Mechanism for Mobile Apps

D. S. Baravade, S.A.Lakade, V.C. Deshmukh.

Lecturer's, Department of Computer, Ashokrao Mane Polytechnic, Vathar, Maharashtra, India

ABSTRACT: In recent years, Smartphone's, tablets, and other mobile devices, has been increased in daily life. Recently ,most of the desktop applications have a corresponding mobile version. Now, To boost the adoption of these apps on mobile devices, the user experience needs to be friendly, and secure. In advance, with business apps now available to employees on their mobile devices, enterprises must ensure they meet the highest standards of security and usability. To achieve this goal, enterprises are trying to make access across apps easier through a one-time secure authentication for users. Mobile Single sign-on (SSO) is a latest growing authentication systemfor enterprise apps . It can broadly be described as a customized home-grown solution by third party vendors targeting specific devices. In this paper, we discuss the outline of a secure SSO client solution for native enterprise apps hosted on popular mobile platforms, namely, the iOS and Android. We also briefly assess some of the different types of mobile SSO solutions in the market and their merits. Based on the architectural outline discussed, the solution formulated can help ensure a secure user experience for enterprise users. The choice of the appropriatesolution will of course vary, depending on an enterprise's individual needs.

KEYWORDS: SSO Mechanism , mobile devices, userIdentification, security analysis, SAML,OAuth lifetime

I. INTRODUCTION

Authentication plays an important role for secure communication, authentication gives assurance that two communicating parties such as user and Distributed service providers [1],[2] are authenticate each otherfor Secure communication. To maintain different pairs of identity and passwords for different serviceproviders is very difficult task , since this couldincrease the workload of both users and serviceproviders , So for that single sign-on authenticationmechanism[3]is introduced ,in which multiple service providers[4]in distributed network authenticate to legal user with single credential may able to access the services without increase the workload on networks Widely, SSO Mechanism has been available for web application[12]. Now days it is also used for mobile devices. The concept is the same, but there are four things your mobile single sign-on solution should include:

A. Revoke access at anytime

Anyone can able to revoke the access anytime when Security threats happen on mobile devices. They are more likely to be lost and more likely to be shared with someone else

B. Native app-friendly standards SAML

SAML is most commonly used standard in web apps, particularly in an enterprise environment. but, it does not work very well on mobile devices. Why? Because SAML assumes its clients to be web browsers, but the mobile native apps do not run in browsers. This also tells us the age of SAML. When it came out in 2005, iOS and Android hadn't been released yet, and Windows Mobile or Symbian only accounted for a small fraction of the overall mobile phone market. This brings us to the new OAuth 2.0 standard, which came out in 2012[3]. Instead of only considering browser use cases, OAuth 2.0 defines several flows, one of which is for mobile devices. The way OAuth 2.0 sends its token between clients and resource servers is specifically designed for mobile native apps, so they can consume the token without any workaround. Therefore, when considering single sign-on solutions for your mobile apps, make sure the

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

solution you choose supports OAuth 2.0. You'll also need SAML support, because in mobile web apps it is still useful, and it's still the most popular single sign-on standard for desktop web apps in an enterprise environment.

C. Risk-based authentication

You won't want the user to be granted access to everything in all the apps once authenticated. For example, when the user is about to do a financial transaction, you will want to use additional authentication methods to ensure better security. The mobile devices are also moving constantly, and using a one-size-fits-all policy for mobile devices is a bad idea. Risk-based authentication solves this problem and is what you should look for when considering single sign-on solutions.[11]

D.A tough and user-friendly authentication method

There's always more security risk when using single sign-on, so you want to make sure that the users aren't using weak authentication methods. But since the input on mobile devices is much harder than on desktop devices, and user experience is critical to the mobile app's success, you also want the authentication method to be as user-friendly as possible. On mobile devices, the popular ways to accomplish both goals are to use a one-time password, certificate-based authentication and multi-factor authentication. [4]

Mobile Ad Hoc Networks (MANETs) consists of a collection of mobile nodes which are not bounded in any infrastructure.Nodes in MANET can communicate with each other and can move anywhere without restriction. This non-restricted mobility and easy deployment characteristics of MANETs make them very popular and highly suitable for emergencies, natural disaster and military operations.

Nodes in MANET have limited battery power and these batteries cannot be replaced or recharged in complex scenarios. To prolong or maximize the network lifetime these batteries should be used efficiently. The energy consumption of each node varies according to its communication state: transmitting, receiving, listening or sleeping modes. Researchers and industries both are working on the mechanism to prolong the lifetime of the node's battery. But routing algorithms plays an important role in energy efficiency because routing algorithm will decide which node has to be selected for communication.

The main purpose of energy efficient algorithm is to maximize the network lifetime. These algorithms are not just related to maximize the total energy consumption of the route but also to maximize the life time of each node in the network to increase the network lifetime. Energy efficient algorithms can be based on the two metrics: i) Minimizing total transmission energy ii) maximizing network lifetime. The first metric focuses on the total transmission energy used to send the packets from source to destination by selecting the large number of hops criteria. Second metric focuses on the residual batter energy level of entire network or individual battery energy of a node [1].

II.LITERATURE SURVEY

The growth in mobile apps, particularly enterprise native apps, has highlighted the security loopholes in unsecure networks. Sensitive and confidential information like login credentials run the risk of being exposed when entered multiple times by users. The need of the hour is a robust and foolproof solution that requires the user to enter critical information only once. This simplified process will not only improve user experience, but also still trust in the minds of users as well as enterprises, thus boosting adoption of these apps.Enterprises have robust IAM infrastructures that mostly deal with desktop computing. With the spurt in the number of mobile devices, particularly in the BYOD context, mobile-specific risks have arisen, which call for a mobile-centric approach to SSO. With SSO, users do not need to repeatedly enter their credentials, introducing convenience and enhancing experience, which could eventually trigger increased adoption[6]. SSO also eliminates the need to repeatedly Transmit credentials over the network - a vulnerability since credentials are more likely to be compromised when transmitted over unsecure Wi-Fi networks. SSO also discourages users from storing credentials in unsafe apps on the device as mobile devices are often lost or stolen, thus increasing security risks. Even though SSO for desktops or web applications [6] is a standard practice, SSO for native mobile apps is still evolving. There has been considerable progress in the custom solutions and vendor products that cater to this space. In the conventional desktop world, SSO is achieved with trusted applications that share

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

information or tokens among themselves. This information exchange does not require users to repeatedly enter details, thus eliminating challenges when they try to access other apps in the same or different domain. The architecture of most mobile platforms differs significantly, but one similarity between desktop and mobile is the sandboxed nature of the apps they contain. In other words, one app is protected from another and cannot access the other's data. This feature can work well in the BYOD ecosystem, where enterprises are wary of their corporate confidential data getting leaked to other, possibly malicious, apps on an employee's personal device. However, this sandboxing of apps in a mobile device could pose a significant challenge for any SSO solution where tokens or signed claims are expected to be shared across trusted apps to enable a secure authentication and authorization experience. An alternate mechanism is thus required to share authentication tokens between apps to enable SSO. Mobile specific SSO is a pressing need, as the enhanced user experience - that the mobile platform needs to deliver – is compromised when they need to enter passwords multiple times across apps. Let us briefly look at the mobile app security infrastructures of some popular platforms, in the context of sandboxing.

A .Access Controls and Security Frameworks of Popular Platforms

1)iOS

In Apple's operating system iOS, apps are sandboxed and there is an option to encrypt sensitive data within the sandboxed area when the device is locked. The system creates a unique folder for every app. In the case of iOS v4-v7, its data and configurations were stored at '/var/mobile/Applications/{APPGUID}'. On the other hand, in iOS 8,apps are stored at 'var/mobile/data/Containers/Bundle/Application/{APPGUID1}' and the documents, and temp
files are stored at 'data/Containers/Data/Application/{APPGUID2}', thus providing a sandboxed environment for each app.[7]

2)Android

Like iOS, in the Google-developed Android operating system too apps are sandboxed and need explicit user permission to access data from other apps such as Contacts and GPS. A unique system user ID and group is assigned to every app and this Linux-based folder and permission structure is used to protect apps from each other. In a BYOD context, it is also possible to provide a corporate sandbox to devices that isolate enterprise apps and data
from personal apps.

3)Blackberry

In case of the BlackBerry OS, third party apps can, in general, only access user data (email, contacts, and calendar update) and a persistent storage location that shares data with other apps. This access can be controlled or restricted using Blackberry Enterprise Server (BES) policies, and a Mobile MDM solution. In response to growing needs, SSO has recently become part of some mobile platforms. However, many of them have limitations and do not suit the needs of enterprise native apps. Samsung, for instance, has implemented SSO in its KNOX platform. The Windows phone also includes some support for SSO.[9]
The iOS 7 SSO feature provides Kerberos-based SSO, which works efficiently in a corporate Wi-Fi network but not in a remote setup. With some improvement, it is possible to proxy Kerberos communications to simulate SSO in a remote setting. The iOS 8 offers certificate-based support for SSO which authenticates users to enterprise apps,
Allowing them to switch between enterprise apps without having to re-enter their password.
Kerberos-based authentication in many enterprises - which use AD and other LDAP services - involves entities such as protected resources, LDAP services, and devices to be a part of the same network with their clocks in sync, as Kerberos authentication includes a timestamp.
In a mobile world, when users are outside the corporate Wi-Fi network, enabling SSO involves exposing the Kerberos infrastructure outside the corporate network, which is a security risk. For the purposes of this white paper, we will only focus on a client-based SSO solution architecture. However, it is worth noting that there are several industry approaches being developed towards achieving a comprehensive SSO solution cantered around mobile. In the case of client-based libraries, standard OAuth or Open ID based tokens are distributed using client-side SSO libraries.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

Although Security Assertion Markup Language SAML may not be preferred for mobile, it is still used. These tokens leverage mobile device security features to store SSO related artifacts and have the ability to provide an authentication certificate. They have configurable policy parameters for app-based, user-based, or device-based validations. The tokens can be integrated with IAM systems or Corporate LDAPs. They can support MDM features for auditing, logging, revoking, and tracking, and have a self-service admin interface to manage policies, and configurations.[8]

B. Reviewing Standards and Protocols that define the SSO Solution

For a clear understanding of the SSO solution, a review of the available standards is necessary. In the federated identity space, the three most relevant standards are SAML, OAuth, and OpenID Connect, with OpenID being the most recent and still maturing entity.

1)SAML

SAML is an XML based standard to share authentication or authorization information between relying parties (the identity provider and service provider) to achieve SSO, Federation and ID Management. It is an XML-based token (Signed XML claim), which implies the token can be large in size. This standard relies on an HTTP binding and the transport format can be SOAP, JMS, and so on. SAML is typically used in the enterprise SSO context and supports server-side SSO where a single user agent (browser) accesses several applications within the same domain or across different ones.

2)OAuth

OAuth is a protocol to allow authorization to API resources on behalf of the resource owner. As mobile apps consume APIs, it is more relevant to the space. The tokens can be JSON, binary or SAML. Since the JSON-based tokens[7] are small in size, they are better suited for mobile. OAuth only makes HTTP bindings and is typically used to access web resources over the internet. SAML and OPEN ID (JWT-based) tokens can also be used as OAuth bearer tokens. However, the access granted is limited by scope, time, and action. Apart from server-side SSO, OIDC or OAuth also support persisted login and client-side SSO, where a single user agent or multiple user agents (multiple mobile apps) will continue to have access to the resource even after the device has been restarted or on switching between apps.

3)OIDC

OIDC is used to achieve SSO, Federation, and ID Management. It is also a JSON-based token, thus better suited for mobile devices. OIDC is an identity layer on top of OAuth 2.0 that can provide the profile information of users from the authorization servers, based on the authentication it performs. It supports RESTful APIs as it is HTTP and JSONbased.OIDC uses and extends OAuth endpoints, namely, authorization (user authentication and consent), token (to fetch and validate tokens), user information (to query identity data), registration or configuration (to manage clients), and end session (to sign out).

Let us also review the API resources or web services that need to be invoked and understand how they compare for usage in mobile.

4)SOAP

SOAP supports the XML data format. The API client authentication is defined by Web Services-Security and WS-Trust standards.

5)REST

REST supports JSON and XML data formats. It supports standard Create, Read, Update, and Delete (CRUD)
HTTP operations and OAuth-based security for client API access. REST-based access can thus be facilitated using OAuth calls.

C. Recommended Solution Architecture

In this section, we will review a client-based SSO solution leveraging an OAuth based AS for iOS and Android.

1)Server Side

In any AS implementation, we usually have token endpoints that are used to retrieve and validate OAuth tokens.The authorization and validation endpoints are RESTful services that can be invoked by native apps as part of the SSO process. The AS can be configured against the corporate trusted authentication data source (LDAP or AD) against which user credentials can be validated and authenticated.

2)Implementing OAuth AS

The OAuth AS REST service is responsible for accepting client API login requests, validating credentials, and returning tokens on authentication. This can also be configured to return user profile information in the response.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

Depending on the grant type configured for the client, the AS may return different types of tokens or authorization codes in the JSON response. The tokens are bearer tokens, which allow the client presenting it to gain access to the requested resource. An open source JSON parser like NSJSON Serialization can be used to parse the response and retrieve the tokens. For example, in the password and client validate grant type, an Access Token and Refresh Token may be returned. While an Access Token is used to access protected resources, a Refresh Token is used to retrieve a new Access Token. Typically, Access Tokens have a short validity period while Refresh Tokens have a longer ttl.

3)Introducing a Device or App in the AS Infrastructure

As a pre-requisite, the client (device or app) should be configured on the AS by assigning a grant type for the OAuth based authorization, configuring a client ID for the calling native app, and/or assigning a secret question or password to authenticate the app or device to the AS.

For example, the grant type is usually set to 'password' for validating user credentials against the AS, and is set to 'client validate' for validating a token that is presented by the calling app.

4)OAuth-enabled Resource Server

This server contains the protected REST resources to which native apps would request access. These servers are OAuth enabled, providing an integration point with the AS. When the client app makes a REST API call to access these protected resources, they package the request along with the Access Token which is validated by the RS against the AS. On successful validation, the resource is returned to the native app and the refreshed token can be
stored on the device. Session management can be handled on the RS side to avoid repeated validation calls to the AS within a given session.

III. BUILDING BLOCKS OF A CLIENT SSO FRAMEWORK

A. FOR IOS

In iOS, apps can invoke other apps using custom URL schemes. However, apps that do not require user interaction or intervention cannot run in the background. Due to the limitations of having an application running in the background in iOS, for the purpose of handling client SSO features, a client SDK approach is suggested below.

B.Client Configuration

The same AS side client configuration parameters can be set on the native client library side, to be in sync with configuration on the AS side. These client configuration parameters (such as client ID, secret, grant type, and so on)will get passed as part of every REST call to the AS token endpoints. For example, a property list file can be used to hold such configuration parameters.[8]

C.Secure Shared Storage Location

One of the pillars of a secure SSO solution is the decision to use a shared storage area where tokens and other SSO artifacts can be stored and retrieved by participating native apps. Access and Refresh tokens can be stocked in shared storage on the device, which can be retrieved by the client SSO library. For example, a keychain store can be used which can be accessed by providing a common KeyChain Group name. A common bundle seed ID for all applications is the pre-requisite for shared keychain access.

D.Client SDK Library

Enterprise apps that wish to leverage SSO functionality can bundle this client SDK into their native app code (for example, the Obj-C package) and make API calls to gain SSO advantages. Every client SSO library would have an initial routine to check and retrieve some elements (for example, presence of authentication tokens, ttl for authentication tokens, and so on) from the shared storage of valid tokens, client configurations, and so on. This initial check will help determine if the login use case needs to be triggered and packaged with the client parameters, or an SSO packet is to be sent with the valid tokens to the AS.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

1)**Single Logout**
SLO is possible by flushing the shared storage location of all tokens, and also making an API call to the token AS endpoint and invalidating the tokens, including Refresh Tokens, in its persistent table. Once done, when the client SDK checks the storage for tokens, it will return null, prompting the login use case.

2)For Android
Given the known limitations with shared storage in Android (using content providers, SD storage, and so on), a background app architecture is more suited where the app is responsible for all SSO features, including the storage and retrieval of OAuth tokens on behalf of participating apps. However, a client library is still required to help apps correspond with the background SSO app.

3)Background SSO App
This app handles the communication with the OAuth AS for login and retrieval of tokens. It also stores the tokens within the app's shared preferences data provider. Mobile apps participating in the SSO can interact with this background app using the service messenger interface.

4)SSO Client Library
This library is bundled within the code base of participating SSO apps to help provide the calling methods that invoke the background app using the service messenger interface. The Access Token retrieved from the background app is used by the applications to package within its API call to protected resources on the RS.

IV.BUSINESS AND TECHNICAL FACTORS TO BE CONSIDERED WHILE EVALUATING SOLUTIONS

A well-executed SSO strategy reduces password-related support incidents and provides users with improved convenience and highly-efficient authentication processes. A sound SSO strategy will give users fewer reasons to write down passwords. However, one password providing access to all in-scope systems can lead to compromised security. The following factors can be considered when identifying an SSO solution and checking if it is appropriate for an enterprise: n The types of apps to be supported in the SSO infrastructure, whether internal enterprise apps or software-as-a service (SaaS) n Mobile devices and platforms to be supported, which would help determine if a third-party product or custom solution should be chosen if there is no out-of-the-box native support for SSO n Need for a hybrid solution, one for internal apps and one for SaaS applications, that may add to complexity and cost n Users who are required to be covered—they could be employees, clients, and contractors—and the levels of access to be given to them n Scope of changes likely to take place in the near term n Technologies that suit the enterprise and leverage the current services and applications n Security, cost, and functionality of the solution for the enterprise.

LIMITATIONS OF SSO

SSO does not eliminate all security risks but it does help mitigate some security of themAlong with choosing the right technology, identity governance is crucial to the improvement of security through SSO n In an SSO regime, account de-provisioning does not cut off access. The user who has been de-provisioned can still access systems if he has any active sessions open on a computer at home, at the time when the SSO access is closed in office n It is very difficult to have all applications to which a user needs access support a single standard, or be mapped to a password bank. One must consider SSO only for applications requiring frequent access and not for those that are rarely used

VI. CONCLUSION

SSO mobile apps promises several advantages such as enhanced secure login, user experience, and increased adoption of enterprise apps. In a dynamic environment where remote operations are a budding cultural fact, the growth of enterprise mobile apps will only pick up pace, heightening the need to accept a solution to secure critical information, irrespective of location.SSO functionality can be successfully implemented for enterprise mobile apps using the approaches mentioned in this paper, even with the sandboxing limitations of mobile devices. For iOS, the recommended architecture is to have a client SDK bundled with mobile apps, to which all activities and callback

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 7, July 2017

functions of SSO can be delegated. The keychain store can be leveraged to share SSO artifacts and tokens across applications.

In Android, a background SSO application can perform tasks related to SSO on behalf of enterprise mobile apps. A client SDK can be bundled within the mobile apps that can use the service messenger interface to interact and call the background SSO app. The SSO app's shared preferences storage can be used to store the SSO artifacts and tokens.

This white paper has only highlighted the architectural approach towards an SSO solution. While there are many third-party products in the market that help implement the above functionality, a custom-solution along the suggested architecture can also be implemented by enterprises requiring a high level of customization. A decision on the ideal solution depends on the organization and their specific application needs. The best-fit solution can be arrived at, based on the different mobile platforms that need to be supported, the total cost of ownership of each solution, and other relevant organizational factors.

REFERENCES

1. C. Weaver and M. W. Condtry,"Distributing internet services to the network's edge," IEEE Trans. Ind. Electron.,vol. 50, no. 3, pp.404–411, Jun. 2003.
2. L. Barolli and F. Xhafa, "JXTAOVERLAY:A P2P platform for distributed, collaborative and ubiquitous computing," IEEE Trans. Ind.Electron., vol.58, no. 6, pp. 2163–2172, Oct. 2010
3. W. B. Lee and C. C. Chang, "User identification and key distribution maintaining anonymity for distributed computer networks," Computer. Syst. Sci. Eng., vol. 15, no. 4, pp. 113–116, 2000.
4. W. Juang, S. Chen, and H. Liaw, "Robust and efficient password authenticated key agreement using smart cards," IEEE Trans. Ind. Electron.,vol. 15, no. 6, pp. 2551–2556, Jun. 2008.
5. X. Li,W. Qiu, D. Zheng, K. Chen, and J. Li, "Anonymity enhancement on robust and efficient password-authenticated key agreement usingsmart cards," IEEE Trans. Ind. Electron., vol. 57, no. 2, pp. 793–800,Feb. 2010.
6. M. Cheminod, A. Pironti, and R. Sisto, "Formal vulnerability analysis of a security system for remote fieldbus access," IEEE Trans. Ind. Inf.,vol. 7, no. 1, pp. 30–40, Feb. 2011.
7. A. Valenzano, L. Durante, and M. Cheminod, "Review of security issuesin industrial networks," IEEE Trans. Ind. Inf., vol. PP, no. 99,2012, DOI 10.1109/TII/2012.
8. T.-S. Wu and C.-L. Hsu, "Efficient user identification scheme with key distribution preserving anonymity for distributed computer networks," Computers and Security, 23(2): 120-125,2004.
9. A. C. Weaver and M. W. Condtry, "Distributing Internet services to the network's edge," IEEE Trans. Ind. Electron., Vol. 50, No. 3, pp. 404-411, Jun.2003.
10. L. Lamport, "Password authentication with insecure communication," Commun. ACM, Vol. 24, No. 11, pp. 770-772, Nov. 1981.
11. C.-C. Chang and C.-Y. Lee, "A secure single sign-on mechanism for distributed computer networks," IEEE Trans.Ind. Electron., vol. 59, no. 1, pp. 629–637, Jan. 2012.
12. L. Harn and J. Ren, "Generalized digital certificate for user authentication and key establishment for secure communications," IEEE Trans. Wireless Commun., vol. 10,no. 7, pp. 2372–2379, Jul. 2011.
13. Guilin Wang,Jiangshan Yu, and Qi,"Security analysis of a single sign-on mechanism for distributed computer networks,"IEEE Trans. Industrial Informatics.,vol. 9,no. 1,Feb 2013.