Web Authentication: The next step in the evolving identity eco-system?

Harry Halpin
CSAIL/MIT
World Wide Web Consortium (W3C) Boston, USA
Email: hhalpin@w3.org

***Abstract*—Currently, the identity eco-system on the Web is fragmented between a number of different flows for authorization with no standardized high-security authentication mechanism outside of usernames-passwords. Current identity solutions such as OpenID Connect and BrowserID are on an abstract level just two different authorization flows that differ across a number of criteria such as privacy. We also detail a number of well-known attacks against each approach. So the "client offline/server-to-server" authorization flow of the OAuth-based approach (OpenID Connect) is actually complemented by the "online client-to-server" authorization flow from BrowserID, each being more or less effective depending on the particular use-case at hand. Finally, we sketch how combining either of these flows with the upcoming W3C Web Cryptography API for public key-based authentication in the browser will allow for a cross-platform 'Web Authentication' that gives users the best of both worlds.**

***Keywords*-authentication, authorization, identity, BrowserID; OAuth; OpenID; WebAuth**

I. Introduction

Currently, the identity eco-system on the Web is fragmented between a number of different flows for authorization with no standardized cross-platform high-security authentication mechanism outside of usernames-passwords. In order to avoid identity on the Web from being further fragmented, we analyze the various flows on an abstract level in order to determine the next steps for making identity a first-class citizen of the Web.

Due to the large amount of terminological confusion over the vague notion of identity, a clarification of basic terms is in order. A *user* is the human being of which identity claims are made, where an *attribute* (sometimes called an "identity claim") is an aspect of the user that may be transmitted as a message, such as "First Name is Bob." A *browser* is, broadly construed, a device capable of executing mobile code (Web Applications) that consists of HTML and Javascript and that acts on behalf a user. This includes both traditional web browsers as well devices such as smartphones. The *platform* is the larger context outside of the browser, including the operating system, such as MacOS, Windows, Linux, iOs, or Android. Perhaps the most confused issue in identity systems is the difference between authentication and authorization. *Authentication* is when a user verifies their identity to a site (commonly an "identity provider," as detailed below), proving that the user matches a given set of attributes (for example, that the user is a human rather than a robot or has a certain legal name). *Authorization* is when a user allows a relying party to access all or some of their identity claims. A *credential* is the digital material that allows authentication (authentication credential) or authorization (authorization credential). A *token* is a string that allows verification. Both symmetric *bearer tokens* such as traditional "shared secret" strings and asymmetric *signed tokens* such as digital signatures are kinds of credentials. On a higher level, the 'identity eco-system' consists of a *relying party* (RP), a website or service that wishes to access attributes in order to provide a service to the user. An *identity provider* (IDP) is then another web service that stores attributes for a user and allows authorized access to attributes.

A. Properties

In order to evaluate an identity solution, we need to investigate its performance with respect to a set of desired properties. Properties of current interest include how does the user *authenticate* to an identity provider, such as using a user-name and password combination or other authentication credentials such as public key material? Also, how does the user *authorize* access to their identity, taking into account questions such as the life-time and revocation of the authorization? Any identity system would ideally be *multi-device*, i.e. able to work across multiple devices. *Privacy* can be thought of as how much control does the end-user have over what information the relying party, the identity provider, and third parties can observe? In particular, we decompose privacy into linkability and anonymity. *(Un)linkability* answers the following: Can users prevent transactions using the same identity from being correlated across relying parties? Internally, unlinkability "means that within a particular set of information, the attacker cannot distinguish whether [items of interest] are related or not." [4]. Items of interest might include multiple transactions using the system, or distinct sets of attributes. External linkability is when the availability of auxiliary information allows the linking of internally unlinkable items.

II. OpenID Connect

While the OpenID brand has undergone several major changes, the latest OpenID Connect [2] is a profile of OAuth 2.0, optimizing certain elements of OAuth for server-side

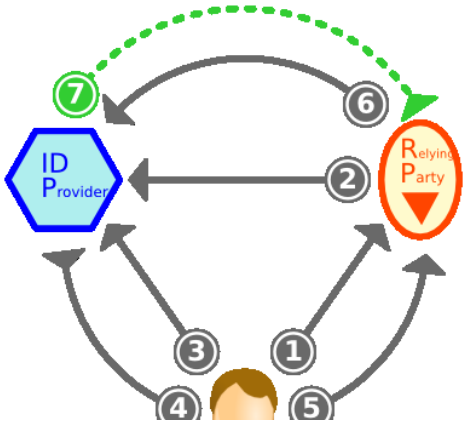

Figure 1. Flow of OpenID Connect (OAuth 2.0)

exchange of attributes and requires no changes to current browsers. OpenID Connect uses OAuth 2.0 for the authorization flow, while adding a small number of non-opaque identifiers for attributes in the response between an identity provider and relying party. One important point is that OpenID Connect, as it is a profile of OAuth, does not standardize any authentication, but only specifies an HTTP redirection flow from the relying party to the identity provider (a possible *postMessage* alternative has been discussed as well). In current implementations authentication is usually done via a username-password combination, although higher security authentication credentials are allowed, but are not specified. The flow of OpenID Connect is illustrated in Figure 1, where the user's browser is given by an human icon and the flow of personal data by the green arrow.

A. Flow

1) A user visits a relying party that needs attributes.
2) The relying party makes a request for attributes to the identity provider.
3) The user is redirected to the identity provider from the relying party.
4) The user authenticates to the identity provider (typically using a username-password combination), and is granted a bearer token.
5) User is redirected back to relying party and grants authorization token to relying party.
6) The relying party sends the authorization token to the identity provider and receives an access token (a bearer token with a scope and limited lifespan).
7) While the access token is valid, the identity provider sends attributes to the relying party.

B. Properties

Authentication: Authentication is out of scope of this specification, but the flow usually uses redirects and usually uses user-names and passwords. *Authorization*: The details of the kinds of scope and limits to authorization rely on OAuth 2.0 and an server-side flow is fully specified that works even when the user is offline once user has granted authorization once, which can be a distinct advantage for use-cases involving aggregation and updating applications. Thus, OpenID Connect is an "offline client / server-to-server" authorization flow at heart. *Extensibility*: Only the standard attributes given by OpenID Connect are specified as opposed to a more general metadata framework. *Multi-device*: As the redirection flow can use username-passwords without any change to a browser or device-specific code, in its most basic form it can be used across devices. *Privacy and Linkability*: The identity provider knows all of a user's interactions, including which attributes are requested by which relying parties and when, although this may be mitigated to some extent by the opportunity to choose among identity providers. *Anonymity*: Nothing in the architecture of OpenID Connect requires that identifiers be persistent or tied to a stable external identity. The specification does not require particular user identifiers.

C. Attacks

Phishing Heaven: Redirection attacks are possible with OpenID Connect. Unless the user is aware of TLS, a malicious relying party can fool them into redirecting to a fake identity provider site and then use that redirection to intercept their real credentials (i.e. usually reusable passwords); the malicious party can then use these credentials to provide a real authorization code to the actual identity provider, thus "stealing" the user's identity. There are a number of variations on this attack, such as a real identity provider being provided with a malicious redirection URI to fake relying party.[1] *Bearer Tokens*: While having set lifespans and one-time authorization codes and tokens should be common practice, bearer token that can be intercepted can be used in an attack as typically the credentials used in OpenID are bearer tokens. As all authorization credentials are transmitted via HTTP user-agent redirections, these credentials can be revealed possibly via HTTP referrer headers and the browser history. *Server-side Privacy*: Once an access token is granted, the relying party can talk to the identity provider without any interaction with the user. The user does not necessarily know the scope, lifetime, and kinds of access to their attributes that a token provides. Furthermore, the identity provider can observe every interaction of the user with any relying party. *Traffic analysis*: Even if the user takes steps to mask their own communications, communications between relying party and identity provider may be observable. Traffic patterns may be observable even if the messages between the identity provider and relying party are encrypted.

[1]See Ben Laurie in *http://www.links.org/?p=187*.

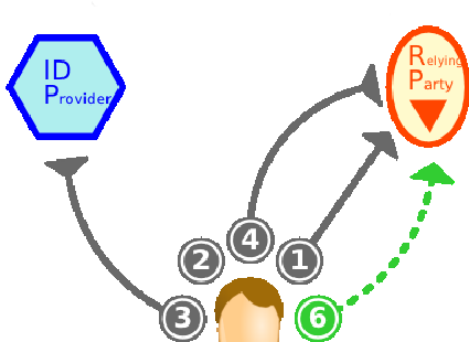

Figure 2. Flow of BrowserID

III. BrowserID

BrowserID [1] is currently a Javascript library to allow users to authenticate their identity via a "verified" email, where an email address is verified via attachment to key material. Compared to the fully specified OpenID Connect, BrowserID is still a work in progress. BrowserID can be decomposed into two primary parts: an authentication scheme that allows a single email and key combination to be re-used in multiple contexts and the identity provider can mediate the transfer of attributes to the relying party via the browser via a "online client-to-server" authorization flow. Unlike OpenID Connect that requires trust in the server, BrowserID requires trust in the browser. The flow of BrowserID is also abstractly illustrated in Figure 2.

A. Flow

1) The user attempts to identify themselves by giving an e-mail address, and wants to bind that address to a particular set of key material (for which the user then provide a proof of possession) by having the identity provider attest to that binding.
2) The browser checks to see if a private key is present in the browser associated with that email address.
3) If no key exists for the email address locally in browser, the browser generates key material and registers the public key with the identity provider.
4) The browser sends a signed authentication credential to the relying party.
5) The relying party checks authentication credentials with their locally stored database of identity provider public keys, authenticates the user if verification succeeds (not shown in diagram as this step does not happen with every transaction).
6) The user sends signed attributes to the relying party from browser.

B. Properties

Authentication: As authentication is done by default with key material as opposed to username-passwords, it is much more secure. Furthermore, there is no need for redirection. *Authorization*: This specification does not explicitly set parameters for authorization scope to protected resources, but enables a browser-mediated flow of verified attributes. However, this only works if the browser is online. *Privacy and Linkability*: Although the relying party knows the identity provider's key and will have to check at least once with the identity provider to determine public key of user, it does not have to check in theory more than once per e-mail (although updates should be done at set time intervals at a larger-grain than user transactions). Importantly, using the email address as identifier enables the linking of transactions. *Anonymity*: Email addresses can be pseudonymous or throw-away, but many are long-term valuable identifiers; using the email address may make the scope of identification visible to the user.

C. Attacks

Re-implementing Key Infrastructure in Javascript: While the Javascript code works and can be deployed, the work currently relies on Javascript cryptographic operations that would be better baked directly into the browser, and Mozilla is planning on doing so. *Trusted Verification Service*: Currently also all verification of key material from the browser is done by a centralized service (*http://www.browserid.org*), which would be a major vulnerability if compromised. Ironically, the registration to the identity provider such as *http://www.browserid.org* is currently done with username-passwords. Plans to decentralize this service are yet specified and Mozilla has claimed this is a "short-gap" measure. *Email Verification*: Note that the BrowserID pattern could simply be considered proof of a "login and check email" capability (like any other system that relies on emails to reset forgotten passwords), and thus the security of the system relies on the security of the identity (email) provider. Given the poor state of the security of many email providers (for example, STARTLS not providing warning message if TLS does not work and thus e-mails being transmitted as cleartext), the possibility of email address compromise undermines this system. *Traffic Analysis*: Although the identity provider itself will not know which relying party are requesting the attributes due to the "online client-to-server" authorization flow, a global passive adversary can still perform timing attacks between both the browser and the relying party as well as the browser and the identity provider if the identity provider stores the attributes and has to transfer them "real-time" to the browser.

IV. Next Steps: Web Authentication

At the W3C Identity in the Browser workshop [3], participants expressed shared belief that the browser could play

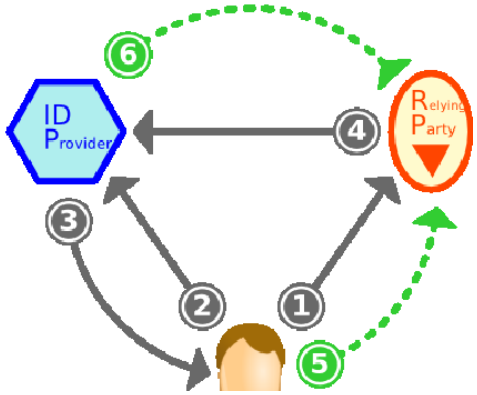

Figure 3. Generalized "offline client/server-to-server" and "online client-to-server" authorization flows plus key-based Web authentication

an important role in identity on the Web by providing more secure authentication. We propose using native Javascript cryptography for key-based Web authentication of the kind done in BrowserID, while allowing both BrowserID and OpenID authorization flows. For Web Authentication, these cryptographic operations are being built as a Javascript API by the W3C Web Cryptography Working Group chartered to create a common cross-browser Javascript cryptography library [3]. This avoids the dangerous redirection pattern of most OpenID Connect deployments and replacing the use of bearer tokens with signed tokens, showing on the W3C Web Cryptography API can help secure OpenID Connect.

Allowing both authorization flows allow attributes to be transmitted while the browser is offline, an advantage of OpenID Connect's "offline client / server-to-server" flow. However, the BrowserID "online client-to-server" flow enables high privacy use-case by having "anonymous" attributes such as "over 21 years old" transmitted via the browser but verified by the identity provider. This would allow an IDP to tell an RP that a certain user had capabilities while minimizing identity disclosure. The flow of the combined flow is also abstractly illustrated in Figure 3.

A. Flow

1) User visits relying party, presents authentication credentials in form of a signed (by identity provider) authentication credential.
2) If user wishes to enable offline attribute access, user must register public key material with identity provider and enable that flow.
3) Identity provider signs authentication credential and any attributes to be transmitted via browser.
4) If authentication credential signed by identity provider, relying party verifies signature.
5) If the signature on authentication credential is valid, then user authenticated and so verified attributes can be sent via the browser if needed.
6) In the case of offline flow authorized, signed attributes sent from identity provider to relying party.

V. Tentative Conclusions

Although lack of space prevents a larger analysis, it seems this approach helps solve the weaknesses of earlier systems. This sort of approach would offer identifiers that can be pseudonymous or throw-away and disclosure of attributes can be done in a capability-based manner. We would consider email verification to be one possibly authentication capability amongst many, including transferring only signed identifiers to the relying party in cases where another form of high-security (such as smartcard) authentication is needed. However, even this system would be vulnerable to traffic analysis. This could be ameliorated through the use of proxies (Tor included) and messages being sent regularly and padded.

The primary advantage of OpenID Connect is that the "redirection-based" authentication based on username-passwords works easily across multiple browsers, unlike BrowserID which requires secret key material to be bound to a browser, which becomes exceptionally problematic across multiple devices/browsers. It seems this can solved by registering public key material for each browser/device to an identity provider, but from a privacy standpoint the identity provider would then know all browsers. More research is necessary in this area in particular.

Username/passwords, are the cornerstone of most current web identity, are inherently insecure. The development of secure cross-platform Javascript cryptography in the browser offers hope for escape from this bind if cross-browser synchronization were to be solved. However, the long-term goal should be a browser environment where a user can chose between different identities, including anonymous identities and identities with site- or time-limited scope, and rely on strong cryptography to secure these identities against misuse. From the side of user experience, we expect a user should be able to log-in without passwords, by simply selecting their identity (including an anonymous option) or automatically logging in using a default identity. These are difficult requirements, but worth pursuing.

References

[1] Mozilla Team. *Verified Email Protocol*, 2012. https://wiki.mozilla.org/Identity/Verified_Email_Protocol.

[2] N. Sakimura et al. *OpenID Connect Standard 1.0*. 2012. http://openid.net/specs/openid-connect-standard-1_0.html.

[3] H. Halpin. *Identity in the Browser Final Report*. 2011. http://www.w3.org/2011/identity-ws/report.html.

[4] M. Hansen. Privacy Terminology. 2012. http://tools.ietf.org/html/draft-hansen-privacy-terminology-03.