Federated Browser-Based Identity using Email Addresses

Mike Hanson
mhanson@mozilla.com

Dan Mills
thunder@mozilla.com

Ben Adida
benadida@mozilla.com

27 April 2011

Abstract

In the long battle to simplify and secure Web identity, we believe an eventual solution: (a) is federated from the start, (b) uses email addresses as identifiers, (c) encompasses a complete identity stack beyond just an identifier, and (d) is baked into the browser as a primary use case. We show how our prior efforts have led us to this vision, and we discuss the Verified Email and API to Me web protocols, pillars of our initiative to bring this vision to life by letting web developers simply ask the browser for the user's email address or other attributes. We briefly describe our pure-JavaScript working implementation. We believe this approach will put users truly in control of their identity using open and convenient technologies, while creating a vibrant landscape of identity providers and relying parties across the assurance spectrum.

1 Background

Web-based identity has seen a multitude of proposed solutions – OpenID[1], Facebook Connect[2], Twitter[3], etc. A few important goals for these efforts can be distinguished:

- **single sign-on**: users can access all of their web resources using a single authentication method, e.g. a password into a keyring.
- **convenient registration**: users can quickly authenticate to a new web site by automatically providing existing profile information.
- **recognizable identity**: users can identify themselves consistently and securely across web sites, e.g. commenting on blogs, thus providing a platform for additional services such as reputation.

These goals often go together: most solutions provide both convenient registration and a subsequent single sign-on mechanism. That said, single sign-on does not necessarily imply a consistent identity or profile sharing at other sites, and recognizable identity is often a non-goal: users may specifically want to partition their online lives. If we examine three successful identity solutions on the consumer Web – Facebook, Twitter and Google –, where success is defined by a combination of developer and user adoption, we note the following interesting properties:

- each uses a *different underlying technology* that is mostly opaque to the user (proprietary, OAuth [2], and OpenID [4]).
- each provides a *streamlined user experience*: click the logo you know, enter your credentials, continue. That said, each experience is slightly different.
- none requires browser intervention, and all are particularly *vulnerable to phishing attacks*.
- each enables *more than just identifier sharing*: Facebook-Connect sites have access to the user's newsfeed and social graph, Twitter sites have access to the user's Twitter stream and followers, and Google sites can request access to the user's email address, contacts, calendar, and much more.

[1] http://openid.net
[2] http://developers.facebook.com
[3] http://dev.twitter.com/pages/auth

- each approach requires the user to *first select a brand*, i.e. Google or Facebook, then to identify themselves.

At Mozilla, we have developed a number of identity-related projects, which have greatly informed our thinking and the solution we propose. We first review these projects. In Section 2, we present the design principles extracted from these experiences, which lead to our proposal in Section 3. We provide a few points for discussion in Section 4.

1.1 Password Store and Sync

Mozilla Firefox, like most other browsers, includes a password store that lets users automatically store the passwords they enter into HTML forms. With the introduction of Firefox Sync in Firefox 4.0 [4] , these passwords can be synchronized across all of the user's devices, including Firefox Mobile running on Android or Maemo. This technology solves, to a certain extent, the single sign-on problem, though the browser is involved to a very minor degree: nothing in the browser chrome indicates the user's login status nor helps them select amongst their personas when logging into a website.

1.2 Account Manager

The Mozilla Labs Account Manager project [5], released as an add-on to Firefox 3.6 and above, defines a mechanism for web sites to declare their login protocol, i.e. which URLs perform login, logout, and login-status-checking, what parameters they expect, etc. The browser can then hook into this declared protocol and provide a trusted user interface for logging in, logging out, quickly checking login status, and easily enabling a user to pick one persona among many used over time at a given web site. This feature, like Password Store, is available to the user across their Firefox-enabled devices and solves the single sign-on problem. This time, however, the process is more intuitive for users, because the browser chrome is closely involved in every step of the authentication process and can provide more consistent information to the user. Unfortunately, Account Manager is not meant to provide recognizable identity across sites, as it is a meta-protocol that automates sites' existing account systems.

1.3 Contacts

The Mozilla Labs Contacts Add-On[6] provides a JavaScript API for web sites to request access to a user's contacts. The contacts data can be pulled from a number of different providers, including Facebook, Google, etc, without the requesting web site needing to know. With this API, the power of a simple, unified JavaScript API, as opposed to backend-focused protocols like OAuth, becomes clear.

1.4 OpenID Sniffer

Much more experimentally, Mozilla Labs also developed a browser add-on for OpenID support. The add-on detects OpenID forms and inserts itself into both the HTML presentation and the redirection-based OpenID protocol. We found that, because relying parties didn't expect this behavior, and because different relying parties employ OpenID in different ways, this approach didn't work consistently enough to reliably implement OpenID in the browser with a simplified user flow.

2 Design Principles

Taking these experiences and others' into account, we came up with a few important design principles:

Federation should be baked in and made simple for developers. One important lesson we can draw from the recent Web Certificate Authority compromise is that trust is not universal: different relationships between different parties require different trust relationships. No identity provider should be relied upon to authenticate all users for all purposes. Even with its incredible success, Facebook is unlikely to be the identity layer for services that users wish to use pseudonymously, or services that are covered by certain privacy regulation (e.g. FERPA, HIPAA, etc.)

[4] `http://www.mozilla.com/en-US/mobile/sync/`
[5] `https://mozillalabs.com/blog/2010/03/account-manager/`
[6] `https://mozillalabs.com/contacts/`

That said, federation cannot become an undue burden on web site developers. A federated login system can be built to be just as simple as Facebook Connect for relying-party developers to use. As we simplify the experience, however, the result should not mask federation: a federated protocol where a relying party is motivated to pre-select or highly encourage certain identity providers negates most of the advantages of federation. It is critically important to Mozilla to give users control over their identity, data, and more generally their online selves. Thus, true federation should be easy for developers *and* users.

An email address is the right identifier. While OpenID and similar technologies often result, by design, in opaque, directed identities to relying-party web sites, we believe that, in almost every use case, a relying party needs at least a means of reconnecting with the user. It is, in fact, well documented that most OpenID relying parties immediately ask their users for an email address, whether via attribute exchange or directly in an HTML form[7]. Email addresses have significant additional advantages:

- they are federated by design, including a local username and a globally unique and addressable domain name.
- they inherently support self-hosting.
- they are well understood by all types of users. Even Facebook users log in using their email address.
- they have been adopted by some of the more privacy-stringent identity use cases, e.g. the Direct Project for health information exchange[8].
- they provide a natural and well understood multi-persona and pseudonym mechanism using email forwarding.

Thus, we propose that authentication transactions simply yield STMP-addressable email addresses. Rather than signing in first with a brand (e.g. Google or Twitter), then with an identity, users can then sign in immediately with their identity, `user@example.com`. We note that this proposal is not novel [9] [10] [1].

A complete identity functionality stack. Simon Willison, co-creator of the Django framework and contributor to the OpenID project, launched his most recent enterprise, Lanyrd [11], with only Twitter as a supported identity provider. The reasoning for this is similar to that used by web sites built predominantly on Facebook's identity system: many relying parties need a lot more than a pseudonymous token. They need a way to contact the user, a way to get at their profile information, a way to let the user share with her friends/contacts, etc. Certainly, not all web sites need these features. However, when they do, and when users are willing to provide the data, the identity service's failure to enable the exchange results, at best, in a disconnected and tedious user experience. We need an identity system that allows rich identity data for rich applications, in an architecture that supports full user transparency and consent, without the need to hard-wire a single identity provider.

Design first for a strong browser role. OpenID was designed first and foremost to be used without browser involvement. This focus on a pure HTML solution makes clean browser integration very difficult. Thus, we believe a good identity system design should consider first the browser-based user experience, then how to gracefully degrade to a pure HTML environment. In addition, we see a strong browser-based use case as necessary to protect users' privacy from all-knowing identity providers, and in defending against advanced attacks such as clickjacking and phishing.

3 Our Proposal: Verified Email and API to Me

Putting these principles together, we envision an ecosystem of federated identity providers which provide email-address-based identity proofing and a set of API calls that relying-party sites can use to request further data from the user. We see the browser mediating all of these calls, so that developers can build against one simple interface

[7] "We've heard loud and clear that sites looking to adopt OpenID want more than just a unique URL; social sites need basic things like your name, photo, and email address." `http://openidconnect.com/about/`

[8] `http://wiki.directproject.org/`

[9] `http://siliconflorist.com/2008/06/20/email-to-id-my-openid-is-an-email-address/`

[10] `http://brad.livejournal.com/2357444.html`

[11] `http://lanyrd.com`

and let their users select Twitter, Identi.ca, Facebook, or any other identity provider with similar feature support. We envision the browser providing a secure user experience for authenticating and granting appropriate persmissions to applications, and a particularly simple API for web developers to hook into this information. We call these the Verified Email Protocol and the API to Me. For example, a web developer can write the following simple JavaScript:

```
navigator.id.onVerifiedEmail = function(identity_assertion) {
  // upload assertion to server...
});
```

which will be called when a user clicks the built-in ”log in” button in a web browser augmented with Verified Email. The identity assertion provided to the developer's code includes a copy of an identity certificate, which certifies the user's email address. The specific assertion is signed by the user's private key held within the browser, with the public-key counterpart xbound to the email address by the identity certificate.

We stress the simplicity of the web developer's task: a few lines of JavaScript, a call to a standard verification library, and the user's verified email address is available to the web site. The disclosure of an identity assertion is triggered only by the user's agent, and provides a strong user interface platform for informed consent.

Generating and Certifying Keys. We also propose a JavaScript API for generating a new keypair:

```
navigator.id.saveVerifiedEmail(<email_address>, function(public_key) {
    // upload public key to server for certification.
});
```

which triggers a keypair generation within the browser. The browser then makes the public key available to the JavaScript code, which can then upload the key to the server for certification. The server is then expected to store this public key in some publicly discoverable location, likely indicated by WebFinger[12] and supporting existing and future key discovery protocols such as Salmon [3]. Crucial to this process is that a relying party can discover the appropriate authority for a given domain and thus determine if the identity claim presented by `user@example.com` is trustworthy.

Secondary Certification. The fully federated model needs an on-ramp. For inspiration, we look to Gmail and other large email providers, which have established practices for sending email on behalf of users' alternate email addresses, even ones on domains not controlled by Google. For example, Gmail will send email on behalf of `bob@example.com` if Bob has proven to Gmail that he can receive email at that address. Gmail is effectively trusted by the rest of the world to send email on behalf of any domain, because the world believes that Gmail's processes are sufficiently solid.

We formalize this approach with Secondary Certification, though we stress that it is only meant as a temporary on-ramp: an identity provider can choose to certify that a user owns a certain email address, even if the identity provider doesn't control the domain in question. Relying parties are then free to trust those assertions if they so choose. We anticipate that only a few identity providers will be trusted to provide such assertions. Mozilla will provide one such identity provider, Firefox Accounts, though we stress, again, that we have not hard-wired our authority into the protocol.

Before Browser Support. We cannot expect every browser to immediately support our proposal. Thus, our design of `navigator.id.*` is such that it can be mimicked by a JavaScript library using IFRAMEs and `postMessage()`, backed by the Mozilla identity provider just mentioned. The advantages of browser-based implementation, including notably the privacy, clickjacking and phishing protections, are obviously not included in this approach. We intend to make this very clear to our users.

The API to Me. Further APIs will be built into the `navigator.*` namespace, including attribute requests, posting data to one's activity stream, forwarding information to friends, etc. Our model is based on a continuation of the Verified Email protocol and the work we did with the Contacts add-on: a browser-mediated, JavaScript API that lets web sites request information about the user, giving the user the opportunity to see and approve the request as it is made and carried out by the user agent.

[12] `http://code.google.com/p/webfinger/`

Implementation. We have implemented the Verified Email protocol, providing relying parties with a cross-browser-compatible JavaScript library that implements the `navigator.id.*` interface. The implementation relies on `postMessage()` to, for now, a trusted domain that implements the identity selector and stores the user's private keys in browser local storage. Signatures are performed using Tom Wu's JavaScript RSA [13].

4 Discussion

Email Expiry. What happens when a user's email address is retired/recycled? In the current web ecosystem, though users employ email to reset their passwords, they can, if they still remember their passwords, use the web sites long enough to update their email address. In the Verified Email approach, either:

- identity certificates are short-lived, in which case email address changes are trickier to implement and may require a "backup email," approach by every relying party, or
- identity certificates are longer-lived.

Revocation. If we have sufficiently long-lived identity certificates, we may need a revocation mechanism. Such a mechanism will almost certainly provide significant information to identity providers, negating much of the privacy advantage we strive for with browser-based support.

Rich private key control APIs and their implications. If the authenticating host can convey to the browser some detail about how long the authentication is valid (for example, for one session only), and how it should be stored (encrypted, in memory only) the verified ID is useful in a number of additional domains (both more secure – think trusted intranet – and less secure – think library kiosk). Supporting automatic reauthentication challenges and key refresh should be a feature of any useful user-agent authentication protocol.

5 Conclusion

Based on our experience with authentication systems, we have proposed *the Verified Email* and *API to Me* stack, a federated, browser-integrated, developer-and-user-friendly approach to online identity. We intend to implement this approach in future versions of Firefox, and welcome the community's input.

6 Credit and Acknowledgments

Mike Hanson and Dan Mills designed the protocols. Ben Adida helped with the write-up. A number of Mozillians participated in the discussions and design of our proposed approach, including: David Ascher, Alex Faaborg, Chris Howse, Ed Lee, Anant Narayanan, Aza Raskin, Ragavan Srinivasan, Sid Stamm. Others in the community also provided key insights and a sounding board, including Blaine Cook and Dick Hardt.

References

[1] Ben Adida. EmID: Web Authentication by Email Address. In *W2SP 2008, Proceedings of the Second Workshop on Web 2.0 Security & Privacy, Oakland, California.*, May 2008.

[2] Eran Hammer-Lahav. The OAuth 1.0 Protocol. `http://tools.ietf.org/html/rfc5849`.

[3] John Panzer. The Salmon Protocol. `http://salmon-protocol.googlecode.com/svn/trunk/draft-panzer-salmon-00.html`, January 2011.

[4] David Recordon and Drummond Reed. OpenID 2.0: a platform for user-centric identity management. In *DIM '06: Proceedings of the second ACM workshop on Digital identity management*, pages 11–16, New York, NY, USA, 2006. ACM.

[13] `http://www-cs-students.stanford.edu/~tjw/jsbn/`