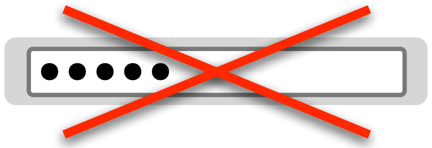


<input type="password"> must die!

Daniel Sandler Dan S. Wallach
Rice University
{dsandler,dwallach}@cs.rice.edu

Abstract

We propose that the HTML password input widget is harmful to user security, as it draws attention away from relevant security indicators, exposes a password's keystrokes to hidden client-side code, and generally conditions users to supply sensitive information in insensitive places. In this paper we advocate private password entry: *a mandatory, common authentication user experience that allows the user to enter a password for any site* in private, *free from snooping JavaScript. We describe a UI design for private password entry called the* password booth *that is backward-compatible with HTML login forms on most existing websites. It can be used to provide timely and relevant security indicators, as well as potentially unify and enhance other advances in authentication on the web. We hope that the password booth approach will, like a voting booth or a bank-card PIN pad, become a security feature that users come to expect for their own peace of mind.*

1 Introduction

The typical internet user of five years ago might have entered a username and password on a bank's website, a handful of e-commerce sites, and webmail. By contrast, 2008's web constantly demands authentication, promising personalization, social networking, advertising, and other features tailored to the individual user. Many websites even request a user's password multiple times in a single session; this is intended to be a security feature.

As a result, users are now regrettably quite accustomed to typing their passwords wherever and whenever requested. Developers have succeeded in training users to divulge this identifying information on demand, and in so doing, have desensitized users to its preciousness. It is remarkably easy to accidentally give a valid password to the wrong website, whether the user is being attacked (i.e., phished) or has simply forgotten which of a handful of passwords is the right one for a legitimate page.

More troubling still is the password field's exposure to JavaScript. Client-side code can extract the password field's contents and even snoop on keystrokes; this is true even in cases where the JavaScript comes from a different server than the main document. The result is that advertising systems, web statistics services, and mashups can silently capture password information from an otherwise innocuous page before the user has even clicked "submit" on a login form.

The HTML password input widget trains users to be phished and exposes their passwords to hidden code; it must therefore be considered deleterious to user security. In this paper we advocate *private password entry*: a common authentication user experience that allows the user to enter web passwords *in private*, free from snooping JavaScript. The idea of a trusted dialog box is not new, but we have found that such an interface is now urgently needed for securing web logins. We sketch a design for private password entry called the **password booth** that works with conventional HTML login forms, and as a result can be deployed immediately into WWW user-agents. The password booth offers an excellent place to forefront security indicators and warnings that are typically inconspicuous or too far removed from the password-entry context to be relevant. The browser shows the booth whenever a password field is encountered, so the user does not need to remember to invoke security features explicitly, and laggard websites (or malicious ones) may not opt out of the security features.

The password booth idea is not revolutionary; it is, however, a straightforward improvement that can be adopted immediately in browsers. We hope that this approach to private password entry, as with a voting booth or a bank-card PIN pad, will become a security feature that users will come to expect for their own peace of mind.

In the next section we elaborate on the security issues surrounding Web password input. In § 3 we propose private password entry as a useful and incremental remedy. We discuss related work in §4 and conclude with §5.

2 Background

2.1 Password leakage

A "password leak" occurs whenever the user gives a password to a remote entity that should not receive it. This may occur during an active attack on the user; a common example is a phishing attack, in which the user is fooled (often via a targeted email masquerading as legitimate correspondence) into logging into a forgery of a legitimate website.

Passwords may also be leaked in an altogether innocuous and, regrettably, quite common situation: when the user forgets which of his many passwords he has used at a given site. The authors can personally attest to typing several high-security passwords into the same reputable website while striving to remember which is the correct one. While the remote party in this case is not an attacker, it has still received passwords to which it should not be privy, and those passwords can be considered to have leaked.

In this paper we focus on password entry because, as a general-purpose authentication token, a password typically provides access to more sensitive, "principal" secrets (e.g., US Social Security number, credit card or bank numbers, information about children). Exacerbating the problem is a sort of "secrecy fade effect:" while users are suspicious about revealing principal secrets, they seem to divulge passwords more readily, even if doing so would allow the recipient access to principal information.

2.2 Security indicators

As a result of intensifying phishing and pharming activity, browser security indicators have lately received a great deal of attention and scrutiny, including a recent study [11] suggesting that existing presentation is insufficient to the task of properly informing the user's security decisions. It is unclear, however, whether larger or more obtrusive security indicators will ameliorate the problem.

We hypothesize that the central issue is one of *locality.* That is, current security indicators are divorced from the moment of critical information loss (i.e., password entry) either by space or time (or both).

Spatial separation. The browser's persistent security indicators are designed to be unobtrusive. (Unnecessary interruption of everyday Web browsing poses an irritation to the user, or worse, threatens to further desensitize the user to security information.) An unfortunate consequence is that the relevant indicators (e.g. SSL padlock, URL bar with domain name) are far from most password input fields, and thus out of the user's focus while interacting with the password form.

Temporal separation. Security warnings (for example, about certificate validity) come either too early or too late. That is, they appear when *encountering* or *submitting* a form, not when typing a password into that form. Early warnings—those presented before a form is shown—force a user to decide whether to abandon progress or to bypass the warning to see what is on the other side. The Emperor study [11] indicates that users are quick to dismiss such a warning; we may speculate that this is at least in part due to the perceived innocuousness of the proscribed action (simply visiting a page). By the time the user engages in risky behavior, namely, entering a password on that page, the warning is long gone. Similarly, if the page on which the form resides is unremarkable but the *destination* of the form is questionable, any warning issued to the user comes too late: his keystrokes may have already been captured by malicious code (as we will see in §2.3).

The end result, for the user, is that the critical security warning is either "out of sight, out of mind" or "out of time, out of mind."

2.3 The all-seeing eye of JavaScript

Although characters entered may be obscured from view to prevent shoulder-surfing, a password field appears to JavaScript no different than any other form object in the Document Object Model. Its current value may be read at any time (using the `.value` DOM attribute), and handlers may be registered on `keyDown` events so that keystrokes may be captured. This means that any JavaScript code running in the context of the current document can capture information from any password input field, before any forms have been submitted, and indeed as the password is being typed. Phishing websites and pharming attacks may take advantage of the same techniques to silently add keylogging code to otherwise innocuous websites, meaning that even if a user realizes she's on an untrustworthy page before submitting her password, that password may already have been captured. This goes directly counter to the user's (quite reasonable) expectation that her keystrokes are private until she clicks "submit." If the user checks for security indicators after typing a password but before submitting a form, it is already too late.

Even trustworthy websites are at risk if they rely on JavaScript from third parties. Whereas JavaScript executing in a document loaded in an `<iframe>` is disallowed access to the DOM of the parent document unless the two documents share a server origin, no such constraint is applied to JavaScript code included by URL in the document (i.e., via `<script src="...">`). Therefore, when a website includes foreign JavaScript code on a page with a password field, **it implicitly trusts that code with its users' passwords.** For example, the Twitter social messaging service includes web statistics code from Google Analytics on its main login page; this means, in effect, that Google has access to all Twitter login credentials.

Perhaps Twitter passwords aren't much to fuss over, but we note that as of this writing:

- Many websites (including Yahoo!, the Apple Store, and E*Trade) use Akamai to host JavaScript on pages that also include login forms.
- E*Trade also includes JavaScript from `clickfacts.com` on its login page.
- PayPal's front page (which includes a login form) includes JavaScript from `doubleclick.net`.

This is merely a cursory survey of different-origin JavaScript on sensitive login forms. It is unknown just how many websites—from newspapers to Web 2.0 services to individual blogs—allow third-party JavaScript for polls, pageview statistics, or advertising to share a page with their login forms. In the advertising scenario, syndication partnerships mean that advertisement JavaScript may even come from a fourth party (the advertiser).

The potential for abuse is clear: a malicious piece of JavaScript—whether included from a compromised third party, installed on a victimized website via SQL injection, included via a questionable mashup or or inserted in a phishing page—has full access to password input. This is particularly dangerous because of its invisibility and counter-intuitiveness: even a security-minded user might not consider that a form on a trusted page whose HTTP `POST` target is another trusted page might be vulnerable. No combination of security indicators will reveal the presence of such a covert channel. [1]

2.4 Whither HTTP authentication?

HTTP Basic and Digest authentication [6] are immune to these problems, for their interaction with the user is entirely external to HTML presentation and beyond the reach of JavaScript. The conventional browser user interface for this exchange is a graphical dialog box. It does not deliver password data to a form (where JavaScript may capture it), and it does attempt to provide an external, non-spoofable experience. In this sense HTTP authentication has the right idea: a password is requested from the user and handed directly to the destination.

However, the design of this dialog may be replicated with web content; even a rough approximation may be enough to fool most users into typing a username and password into a hostile environment posing as an HTTP authentication dialog. Furthermore, some portions of the HTTP authentication user interface are freeform and may be filled by a malicious site with "comforting" information that appears to come from the browser itself [9].

Finally, we note that HTTP authentication has been largely abandoned by the web development community. Digest-Auth, introduced as a security improvement over Basic-Auth, suffers from interoperability problems [7]. Both authentication schemes have limited support for controlling session duration (no obvious user interface for logging out, for example). Perhaps most importantly, however, web developers have chosen `<form>`-based login boxes over HTTP authentication because of their presentation flexibility; forms can be styled and branded to match the website instead of the user's operating system. Regrettably, this decision offers users the worst of all worlds: passwords sent in cleartext like Basic-Auth, but in a phishing-friendly, mutable visual presentation that also allows JavaScript side channels.

3 The Password Booth

We believe that, given the foregoing, it is time to revisit the safeguards in place to protect HTML form inputs—in particular, password fields, the most sensitive of these—during entry, and to properly inform the user when choices must be made. While our ultimate goal must be a trusted path from the user to the remote site, fully addressing the general trusted-path problem will require backward-incompatible changes (see § 4). We focus instead on cutting a trusted "tunnel," through untrusted client-side JavaScript, that can be decorated with just-in-time secu-

[1]Note also that security indicators fail to inform about network connections made by JavaScript during `XMLHTTPRequest`s; this means the security of login forms implemented as AJAX widgets cannot be accurately conveyed to the user. Our instinct here is that AJAX login schemes are an effective and succinct demonstration of the very tenuous security situation of password fields in the modern web.

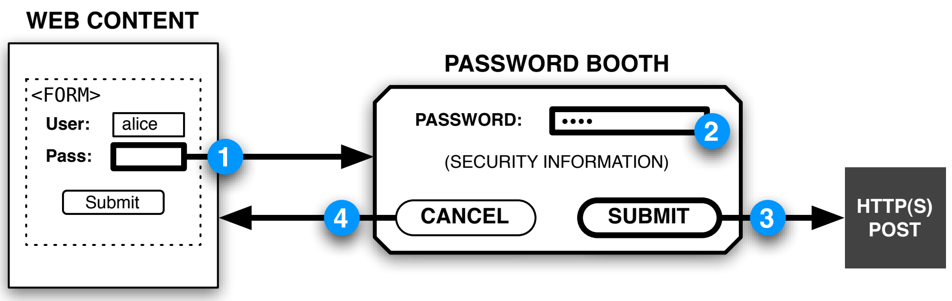


Figure 1: Flow of the password booth. (1) The user gives focus to a conventional password field, but rather than being allowed to enter text, she is taken to the private password entry interface where she may enter her password (2). From here she may submit the form (3) without returning to the previous page, or cancel (4), returning to the previous form without providing it any information (even if a password was partially entered).

rity information to help the user make an informed decision.

To be practical and useful in the near term, any such solution must be backward-compatible with existing web sites. It must also be compulsory, so that the user does not need to remember to explicitly invoke the security features, and also so that a phishing attack cannot plausibly claim that "our site's security features are unavailable today, please type your password." It should also allow the user to *roam* away from her primary machine without sacrificing all the security benefits of the system.

Our proposed solution is to deactivate the HTML password input widget altogether, so that the user is no longer asked (or, indeed, able) to enter a password in arbitrary unsafe environments. Its functionality is replaced by the **password booth**, a design that provides a *private password entry* context for the user. Like a voting booth, an ATM vestibule, or point-of-sale PIN pad, the password booth is designed to provide the user with a "safe" environment. In particular, in a password booth,

- **The user** can comfortably enter a password without fear of divulging it to any party but the intended `POST` action listed in the `<form>`.

- **The system** can provide critical security advisories to the user relevant to the entry of a password.

This approach provides immediate benefits to any site currently using conventional password entry based on `<input type="password">`; it therefore composes nicely with single sign-on systems (including OpenID) as well as site-authentication systems (e.g., SiteKey; Yahoo! Sign-in Seals).

We show the user's flow through the password booth in Figure 1 and describe its feature set below.

3.1 Functionality

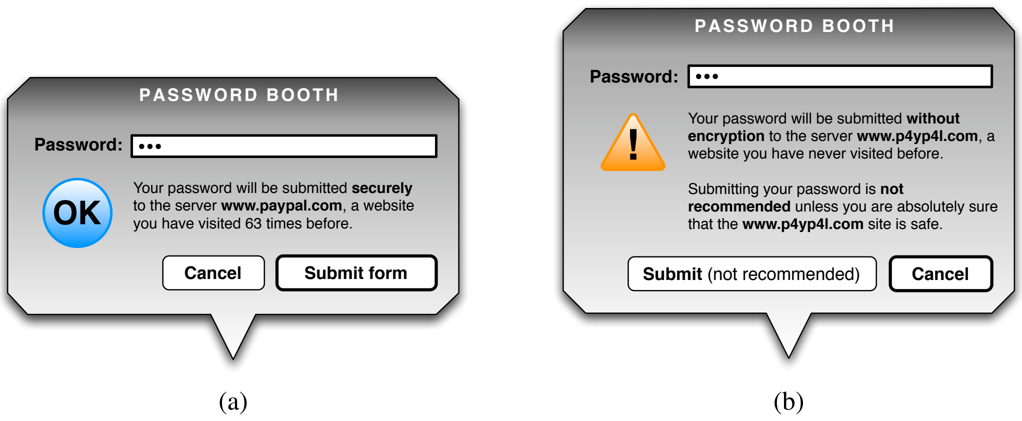
Password entry. The password booth allows the user to enter a password. The password's characters can be obscured as is customary, though this is not required (and may even be counterproductive if we are to encourage users to use longer pass phrases that require more care to correctly enter).

Non-re-entrance. There are only two exits from the password booth: Cancellation (returning to the current page, without supplying any password data to it) and submission (constructing and sending the HTTP `POST` request). The user's password data must never be handed to the possibly-insecure webpage on which the form resides.

Integration with HTML4. To be practical, a private password entry solution must be compatible with existing web login forms and should require only minor improvements to browsers. The password booth can be integrated into the browsing experience without modification to most existing sites. When the user gives focus to a password input field, the text editing behavior of the widget is suppressed; instead, the password booth is displayed (Figure 1). Use of the password booth is compulsory, thus *security is the default mode*: the user does not need to invoke any special UI function to use it.

Websites with a conventional HTML form containing a single password field need no modification; forms that allow password input for purposes other than login (e.g. setting passwords) may be modified not to use `type="password"` inputs.[2] Alternatively, we may pig-

[2]We must not be tempted to introduce an extension to HTML that indicates that for a given password field, the booth should not be used. Our goal is to forcibly eliminate insecure password entry, and a `NOBOOTH` addition to the password input would quickly be adopted by web developers uninterested in the password booth's security features. Users would consequently fail to develop a suspicion of pages that do not use the booth.



Your password will be submitted **securely** to the server **www.paypal.com**, a website you have visited 63 times before.

Your password will be submitted **without encryption** to the server **www.p4yp4l.com**, a website you have never visited before.

Submitting your password is **not recommended** unless you are absolutely sure that the **www.p4yp4l.com** site is safe.

Figure 2: Security information in the password booth. In subfigure (a), the browser confirms that the form will be posted to `paypal.com` using HTTPS. Additionally, since this is the user's home machine, the browser is able to show that she has posted to this form in the past. Subfigure (b) shows a more dangerous scenario; the browser recommends in this case that the user not provide her password.

gyback on the browser's detection of a password-change page (used in form auto-filling) and present a different, multiple-field version of the booth.

Note that for login forms, the "submit" control is no longer directly accessed by the user; only testing will tell us if this is a problem for users, but it is a necessary side effect of avoiding entry of password information in the form itself. Additionally, forms that use JavaScript buttons to log the user in are fundamentally insecure and must be replaced with conventional `POST`ed forms to allow private password entry.[3]

Security indicators. The moment when the user is entering her password is the right time to forefront the security properties of the current site and the form `POST` destination, including:

- The domain name of the destination.
- Whether the destination resides on a different domain than the one originating the form.
- Whether the destination is a `https://` URL.
- Relevant information from the site's SSL certificate in use (e.g., organization and signing authority).
- If available, the number of times the user has logged into this destination in the past.

Figure 2 sketches some of these security indicators as they might be provided in the booth.

[3] The trend toward JavaScript-based login forms is regrettable not only for the security reasons we describe here, but also on the basis of accessibility. Older browsers, mobile browsers, and users who have disabled JavaScript are all prevented from using such a website. Backward-incompatible and inaccessible features, frivolously introduced, are very much not in the spirit of the WWW.

Externality; non-spoofability. The trustworthiness of the password booth depends upon it being situated outside the usual HTML and DOM where foreign JavaScript may impinge upon it. It is equally essential that the booth be *presented* in such a way that web content may not replicate its appearance, in so doing fooling the user into believing she is entering a password in a safe area. As we have seen, the malleability of HTML presentation allows phishers to carefully replicate legitimate websites, and it is no less potent for mimicking native user interface elements. Possible techniques include integrating the booth into the browser's "chrome" (such as an expanding pane among the existing browser control cluster), or using a separate window whose design is not available to Web content (e.g., a non-rectangular shape, or a full-screen alpha-blended overlay). If the roaming requirement is relaxed, non-spoofability may be achieved using a per-user-per-host secret such as the "personal image" proposed by Dhamija and Tygar [5].

3.2 Future opportunities

The password booth is the first step in eliminating insecure password entry. It offers a solution that can be deployed now, incrementally, without modifying most websites, and that offers protection for users entering passwords. It also hints at more sophisticated authentication techniques that might be introduced in the private environment of the booth if we are willing to modify websites as well. The promise of two-way authentication is particularly appealing, as it may obviate many of the security indicators currently required to decide whether a website is trustworthy or not. §4 surveys current techniques that can potentially be brought under the umbrella of private password entry. More broadly, it is safe to assume that any further advances in web authentication will need trusted UI, and the password booth is a step in that direction that can provide benefits now and set user expectations for the future.

4 Related Work

Some of the problems we have identified with password input on the web have been addressed in prior work. These solutions, broadly, fall into two categories: content-based and browser-based approaches. We examine prominent examples here.

4.1 Web-based solutions

Bank of America's SiteKey [1] system is an example of an early attempt at authenticating a site to the user before she enters sensitive information. A SiteKey user registers an image and a phrase with the authentic site; upon returning to (what purports to be) the same site, she inputs her username only. Unless the site responds to this challenge with the user's chosen image and text, the user should not enter her password.

SiteKey has been shown to be vulnerable to man-in-the-middle attacks [18], however, as it is straightforward to request a user's personal image from the server and then re-present it to the unwitting user. Recognizing this vulnerability, locally-bound site authentication mechanisms like the Yahoo! Sign-In Seal [2] bind the user's security image to the user's *computer* (in fact, a particular browser on said computer, via a cookie) instead of to a username. This curbs the ability of an attacker to capture a user's chosen seal image. Unfortunately, due to the fragility of the cookie-based approach, there are many benign reasons that the sign-in seal may not appear even if the user is not being phished [16]. This allows an attacker to simulate one of these innocuous scenarios and request the user's password (along with a new seal image). Furthermore, any solution that relies on cookies is awkward for roaming; the user is shackled to her own computer if she wants phishing protection.

OpenID[4] is a decentralized authentication system. A user creates an account with an identity *provider*, whose URL she uses in lieu of a username when logging in to a website. She is directed to log in with her identity provider using whatever mechanism the provider demands (a username/password is common). Finally, she is redirected (with the identity provider's affidavit of her successful login) to her destination. Like many single sign-on systems, OpenID is highly vulnerable to phishing attacks that interpose themselves between the user and her OpenID provider [8]. Therefore, when used with password-based authentication, OpenID is exemplary of the weak security of current web-based password entry.

[4]http://openid.net/

4.2 Browser extensions

PwdHash [10] addresses the problems of password diversity and complexity by generating per-site passwords based on a hash of the user's (memorable) master password and the site's domain name PwdHash elegantly requires no modification to existing websites, but it has a very severe roaming issue: the actual passwords it establishes with websites are unmemorable hash values. It is therefore difficult to login to a website without your access to your own computer; PwdHash proposes a public "password calculator" service which users must trust and learn how to use while on the road.

Passpet [17] combines the PwdHash technique with personalization and automatic form filling. As such, it frees the user from typing in passwords altogether, so she cannot accidentally type them into a malicious form. Its solution to the roaming problem is a server which stores a user's data. Trust issues aside, this is indeed effective for systems that have Passpet installed, but if the software is missing the user is not only afforded no security benefits, she can't log in at all (as with PwdHash).

The Web Wallet [14] extends the notion of saved passwords (a common feature of modern browsers) to a browser sidebar. The Web Wallet requires a user to press a special security key to open the wallet, allowing her to choose one of her personal ID cards from the wallet to insert into a login form. Unfortunately, in addition to being non-portable, the Web Wallet requires explicit invocation by the user; the authors acknowledge that users have been so effectively trained to type passwords directly into web forms that it is difficult to remind them to take an additional step to activate security features.

Microsoft CardSpace (née InfoCard) combines content-based and browser-based methods in service of a larger identity metasystem [4] that manages a user's personal information, institutionally-assigned identification data, and website authentication. This is a vastly larger problem than the one we are trying to tackle here, but the CardSpace solution shares our insight that personal information must not be entered into untrusted web forms but must instead be handled in a safe local environment. It is worth noting that CardSpace suffers from the same problems that other browser extensions do (portability problems for roaming users; must install additional software) as well as the problems faced by content-based approaches (requiring substantial modifications to websites wishing to participate).

Steiner et al. [12] integrated into SSL a protocol for *mutual authentication* based on an encrypted key exchange (EKE) zero-knowledge password protocol [3]. This approach allows the server to be convinced that the user knows the proper password while even an active attacker would learn nothing at all. More importantly, however, it offers two-way authentication: the user has confidence that the *server* also knows the user's password (or some derived value thereof). Dynamic Security Skins [5] apply a similar approach to web security, but instead use changes in the browser's user interface (determined programmatically from the results of the key exchange protocol) to give the user clues as to the authenticity of the website. The Secure Remote Password (SRP) is a similar eavesdropper-proof password protocol [15]; SRP-TLS, recently proposed as RFC 5054 [13], integrates SRP as an authentication method in the TLS standard; however, it is not yet widely supported by web servers and not at all by browsers.

These techniques are powerful, but deployment requires modifications to both the server and client; more importantly, it will necessitate a trustworthy password entry user interface not unlike the password booth. While we wait for adoption of these more sophisticated authentication schemes,[5] we can begin accustoming users *now* to the password booth for conventional `<form>` logins.

5 Conclusion

We have argued that `<input type="password">` is a detriment to user security. The power of HTML and JavaScript is sufficient to allow any web developer to precisely mimic any website, which is sufficient for most users to believe that the site is authentic. We have unfortunately conditioned users of the web to type passwords wherever and whenever they are asked.

This must end; we must give users a secure, *private* place to enter passwords—the password booth—and train them to feel *unsafe* entering passwords outside of the booth. This naturally leads to explorations of how we might eventually capture other sensitive information in this secure space (in a way that will still offer some security when she is away from her primary computer). Designing this more general "web booth" is future work.

[5]Perhaps related to the pending expiration of relevant EKE-related patents (U.S. Pat. #5,241,599 and #5,440,635).

We are currently working to implement the password booth design as an extension for existing browsers, with the intent to eventually build this mechanism into the browser itself. The next step will be to perform a user study to assess the password booth's ability to alert users to unsafe situations by focusing attention on security information when and where it is necessary.

The password booth is hardly a complete solution to the threat of phishing and the problems of authentication on the web, but it is a substantial step that can be taken with minimal impact to existing websites and that may finally make security indicators relevant.

Acknowledgments

This work was funded in part by NSF grants CNS-0524211 and CNS-0509297. We thank the anonymous reviewers for their critical notes and useful suggestions.

References

[1] SiteKey at Bank of America. http://www.bankofamerica.com/privacy/sitekey/. Accessed February 2008.

[2] Yahoo! Security Center: What is a sign-in seal? http://security.yahoo.com/article.html?aid=2006102507. Accessed February 2008.

[3] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1992.

[4] K. Cameron and M. B. Jones. Design rationale behind the identity metasystem architecture, 2006. http://www.identityblog.com/wp-content/resources/design_rationale.pdf.

[5] R. Dhamija and J. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the Symposium on Usable Security and Privacy (SOUPS)*, Pittsburgh, PA, July 2005.

[6] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[7] J. Gregorio. Problems with http authentication interop, Jan. 2006. http://bitworking.org/news/Problems_with_HTTP_Authentication_Interop.

[8] B. Laurie. OpenID: Phishing heaven, Jan. 2007. http://www.links.org/?p=187.

[9] A. Raff. Yet another dialog spoofing—Firefox basic authentication, Jan. 2008. http://aviv.raffon.net/2008/01/02/YetAnotherDialogSpoofingFirefoxBasicAuthentication.aspx.

[10] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser

extensions. In *Proceedings of the 14th USENIX Security Symposium*, Aug. 2005.

[11] S. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *IEEE Symposium on Security and Privacy*, Oakland, CA, May 2007.

[12] M. Steiner, P. Buhler, T. Eirich, and M. Waidner. Secure password-based cipher suite for tls. *ACM Transactions on Information and System Security (TISSEC)*, 4(2):134–157, May 2001.

[13] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) protocol for TLS authentication. RFC 5054, Nov. 2007. http://www.ietf.org/rfc/rfc5054.txt.

[14] M. Wu, R. C. Miller, and G. Little. Web wallet: Preventing phishing attacks by revealing user intentions. In *Proceedings of the Symposium on Usable Security and Privacy (SOUPS)*, Pittsburgh, PA, July 2006.

[15] T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, San Diego, CA, Mar. 1998.

[16] Yahoo! I don't see my sign-in seal. should i be concerned? http://help.yahoo.com/l/us/yahoo/edit/privacy/edit-35.html. Accessed February 2008.

[17] K.-P. Yee and K. Sitaker. Passpet: Convenient password management and phishing protection. In *Proceedings of the 2nd Symposium on Usable Security and Privacy (SOUPS)*, Pittsburgh, PA, July 2006.

[18] J. Youtl. Fraud vulnerabilities in SiteKey security at Bank of America. Technical report, Challenge/Response, LLC, July 2006. http://www.cr-labs.com/publications/SiteKey-20060718.pdf.