Characterization of Web Single Sign-On Protocols

Single Sign On (SSO) protocols are today integrated in millions of Web services so end users can authenticate to a third-party identity provider (IdP) to access multiple services. IdPs normally provide integration tools that hide almost all implementation details and allow developers to implement SSO in some minutes. Such integration tools along with cumbersome protocol specifications result in developers without a clear view of the underlying SSO protocol.

Victoria Beltran

Abstract

Single Sign On (SSO) protocols are today integrated in millions of web services so end users can authenticate to a third-party identity provider (IdP) to access multiple services. IdPs normally provide integration tools that hide almost all implementation details and allow developers to implement SSO in minutes. Such integration tools along with cumbersome protocol specifications result in developers without a clear view of the underlying SSO protocol. This article presents a conceptual characterization of web SSO protocols through their assertions and their features that help preserve the privacy of the user resources involved in SSO.

Introduction

Web services need to verify the identity of their users to guarantee user security and privacy, or offer a personalized user experience. Web users have traditionally had to replicate the same personal and contextual data (e.g. name, address, bank account, telephone number, etc.) to create their user accounts. Such site-centric identity management forces users to replicate their identities all over the web and to authenticate to services, each with their own credentials, one after the other.

The well known Single Sign-On (SSO) approach permits users to log in once and to access multiple services by delegating user authentication from service providers to authentication services [1]. Service providers and authentication services form identity federations that enable importing identity information from one domain to another [2].

On the web, identity federation protocols like OpenID allow web services to redirect users to third-party services, which are known as identity providers (IdPs) for authentication [3]. The IdP tells the web service who the present user is by delivering an identity assertion that contains at least the user's unique identifier and may include other identity-related attributes such as name, address, and email.

In SSO, web services only rely on identity information to grant users access. Although the HTTP flows of identity federation protocols have already been extensively studied, these protocols' data model for assertions has not yet been sufficiently analyzed. Only in the framework of Web Real-Time Communications (WebRTC), has the impact of identity assertions on user privacy been addressed [4].

This article provides a characterization of SSO assertions and interprets four protocols through their assertions: OpenID, OAuth2.0, OpenID Connect, and BrowserID. This article also conceives a high-level view on how these protocols' design choices impact user privacy.

The Web Flavor of Identity Federation

Identity federation and SSO originally came into being in the security domains of corporate and governmental networks. Corporate users can transparently access any service in the security domain of the authentication server to which they have authenticated. Kerberos is the de facto authentication protocol for centralized authentication services. For inter-domain scenarios, the SAML/SOAP paradigm is deeply integrated in corporations, government networks, and verticals such as healthcare, banking and education. Similar to SAML, WS-* (e.g. WS-Federation and WS-Security) is an extensive set of OASIS specifications for identity federation promoted by Microsoft.

Although SAML and WS-* specifications enable importing identity assertions from one domain to another, their complexity and strong requirements on authorization rules and trust relationships between services and authentication servers have not fit in with the ever-changing World Wide Web (WWW). Identity federation for Internet services has a different nature than its corporate counterpart. There are no centralized security servers but millions of APIs, mashups, and applications that liberally communicate through basic exchanges of HTTP requests and responses, based on JSON or XML. Thus, flexible REST-based protocols such as OpenID[1] and OAuth2.0 [5], that simplify loosely-coupled identity federation and speed up application development, have found their way onto the web. Web identity federation protocols mainly define how to deliver identity information from IdPs to web services that are called relying parties (RPs) in web SSO terminology. These protocols are independent from the authentication process carried out by the IdP. Indeed, they do not provide SSO by themselves. The SSO experience is given by the user browser's HTTP cookie storage. As long as the IdP's session cookie is active, the user will not have to re-authenticate to log on to a web service. Nevertheless, for the sake of brevity, in the rest of this article I refer to web identity federation protocols as Web SSO protocols.

Although OpenID was the first Web SSO protocol to be widely adopted by Internet service providers by 2005, OAuth gained ground rapidly due to its ability to delegate access to protected APIs along with the user's identity. In fact, the biggest driver of OAuth adoption was the release of a set of OAuth-based Facebook APIs, known as Facebook Connect, that allow web services to log users in and get access to social network

The author is with Orange Labs.

[1] OpenID, http://openid.net/specs/openid-authentication-2_0.html

features based on the users' Facebook identities. OAuth is a generic delegated authorization protocol that provides an access token to a client so that it can access a protected resource, based on the permission of the resource owner. Although OAuth is not a SSO protocol and leaves the resources being authorized out of scope, it has been massively used to build authentication and identity protocols.

The fact that OAuth does not specify how to convey authentication and identity information has led to interoperability and even security problems. Each IdP defines their own customized identity layer on top of OAuth to provide delegated access to identity information. Authorization and user authentication are often confused, resulting in vulnerable implementations.[2] To address these limitations, OpenID Connect (OIDC)[3] emerged to provide an interoperable authentication and identity framework on top of OAuth. Finally, BrowserID[4] is a browser-side API specification for SSO based on user certificates.

Native Apps

OAuth has become the de facto protocol for implementing federated authentication and authorization functionality in mobile applications. Since OAuth evolved during the same period that the adoption of mobile applications exploded (by 2007–2008), this protocol was the natural choice for merging authentication and authorization in mobile platforms.

Mobile RPs can integrate third party login natively (by means of the SDKs provided by some IdPs) or by the same web-based protocol flows (through the mobile browser or WebViews) implemented by web RPs. On the web, SSO relies on the browser's cookie storage to verify whether the user has an active session with the IdP. On mobile platforms, mobile browsers also keep a cookie storage that can be shared by mobile applications for SSO. However, redirecting users to mobile browsers for login provides a poor user experience. Furthermore, if something goes wrong with the login flow, the user might not be redirected back to the application. Native or WebView-based implementation of third party login offers a much better user experience. However, mobile applications are isolated and do not share any space of user authentication proofs, and neither do WebViews.

A shared space of authentication proofs for mobile SSO can be provided by native SSO applications or platform-integrated SSO. Some IdPs are already providing native SSO applications that handle protocol flows and authentication proofs on behalf of mobile applications. When a mobile app wishes to authenticate a user, it will invoke the SSO native application of the user's IdP, instead of implementing a third party login by itself (by an SDK, WebView, or the mobile browser). Nevertheless, to install one SSO app for each IdP to which the user wishes to authenticate can be tedious. Thus, there already exist third party providers that offer a single native app for SSO with multiple IdPs. For the sake of standardization, the recent Native Applications (NAPPS) Working Group[5] is defining a profile of OpenID Connect (OIDC) for a native token-issuing agent[6]. This profile will provide native SSO through standardized interfaces to mobile applications. Regarding platform-integrated SSO, some solutions already exist. Samsung Knox and Apple iOS (from iOS7 releases) implement Kerberos to address the mobile security needs of enterprise users. For customer services, both Google and Microsoft integrate OAuth into Android and Microsoft's Windows Phone 8.1 to enable authenticating to their respective IdPs (i.e. Google Sign-In and Microsoft Live Connect SSO).

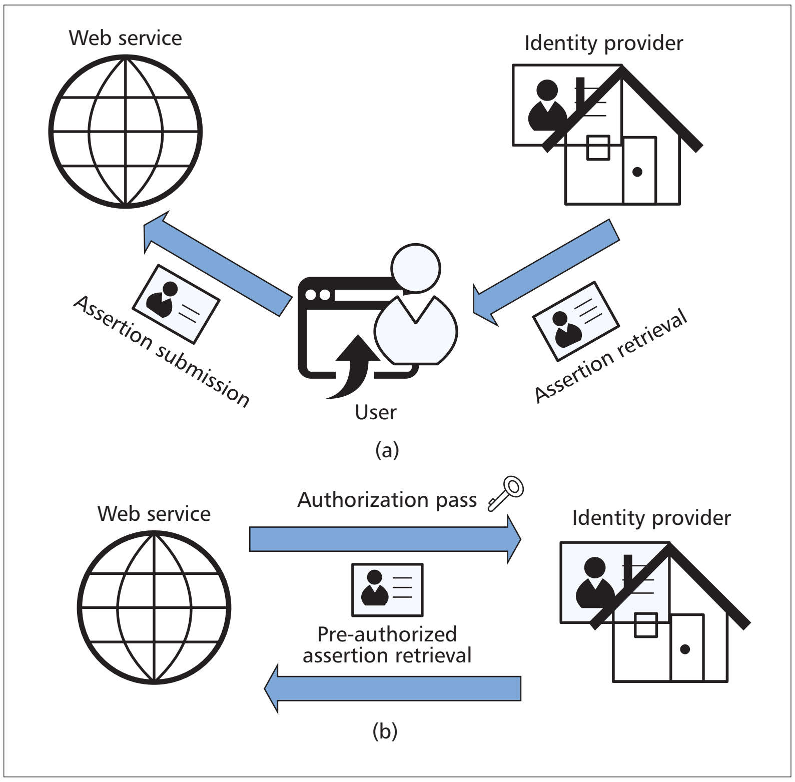

Figure 1. Logical steps in the immediate (a) and pre-authorized (b) delivery modes of SSO assertions. Immediate assertions (a) are sent directly by the IdP, without any previous authorization, and involve two steps: assertion retrieval, in which the assertion arrives to the browser, and assertion submission, in which the assertion is sent to the Web service. Pre-authorized assertions (b) require the requester to possess an authorization assertion.

SSO Assertions

The ultimate goal of any SSO system is the delivery of identity assertions that allows users to sign into web services without re-authenticating. The format of identity assertions depends on the underlying SSO protocol. OpenID encodes user identity information as standard plaintext key-value attributes.[7] OIDC defines two different JSON objects for user identity information: ID tokens and UserInfo objects. Both options are JSON Web Token (JWT) objects that can include the same identity attributes, which are called claims in OIDC terminology. Both OpenID and OIDC define a set of standard "attributes" and "claims," respectively, but also allow defining non-standard identity information. In BrowserID, user identity information is encoded in a JSON Web Signature (JWS) object that only contains an email address that identi-

[2] J. Richer, User authentication with OAuth 2.0 http://oauth.net/articles/authentication/

[3] OpenID Connect, http://openid.net/connect/

[4] BrowserID, https://github.com/mozilla/idspecs/blob/prod/browserid/index.md

[5] NAPPs working group, http://openid.net/wg/napps/charter/

[6] OIDC Native Application Token Agent Core 1.0 web SSO v2, http://openid.bitbucket.org/draft-native-application-agent-core-01-working-draft.html

[7] OpenID Attribute Exchange 1.0 , http://OpenID.net/specs/OpenID-attribute-exchange-1_0.html

Protocol	Name	Type	Format	Delivery mode
OpenID	Positive assertion[1]	Identity	(key, value) pairs	I
OAuth2.0	Authorization code	Authorization	Opaque string	I
	Access token	Authorization	Opaque string	I[2] A[3]
OpenID Connect	Authorization code	Authorization	Opaque string	I
	Access token	Authorization	Opaque string	I[4] A[5]
	ID token	Authentication and identity	JWT/JWS/JWE	I[6] A[7]
	UserInfo	Identity	JWT/JWS/JWE	A
BrowserID	Backed identity assertion	Identity	JWS	I

[1] The "positive assertion" term is used to refer to the set of attributes that results from a successful user authentication event
[2] OAuth2.0 implicit flow
[3] OAuth2.0 authorization code flow
[4] All the OIDC authorization flows except "code" and "code id_token"
[5] All the OIDC authorization flows except "id_token token"
[6] All the OIDC authorization flows except "id_token token" and "id_token"
[7] All the OIDC authorization flows except "code" and "code token"

Table 1. Assertion characteristics in SSO protocols (in immediate (I) and pre-authorized (A) delivery modes).

fies the user. OAuth-based IdPs define their own identity attributes and their format since OAuth leaves the format of the resources being authorized unspecified.

Apart from identity assertions, SSO protocols can exchange authentication and authorization assertions. An authentication assertion tells a web service about the user's authentication event. For example, OIDC ID tokens can specify the exact time when the user authenticated and the authentication method. An authorization assertion serves as a pass given by the IdP to an RP to retrieve another assertion. SSO assertions can be classified, no matter their nature (i.e. authentication, identity, and authorization), based on their delivery mode, i.e. immediate or pre-authorized modes, as depicted in Figure 1. Table 1 summarizes the characteristics of the assertions provided by OpenID, OAuth, OIDC, and BrowserID.

A View of Web SSO Protocols Through Their Assertions

SSO protocols differ in their assertions' types, formats, and delivery modes. Figure 2 shows how the user browser's software components, and the IdP's and RP's servers, interact to deliver SSO assertions to the RP once the user has authenticated (and authorized the RP) in OpenID, OAuth2.0, OIDC, and BrowserID.

As shown in Fig. 2a, OpenID delivers immediate identity assertions by means of an HTTP redirection response that contains the identity information in the response body.

OAuth2.0 only defines how to obtain authorization assertions that are called access tokens, which can be used to retrieve server resources based on the resource owner's authorization. Thus, OAuth-based IdPs only provide pre-authorized identity assertions by requiring the RP to possess an access token. Access tokens can be immediate, or pre-authorized by another (immediate) assertion called authorization code. The latter case is referred as the authorization code flow and is depicted in Fig. 2b. An authorization code is attached as a parameter of the redirection URL in the HTTP redirect response sent by the IdP. The RP exchanges the authorization code for an access token; the pre-authorized access token is included in a JSON object in the HTTP response's body. When an access token is immediate, it is directly sent to the Web service (without a previous authorization code). Figure 2c shows this flow, which is referred to as the OAuth2.0 implicit flow. The IdP includes the token as a fragment of the HTTP response's redirection URL. The redirection request will then download a script that retrieves the token from the redirection URI and sends it to the server-side RP.

OIDC defines an identity layer on top of OAuth2.0 that enables web services to obtain immediate and pre-authorized identity assertions. UserInfo objects are identity assertions that are always pre-authorized by an access token (see right part of Fig. 2d) and are included as JSON objects in the IdP's response to the RP. As in OAuth2.0, an access token can be immediate or pre-authorized by an authorization code. ID tokens can also contain identity assertions and can be retrieved in a pre-authorized or immediate mode. Figure 2d shows the OIDC authorization code flow that provides ID tokens and access tokens pre-authorized by authorization codes. Other OIDC flows combine immediate and pre-authorized access tokens and ID tokens. An immediate ID token is sent immediately after an authentication request as a URL fragment of the HTTP redirection response (as in the OAuth2.0 implicit flow for access tokens).

BrowserID flows for delivering identity assertions are different from those of the other three protocols, as shown in Fig. 2e. Although BrowserID identity assertions are immediate, they are not sent directly to the RP. The IdP sends a normal response to the user browser rather than a redirect response toward the RP. The IdP and RP are therefore completely disconnected. Indeed, the IdP does not reply with the final user identity assertion for the RP, but with a user certificate. The user certificate is a JWS object that contains the user's email address and public key signed by the IdP. When the user browser receives the user certificate, it generates another JWS object that includes the target service's URL. The browser signs this object, which is called the user assertion, with the user's private key and passes this assertion along with the user certificate to the client-side RP code on the browser. A BrowserID identity assertion (so called user backed identity assertion) is therefore composed of two JWS objects: one generated by the IdP (the user certificate) and another generated by the browser (the user assertion).

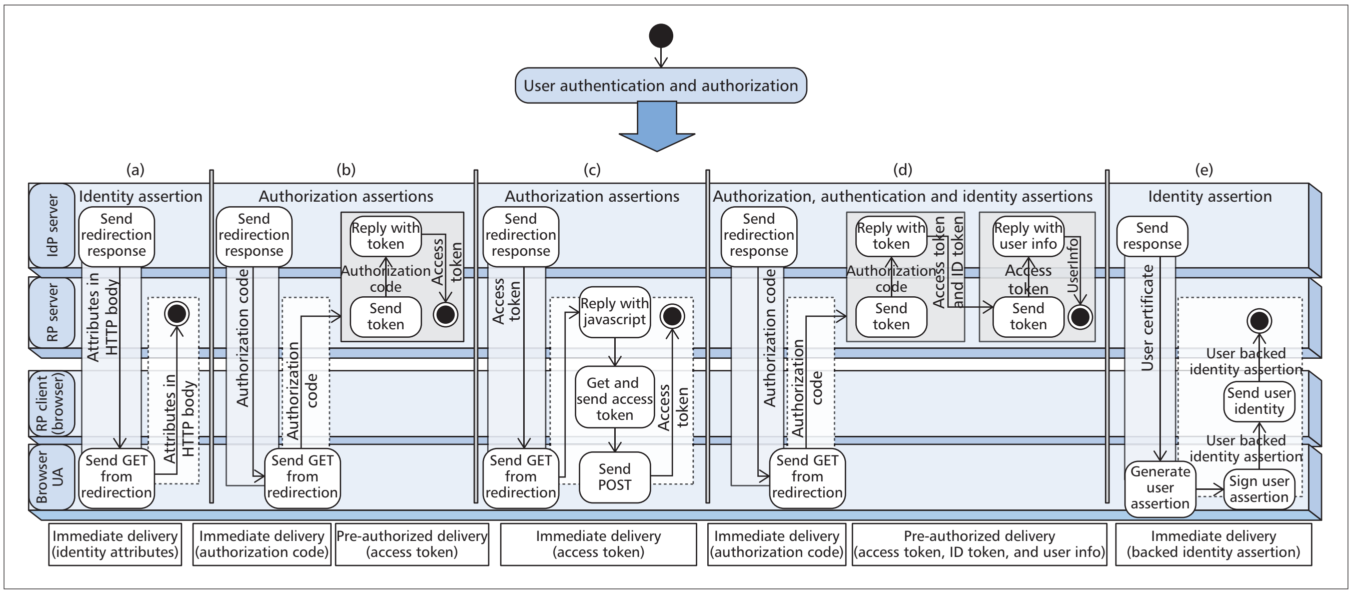
User authentication and authorization
(a)
(b)
(c)
(d)
(e)
IdP server
RP server
RP client (browser)
Browser UA
Identity assertion
Authorization assertions
Authorization assertions
Authorization, authentication and identity assertions
Identity assertion
Send redirection response
Attributes in HTTP body
Send GET from redirection
Attributes in HTTP body
Authorization code
Reply with token
Authorization code
Send token
Access token
Access token
Reply with javascript
Get and send access token
Send POST
Reply with user info
Access token and ID token
UserInfo
Send response
User certificate
Generate user assertion
Sign user assertion
User backed identity assertion
Send user identity
Immediate delivery (identity attributes)
Immediate delivery (authorization code)
Pre-authorized delivery (access token)
Immediate delivery (access token)
Immediate delivery (authorization code)
Pre-authorized delivery (access token, ID token, and user info)
Immediate delivery (backed identity assertion)

Figure 2. How SSO assertions are retrieved from the IdP (grey boxes) and submitted to the RP (clearer boxes) when a user is authenticated, in OpenID (a), OAuth2.0 authorization code flow (b), OAuth2.0 implicit flow (c), OIDC authorization code flow (d), and BrowserID (e). For OAuth2.0 and OIDC, RP server integrates all the logical endpoints (e.g., authorization, token and UserInfo endpoints).

User Privacy in Web SSO

An IdP asserts the veracity of a user's identity assertion, thereby ensuring that such assertion belongs to the user that has been correctly authenticated. While the user authenticates to the IdP, generally by proving possession of something that they know (e.g. a password) and/or have (e.g. a SIM card), an identity assertion is the only authentication proof that the web service needs to grant the user access, as depicted in Fig. 3.

Compared to traditional client-server authentication systems, third party login entails more security risks, i.e. there are simply more resources to protect. Figure 3 shows that in a single SSO authentication event the user gets access to two private resources, i.e. their accounts at the IdP and at the web service. Moreover, identity assertions themselves can entail sensitive information, which can motivate attackers to steal or eavesdrop on them.

User privacy does not only concern the confidentiality of the user information stored in the RP's and IdP's user accounts and in identity assertions, but also the protection of user activity on the web. User privacy in a SSO system depends on the underlying SSO protocol's functionality, architecture, and user identification scheme.

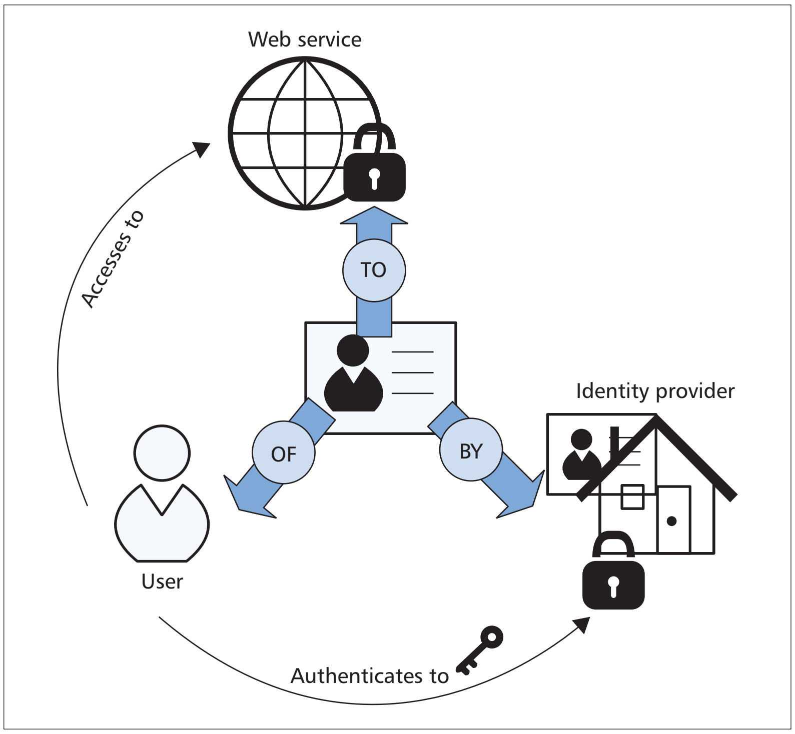

Figure 3. Triple conceptual relation of SSO assertions.

User Accounts, Assertions, and the User: The Three Points of Attack

SSO systems take part in a complex web ecosystem that exposes them to a variety of attacks that can break user privacy. Web application vulnerabilities (e.g. no transport layer protection, Cross-Site Request Forgery (CSRF), and Cross-site scripting (XSS)), the misuse of opaque SDKs provided by IdPs, and the misunderstanding of the underlying SSO protocol, result in unsecure SSO implementations [6–8].

Web attackers can take advantage of unsecure SSO systems to break user privacy by illegitimately obtaining user information from some of the user information sources in a SSO system: the user accounts at the RP and IdP, the identity assertion, and the user. Web attacks can be classified into user impersonations toward the RP and phishing attacks through fake RPs, through fake IdPs, or even through fake RP user accounts. Table 2 shows the sources of user information that are compromised in each possible attack.

User impersonation attacks aim to get control

	RP's user account	IdP's user account	Identity assertion	Own user
User impersonation	X		X	
Fake IdP	X	X		X
Fake RP			X	X
Fake RP account				X

Table 2. SSO attacks and compromised user information sources.

of the RP's user account by reusing a stolen or eavesdropped user identity assertion against a benign RP. If an attacker succeeds in impersonating the user, it will then be able to read any personal information stored on the user account. User impersonation has proven its success in SSO systems mainly due to application vulnerabilities [6, 8]. In OpenID, man-in-the-middle vulnerabilities can lead to user impersonation attacks [9]. BrowserID has also been proven vulnerable to identity forgery and user impersonation [8] [11]. The OAuth2.0 implicit flow is inherently vulnerable to user impersonation attacks since access tokens are not bound to the RP to which the token was issued [5]. Thus, an RP cannot be aware of whether an attacker replaced the real token in the IdP's HTTP response with another one (i.e. one previously stolen or eavesdropped, or even legitimately emitted to a malicious RP[8]). For mobile applications, bad practices can make OAuth-based third party login vulnerable to user impersonation. In [12], almost 60 percent of considered implementations are found to be faulty and vulnerable to attacks, mainly due to the local storage of RP secrets, the usage of the OAuth2.0 implicit grant, or unsecure mobile redirections. If an attacker succeeds in stealing the secret from the client-side RP, it may be able to retrieve the user's access token from the IdP (breaking the security of the OAuth authorization code flow) and use it for user impersonation attacks. iOS custom schemes and Android Intent mechanisms simulate the HTTP redirections that are fundamental for any web SSO protocol. These mechanisms allow mobile applications to register custom URI schemes for themselves. Thus, an IdP can send assertions to the URI with the custom scheme of the RP that sent the request. Although these mechanisms resemble web browser redirection, most of the time they are unsecure because mobile applications cannot be globally and uniquely identified. Thus, an IdP cannot verify that an access token is sent to the RP that requested it. Mobile OAuth implementations are only secure in the Android OS under the condition that they are native and the IdP verifies the RP mobile application's developer key hash [12]. Web-based implementations in Android are unsecure because the developer key hash can only be verified using a native mobile application. For iOS, third party login is always vulnerable to user impersonation attacks, since it is not possible to guarantee that access tokens are sent to the RP that requested them. The redirection behavior for a custom scheme is undefined if two applications registered to it.[9]

A fake IdP can imitate the appearance of a benign IdP's web site and trick the user into giving their account credentials, which is well known as phishing. Having a look at Fig. 3, it is easy to realize that the loss of control on the IdP's user account is the worst security risk in a SSO scenario. The attacker will be able to sign into the IdP as the victim user and hence to impersonate the user in any RP. Phishing attacks have largely exploited web vulnerabilities such as XSS. The redirection mechanism of OpenID [9] and the characteristics of BrowserID windows [10] make these protocols especially vulnerable to IdP impersonation attacks.

A fake RP attack aims to redirect the user, after they have authenticated to their IdP, to a malicious RP's web site instead of the RP to which the user is authenticating. If the user does not realize that the website is fake, the malicious RP can trick the user by getting them to enter confidential information. The OpenID realm spoofing attack abuses the redirection mechanism in OpenID to perform this attack [9]. The malicious RP, rather than the RP to which the user thinks they have signed in, will receive the identity assertion. Thus, the malicious RP may be able to impersonate the user toward other RPs in the presence of vulnerabilities between the benign RPs and IdPs (e.g. the lack of contextual binding such as nonce between authentication request and responses).

Finally, a fake RP user account is the result of a user session swapping attack by which the attacker supplants the victim user's SSO credentials by its own credentials. Thus, the victim user logs in the RP without noticing that he is registered on an account under the attacker's control rather than their own account. Any personal information entered by the user will therefore be accessible to the attacker. In OAuth2.0 and OpenID, the lack of contextual binding in assertions makes session swapping attacks possible [6, 8]. In BrowserID, vulnerabilities of login persona can be exploited to inject an attacker's certificate [11].

Protocol Properties

Table 3 shows user privacy properties on user information and activity, and the fulfillment of these properties by Web SSO protocols.

Privacy on User Information: The first property, i.e. free choice of IdP, will determine the identity federation protocol to use. Thus, it affects the overall privacy of both user information and activity. All the protocols except OAuth provide freedom of choice of IdP since they support IdP discovery.

Regarding private user information, the first four properties are concerned with the contents of identity assertions. OpenID, OIDC, and BrowserID guarantee the integrity of the exchanged assertion. In OpenID, IdPs rely on a shared symmetric key to sign the identity assertion. This key is negotiated between the IdP and RP by the Diffie-Hellman algorithm. OIDC and BrowserID sign identity assertions using JWS. OIDC is the only protocol that enables assertion confidentiality by encrypting assertions using JSON Web Encryption (JWE) with the cryptographic material registered by the RP.[10] Although the other protocols do not

[8] http://www.thread-safe.com/2012/01/problem-with-oauth-for-authentication.html

[9] https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/AdvancedAppTricks/AdvancedAppTricks.html

[10] OpenID Connect Dynamic Registration, http://openid.net/specs/openid-connect-registration-1_0.html

provide built-in solutions for data confidentiality, transport layer encryption (i.e. HTTPS certificates) may be used. None of the protocols support anonymous identity assertions that allow users not to disclose their identities while asserting that they have authenticated. Recyclable user identities can also compromise the privacy of user information. When a user abandons an identifier, it may become available to others. The new owner of the recycled identifier then will be able to authenticate as the previous owner to all the RPs that only rely on user identifiers to identify user accounts, thereby getting access to all user information stored in the previous owners' accounts at these RPs. Both BrowserID and OpenID rely on public identifiers (emails and OpenID URLs, respectively) that can potentially be recycled by IdPs [9, 10].

The last two privacy properties of user information are concerned with protocol functionality rather than the content of identity assertions. Selective disclosure refers to the protocol's capacity to allow the user to select the information to send out to the web service. The OpenID attribute exchange specification determines that the user can choose from a series of optional attributes to be disclosed. OIDC and OAuth2.0 provide scopes that indicate the information requested to the IdP. Nonetheless, the RP rather than the user decides these scopes. BrowserID does not provide more user information apart from their identity, and hence this property is not applicable. Single Log Out is the protocol capability to synchronize logout events between RPs and IdPs. If users are not aware that their sessions at RPs are still active when they sign out of their IdP (or vice versa), user privacy may be compromised in shared devices. When the user leaves, any other user that can access the same browser and will be able to access the user accounts that have not been terminated [13]. Only OIDC tackles the process of logging the user off when they finish the session at the RP or IdP. The behavior of logout in BrowserID has been especially criticized[11] [10, 11]. When the user logs out of the IdP, the user certificate installed on the user browser is not deleted. Since this certificate is used by BrowserID to automatically generate user assertions, anyone that has access to the same browser can sign into web services as the user certificate's owner. Furthermore, since IdPs are not aware of the RPs to which assertions are sent, they cannot notify the RPs when the user logs out.

On mobile applications, the back-end nature of OAuth and OIDC authorization brings out privacy concerns. OAuth and OIDC access tokens are independent from the mobile app and device that requested them. Once a user authorizes an app (i.e. the client-side RP), the app's back-end servers (i.e. the server-side RP) can retrieve the user's information as long as the user's access token does not expire, even after the user uninstalls the application. Front-end authorization should therefore be a future privacy property of SSO protocols for mobile platforms in order to limit the disclosure of user information to mobile apps [14].

	OpenID	OAuth2.0	OIDC	BrowserID
Free choice of Idp	✓	N/A	✓	✓
USER INFORMATION				
Assertion integrity	✓	×	✓	✓
Assertion confidentiality	×	×	✓	×
Anonymous identity assertion	×	N/A	×	×
Non-recyclable user identifiers	×	N/A	✓	×
Selective disclosure	✓	RP request	RP request	N/A
Single log out	×	N/A	✓	×
USER ACTIVITY				
Unobservability	×	×	×	✓
Unlinkability by different RPs	×	N/A	✓	×
Unlinkability by the same RP	×	N/A	×	×

Table 3. User privacy properties in SSO protocols. Since OAuth is an authorization rather than an identity federation protocol, many of the privacy features are not applicable (N/A) to this protocol (i.e., they depend on each IdP's OAuth-based identity layer).

Privacy of User Activity: Privacy of user activity depends on how users are identified in identity assertions and how these assertions are delivered. Unobservability means that the IdP is incapable of tracking the users as they visit web services. BrowserID is the only one to provide unobservability, since it does not involve the web service communicating with the IdP to obtain identity assertions. Identity unlinkability is the inability of web services to track user activity based on the user's identity assertions across communications [15]. It is possible to differentiate between identity unlinkability by the same RP and by different RPs. In the former, a web service cannot correlate a user's identity assertions across different login events. Conversely, identity unlinkability by different RPs means that a web service is able to correlate a user across login events but different web services cannot collaboratively link the same user's identity assertions. Identity assertions must contain anonymous and pseudonymous identifiers[12] in order to provide identity unlinkability by the same RP and by different RPs, respectively. None of the considered protocols can fulfill identity unlinkability by the same RP because none of them provide anonymous user identifiers. OIDC is the only protocol that defines the use of pairwise pseudonymous identifiers that are different for every web service. Thus, the unlinkability by different RPs' property is marked for OIDC in Table 3 (analysis techniques to correlate assertions based on their contents are not considered). BrowserID identity assertions can always be correlated based on the user's email address. Likewise, OpenID user identifiers are public URLs or eXtensible Resource Identifiers (XRIs).[13] However, the "directed identity" term[14] emerged in the OpenID community to refer to pseudonymous OpenID identifiers that provide unlinkability by different RPs. Nevertheless, since this type of

[11] https://groups.google.com/forum/#!topic/mozilla.dev.identity/Wm0Cd-rjczc

[12] Pseudonymous identifiers conceal a user´s identity but allow them to be recognised on a return visit

[13] XRI 2.0 is an OASIS specification aimed to be a standard syntax and discovery format for abstract identifiers that was discontinued in 2008, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xri

[14] http://barelyenough.org/blog/2007/12/openid-20s-killer-feature/

As of now, OIDC is the protocol that can satisfy the most privacy and hence developers should always choose OIDC in preference to OAuth2.0 and OpenID. Although BrowserID lacks many desirable privacy properties, it is the only one that provides unobservability and hence it should be chosen if this property is paramount.

OpenID identifier is not included in the OpenID specification, the unlinkability by different RPs' property is not marked for OpenID in Table 3.

Conclusions

This article has provided an assertion-driven analysis of web SSO protocols to help developers assessing the nature of the flows and data items of these protocols. The article highlights the importance for service developers to be aware of the user resources involved in SSO login events and how these resources may be compromised. Before developing a SSO system, the requirements for user privacy should be determined to choose the underlying SSO protocol that best meets them.

None of the studied protocols address all the user privacy concerns. Indeed, some privacy properties are contradictory and hence can hardly be implemented by the same protocol. The most illustrative example is unobservability and Single Log Out. The former prevents the IdP from tracking the user's activity through their login events to web services. This property involves the IdP and RP being incapable of knowing each other. Thus, unobservability makes Single Log Out impossible since the IdP cannot communicate user logouts to the RP (and vice versa). None of the protocols provide anonymous identifiers and hence web services are always able to track users through login events (i.e. unlinkability property is not satisfied). SSO in mobile apps is especially vulnerable. iOS mobile apps are not secure because apps cannot be uniquely identified. Moreover, none of the protocols offer a suitable authorization model for mobile environments that grants authorization to mobile apps rather than back-end servers.

The article aims to bring out the need for further improvements in user privacy in SSO on the web. As of now, OIDC is the protocol that can provide the most privacy and hence developers should always choose OIDC before OAuth2.0 and OpenID. Although BrowserID lacks many desirable privacy properties, it is the only one that provides unobservability and hence it should be chosen if this property is paramount.

Acknowledgment

I thank Emmanuel Bertin for his valuable suggestions on this work.

References

[1] J. De Clercq, "Single Sign-On Architectures," *Infrastructure Security*, Springer LCNS, vol. 2437, 2002, pp. 40–58.

[2] A. Perez-Mendez *et al.*, "Identity Federations Beyond the Web: A Survey," *IEEE Commun. Surveys & Tutorials*, vol.16, no. 4, pp. 2125–41.

[3] L. Lynch, "Inside the Identity Management Game," *IEEE Internet Computing*, no. 5, 2011, pp. 78–82.

[4] V. Beltran, E. Bertin, and N. Crespi, "User Identity for WebRTC Services: A Matter of Trust," *IEEE Internet Computing*, no. 6, Nov. 2014, pp. 18–25.

[5] D. Hardt, "The OAuth2.0 Authorization Framework," IETF RFC 6749, Oct. 2012.

[6] S.T. Sun and K. Beznosov, "The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems," *Proc. 2012 ACM Conf. Comp. and Commun. Security*, Raleigh, USA, Oct. 2012.

[7] R. Wang *et al.*, "Explicating SDKs: Uncovering Assumptions Underlying Secure Authentication and Authorization," *Proc. USENIX Security Symp.*, 2013, pp. 399–14.

[8] G. Bai *et al.*, "AUTHSCAN: Automatic Extraction of Web Authentication Protocols from Implementations," *Proc. Network and Distributed System Security Symp.*, 2013.

[9] B. Van Delft and M. Oostdijk, "A Security Analysis of OpenID," *Policies and Research in Identity Management*, Springer Berlin Heidelberg, 2010, pp. 73–84.

[10] M. Hackett and K. Hawkey, "Security, Privacy and Usability Requirements for Federated Identity," *Proc. Wksp. Web 2.0 Security & Privacy*, v2012.

[11] D. Fett, R. Küsters, and G. Schmitz, "An Expressive Model for the Web Infrastructure: Definition and Application to the Browser ID SSO System," *Proc. 2014 IEEE Symp. Security and Privacy (SP '14)*, Washington, DC, USA, pp. 673–88.

[12] E. Y. Chen *et al.*, "OAuth Demystified for Mobile Application Developers," *Proc. 2014 ACM SIGSAC Conf. Comp. and Commun. Security*, pp. 892–903.

[13] S. Suoranta *et al.*, "Logout in Single Sign-On Systems: Problems and Solutions," *J. Information Security and Applications*, vol. 19, no. 1, 2014, pp. 61–77.

[14] M. Nauman *et al.*, "POAuth: Privacy-Aware Open Authorization for Native Apps on Smartphone Platforms," *Proc. ACM 6th Int'l. Conf. Ubiquitous Information Management and Communication*, Kuala Lumpur, Malaysia, Feb. 2012, no. 60.

[15] E. Birrell and F. B. Schneider, "Federated Identity Management Systems: A Privacy-Based Characterization," *IEEE Security & Privacy*, vol. 11, no. 5, October 2013, pp. 36–48.

Biography

Victoria Beltran (vicbelma@gmail.com) has been a senior researcher on Internet services and identity protocols at Orange Labs in France. She holds a Ph.D in telematics engineering from the Universitat Politècnica de Catalunya in Barcelona. She has approximately 10 years experience in telecommunications research and practice in different research institutions such as Bell-Labs Alcatel-Lucent (Murray-Hill), Institut Mines-Telecom (Paris), and Columbia University (New York). She is currently an associate teacher and senior researcher at the University of Murcia.