OpenID$_{email}$ Enabled Browser: Towards Fixing the Broken Web Single Sign-On Triangle

San-Tsai Sun, Kirstie Hawkey, Konstantin Beznosov
Laboratory for Education and Research in Secure Systems Engineering
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
{santsais,hawkey,beznosov}@ece.ubc.ca

ABSTRACT

Current Web single sign-on (SSO) solutions impose a cognitive burden on web users and do not provide content-hosting and service providers (CSPs) with sufficient incentives to become relying parties (RPs). We propose a browser-based Web SSO solution that requires *minimal* user interaction and provide RPs with clear value propositions to motivate their adoption. Our approach builds OpenID support into web browsers, hides OpenID identifiers from users by using their existing email accounts, extends the OpenID protocol to perform authentication directly by browsers, and introduces an OpenIDAuth HTTP access authentication scheme to convey authenticated identities automatically into websites that support OpenID for authentication. Our solution embeds an intuitive and consistent login experience for web users in the browser; to motivate adoption by RPs, it provides them with instant marketable leads and the potential for gradual engagement of site visitors.

Categories and Subject Descriptors

D.4.6 [**Security and Protection**]: Authentication

General Terms

Security

Keywords

OpenID, Web Single Sign-On, Web Identity Management, Authentication, Identity-Enabled Browser

1. INTRODUCTION

Today's Web is site-centric; a user has to maintain a separate copy of their identity and corresponding password for each content-hosting and service provider (CSP). A large-scale study of password habits found that a typical web user has about twenty-five accounts that require passwords and types eight passwords per day [11]. Web users face the burden of managing this increasing number of accounts and passwords, which leads to "password fatigue" [48]. Aside from the burden on human memory, password fatigue may cause users to devise password management strategies that degrade the security of their protected information [14, 11].

Web single sign-on (SSO) systems are meant to address the root causes of the password fatigue problem [7]. A Web SSO system separates the role of identity provider (IdP) from that of relying party (RP) to enable users to leverage one identity across multiple RPs. An IdP issues identities or credentials to users, while an RP depends on the IdP(s) to assert the user credentials before allowing them access to its services. In addition to reducing users' memory burden, a globally adopted Web SSO solution can enable content sharing across and beyond the boundaries of CSPs [42].

OpenID [37] and InfoCard [32] are mainstream Web SSO solutions targeted for Internet-scale adoptions [26, 7]; however, they are facing an RP adoption problem [43]. Evidence shows that although major CSPs acted quickly to become OpenID IdPs (there are currently over one billion OpenID-enabled accounts), only a limited number of websites have adopted the role of RP [35, 22, 31, 34]. This is similar to having more than a billion keys, but few locks to use. For InfoCard, the list of RPs and IdPs is almost empty [46]. Figure 1, the broken triangle of the Web SSO identity ecosystem, illustrates this RP adoption problem. Each dashed line indicates a lack of driving force or incentive between two actors in the ecosystem. For instance, line `B` shows the lack of incentive for a relying party to support SSO for users.

Fundamentally, Web SSO systems shift the functions of identity collection and authentication from RPs to IdPs. However, the incentive for RPs to rely on the identity assertion services provided by IdPs is insufficient, as illustrated by line `A` in Figure 1. RPs are not willing to relinquish control over their user base unless they can obtain user data and verify the IdP's authentication and data collection policies [23, 7]. In addition, RPs have to choose which IdPs to trust as they are liable for any losses if IdPs get compromised [30].

Current Web SSO solutions do not provide RPs with sufficient business incentives for supporting Web SSO for users (line `B` in Figure 1). CSPs are reluctant to modify their login UI and process because new login procedures might confuse and upset users [12, 38]. In addition, RPs might not want to expose their users to potential business competitors because

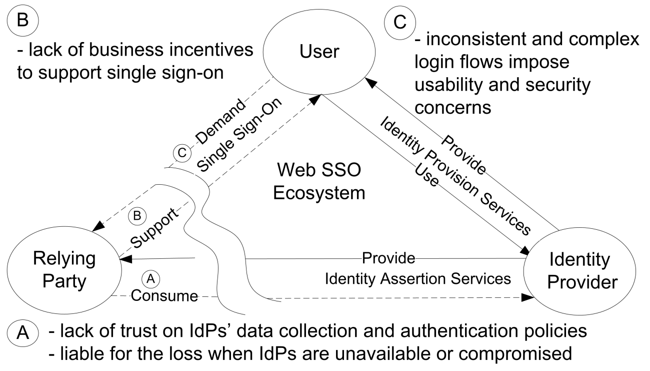

B
- lack of business incentives to support single sign-on
User
C
- inconsistent and complex login flows impose usability and security concerns
C Demand Single Sign-On
Identity Provision Services Use
Provide
Web SSO Ecosystem
B Support
Relying Party
Provide
Identity Assertion Services
A Consume
Identity Provider
A - lack of trust on IdPs' data collection and authentication policies
- liable for the loss when IdPs are unavailable or compromised

Figure 1: The broken triangle of the Web SSO identity ecosystem. Each dashed line indicates a lack of driving force or incentive between two actors.

once the attention of users has been redirected to an IdP during the login process, they might not return [7]. As early adoption does not appear to provide RPs with competitive advantages, CSPs may be waiting until Web SSO technology is mature and the cost of user training has already been absorbed by other websites.

To encourage adoption by RPs, Web SSO systems have to rely on the demand from users as the driving force (line C in Figure 1). However, the login interaction flows provided by today's Web SSO solutions are *shared-identity* sign-on (SISO) ones rather than true *single* sign-on. With SISO solutions, users can use one identity to sign into multiple RPs. Nevertheless, when accessing N RPs using one IdP, the user must visit $N + 1$ different login forms (one for each RP website and one on the IdP), choose an IdP to login N times via N possible ways, read the consent page (e.g., consent to release identity attributes, setup a custom identifier) on the IdP N times, and log out $N+1$ times through $N+1$ different interfaces. These complex and inconsistent user experiences impose a cognitive burden on web users [12, 38, 7].

SISO redirects the user's attention during the login process. However, in addition to usability issues [12, 7], redirection exposes users to phishing attacks [24, 7, 27] and makes IdP tracking possible [26]. InfoCard [32] redirects users only to their identity selector. However, for N RPs using one IdP, the user has to provide her credential to the IdP N times unless a self-issued card (without password protection) is selected. Moreover, using multiple identities in one browsing session complicates the process for users even further. When users sign on with multiple IdPs in one browser session, they have to remember which identity was used for accessing which RP. Mixing identities in one browser session can make it difficult for users to determine why an access failed, and whom to contact when a problem is encountered.

SISO requires usable interfaces and login interaction flows from both RPs and IdPs. Simply adding an OpenID textbox or InfoCard icon to the traditional login page is not an option [12, 38]. To improve the user experience, some OpenID RP adopters provide a list of IdP logos on their login form for users to choose from. The users can simply click on an IdP icon to initiate a sign-on process. However, this approach leads to the "NASCAR" problem [27] when the list of IdPs grows too long to fit on the login screen. In addition, using a list of IdPs restricts users' freedom of choice, which impairs healthy competition in the ecosystem.

SISO is especially problematic in Web 2.0 applications that require access to personal data located on multiple CSPs. For OAuth-based applications [33] that process a user's personal content from different providers, being presented with a login form on each CSP is annoying, and imposes a cognitive impact on the user [3]. For client-side mashups that use Ajax-style web services to acquire user data from several websites, login forms will block such communications. In addition, SISO-based solutions may be more difficult to use on mobile devices that have limited input capabilities.

Our research goal is to develop a Web SSO solution that requires *minimal* user interaction and provides RPs with clear value propositions to motivate their adoption. In our vision of a true Web SSO system, a user should log into her IdP once and gain access to all websites that she has an account with, *without being prompted to login again on each website*. In other words, when accessing N RPs using one IdP, the user should provide her credential exactly once to the IdP, consent at most N times (one for each RP if the consent was not recorded for future use), and should perform a logout process only once from the IdP.

Designing an usable Web SSO solution that fulfills our vision and motivates adoption by RP is challenging. To be usable, the solution must leverage the skills and experiences that an average Web user already has. It must not require any special software being installed on end-user computers or require users to manage public/secret keys or X.509 certificates for performing cryptographic operations. There are currently over one billion OpenID-enabled "keys" provided by major CSPs [34], and they are too valuable to be ignored. Thus, the solution should be backward-compatible with existing IdPs and RPs and support gradual adoption. To facilitate adoption by RPs, the solution must not require them to modify their login UI; how to design a usable login UI and login interaction flow for web users is still an open problem that ongoing single sign-on research is attempting to address [38, 36]. Furthermore, the solution should be readily employable for emerging Web 2.0 applications that process personal data located on multiple CSPs.

To achieve our goal, we propose a new approach for Web SSO that leverages OpenID and email, and builds identity support into the browser. Our approach (1) builds OpenID support directly into web browsers, (2) hides OpenID identifiers from users through the use of their existing email accounts, (3) extends the OpenID protocol to perform authentication directly with user-agents such as browsers (an OpenID$_{ua}$ extension), and (4) introduces an OpenIDAuth HTTP access authentication scheme to convey authenticated identities automatically to websites that support OpenID for authentication. To evaluate the feasibility of our approach, we implemented the proposed protocols in Java, and set up an OpenID$_{email}$-enabled IdP and five RPs. In addition, we designed an OpenID$_{email}$ Firefox extension to demonstrate our vision of a true Web SSO solution.

With our approach, web users authenticate with their existing email accounts via an OpenID$_{email}$-enabled browser. With the user's consent, their identity information transparently "flows" into websites that require it. Our approach provides users with intuitive and consistent login experiences and does not require RPs to modify their existing login forms. In addition, our approach can turn an anonymous visitor into a marketable lead with one simple click, and it could potentially decrease the sign-up form abandonment rate on RPs' websites through gradual engagement with visiting users.

This paper makes the following contributions: (1) a novel architecture and design of a browser-based Web SSO solution, (2) an OpenID$_{ua}$ extension that enables the OpenID protocol to perform authentications directly with user agents, (3) a new HTTP access authentication scheme that enables an authenticated OpenID to be automatically provided to RPs, and (4) a prototype implementation of the proposed approach.

The rest of the paper is organized as follows. The next section discusses background and related work. Section 3 discusses our design considerations, and Section 4 presents the detailed design of our proposed solution. The prototype implementation and evaluation of our approach is discussed in Section 5. Section 6 discusses the implications of our approach, and Section 7 summarizes the paper and outlines future work.

2. BACKGROUND AND RELATED WORK

In this section, we provide background information on Web SSO systems and present related work on browser-supported login solutions. Readers familiar with the subject can proceed directly to the next section.

2.1 Web SSO systems

To achieve Web SSO, major CSPs have provided a way for other CSPs to accept user credentials from their domain (e.g., Microsoft Live ID, Yahoo BBAuth). However, these systems are proprietary and centralized; identity information is maintained and controlled by a single administrative domain. Federated identity solutions such as Liberty Alliance Project [25] and Shibboleth [21] enable cross-domain single sign-on. However, these solutions require pre-established trust relationships and agreements between organizations in the federation, making them hard to scale on the Web.

OpenID [37] is an open, user-centric identity solution, which avoids the scalability problems of federated identity solutions. OpenID is user-centric in the sense that users are free to choose or setup their own OpenID providers. One key scalability feature of OpenID is that it does not require any pre-established trust relationships between IdPs and RPs. According to the OpenID Foundation [34], as of September 2009, more than one billion OpenID enabled user accounts were provided by major CSPs (e.g., Google, Yahoo, AOL, Microsoft). OpenID is a promising solution. However, OpenID suffers from adoption [35, 22, 31, 34] and usability problems [24, 7, 27], is vulnerable to phishing attacks [24, 7, 27], and imposes privacy concerns [26].

Information cards (known as InfoCard) [32] are personal digital identities that are analogous to real-world identity cards such as passports, driver licenses, and credit cards. Each card contains assertions about a user's identity that are either self-issued or issued by an identity provider. When logging into a web site, the user selects a card instead of typing a user name and password. Information cards have important features such as phishing-resistant authentication, IdP-to-RP unlinkability, and real-time user consent. However, in comparison to OpenID, InfoCard is a heavy-weight protocol. In particular, users need to install an identity selector and relying parties must have a valid SSL certificate configured to provide secure channels when communicating with identity selectors. Furthermore, because cards are stored on the end user's computer, they could be stolen and used to impersonate the victim if the user's computer is compromised [18]. Additionally, InfoCard raises privacy issues when used on shared/public computers and is difficult to employ when users switch between multiple computers.

The OpenID Information Card specification [19] enables OpenID IdPs and RPs to perform OpenID authentication using InfoCards. The OpenID assertions from OpenID IdPs to the RPs is performed by the identity selector, which makes OpenID authentication resistant to phishing. Nevertheless, the fundamental usability, security, and adoption problems of InfoCard are not addressed.

2.2 Browser-supported login solutions

Sxipper [45] is a form manager that helps users fill-in web forms such as registration or order forms. It also recognizes login forms and leverages the Firefox built-in password manager for single-click login. This Firefox add-on enables web users to maintain several personae; each persona consists of a set of user attributes such as name, email, and address. When encountering a web form, Sxipper prompts the user to pick a persona and then fills the corresponding form automatically. The main limitation of Sxipper is that users still have to maintain a separate copy of their identity for each CSP, which makes online profile management and controlled content sharing difficult. In addition, Sxipper might not detect forms correctly; and it stores sensitive information such as credit card numbers on the user's local machine. This poses a security threat if the user's computer is compromised and raises portability issues when users switch between computers or want to use a shared/public computer.

VeriSign's Seatbelt Firefox add-on [47] is designed to make OpenID more convenient to use by automatically filling in a user's OpenID URL when visiting relying parties. This extension also provides an OpenID user with information about their login state with their OpenID provider and automatically monitors OpenID transactions to help prevent phishing attacks. Seatbelt is easy to use; however, it may not detect OpenID login form fields precisely as a simple text matching technique (e.g., openid, oidurl, open-id, open_id) is used to identify them. In addition, it requires Seatbelt specific configurations and login state provision from the participating OpenID IdPs. Furthermore, it is unable to detect "rogue relying party proxying" phishing attacks (e.g., content scraped from the real site) that do not rely on HTTP redirections when spoofing victims.

Weave Identity [29] from Mozilla Labs is a Firefox add-on that leverages a Firefox built-in password manager for single-click and automatic logins and integrates Weave server accounts for automatic OpenID sign-on. Similar to VeriSign's Seatbelt, it might not detect and submit login forms correctly; and automatic OpenID login support is limited only to Weave accounts. Moreover, Weave Identity and Seatbelt are SISO solutions rather than true SSO ones.

3. DESIGN CONSIDERATIONS

In addition to the usability and backward-compatibility requirements discussed in Section 1, we discuss other considerations that affect our design decisions in this section.

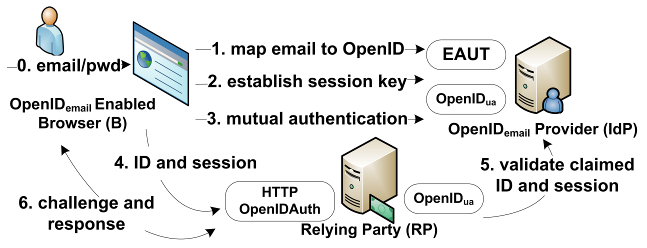


Figure 2: System architecture and high-level data flow of the proposed Web single sign-on system.

3.1 Build Identity Support Into Browser

The adoption of current Web SSO solutions is facing a classic chicken-and-egg problem: CSPs do not want to change their authentication procedures until a critical mass of users have adopted Web SSO, and users have little incentive to employ the technology unless many of their CSPs are supported as RPs [39]. In addition, users have no urgent need for SSO as they can use a password manager as a limited version of a personal identity manager [28].

To resolve this chicken-and-egg problem, future Web SSO development needs additional forces from other sources beyond the actors in the Web SSO triangle [43]. As the browser is the central piece that communicates with all actors in the identity ecosystem, we conjecture that the browser can potentially provide driving forces for RPs to adopt SSO when it is directly augmented with identity support. Currently, most web users are not aware that they already own SSO "keys" hosted on major CSPs. To advocate users' awareness, an identity-enabled browser could prompt users to sign in before browsing, provide users with an intuitive way for selecting an identity to use when visiting websites, and make it clear that they can sync their personal data from IdPs to RPs right within the browser. By embedding the SSO experience into most web users' daily web-surfing activities, the browser could drive users to reach the necessary critical mass to overcome the resistance of CSPs to become RPs.

3.2 Motivate Adoption by RPs

A high conversion rate (i.e., the ratio of visitors who become registered users) is desirable for many websites. However, most websites convert only a fraction of visitors into customers (the average online conversion rate is around 3%, with the highest at approximately 9%) [41]. One key factor that affects a website's conversion rate is the *form abandonment rate*—the ratio of visitors that fail to complete a sign-up form [49]. Traditionally, websites redirect visitors to sign up for an account before granting them access to the protected resources or allowing them to create personal content. For password recovery and to ensure future communication with users, most websites also require validation via email before activating an account. Many web services need to identify each individual user before providing the requested service (e.g., access to shared personal content that is controlled by an access-control-list). However, this requirement discourages potential customers from trying a new web service. Thus, any new solutions should provide RPs with mechanisms that could improve the conversion rate on their websites to motivate their adoption.

3.3 Security Considerations

We focus on the threats posed from the interception of user input and network traffic. The adversary's goals are to steal a user's password or to gain unauthorized accesses to an RP by impersonating an OpenID owner. To steal a user's password, an attacker could capture user passwords directly or infer them from collected data. We assume a user's computer could be compromised. An attacker could install keyloggers to capture the user's key strokes and use proxies to watch, replay, and modify network traffic between all actors in the system. The solution should be secure even under the threats of malicious CSPs, compromised user computers, and network traffic sniffing and modification.

OpenID and other similar HTTP-redirection based protocols (e.g., Google AuthSub [15], AOL OpenAuth [2], Yahoo BBAuth [52]) may habituate users to being redirected to IdP websites for authentication. If users do not verify the authenticity of these websites before entering their credentials (and they usually do not [40, 9]), phishing attacks are possible. To prevent phishing attacks, users must confirm the authenticity of an IdP before entering their credentials.

Research on methods of authenticating websites to users include security indicators [6, 20], secure bookmarks for known websites [8, 51, 53], and automated detection and blacklisting of known phishing sites [10]. However, studies suggest that security indicators are ineffective at preventing phishing attacks [9, 40]; and blacklisting known phishing sites still cause the problem of a high rate of false-positives and false-negatives [54]. Even with improved security indicators, users may ignore them [50, 40, 44].

Based on the results of user studies that evaluate the effectiveness of anti-phishing techniques [50, 9, 54, 40, 44], a Web SSO solution should avoid relying on users' cognitive capabilities to detect phishing sites.

4. APPROACH

Our approach is based on the identity flow metaphor from the design of operating systems (OS). In an OS, a user authenticates to the OS and that authenticated identity automatically "flows" into all processes launched by the user. Our approach treats a browser as an operating system and each web site the user visits as a process. A user enters her existing user name/password on the Web into a browser; and with the user's consent, that authenticated identity automatically flows into all websites that require an authenticated identity.

4.1 System Architecture and Data Flow

The main actors in our system are an OpenID$_{email}$ identity provider, an RP that supports the OpenIDAuth HTTP authentication scheme, and an OpenID$_{email}$-enabled browser. Figure 2 illustrates the system architecture and high-level data flow among the actors.

An OpenID$_{email}$ provider is an existing OpenID identity provider augmented with both an email-to-OpenID translation service (e.g., EAUT [13], WebFinger [17]) and an OpenID$_{ua}$ extension. Web users are not accustomed to using an OpenID URL as an identifier [9, 1]; email addresses on the other hand, serve as user identifiers for many CSPs [1]. Our approach is not bound to any specific email-to-OpenID translation service. In our implemented solution, we combine EAUT and OpenID so that web users can login using their email addresses to IdPs while transparently conveying an OpenID identifier to RPs for identification. With EAUT, an OpenID$_{email}$ provider is free to implement any custom logic to map or translate an email to an OpenID. OpenID$_{ua}$ is our proposed OpenID extension that allows IdPs to authenticate directly with user-agents such as browsers. Details of the OpenID$_{ua}$ extension are discussed in Section 4.2.

To "flow" authenticated identities automatically into websites that support OpenID for authentication, we introduce an HTTP access authentication scheme named OpenIDAuth. Similar to the HTTP Basic or Digest authentication schemes, OpenIDAuth is designed to allow a web browser, or other client program, to provide credentials when making an HTTP request. However, instead of utilizing the username and password as credentials, OpenIDAuth uses OpenID and a challenge/response protocol to ensure that a user "owns" the claimed OpenID. Details of the HTTP OpenIDAuth scheme are discussed in Section 4.3.

An OpenID$_{email}$-enabled browser is a browser extended with the OpenID$_{email}$ protocol. To login, a user *mutually* authenticates to her IdP directly in the browser, instead of performing authentication on the IdP's web site. There are three main steps to this login process (Steps 1 to 3 in Figure 2):

1. **Map an email address to an OpenID**: After a user enters her email address into an OpenID$_{email}$-enabled browser, the browser maps it to an OpenID identifier via an email-to-OpenID mapping service and uses that to discover the end-point of the IdP.

2. **Establish a session key with IdP**: The browser uses an extended `associate` operation (defined in OpenID$_{ua}$) to exchange a shared session key with the IdP.

3. **Mutually Authenticate with IdP**: The browser mutually authenticates with the IdP via an extended `checkid_immediate` operation. This allows the IdP to assert that the user owns the claimed OpenID identifier and the browser to validate the authenticity of the IdP. During this process, the password hash of the corresponding claimed OpenID is used as the shared secret between these two parties.

Once mutual authentication has been successfully completed, the browser and the IdP share a tuple: OpenID i, session handle h, and session key k.

When the user accesses protected content of an RP, the RP responds with an HTTP 401 "Unauthorized" message to the browser with the WWW-Authenticate scheme set to `OpenID:session`. There are three steps required for the browser and the RP to complete an OpenIDAuth authentication (Steps 4 to 6 in Figure 2):

4. **Supply a claimed OpenID and the session handle**: The browser makes an HTTP request again with the claimed OpenID i and the corresponding session handle h in the request header.

5. **Validate a claimed OpenID and the session handle**: The RP discovers an IdP based on i and then sends i and h to the IdP to ensure that i and h are valid. The IdP responds to the RP with a validation result comprised of a nonce n and signature $s = HMAC(i||h||n, k)$.

6. **Compute a response for a given challenge**: If the claimed OpenID is valid, the RP responds to the browser with an HTTP 401 "Unauthorized" message; it includes i, h, n in the response header and sets the WWW-Authenticate scheme to "`OpenID:challenge`". The browser computes a signature $s' = HMAC(i||h||n, k)$ based on the stored (h, k, i) tuple and the received nonce n, and it sends s' to the **RP** to check whether $s' = s$. If the check is successful, access is granted.

4.2 OpenID$_{ua}$ Extension

To allow an IdP to communicate directly with a browser rather than relying on redirections with a RP, we propose OpenID$_{ua}$, which is an OpenID extension that "piggybacks" extra information on the `associate` and `checkid_immediate` operation messages. OpenID$_{ua}$ extends the `associate` operation to prevent man-in-the-middle (MITM) attacks. The original OpenID protocol uses the Diffe-Hellman (DH) key exchange protocol to establish a session key; however, plain DH is vulnerable to MITM attacks. To prevent this attack between a browser and an IdP, our design piggybacks the `associate` operation with a claimed OpenID. As described in Section 8.2.3 of the original OpenID Specification [37], the session key is only XORed with the hash of a DH shared key (i.e., $k_{enc} = (k) \otimes H(g^{ab} \bmod q)$. It is possible for an MITM attacker to perform two distinct DH key exchanges with each party, which allowing the attacker to decrypt then re-encrypt the messages passed between them. With our extension, the additional `claimed_id` field can be used by an IdP to find the corresponding password hash which is only known to the IdP and the browser, but not to the adversary. The function used to hash the password is indicated in `pwd_hash_type` field. Based on our extension, the IdP responds to the browser with an encrypted session key by XORing the session key with the password hash and then XORing the result with the hash of a DH shared key: $k_{enc} = (k \otimes H_p(p)) \otimes H(g^{ab} \bmod q)$. Upon receiving the IdP's response, the browser computes the session key: $k = H_p(p) \otimes (H(g^{ab} \bmod q) \otimes k_{enc})$.

OpenID$_{ua}$ extends the `checkid_immediate` operation with an extra field named `enc_pwd_hash` that contains a password hash encrypted with the session key k. This information is used by an IdP to check whether a user "owns" the claimed OpenID identifer. The encryption method is indicated in the `enc_type` field. As this is a direct communication between a browser and an IdP, the value of the original `openid.return_to` field should be ignored by the IdP. In our prototype implementation, we set `openid.return_to` to the browser's `User-Agent` HTTP header. Below is a sample request from a browser (text in bold font indicates our extension to the message):

[Browser request:]
openid.ns.ua=http://lersse.ece.ubc.ca/openid/ext/ua/1.0
openid.ua.enc_pwd_hash=QAWSDERF412QA
openid.ua.enc_type=AES-256
openid.mod=checkid_immediate
openid.claimed_id=ece.ubc.ca/alice
openid.assoc_handle=123456789
openid.return_to=**User-Agent: Mozilla/5.0 ...**

The message format of the response from an IdP is identical to the one for the original `checkid_immediate` operation.

This operation is also used by an RP to check whether a given claimed OpenID and its corresponding session handle have been authenticated with the IdP (without `enc_pwd_hash` field) when processing a request for protected content.

4.3 HTTP OpenIDAuth Scheme

To transmit a claimed OpenID transparently to an OpenID-enable web site, we introduce OpenIDAuth (Steps 4 and 6 in Figure 2), which is an HTTP access authentication scheme. For a given HTTP request to a protected resource, the RP responds to the browser with the following message in order to solicit a claimed OpenID and the corresponding session handle:

[RP response:]
HTTP/1.1 401 Authorization Required
WWW-Authenticate: OpenID:session realm="*.ubc.ca" auth-domain="www.ubc.ca"

The `realm` value and `auth-domain` specified in the response header are used to define the realm of an authentication session, which is the area of protected resources that shares the same user credentials. To respond to an `OpenID:session` message, the browser prompts the user to select an authenticated identity for the specific `realm` on the RP. The browser can optionally record this identity-to-realm mapping and use it for future requests without prompting the user again (for a specific period of time designated by the user). Once an authenticated OpenID is selected, the browser makes another HTTP request:

[Browser request:]
GET /private/content.html HTTP/1.1
Authorization: OpenID:session user-id="http://ece.ubc.ca/alice", session-id="1234"

When an RP receives such a request, it sends the supplied OpenID and session id to an IdP via an extended OpenID `check_immediate` operation to ensure the supplied information is valid. If it is valid, the IdP responds with a positive assertion that consists of a nonce n and signature $s = HMAC(i||h||n, k)$. The RP then uses n to challenge the browser. To respond to the `OpenID:challenge` message, the browser computes a signature over the list of fields specified by the `signed` field (e.g., OpenID i, session handle h, nonce n), and then sends the signature to the RP.

The RP then checks whether the response matches the signature generated by the IdP. If it does, the RP responds to the browser with the requested resource and a logout URL in the header. The browser uses the logout URL to notify the RP when the single logout event is triggered by the user.

4.4 Log into an OpenID$_{email}$ Provider

We now provide the sequence of steps for logging into an OpenID$_{email}$ provider as illustrated in Figure 3:

1. User **U** enters email e and password p into browser **B**.
2. **B** parses the domain d from e (email is in the form `user@domain`) and prepends the string "http://" to d to form an EAUT discovery URL. **B** retrieves an XRDS document [16] on the URL, and lookups values representing an EAUT service **E**.

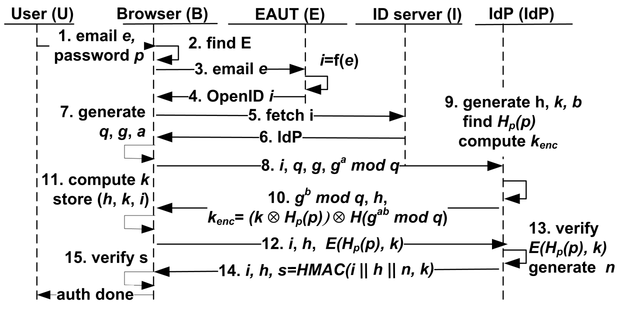


Figure 3: Flow for logging into an OpenID$_{email}$ provider.

3. **B** sends e to **E** (i.e., https://lersse.ece.ubc.ca/eaut/).
4. **E** maps e to an OpenID identifier i and sends it back to **B**.
5. **B** makes an HTTP request on i to fetch the document hosted on **I**.
6. **I** responds with either an XRDS or HTML document that contains the IdP endpoint URL **IdP**.
7. **B** generates a Diffie-Hellman (DH) modulus q, generator g, and a random DH private key a to initiate an `association` operation that establishes a session key k with **IdP** (Steps 7 to 11).
8. **B** sends i, q, g, and the DH public key $g^a \texttt{ mod } q$ to **IdP**.
9. **IdP** generates a new session handle h, a session key k, and a random DH private key b. **IdP** then retrieves the password hash $H_p(p)$ based on i from its credential store.
10. **IdP** sends $g^b \texttt{ mod } q$, h, and an encrypted session key $k_{enc} = (k \otimes H_p(p)) \otimes H(g^{ab} \bmod q)$ to **B**. Note that k is XORed with $H_p(p)$ to prevent MITM attacks.
11. **B** computes $k = H_p(p) \otimes (H(g^{ab} \bmod q) \otimes k_{enc})$ and then stores the tuple $(h,\ k,\ i)$.
12. To check whether **U** owns the claimed OpenID identifier i, **B** sends i, h, and $E(H_p(p), k)$ to **IdP** via an extended `check_immediate` operation.
13. **IdP** decrypts the encrypted password hash using k, and checks whether $H_p(p)$ matches the stored password hash for i.
14. After password verification, **IdP** sends back i, h, a nonce n, and a signature $HMAC(i||h||n, k)$ to **B**.
15. **B** verifies the signature using the session key computed at Step 11 to ensure **IdP** holds the same session key. Once the signature is verified, **B** acknowledges to **U** that the authentication process has been successfully completed.

Our design allows users to log in with multiple IdPs. Users are prompted to choose an appropriate identity when accessing protected content on RPs.

4.5 Access Protected Content

When the login process has completed, the browser and the IdP have been mutually authenticated and each has established a tuple of $(h,\ k,\ i)$. We now illustrate the data flow for accessing protected content on an RP (Figure 4):

1. **B** makes an HTTP request r for the protected content.

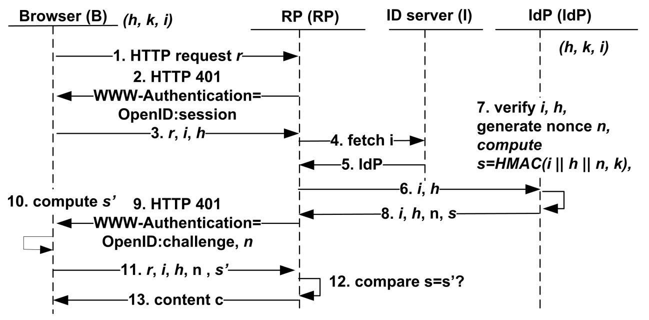


Figure 4: Flow for accessing protected content.

2. **RP** responds with an HTTP 401 "Unauthorized" to **B** with `WWW-Authenticate` scheme set to `OpenID:session`.
3. **B** presents an identity selection dialog for **U** to select a claimed OpenID i. **B** sends r to **RP** again with i and the corresponding session handle h in the request header.
4. **RP** makes an HTTP request on i.
5. **I** responds with either an XRDS or HTML document that contains the IdP endpoint URL **IdP**.
6. **RP** sends i, h to **IdP** via an extended `check_immediate` operation to check whether i has associated with an authenticated session h.
7. **IdP** verifies i and h based on the stored (h , k ,i) tuple. If i and h are valid, **IdP** generates a nonce n (e.g., `2009-09-15T17:11:51 ZUNIQUE`) and computes signature $s = HMAC(i||h||n, k)$.
8. **IdP** sends i, h, n, and s to **RP**.
9. **RP** responds with an HTTP 401 "Unauthorized" to **B** with `WWW-Authenticate` scheme set to `OpenID:challenge` and uses the nonce n as a challenge.
10. **B** computes signature $s' = HMAC(i||h||n, k)$ based on the stored (h, k, i) tuple and the received nonce n.
11. **B** sends r to **RP** again with i, h, n, and s'.
12. **RP** checks whether $s = s'$. If it does, **RP** returns the protected content to **B** and a logout URL in the response header.

Once the OpenIDAuth authentication process has completed, the **RP** can issue a cookie for the browser **B** to represent the current authenticated session. **B** then includes this cookie in the HTTP request header for future communications with **RP**, instead of re-initiating an OpenIDAuth process.

5. PROTOTYPE IMPLEMENTATION AND EVALUATION

To evaluate our approach, we implemented the proposed protocols in J2EE, and developed a Firefox extension to communicate with the new protocols. For OpenID$_{ua}$, we extended OpenID4Java [5], which is an open-source Java library that offers support for implementing OpenID identity providers and relying party websites. We setup an OpenID$_{email}$ provider by augmenting an existing email server with EAUT and the OpenID$_{ua}$ extension. We also augmented five open-source J2EE web applications (`Bookstore`, `Employee Directory`, `Classifieds`, `Events`, and `Portal`) from `gotocode.com` to become OpenID$_{email}$ RPs.

Figure 5a shows a screen shot of the OpenID$_{email}$ Firefox add-on when a user launches Firefox, but before the UI of the browser is visible to the user. The user enters her email account and types or "clicks through" her password to start the login sequence discussed in Section 4.1, steps 1 to 3. Once logged in, the user's current login information will be shown on an icon located on the status bar of the browser. The user can click on the icon to log out or sign in with additional accounts via a popup menu (Figure 5b). When the user is browsing to a protected resource on an RP that supports OpenIDAuth, the Firefox add-on will prompt the user to choose an identity (Figure 5c), before starting the automatic identity provisioning discussed in Section 4.1, steps 4 to 6. The add-on also records this action (i.e., "Remember it" in Figure 5c), which allows automatic login for future access and assists users in determining which identity was used for accessing which RP.

5.1 Web 2.0 Mashup Applicability Evaluation

To evaluate the applicability of our approach in Web 2.0 applications that aggregate personal data, we designed an AJAX-style client-side mashup that combines Google Map and personal photo albums from a prior developed Facebook photo sharing application [42]. We augmented the photo sharing application with OpenIDAuth and OpenID$_{ua}$ extension and added a location tagging function for album owners to indicate where the photos were taken. We also added a web service that returns a set of albums (in the form of a JSON object) based on the authenticated user.

When a user loads the mashup, the Firefox add-on will prompt the user to choose an identity when the sharing application uses the OpenIDAuth protocol to solicit a claimed identity (Figure 5d). Based on the authenticated identity, the sharing application then returns the corresponding albums to the mashup. For each album, the mashup places a marker on the map using the album's cover picture, as shown in Figure 5e.

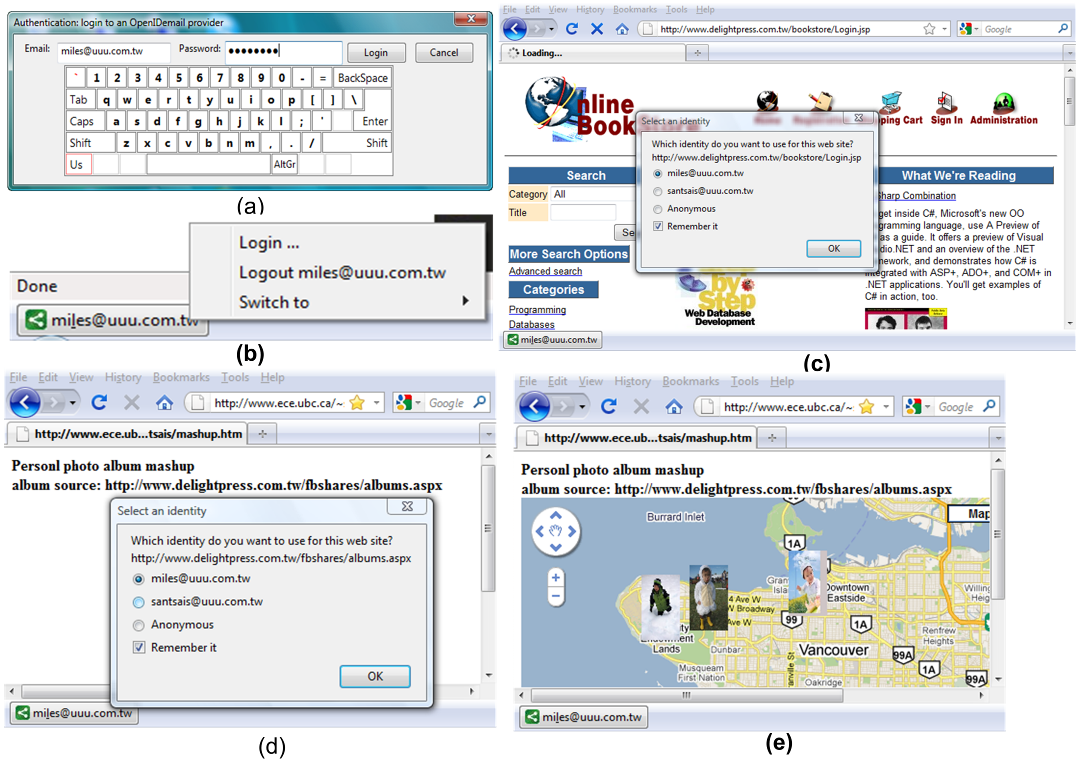
6. DISCUSSION

We now discuss why our approach is backward-compatible with existing OpenID IdPs and RPs, how it provides RPs with instant marketable leads and the potential for gradual engagement, and how it improves the security of the OpenID protocol.

6.1 Backward-Compatibility

Our solution is backward-compatible with existing OpenID IdPs and RPs, and therefore supports gradual adoption. First, the design of the OpenID$_{ua}$ extension is based on the standard extension framework specified in the OpenID specification. Therefore, OpenID$_{email}$ providers can still communicate seamlessly with existing OpenID RPs. Second, OpenIDAuth is activated by an RP *only* when an OpenID$_{email}$ enabled browser is detected. In our design, the browser's OpenID$_{email}$ support is indicated in the `user-agent` HTTP header by appending special text such as `OpenIDemail/0.2.1` to this header. Thus, OpenIDAuth-enabled RPs can use the original OpenID protocol to communicate with browsers that do not have the OpenID$_{email}$ add-on installed.

We designed the OpenID$_{email}$ Firefox add-on to demonstrate the benefits of our approach. We acknowledge that, in order to achieve widespread adoption, the proposed solution



Authentication: login to an OpenIDemail provider

Email: miles@uuu.com.tw Password: ••••••••

Login ...
Logout miles@uuu.com.tw

miles@uuu.com.tw

http://www.delightpress.com.tw/bookstore/Login.jsp

Select an identity

Which identity do you want to use for this web site?
http://www.delightpress.com.tw/bookstore/Login.jsp
- [✓ miles@uuu.com.tw]
- santsais@uuu.com.tw
- Anonymous
- [✓ Remember it]

http://www.ece.ubc.ca/~

http://www.ece.ub...tsais/mashup.htm

Personl photo album mashup
album source: http://www.delightpress.com.tw/fbshares/albums.aspx

Select an identity

Which identity do you want to use for this web site?
http://www.delightpress.com.tw/fbshares/albums.aspx
- [✓ miles@uuu.com.tw]
- santsais@uuu.com.tw
- Anonymous
- [✓ Remember it]

Figure 5: Screen shots of the OpenID$_{email}$ enabled browser.

should be built directly into browsers instead of provided as an add-on to the browsers.

6.2 Gradual Engagement

With our approach, an RP can avoid sign-up forms in favor of gradual engagement. When an anonymous visitor consents to use one of her authenticated identifiers for the visiting RP, the RP can grant the user the required permissions for the task at hand without any interruption. This instantly turns the visitor into a marketable lead, who is identifiable by the user's OpenID identifier and email address. Once the visitor is identifiable, the RP can gradually engage with the user to acquire additional attributes (e.g., gender, date of birth) when there is value for the user to provide them. Ultimately, the RP may be able to convert the user from performing actions, such as simple page browsing, to performing more desired transactions, such as sales of products or software downloads.

Our approach allows users to choose (or switch) an identity from the browser with one simple click and their trust may be built through gradual engagement with RPs. We hypothesize that the one-click conversion rate and the gradual engagement conversion rate would be higher than the conversion rate with traditional login options. We plan to conduct experimental studies as future work to assert this hypothesis.

6.3 Security

Based on our design, phishing is prevented because users enter their password only into the browser instead of on each individual IdP web site, and the IdP has to know the password hash of the corresponding claimed OpenID in order to complete the mutual-authentication process. To prevent key stroke logging attacks, we designed a virtual keyboard (Figure 5a) for user to "click-through" their passwords instead of keying them. The virtual keyboard is positioned differently each time it is displayed to prevent malware from inferring the password based on the captured mouse-click coordinates. The user's email and password are only entered into the chrome area of the browser to prevent login form overlay attacks [27]. In addition, the proposed protocol transmits only the encrypted password hash to the IdP to prevent password sniffing or password-hash replay attacks during the login processes.

To impersonate an OpenID owner, an attacker must know the session key. Assume an attacker could route all traffic to a malicious IdP proxy (local or remote) during login processes. The proxy can then act as a MITM to watch and forward traffic back-and-forth between the browser and the real IdP. Thus, a MITM attacker can learn the claimed OpenID, the session handle, and all DH-related information; however, the session key cannot be derived as the password hash is unknown to the attacker. The MITM attacker can also learn the value $(k \otimes H_p(p))$ from the IdP to the browser during association stages. Later, the attacker may discover both the password p and the key k by using an online brute-force or dictionary attack on all possible p's to discover all possible k's. Nevertheless, similar to all other password-based authentication schemes, we assume IdPs employ certain brute-force attack countermeasures (e.g., exponential delay response, n-trial account locking) to prevent online dictionary attacks.

Another way for an attacker to attempt to compromise a session key is to break in to the browser's process to search for the heap-allocated session key directly. To prevent such an attack, techniques such as data space randomization [4], which randomizes the representation of data stored in program memory, could be employed to effectively prevent non-control data attacks as well as code injection attacks. Even in the extreme circumstance where a session key is maliciously discovered, only the current login session is compromised. The protection of heap-allocated session keys is an important research topic, but it is outside the scope of this paper.

While our approach attempts to address OpenID's phishing problem, other OpenID inherited security threats still exist (as described in the Section 15 of the OpenID Specification [37]). The main threats are denial of service (DoS) attacks against an IdP, and MITM attacks between an RP and an IdP. How to make the OpenID protocol more resilient to DoS and MITM attacks between an IdP and an RP is an open technical problem.

7. CONCLUSION

Similar to how credit cards reduce the friction of paying for goods and services, Web SSO systems are built to reduce the friction of using the Web. However, current Web SSO solutions impose a cognitive burden on web users and do not provide CSPs with sufficient incentives to become RPs. In addition, web users do not have an urgent need for SSO and RPs would rather wait until SSO technology is mature and pervasive. In this paper, we present the architecture, design, and implementation of a browser-based Web SSO system that is a step towards fixing the broken Web SSO triangle. Our approach embeds an intuitive and consistent login experience in the browser, and can provide RPs with instant marketable leads and the potential for gradual engagement of website visitors to motivate their adoption.

In addition to normal browsing, our solution could be employed to make the login process more usable in other emerging application domains. For instance, our approach can be applied to Web 2.0 mashups that aggregate personal data from multiple CSPs or mobile devices that have limited input capabilities. In addition, the proposed OpenID$_{ua}$ extension can be employed to leverage those billion OpenID-enabled accounts into traditional rich-client applications and appliance devices such as Netflix, XBox, and Zune.

For future work, we plan to conduct usability studies of our prototype implementation to ensure the proposed mechanisms are usable to average web users. To motivate RPs adoption, we plan to extend our solution with user data exchange to enable RPs to import or synchronize user's personal data from IdPs to their website. We also plan to perform a rigorous security analysis of the proposed protocols using automatic security protocol validation tools.

8. ACKNOWLEDGMENTS

We thank members of the Laboratory for Education and Research in Secure Systems Engineering (LERSSE) who supplied valuable feedback on the earlier drafts of this paper. Research on the OpenID$_{email}$-enabled browser has been partially supported by the Canadian NSERC ISSNet Internetworked Systems Security Network Program.

9. REFERENCES

[1] B. Adida. EmID: Web authentication by email address. In *Web 2.0 Security and Privacy Workshop 2008*, Oakland, California, USA, 2008.

[2] AOL LLC. AOL Open Authentication API. http://dev.aol.com/api/openauth, January 2008.

[3] P. Austel, S. Bhola, S. Chari, L. Koved, M. McIntosh, M. Steiner, and S. Weber. Secure delegation for web 2.0 and mashups. In *Workshop on Web 2.0 Security And Privacy*, 2008.

[4] S. Bhatkar and R. Sekar. Data space randomization. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 5137, pages 1–22, Paris, France, 2008. Lecture Notes in Computer Science.

[5] J. Bufu. OpenID4Java. http://code.sxip.com/openid4java/, 2009.

[6] CoreStreet Ltd. Spoofstick. http://www.spoofstick.com/, 2005.

[7] R. Dhamija and L. Dusseault. The seven flaws of identity management: Usability and security challenges. *IEEE Security and Privacy*, 6:24–29, 2008.

[8] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security*, pages 77–88, New York, NY, USA, 2005. ACM.

[9] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 581–590, Montréal, Québec, Canada, 2006. ACM.

[10] Earthlink Inc. Earthlink toolbar: scambloker for Windows users. http://www.earthlink.net/, 2008.

[11] D. Florencio and C. Herley. A large-scale study of web password habits. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 657–666, New York, NY, USA, 2007. ACM.

[12] B. Freeman. Yahoo! OpenID:One Key, Many Doors. http://developer.yahoo.com/openid/openid-research-jul08.pdf, July 2008.

[13] D. Fuelling and W. Norris. Email Address to URL Transformation 1.0. http://eaut.org/specs/1.0/, June 2008.

[14] S. Gaw and E. W. Felten. Password management strategies for online accounts. In *Proceedings of the second Symposium on Usable Privacy and Security*, pages 44–55, 2006.

[15] Google Inc. AuthSub authentication for web applications. http://code.google.com/apis/accounts/docs/AuthSub.html, December 2008.

[16] E. Hammer-Lahav. XRDS-Simple 1.0. http://xrds-simple.net/core/1.0/, March 2008.

[17] E. Hammer-Lahav. WebFinger. http://webfinger.net/, August 2009.

[18] X. Hao. Attacking Certificate-based Authentication System and Microsoft InfoCard. In *Power of Community Security Conference*, 2009.

[19] D. Hardt and J. Bufu. OpenID Information Cards 1.0 - Draft 01. https://openidcards.sxip.com/spec/openid-infocards.html, August 2007.

[20] A. Herzberg and A. Jbara. Security and identification indicators for browsers against spoofing and phishing attacks. *ACM Trans. Interet Technology.*, 8(4):1–36, 2008.

[21] Internet2. Shibboleth System. http://shibboleth.internet2.edu/, 2008.

[22] JanRain Inc. Relying Party Stats. http://blog.janrain.com/2009/01/relying-party-stats-as-of-jan-1st-2008.html, 2009.

[23] A. Jøsang, M. A. Zomai, and S. Suriadi. Usability and privacy in identity management architectures. In *ACSW '07: Proceedings of the fifth Australasian symposium on ACSW frontiers*, pages 143–152, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[24] B. Laurie. OpenID: Phishing Heaven. http://www.links.org/?p=187, January 2007.

[25] Liberty Alliance. Liberty Alliance Project. http://www.projectliberty.org/, 2002.

[26] E. Maler and D. Reed. The venn of identity: Options and issues in federated identity management. *IEEE Security and Privacy*, 6:16–23, 2008.

[27] C. Messina. OpenID Phishing Brainstorm. http://wiki.openid.net/OpenID_Phishing_Brainstorm, 2009.

[28] D. Mills. Identity in the Browser (Mozila Labs). https://mozillalabs.com/blog/2009/05/identity-in-the-browser/, 2010.

[29] Mozila Labs. Weave Identity Account Manager. https://wiki.mozilla.org/Labs/Weave/Identity/Account_Manager, 2009.

[30] S. J. Murdoch and R. Anderson. Verified by visa and mastercard securecode: or, how not to design authentication. In *In the proceedings of Financial Cryptography and Data Security 2010*, January 2010.

[31] MyOpenID. OpenID Site Directory. http://openiddirectory.com/, 2010.

[32] A. Nanda and M. B. Jones. Identity Selector Interoperability Profile V1.5. http://informationcard.net/specifications, July 2008.

[33] OAuth Core Workgroup. Oauth core 1.0 specification. http://oauth.net/core/1.0/, December 2007.

[34] OpenID Foundation. Promotes, protects and nurtures the OpenID community and technologies. http://openid.net/foundation/, 2009.

[35] OpenID Foundation. OpenID Directory. http://openiddirectory.com/, 2010.

[36] OpenID Wiki. Openid user experience. http://wiki.openid.net/browse/view=ViewFolder&param=user-experience, April 2010.

[37] D. Recordon and B. Fitzpatrick. OpenID authentication 2.0. http://openid.net/specs/openid-authentication-2_0.html, December 2007.

[38] E. Sachs. Usability Research on Federated Login. http://sites.google.com/site/oauthgoog/UXFedLogin, October 2008.

[39] R. Sausner. Authentication Software: Widespread Adoption Seen As Unlikely Before 2011. http://www.americanbanker.com/usb_issues/116_4/-274048-1.html, 2006.

[40] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 51–65, Washington, DC, USA, 2007. IEEE Computer Society.

[41] I. Strouchliak. Conversion rate optimization. http://www.seochat.com/c/a/Website-Marketing-Help/Conversion-Rate-Optimization/, 2009.

[42] S.-T. Sun, K. Hawkey, and K. Beznosov. Secure Web 2.0 content sharing beyond walled gardens. In *Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC)*, pages 409–418. ACSA, IEEE Press, December 7-11 2009.

[43] S.-T. Sun, Y. BoshMaf, K. Hawkey, and K. Beznosov. A Billion Keys, but Few Locks: The Crisis of Web Single Sign-On. In *Proceedings of the New Security Paradigms Workshop (NSPW)*, Concord, MA, USA., to appear.

[44] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying Wolf: An empirical study of SSL warning effectiveness. In *Proceedings of 18th USENIX Security Symposium*, pages 399–432, 2009.

[45] Sxipper Inc. Sxipper form manager Firefox extension. http://www.sxipper.com/, 2009.

[46] The Information Card Foundation. Advance the use of Information Card. http://informationcard.net/foundation, 2009.

[47] VeriSign Inc. VeriSign OpenID SeatBelt Plugin. https://pip.verisignlabs.com/seatbelt.do, 2009.

[48] Wikipedia. Password fatigue. http://en.wikipedia.org/wiki/Password_fatigue, 2009.

[49] L. Wroblewski. *Web Form Design: Fill in the blanks*, chapter Gradual Engagement. Rosenfeld media, 2008.

[50] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 601–610, New York, NY, USA, 2006. ACM.

[51] M. Wu, R. C. Miller, and G. Little. Web wallet: preventing phishing attacks by revealing user intentions. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 102–113, New York, NY, USA, 2006. ACM.

[52] Yahoo Inc. Browser-Based Authentication (BBAuth). http://developer.yahoo.com/auth/, December 2008.

[53] K.-P. Yee and K. Sitaker. Passpet: convenient password management and phishing protection. In *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, pages 32–43, New York, NY, USA, 2006. ACM.

[54] Y. Zhang, S. Egelman, L. Cranor, and J. Hong. Phinding phish: Evaluating anti-phishing tools. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, 2007.