BeamAuth: Two-Factor Web Authentication with a Bookmark

Ben Adida
School of Engineering and Applied Sciences
Harvard University
33 Oxford Street
Cambridge, MA 02138
ben@eecs.harvard.edu

ABSTRACT

We propose BeamAuth, a two-factor web authentication technique where the second factor is a specially crafted bookmark. BeamAuth presents two interesting features: (1) only server-side deployment is required alongside any modern, out-of-the-box web browser on the client side, and (2) credentials remain safe against many types of phishing attacks, even if the user fails to check proper user interface indicators. BeamAuth is deployable immediately by any login-protected web server with only minimal work, and it neither weakens nor interferes with other anti-phishing techniques. We believe BeamAuth may be most useful in preventing a number of phishing attacks at high-value single sign-on sites, e.g. OpenID providers.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*; K.4.2 [**Computers and Society**]: Social Issues

General Terms

Design, Human Factors, Security

Keywords

phishing, two-factor authentication, web security

1. INTRODUCTION

Web-based authentication is vulnerable to a staggering number of social engineering attacks, typically called phishing [19]. Attackers provide a spoofed web page, where the user is fooled into entering her credentials. The spoof can take the form of a simple user-interface deception, sometimes with a URL crafted to resemble the purported destination in order to trick even users who check the address bar. Recent variants, called pharming attacks [37, 38], are significantly more cunning: by spoofing DNS or even IP addresses, the attacker's phishing URL matches *exactly* the purported destination. Pharming attacks are becoming easier to carry out using, for example, malicious base stations to which wi-fi users might innocently connect. The only remaining defense is the SSL certificate warning, which many users ignore [11].

Much recent work proposes defenses against phishing attacks, including site-specific password pre-processing [41], cryptographic protocols combined with trusted-path user-interface indicators [12], and altogether novel methods of web authentication [8]. Unfortunately, all of these solutions require new client-side code, which greatly limits their deployability until major web browsers implement the feature and a large portion of web users upgrade accordingly. When the proposed change is implemented as a browser add-on, new trust and attack surface issues arise: the add-on usually has full control over the user's browser.

At a high level, it is well known that multi-factor authentication is preferable, though not foolproof, in defending against social engineering attacks. Yet multi-factor authentication is difficult to implement in an out-of-the-box browser. One extension-free approach to web-based two-factor authentication is site-image verification, e.g. BankOfAmerica's SiteKey [3] or Yahoo's sign-in seal [52]: the server provides a personalized login image to browsers previously tagged with a long-lasting cookie, and the user is expected to enter her password only if she notices her expected personal login image. The long-lasting cookie plays the role of a second factor, and the login image provides some form of human authentication of the server requesting the user's credentials.

Our solution, BeamAuth, provides second-factor authentication using a specially crafted bookmark instead of a cookie. We believe this approach provides a few notable advantages:

1. our token is hidden inside a bookmark rather than a cookie so that it is less vulnerable to cross-site scripting (XSS) attacks [9],

2. a bookmark has fewer privacy side-effects than a cookie, making it less likely to be deleted by routine cookie deletion, and

3. a user's multiple browsers and computers can be automatically set up for BeamAuth using any one of numerous existing bookmark synchronization tools.

CCS'07, October 29–November 2, 2007, Alexandria, Virginia, USA.

Target Audience. Our proposal's major downside is its requirement of a distinct bookmark for each site protected by BeamAuth. Thus, BeamAuth is useful mostly to secure a user's few high-value sites, i.e. banks or single-sign-on systems. We believe our approach may be of particular interest to OpenID [39] and other single-sign-on providers that automatically redirect users to their login page. Some consider that such auto-redirect behavior leads to increased phishing risk [7, 30]. BeamAuth should help alleviate some of this risk while furthering the goal of a login solution that does not require browser changes.

1.1 The Fragment Identifier

The fragment identifier is a long-standing and well-specified web feature for accessing portions of URLs [6]. As an example, the URL `http://example.org/stuff#paragraph4` specifies the fragment named `paragraph4` within the primary resource designated by `http://example.org/stuff`. When encountering a fragment identifier, web browsers scroll the viewport to the designated chunk, e.g `paragraph4`.

In accordance with the specification, major web browsers never send the fragment identifier over the network: the primary URL is requested, and the fragment is used only for client-side scrolling. In addition, when a user navigates from one fragment identifier to another within the same primary URL, the page is not reloaded, it is only scrolled appropriately[1]. If no fragment with the requested identifier exists, all major browsers simply ignore the fragment identifier, though it remains in the address bar. (It is interesting to note that browsers also omit the fragment identifier from the `HTTP_REFERER` field, though we do not make use of this property in this work.)

Notably, a page's JavaScript code can read, update, and delete the fragment identifier at will, without causing a reload or causing any data to be sent over the network. All updates are immediately reflected in the address bar, and the JavaScript code can even choose whether this change should be reflected in the browser history or if the previous fragment should be forgotten altogether, inaccessible even using the "back" button.

1.2 Our Contribution: Second-Factor Authentication with a Bookmark

We propose BeamAuth, which treats the fragment identifier as a *local cryptographic input to the web page's JavaScript code*. Alice, the user, installs a bookmark containing a secret token in its fragment identifier: `https://site.com/login#[TOKEN]`. She can use this bookmark to reach the site directly. Alternatively, if she is directed to `https://site.com/login` by normal web surfing activity, she can be prompted to click her BeamAuth bookmark. Since the bookmark click doesn't change the primary URL, the page *does not reload or cause any network access* [2]: the only change is that `#[TOKEN]` is now appended to the URL. In either case, by polling the address bar field, the login page's JavaScript notices this fragment identifier, stores the token in a variable, clears it from the address bar, and prompts Alice for her password. Once entered, this password is cryptographically combined with the token and used as the credential for the login.

[1]with some exceptions addressed in Section 3.

[2]with some exceptions, which we work around in Section 3.

Importantly, if Alice is being phished and is thus *not at her real login page*, the bookmark click will cause the browser to load the *real* login page. If she notices this page load, Alice may realize she was being phished. However, even if she doesn't, her credentials remain safe. If the phishing site convinces her not to click her bookmark (or she forgets), her password may be compromised, but this is only one of two required tokens for login. In other words, it becomes much more difficult to phish Alice: even if she makes a simple mistake, her credentials remain safe.

Auto-Redirect Single Sign-On Sites. Each site that uses BeamAuth requires a setup procedure and some bookmarks-bar real-estate. Thus, this technique is best left to high-value sites, like financial institutions or single sign-on systems. BeamAuth may be particularly interesting for auto-redirect single-sign-on sites like OpenID identity providers [39], where Alice may be more easily phished because she is accustomed to being redirected to her login site by the site requesting authentication. The BeamAuth bookmark click prevents the most common phishing attacks in this scenario.

1.3 Limitations

Though we believe BeamAuth provides significantly enhanced security against phishing with close-to-optimal deployability, it is by no means perfect.

JavaScript Required. BeamAuth requires JavaScript. A user who turns off JavaScript cannot get the proposed benefits. Fortunately, the login provider can detect JavaScript during bookmark setup, and either prompt the user to enable it or fall back to another authentication mechanism which doesn't require JavaScript.

Certain Attacks May Still Succeed. Malware on the user's computer, pharming attacks on non-SSL sites, or pharming attacks on SSL sites where users pay no attention to certificate validity indicators may succeed in defeating BeamAuth. These are considered out of scope for this proposal, though we note that BeamAuth generally does not interfere with other proposed defenses against these more advanced attacks.

1.4 Related Work

Anti-Phishing. The range of anti-phishing solutions is vast. Some techniques focus on detecting spoofed emails in email-based phishing attacks using cryptographic techniques [4, 46, 2], origin or path-based verification [23, 32, 31, 20], or heuristics similar to those used for spam detection [42, 33, 35]. Other techniques focus on providing web-browser toolbars that use various heuristics or collaborative filtering to determine what might be a phishing attack [36, 22]. Both Internet Explorer 7 and Firefox 2 consult databases of known phishing web sites to warn users. More recently, techniques have emerged that strengthen the cryptographic and trusted-user-interface-path capabilities of web browsers in order to implement more intricate authentication protocols [41, 12, 8]. It takes a whole book to describe the range of phishing variants and known defenses [25]. We note that most of these defenses require software-install updates to either the user's web or email client.

JavaScript Use of the Fragment Identifier. The fragment identifier has been usurped in other ways, usually as a mechanism to maintain state in a single-page JavaScript web application. S5 [34], an HTML slide presentation tool, uses the fragment identifier to indicate which slide to display, with the whole slideshow contained in a single HTML file. The Dojo JavaScript toolkit [27] and other JavaScript libraries use the fragment identifier to maintain state information, so that a user may click the forward and back buttons normally in an AJAX [17] web application without full page reloads. As far as we know, fragment identifiers have not yet been used as secure tokens in cryptographic authentication protocols.

Security in the Web Application Layer. Others have proposed security protocols that make use of existing browser features in novel ways, effectively building security into the web application layer. Juels et al. [28] propose to use "cache cookies" for security: the browser cache stores secret tokens for two-channel authentication at secure sites, e.g. online banking. Jackson and Wang [24] explore various existing browser features to enable secure cross-domain communication for web mashups. BeamAuth aims to achieve the same deployability for phishing-resistant web authentication.

Bookmarks for Security. Two recent Internet web postings [7, 30] suggest the use of a bookmark to curtail phishing attacks against OpenID servers, or generally any web single sign-on system where users are automatically redirected to their login page. Though these are very interesting and useful suggestions, our technique is a bit different: we do not need to change existing single sign-on protocols; we only tweak how Alice enters her credentials. We also use the bookmark as more than a server locator: the BeamAuth bookmark serves as a second authentication factor.

1.5 This Paper

We present the core technical components of the BeamAuth technique in Section 2. Section 3 describes the details of the protocol, in particular the "user login ritual" and the initial browser setup procedure. Implementation details and an overview of performance metrics are described in Section 4. Threats, defenses, comparison to other solutions, and potential impact are discussed in Section 5.

2. BEAMAUTH BASICS

The core BeamAuth features are relatively simple: a secret cryptographic token is "injected" into the local page scope via the fragment identifier, a portion of the URL never sent over the network but accessible from JavaScript code. In this section, we present these issues in detail, especially their specific implementation differences across the four major browsers: Internet Explorer (6 and 7), Firefox (1.5 and 2.0), Safari (2.0), and Opera (8 and 9).

2.1 The URL Fragment Identifier

A URL [6] may contain a fragment identifier, which, as its name implies, addresses a fragment of the resource denoted by the primary URL. Specifically, a URL with a fragment identifier looks like:

```
http://hostname/rest/of/url#fragment_id
```

The resolution of a fragment identifier within a given document depends on the document's MIME type. In the case of an HTML document, with MIME type `text/html`, the fragment identifier specifies a portion of the HTML page identified accordingly. Importantly, a web browser will resolve the above URL as follows:

1. connect to host `hostname` on port 80,
2. request `/rest/of/url` and render the HTML page,
3. scroll the viewport to the position indicated by `fragment_id` if it exists, ignore it otherwise.

Note how `fragment_id` is never sent over the network. This property has been confirmed on all major browsers, and it is, in fact, part of the URI specification.

Navigation. When a user navigates from one fragment identifier to another within the same primary URL, the browser *does not trigger a page reload*: the page simply scrolls to the position indicated by the new fragment identifier (or, again, does nothing if no such position exists). In the two dominant browsers, Internet Explorer and Firefox, this absence of reload remains true no matter how the initial URL was loaded, regardless of the cache preferences on the downloaded page: even a page with strict no-cache HTTP headers that results from a `POST` operation is not reloaded when only the fragment identifier changes. Thus, all page state, be it local JavaScript variables or HTML form inputs entered by the user, remains unaffected when the fragment identifier changes.

There are small exceptions to these otherwise consistent rules. In Opera, changing the fragment identifier on a page that results from a `POST` does, in fact, trigger a reload, this time as a `GET`. In Safari, a change in fragment identifier triggered by an external source, e.g. clicking a bookmark or manually entering a new fragment identifier rather than clicking a link within the web page itself, does trigger a reload even if the primary URL does not change. Because these two browsers make up a notable 5% of web users [49], we special-case their support with extra server-side overhead: any form parameters are stored in a server-side session, so that the user can always end up at a `GET` URL without any URL parameters.

2.2 JavaScript Features

Fragment Identifiers. In all browsers, `window.location.hash` is a read/write JavaScript variable that corresponds to the fragment identifier as it appears in the browser's address bar. Changing the value of this variable updates the address bar without reloading the page, scrolls the viewport to the appropriate location (if it exists), and results in the new URL being added to the browser's history.

When we want to change the value of this fragment identifier without leaving a trace in the browser history, for example to clear a secret token from view, we use a slightly different mechanism. The JavaScript function `window.location.replace()` updates the URL (including the fragment identifier) without adding the previous URL to the history. It is as if the previous URL was never visited.

JavaScript Bookmark. It is tempting, for our purposes, to use a JavaScript bookmark, also known as a bookmarklet or favelet, which is effectively a small piece of JavaScript code that is executed when the bookmark is clicked. Some early prototypes of this work were implemented accordingly. Unfortunately, this code cannot be expected to behave correctly, because the bookmark's JavaScript is executed in the context of the current page. A malicious page might override any command, even the standard JavaScript API, thereby completely altering the behavior of the bookmark code and likely revealing the secret token to the attacker. (Some variables are declared `constant` in the JavaScript specification and should provide a safe baseline for such techniques, but most browsers do not respect these constraints, and it is risky to rely on consistent JavaScript behavior across all browsers for security purposes.)

2.3 Web Authentication Security

We briefly review the types of attacks that web users most often face when performing online authentication and how current HTTP security features address them.

1. **passive sniffing**: users often access web sites over open or insecure wi-fi access points, unswitched local wired networks, or corporate proxies. The URLs they request and the content they receive are easily sniffable when SSL is not used. The damage from these kinds of attacks is unclear, as most non-SSL-using web sites are small providers. However, the threat is well understood: while the W3C does not mandate SSL, the W3C's technical advisory group is considering recommending that login credentials never be sent in the clear [40].

2. **social engineering**: users are easily fooled by malicious sites that visually spoof legitimate sites to steal credentials. Financial institutions are the typical target, though other e-commerce sites are also targeted when there is an eventual financial gain. Users generally don't check the URL or even the SSL padlock of their connections [11]. The damage from these attacks is well documented and significant [19], and carrying out such an attack is fairly trivial.

   The most advanced type of attack in this category is the pharming attack, where a DNS record or even an IP address is spoofed to make the user believe she is visiting the correct site. This type of attack is on the rise via malicious open wi-fi base stations, which users tend to trust in their thirst for Internet access "on the go." Even when an incorrect SSL certificate raises a flag, users tend to ignore the warning [11]. This problem may be somewhat alleviated with Internet Explorer 7's strong disincentive to visit inconsistent SSL sites. However, to our knowledge, there is no reliable data yet as to whether user behavior is significantly affected.

3. **desktop compromise:** a surprisingly high number of desktop computers are compromised with malware [29]. Users of these compromised machines have zero guarantee of any security: all security indicators may be faked, and all host names may be hijacked. SSL is useless. Damage from these attacks is significant, though carrying out such an attack is typically more involved than either passive sniffing or social engineering.

SSL is not enough. It is clear that SSL is not enough to protect against desktop compromise attacks. It is also relatively well understood that, for high-value applications, SSL is still not enough to protect against social engineering attacks, as evidenced by the depressingly high success of such social engineering attacks. The key issue is that, even with SSL, the web remains treacherous: a momentary lapse in judgment, and Alice may be tricked into thinking that two 'v's are actually a 'w' [11]. As a result, some suggest that high-value sites resort to two-factor authentication, where at least one factor is not easily stolen from an inattentive user.

2.4 Goals of Our Proposal

We aim to make it more difficult to carry out social engineering attacks against customers of high-value web sites. High-value web sites should have an easy and relatively secure way to implement two-factor authentication without resorting to browser plugins or physical tokens. We specifically aim to provide a "safety net" for users, so that a moment of inattention will not immediately result in identity theft. In other words, we are attempting to make phishing significantly more difficult for the attacker. Importantly, we also aim aim to not interfere with other proposals that may help address sophisticated pharming attacks, which we do not address.

3. THE BEAMAUTH PROTOCOL

We consider high-value web sites, including in particular the single-sign-on use case in its many forms, where Alice is sent to her login page by a third-party web site, sometimes called the *relying party* because it relies on an authentication process performed by another party. For example, Flickr sends its users to Yahoo for authentication, and any web application can use Yahoo in the same way with Yahoo BBauth [51]. A growing number of web applications use OpenID [39] for authentication, where the relying party is expected to redirect Alice to her OpenID server. A number of university networks also use this same technique: Harvard University's PIN system [21] and Stanford's WebLogin system [44] are two prominent examples, where peripheral sites send users to the central login site which, after authentication, redirects the users back to the peripheral site with an authentication token.

In all of these cases, phishing is of great concern, since Alice is sent to her login page *by the site requesting authentication.* It has been noted in particular that OpenID may make phishing easier because Alice explicitly discloses her identity provider, and thus the identity provider's look-and-feel, to a potentially evil site [7, 30]. We aim to mitigate phishing attacks in this widespread scenario.

3.1 The Bookmark

With BeamAuth, we transform a typical browser bookmark into a second factor for web-based authentication, using a secret token in the bookmarked URL's fragment identifier:

```
http://site.com/login#[username|secret_token]
```

The User Login Ritual. Alice may reach a site she wishes to visit either by normal navigation (e.g. entering a URL in the address bar, clicking a link, etc.), or by choosing one of her bookmarks. Assuming Alice has already set up her BeamAuth bookmark at a particular site, we consider her login ritual when she happens upon the login page by navigation or manual URL entering, and we note that it is only slightly more complicated than the typical username/password process:

1. The web site prompts Alice: "click your BeamAuth Bookmark."
2. Alice clicks her bookmark, which updates the login page with her username, and the page now prompts her for her password.
3. Alice enters her password and clicks "Submit."
4. If both the bookmark token and the password are correct, Alice is correctly logged in.

Note how, by including Alice's username in her bookmark, this process may be immediately advantageous to Alice: she has less typing to do. This optimization should probably not be used if Alice's username has some external secret significance, e.g. a social security number, as an attacker who gains momentary physical access to Alice's machine could then read this data easily.

Interestingly, if Alice chooses to use her bookmark as one might expect – to *reach her site in the first place* – the login ritual skips immediately to Step 3. The two-factor protection remains, and Alice's username is also filled in automatically.

Setting up the Bookmark. To set up the BeamAuth bookmark within her browser, Alice must follow an initial authentication process that is inherently more involved than the everyday login. This should be done using a second-channel authentication mechanism, using, for example, a cell phone SMS [50], or an email mail-back [16]. We specifically recommend the email mail-back option, as it requires only an email client, which can easily provide a clickable URL containing a verification code that sends Alice right back to her browser. Many web sites already perform a mail-back verification to ensure that the user's email address is correct: BeamAuth can easily bootstrap off this existing process.

When Alice clicks on this verification URL, the web page she reaches provides her with a link that she can easily drag-and-drop onto her bookmarks/favorites toolbar. Of course, the verification link sent via email should be secure in authentication *and in content*: the verification code in the URL should never be sent in the clear. This can be achieved using SSL, so that a verification URL sent by email looks like:

```
https://site.com/confirm?vc=<verification_code>
```

It can even be done by placing the actual token in the fragment identifier, so that the token is truly never sent over the network, never logged by the web server, etc.:

```
https:
//site.com/get-bookmark#[username|secret_token]
```

The `get-bookmark` page is then a simple HTML template with bundled JavaScript that fills in the template on the client side by extracting the secret token in the fragment identifier:

```
bookmark_link.href =
  'https://site.com/login' + window.location.hash;
```

Re-Initialization and Multiple Computers. Though it is less likely than a cookie deletion, it is certainly possible that Alice will delete her BeamAuth bookmark by mistake. It is also very likely that Alice uses more than one browser, on more than one computer. All of these cases amount to the same problem: how often does Alice need to perform the initialization procedure, and will this be an impediment to using BeamAuth? Significant user testing will be required to answer these questions, but a few signs indicate that, for high-value sites, the situation may well be acceptable.

The simplest approach is to instruct Alice to keep the signup email around until she has set up all of her web browsers. She can visit the setup page and install the BeamAuth bookmark once on every browser she uses. If she loses the signup email, the BeamAuth-protected web site can lead Alice through a sequence of verification questions—similar to SiteKey [3]—and eventually send her a new copy of the signup email. Note that the token is *always* sent via a second channel, even if the verification questions are answered via the primary web channel.

Note also that bookmark synchronization is useful for many other purposes, not just BeamAuth. Google's BrowserSync [18] provides encrypted bookmark synchronization for Firefox, while Sync2It [45] provides the same functionality for all browsers, and Apple's dotMac [13] provides it for Safari. Any of these solutions is a good way to enable Alice to initialize one browser and synchronize the BeamAuth bookmark to all of her other browsers. Though BeamAuth makes special use of the fragment identifier, the "secret sauce" is in the site's JavaScript, not in the bookmark itself: the BeamAuth bookmark will be synchronized just like any other.

3.2 The Mechanism

A BeamAuth login page contains JavaScript that regularly polls the value of the fragment identifier (entirely locally, causing neither network activity nor server-side processing). When Alice clicks her bookmark, the URL is updated (without reloading) to include the token `[username|secret_token]`, and the BeamAuth JavaScript poller reads this token from the fragment identifier. It then fills in the login form with Alice's username, and saves the secret token into a local variable. The JavaScript then clears the fragment identifier so that the secret token is no longer visible in the URL address bar nor the browser's history.

When Alice submits the form with her password, the BeamAuth JavaScript code intercepts the form submit, HMAC's the password with the secret token, and securely submits this resulting credential to the server. Security of this transfer is ensured either via SSL or, if SSL is not available, using some kind of challenge-response approach such as that implemented by Yahoo [15].

Interestingly, the server need only store

$$\mathsf{hmac}_{secret\_token}(password)$$

never the password in the clear. This is effectively the same thing as keeping passwords stored hashed with a salt, a common recommendation for any password-based login system, except the salt-and-hash operation is performed on the client side in JavaScript, with the salt provided via a second channel—the bookmark. The security of this setup is not weakened, because we expect that the transfer of the resulting $\mathsf{hmac}_{secret\_token}(password)$ will be secured either by SSL or by an extra layer of HMAC-based challenge-response. The server may also store *secret_token* if it wants to let Alice regenerate a bookmark in the future without invalidating her other already installed bookmarks (e.g. on her other computers.)

Behavior under Attack. At a high level, we note that it is difficult for Alice to be tricked into revealing her secret token, because it is hidden inside her bookmark. This token appears in the address bar only for a short period of time, usually never long enough for Alice to even see it, and is then removed from the address bar and from the browser's history when the login page's code calls the `window.location.replace()` standard function call.

If Alice is being phished, she may forget to click her bookmark and reveal her password to the attacker, or she may remember to click her bookmark and be immediately whisked away to the real login site (again assuming no DNS or IP spoofing, or at least a user who takes the SSL certificate warning seriously.) The security of BeamAuth relies on the fact that it takes both tokens to log in, and that it is difficult to trick Alice into revealing her bookmark token. We explore attacks in greater detail in Section 5.2.

3.3 Limitations

The URL of the login page must match exactly what the bookmark expects, otherwise a reload will be triggered. If the login page needs certain parameters, they should thus be sent via `POST`.

Unfortunately, Safari and Opera do not fully support this approach. In Opera, the login page must be loaded via a `GET` operation if the local secret-token injection is to succeed without a page reload. In Safari, the page will *always* reloads on a bookmark click. Thus, for both of these web browsers, the login server should store any parameters in a server-side session, so that a page reload will not mistakenly delete them. This work-around requires more server-side state. As Safari and Opera together make up about 5% of the browsing public, it is important to build this work-around, but it is also reassuring that the extra overhead will be required only for a small fraction of users.

4. IMPLEMENTATION

In this section, we describe our BeamAuth implementation, which is fully functional online and available at:

`http://ben.adida.net/projects/beamauth/`

4.1 Setup

Hosted Server and Web Client. We used a typical shared-hosting provider, using a small portion of a quad-processor Intel Xeon 3.2Ghz server with 4GB of RAM, located in Houston, Texas. We tested Firefox 2.0.1, Safari 2.0.3, and Opera 9 on a Macintosh Powerbook G4 running at 1.5Ghz with 1.5 GB of RAM. We tested Internet Explorer 6 and 7 on Windows XP Professional running on a 1.8Ghz Intel Core Duo with 1 GB of RAM. Both client PCs were connected via a Comcast home broadband connection in Boston, Massachusetts.

Web Server and Application Logic. We use Python 2.4 [47] as the back-end programming language, with the CherryPy web environment [10] that simply maps URLs to Python class methods. The code is simple enough that it should be fairly easy to read even if one is not familiar with Python or CherryPy. We use the Apache [5] web server to handle all HTTP requests, with a `mod_proxy` interface to bridge Apache and CherryPy. We built the server-side BeamAuth features using the built-in CherryPy session support and the built-in Python HMAC API. The back-end code, including the mail-back implementation, contains approximately 200 lines of BeamAuth-specific code plus a few HTML templates and generic utilities to render these templates.

JavaScript. We use a JavaScript library [26] that implements HMAC-SHA1. Note that, while SHA1 has recently been shown to have certain weaknesses [48], its security in an HMAC setting has not been compromised. If it were to be compromised, a move to SHA256 would be fairly straightforward and only slightly more computationally intensive. Our small BeamAuth JavaScript library implements:

- polling, reading, and updating the fragment identifier,
- performing the login process, including UI updates, HMAC, and an HMAC-based challenge-response.

Our BeamAuth-specific JavaScript, not counting the HMAC library, is less than 50 lines of code.

4.2 Performance

We evaluated client- and server-side computational needs for performing HMACs. We determined that, on the slowest browser (Safari) using the specified Mac laptop, an HMAC operation requires just under 50ms. As this is entirely client-side computation, it is negligible and barely noticeable to the user. On the server side, in Python, one HMAC operation took 300μs on our setup, a modest computational requirement compared to the average database query.

5. DISCUSSION

We first explore, in greater detail, the threat model we considered. We then examine specific attacks and how BeamAuth fares. We briefly compare BeamAuth to site-image systems such as SiteKey [3], and we briefly note the interesting aspect of no-install security deployment.

5.1 Threat Model

We attempt to protect *authentication credentials* against phishing attacks, including the simplest user-interface deceptions, deceitful URLs, and pharming attacks that omit the SSL component on sites that are SSL-enabled. We specifically point out that we do not try to defend against some of the more involved attacks, including malware that effectively turns a user's machine against her, or pharming attacks that use SSL and expect the user to ignore the certificate warning. Certainly, these other attack vectors are worth

considering, though we believe that defending against them likely requires significant changes to browser code. We've attempted to provide a strong defense against a large class of attacks using only existing deployed web browsers.

We assume that a site implementing BeamAuth will not allow logging in with only one of the two authentication factors. When one of the authentication factors needs to be recovered or reset, it must happen via a secondary channel, e.g. SMS, voice, or email. We specifically recommend email and explore potential attacks accordingly.

We do not attempt to prevent denial of service attacks: we accept the possibility that a user may get frustrated and may believe she can no longer log in, as long as this frustration does not lead to a trivial compromise of both her authentication factors.

We also do not attempt to protect against attackers that convince Alice she has successfully logged in when she, in fact, has not, and then proceed to request further confidential information from her. Our focus is on preventing the attacker from successfully impersonating Alice to the BeamAuth-protected site.

5.2 Specific Attacks

Phishing with a Deceitful URL, Discouraging the Bookmark Click. In a classic, unsophisticated phishing attack where the URL is made to *look* similar to the purported destination with exactly the same appearance and instructions, Alice will click her bookmark and be whisked away to her true login site. Her credentials remain safe. If Alice forgets to click her bookmark, possibly because the phishing site omits that instruction or actively encourages her not to click, she may reveal her password to the malicious site. Because her bookmark token remains safe, this should not allow the attacker to log in on her behalf. However, it may be problematic if Alice uses this same password at another site that requires only a single authentication token.

Overriding Page Unload. Surprisingly, a site can include JavaScript that locks the user in and prevents her from leaving, even if she clicks on her bookmark:

```
window.onunload = function() {
    window.location =
     'http://evil.com/stay-here';
};
```

In this case, when Alice clicks her BeamAuth bookmark, she may think she is being taken to her login site, when in fact the malicious JavaScript interception has sent her to a spoofed post-bookmark-click page. Though she will not see her username automatically filled into the login form, she may ignore this inconsistency and fill it in herself, enter her password, and submit the form. In this case, as in the previous, Alice's password is compromised. However, again, her BeamAuth secret token, and thus her login, remain safe.

Importantly, the malicious JavaScript interception *cannot* access the URL to which the user intended to navigate: a call to `window.location` yields the *current* URL, not the new one. Thus, a malicious web site can prevent a user from navigating to another site, but it cannot determine to which site the user meant to navigate: the BeamAuth token remains safe.

Malicious Bookmark Replacement. An attacker might trick Alice into replacing her BeamAuth bookmark with a malicious one, using, for example, a spoofed email that mimics the BeamAuth setup email. No matter how intricate a procedure the attacker asks Alice to perform, one important point remains: without a significant browser bug, an attacker cannot access the content of Alice's bookmarks, and thus her BeamAuth token. However, this scenario can certainly become another mechanism for the attacker to steal Alice's password. Again, Alice's BeamAuth token, and thus her login, remain safe.

Spoofing the Browser Interface. An attack could spoof the browser interface [53] by opening up a new browser window, hiding the bookmarks bar, and displaying its own fake bookmarks bar. Though the attacker would be hard-pressed to know what bookmarks Alice is supposed to have, there is a chance it could fool Alice into clicking its fake bookmark rather than Alice's real BeamAuth bookmark. This case then reduces to the previous attack, where the attacker convinces Alice to replace her BeamAuth bookmark. Alice may compromise her password, though her BeamAuth token should remain safe.

Explicit Bookmark Theft. An attacker might use the above `onunload` hijacking attack to frustrate Alice because she can no longer log in. The attacker may then follow up and ask Alice to reveal the contents of her BeamAuth bookmark as part of a purported debugging process. BeamAuth attempts to mitigate this attack: the token is cleared from the address bar within milliseconds of the bookmark click, so that Alice would have to manually copy and paste the BeamAuth bookmark content and send it to the attacker. For high value sites, this attack cannot be discounted. It will be important to tell users to never, under any circumstance, send the content of their bookmark to anyone, even someone claiming to be a customer service agent. Even then, one cannot expect this to be foolproof: a determined attacker may convince Alice to manually send him her bookmark token, in which case BeamAuth has been defeated.

Attacking the Email Account. Given the high value of the BeamAuth token sent via email, an attacker might opt to attack Alice's credentials for accessing her email account. If the attacker succeeds, then the BeamAuth token is compromised, and the attacker might then succeed at obtaining Alice's password through normal phishing means. In other words, if an attacker can compromise both Alice's web and email channels, he can successfully defeat BeamAuth.

Using the Victim's Computer. An attacker might temporarily gain access to Alice's computer, e.g. while she is at lunch. This attack is worth considering specifically because the BeamAuth token is inside a bookmark, which is easily accessible to an attacker sitting in Alice's chair. Fortunately, the attacker will not be able to immediately log in as Alice, as he still needs her password. However, an attacker can steal Alice's full credentials if he gains access to Alice's computer and later phishes her for her password.

A usually benign variant of this situation is when a family shares a common computer and even a single account to this common computer: all bookmarks are shared between

users. In this case, each family member would have their own BeamAuth bookmark for each site. Each account remains protected by the family member's password, though it is certainly easier for one family member to steal another's credentials, since one authentication factor is immediately compromised.

Pharming Inattentive Users. An attacker who hijacks a DNS entry or spoofs an IP address can effectively intercept and respond to HTTP requests, including the BeamAuth bookmark click, destined for the legitimate login host. By sending down malicious code that reads the fragment identifier, an attacker can thus compromise the BeamAuth token. If the protected site does not use SSL, BeamAuth is completely vulnerable to this kind of attack. If the protected site uses SSL, BeamAuth remains vulnerable, except when users are attentive enough to take the browser's certificate warning seriously.

Malware and Client Compromise. An attacker who injects untrusted code into the user's client computer can completely control the system, read the browser's bookmark content, and keylog the user's password. BeamAuth is completely vulnerable to this kind of attack.

5.3 Comparison to Long-Lasting Cookies

Other two-factor web authentication techniques that do not require additional client-side code exist, e.g. BankOfAmerica's SiteKey [3] and Yahoo's Sign-In Seal [52]. In these schemes, the second-factor token is stored as a long-lasting cookie, sent only over SSL. A browser is initialized with this long-lasting cookie if the user successfully answers a number of verification questions. It is worth noting that a recent study [43] showed that users are still highly vulnerable to phishing with such site-image systems.

Second-Channel Setup. One important difference between such site-image systems and BeamAuth is the means by which the second factor is set up. In BeamAuth, we opt for a completely separate communication channel—email—so that a successful man-in-the-middle attack would require intercepting both a web and email connection. We note, of course, that site-image systems could adopt the same technique for their long-lasting cookie setup, and that such a defense could mitigate some of the concerns, as a phisher would not be immediately able to carry out a man-in-the-middle attack.

Cross-Site Scripting. A more fundamental difference is BeamAuth's use of a bookmark rather than a cookie to hold the token. A cookie is vulnerable to cross-site scripting attacks [9]: if an attacker finds a way to inject HTML and JavaScript into a target server's web space, a user who visits this page could have his cookie hijacked and sent to the attacker.

A BeamAuth bookmark, on the other hand, cannot be accessed unless it is explicitly clicked: it takes a deep browser bug or a pharming attack, not just a single-web-site bug, to reveal the contents of a bookmark to an attacker. Thus, in site-image systems, every web page at the protected host must be checked for potential XSS vulnerabilities, while in BeamAuth, only the login page needs to be verified. Of course, it's a good idea to check all web pages for cross-site scripting vulnerabilities regardless, but a BeamAuth-style bookmark token reduces this particular attack surface significantly.

Browser Initialization. It is common for a user to delete her cookies regularly, in particular because of the privacy implications of cookies. It is a lot less common for a user to delete one of her *bookmarks*, except if she explicitly means to remove it. Thus, we believe it is less likely that a user would lose her BeamAuth bookmark than her site-image cookie. This advantage is partially weakened by the recent use of Flash cookies [14], a technique which mirrors the cookie inside the embedded Flash application, which users clear far more rarely. However, as these Flash cookies still have significant privacy implications, it is only a matter of time before privacy-protection tools allow users to regularly clear these cookies, too.

Another interesting aspect of bookmarks is that, using one of the many bookmark synchronization tools mentioned earlier, it is relatively easy for a user to transmit her BeamAuth bookmark to all of her browsers. There are no known tools to accomplish this task for cookies. Overall, we expect the BeamAuth token to be more resilient to regular activity, multi-browser and multi-computer use.

5.4 Usability Testing

BeamAuth has undergone only very informal user testing on a handful of willing volunteers. Initial feedback was along the lines of what was expected: the signup procedure is a tad tedious, while the login process is relatively straightforward. Further usability testing is required, of course, in particular to better gauge the use across multiple computers and multiple sites.

5.5 Security in the Web Application Stack

With browsers installed on hundreds of millions of computers, and browser upgrades a fairly rare occurrence with typically conservative goals, it may become increasingly useful to think about implementing security in the web application stack, where the web *site* developer, rather than only the web *browser* developer, can innovate. It will be interesting to think about what small changes can be made to the browser platform to enable more innovation in the web application stack, so that the browser need not commit to one security solution, only to becoming a better platform for additional security [1].

5.6 Impact on Single Sign-On

Single sign-on is a growing use case which stands to benefit the most from our proposals. In particular, systems like OpenID, Yahoo BBauth, and university web-based login systems often expect relying parties to redirect users to the login site for authentication. Users are thus accustomed to arriving at their login site automatically, which makes the act of phishing somewhat more likely: an evil relying party can simply redirect a user to a fake login site.

BeamAuth stands to significantly reduce this specific phishing threat by introducing the "BeamAuth bookmark click" as an inescapable component of the user login ritual, thereby redirecting the user away from potential phishing attacks. BeamAuth is particularly appropriate because, like these single sign-on solutions, it works on existing vanilla web browsers without any add-ons or other additional client-side software.

6. CONCLUSION

Using only existing features of HTTP and modern web browsers, we have designed and implemented BeamAuth, a two-factor authentication technique to combat a number of types of phishing attacks. We believe our proposal makes the most common phishing attacks noticeably more difficult. Because our approach requires boomark-bar real estate and a browser setup procedure, it is best for high-value web sites, in particular single sign-on sites, where the user is inclined to make an additional small effort at registration time. BeamAuth exploits the URL Fragment Identifier and its unusual properties: it is never sent over the network, and changing it does not trigger a page reload.

Looking ahead, we suspect that the web platform is now flexible enough for some aspects of application-layer security to be implemented in JavaScript and HTML. In this model, new security features can be tested and deployed rapidly, on a per-web-application basis, without updating the client. It will be interesting to see if other existing browser features can be usurped to yield additional security properties.

7. ACKNOWLEDGMENTS

The author would like to thank David Wagner and Chris Karlof for extensive and insightful feedback on a draft of this paper, an anonymous CCS reviewer for pointing out that Alice might as well use the BeamAuth bookmark to reach her login site in the first place, Filipe Almeida and Ben Laurie for crucial feedback on an earlier (and broken) version of this scheme, and a number of folks who provided bleeding-edge user feedback: Rachna Dhamija, Simson Garfinkel, Susan Hohenberger, and Alon Rosen.

8. REFERENCES

[1] Ben Adida. The Browser as a Secure Platform for Loosely Coupled Private-Data Mashups. In *W2SP 2007, Proceedings of the First Workshop on Web 2.0 Security Privacy, Oakland, CA, USA*, May 2007.

[2] Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest. Lightweight Email Signatures (Extended Abstract). In *Fifth Conference on Security and Cryptography for Networks (SCN'06)*, volume 4116 of *Lecture Notes in Computer Science*, pages 288–302. Springer Verlag, 2006.

[3] Bank Of America. SiteKey. `http://www.bankofamerica.com/privacy/sitekey/`.

[4] Anti-Phishing Working Group. Digital Signatures to Fight Phishing Attacks. `http://www.antiphishing.org/smim-dig-sig.htm`.

[5] Apache Software Foundation. Apache HTTP Server Project. `http://httpd.apache.org`, last viewed on February 3rd 2007.

[6] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): General Syntax, January 2005. `http://www.ietf.org/rfc/rfc3986.txt`.

[7] Kim Cameron. As simple as possible – but no simpler. `http://www.identityblog.com/?p=649`, last visited on February 3rd 2007.

[8] Kim Cameron and Michael B. Jones. Design Rationale behind the Identity Metasystem Architecture, 2006. `http://www.identityblog.com/wp-content/resources/design_rationale.pdf`.

[9] CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests. `http://www.cert.org/advisories/CA-2000-02.html`.

[10] Remi Delon. CherryPy HTTP Framework. `http://cherrypy.org`, last viewed on February 3rd 2007.

[11] Rachna Dhamija, Doug Tygar, and Marti Hearst. Why Phishing Works. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. ACM Special Interest Group on Computer-Human Interaction, January 2006.

[12] Rachna 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 Press.

[13] Apple dotMac. `http://www.apple.com/dotmac/`, last viewed on 8 May 2007.

[14] Electronic Privacy Information Center. Local Shared Objects – "Flash Cookies". `http://www.epic.org/privacy/cookies/flash.html`, last viewed on August 12th, 2007.

[15] Simson Garfinkel. Fingerprinting Your Files. *MIT Technology Review*, August 2004. `http://www.technologyreview.com/read_article.aspx?id=13718&ch=infotech`.

[16] Simson L. Garfinkel. Email-Based Identification and Authentication: An Alternative to PKI? *IEEE Security & Privacy*, 1(6):20–26, November 2003.

[17] Jesse James Garrett. Ajax: A New Approach to Web Applications, February 2005. `http://www.adaptivepath.com/publications/essays/archives/000385.php`.

[18] Google. Google Browser Sync. `http://www.google.com/tools/firefox/browsersync/`.

[19] Anti-Phishing Working Group. Phishing Activity Trends, November 2006. `http://www.antiphishing.org/reports/apwg_report_november_2006.pdf`.

[20] T. Hansen, D. Crocker, and P. Hallam-Baker. DomainKeys Identified Mail (DKIM) Message Signing Service Overview, March 2007. `http://www.dkim.org/specs/draft-ietf-dkim-overview-04.html`.

[21] Harvard University. Harvard University PIN System. `http://pin.harvard.edu/`, last viewed on February 3rd 2007.

[22] Amir Herzberg and Ahmad Gbara. TrustBar: Protecting (even Naive) Web Users from Spoofing and Phishing Attacks. Cryptology ePrint Archive, Report 2004/155, 2004. `http://eprint.iacr.org/2004/155`.

[23] IETF. MTA Authorization Records in DNS (MARID), June 2004. `http://www.ietf.org/html.charters/OLD/marid-charter.html`.

[24] Collin Jackson and Helen Wang. Subspace: Secure Cross-Domain Communication for Web Mashups. In *Proceedings of the 16th international conference on World Wide Web (WWW 2007), Banff, Canada*, 2007.

[25] Markus Jakobsson and Steven Myers. *Phishing and Countermeasures: Understanding the Increasing Problem of Electronic Identity Theft.* Wiley-Interscience, 2006.

[26] Paul Johnston. A JavaScript implementation of the

Secure Hash Algorithm. http://pajhome.org.uk/crypt/md5.

[27] JotSpot. DojoDotBook. http://manual.dojotoolkit.org/WikiHome/DojoDotBook/Book0.

[28] Ari Juels, Markus Jakobsson, and Tom N. Jagatic. Cache cookies for browser authentication (extended abstract). In *S&P*, pages 301–305. IEEE Computer Society, 2006.

[29] Brian Krebs. Microsoft Releases Windows Malware Stats, June 2006. http://blog.washingtonpost.com/securityfix/2006/06/microsoft_releases_malware_sta.html.

[30] Ben Laurie. OpenID: Phishing Heaven. http://www.links.org/?p=187, last visited on February 3rd 2007.

[31] J. Levine and A. DeKok. Lightweight MTA Authentication Protocol (LMAP) Discussion and Comparison, February 2004. http://www.taugh.com/draft-irtf-asrg-lmap-discussion-01.txt.

[32] John R. Levine. A Flexible Method to Validate SMTP Senders in DNS, April 2004. http://www1.ietf.org/proceedings_new/04nov/IDs/draft-levine-fsv-01.txt.

[33] Justin Mason. Filtering Spam with SpamAssassin. In *HEANet Annual Conference*, 2002.

[34] Eric A. Meyer. S5: A Simple Standards-Based Slide Show System. http://meyerweb.com/eric/tools/s5/, last viewed on October 26th, 2006.

[35] T.A. Meyer and B. Whateley. SpamBayes: Effective open-source, Bayesian based, email classification system. In *Conference on Email and Anti-Spam 2004*, July 2004.

[36] Netcraft. Anti-Phishing Toolbar. http://news.netcraft.com/archives/2004/12/28/netcraft_antiphishing_toolbar_available_for_download.html.

[37] Gunter Ollmann. The Pharming Guide. http://www.ngssoftware.com/papers/ThePharmingGuide.pdf.

[38] V. Ramasubramanian and E. Sirer. Perils of transitive trust in the domain name system. In *Proceedings of the 2005 Internet Measurement Conference (IMC 2005), Berkeley, CA, USA*, 2005.

[39] D. Recordon and B. Fitzpatrick. OpenID Authentication 1.1, May 2006. http://openid.net/specs/openid-authentication-1_1.html.

[40] Ed Rice. Passwords in the Clear, 2006. http://www.w3.org/2001/tag/doc/passwordsInTheClear-52, last viewed on February 3rd 2007.

[41] Blake Ross, Collin Jackson, Nicholas Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions. In P. McDaniel, editor, *14th USENIX Security Symposium*, 2005.

[42] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, May 1998.

[43] Stuart Shechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The Emperor's New Security Indicators. In *S&P*. IEEE Computer Society, 2007.

[44] Stanford University. Stanford WebAuth. http://www.stanford.edu/services/webauth/, last viewed on February 3rd 2007.

[45] Sync2it. http://www.sync2it.com/, last viewed on 8 May 2007.

[46] Tumbleweed Communications. Digitally-Signed Emails to Protect Against Phishing Attacks. http://www.tumbleweed.com/solutions/finance/antiphishing.html.

[47] Guido van Rossum. The Python Programming Language. http://python.org, last viewed on October 26th, 2006.

[48] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

[49] Wikipedia. Usage share of Web Browser. http://en.wikipedia.org/wiki/Usage_share_of_web_browsers, last visited on February 3rd 2007.

[50] Min Wu, Simson L. Garfinkel, and Robert Miller. Secure Web Authentication with Cell Phones. http://groups.csail.mit.edu/uid/projects/cellphone-auth/.

[51] Yahoo. Browser-Based Authentication. http://developer.yahoo.com/auth/, last viewed on October 26th, 2006.

[52] Yahoo. What is a sign-in Seal? http://security.yahoo.com/article.html?aid=2006102507, last viewed on 8 May 2007.

[53] Zishuang (Eileen) Ye and Sean Smith. Trusted Paths for Browsers. In Dan Boneh, editor, *USENIX Security Symposium*, pages 263–279. USENIX, 2002.