Beware of Finer-Grained Origins

Collin Jackson
Stanford University
collinj@cs.stanford.edu

Adam Barth
Stanford University
abarth@cs.stanford.edu

Abstract

The security policy of browsers provides no isolation between documents from the same origin (scheme, host, and port), even if those documents have different security characteristics. We show how this lack of isolation leads to origin contamination vulnerabilities in a number of browser security features, such as cookies, encryption, and code signing. A tempting approach to fixing these vulnerabilities is to refine the browser's notion of origin, leveraging the browser's built-in isolation between security contexts. We demonstrate that attackers can circumvent these "finer-grained origins" using the library import and data export features of browsers. We discuss several approaches to preventing these attacks.

1 Introduction

Browsers grant scripting privileges to documents based on their *origin*. A document's origin is comprised of the scheme (protocol), host, and port of the URL from which the browser retrieved the document. One document can read and modify (i.e., control completely) another document if the two documents were retrieved from the same origin. For web URLs, the same-origin, and therefore the "can-script," relation is transitive and symmetric, partitioning documents into equivalence classes. These equivalence classes provide confidentiality and integrity for sensitive information by isolating different protection domains.

Origin Contamination. Many current and proposed browser security features ignore the same-origin equivalence classes and treat documents differently depending on some security characteristics of the document. For example, the browser restricts which cookies a document can read based on the document's path. A *sub-origin privilege* is a privilege granted to some, but not all, of the documents from an origin. Table 1 contains a number of browser security features that attempt to grant sub-origin privileges.

These sub-origin privileges interact poorly with the browser's scripting policy. If left unmodified, the browser's scripting policy lets a document obtain all the sub-origin privileges granted to any of the documents in its origin, a phenomenon we refer to as *origin contamination*. For example, a document can circumvent path-based restrictions on cookies because the scripting policy permits the document to inject script into a document with the appropriate path. One approach to avoiding this privilege escalation is to revise the browser's scripting policy to recognize *finer-grained origins* that isolate documents with different privileges, preventing documents from gaining additional privileges by injecting scripts into other documents.

Library Import and Data Export. Although finer-grained origins prevent origin contamination as such, an attacker can often circumvent the protection they provide by using browser features that explicitly import libraries or export data. For example, importing a library (such as a script, cascading style sheet, or a SWF movie) lets a document *endorse* the contents of the library and execute the library in the protection domain of the document. Submitting a form lets a document *declassify* some information and export the information to the URL specified by the "action" attribute of the form element.

When importing a library or exporting data, documents specify the library to import or the recipient of the data by *URL*. If a browser security feature depends on information not present in the URL, these import and export facilities lead to undesired behavior. For example, the locked same-origin policy [8] attempts to refine origins by segregating documents retrieved over HTTPS using different public keys, but this policy does not protect documents that import libraries, for example via the `script` element, because URLs fail to designate public keys.

Solutions. We describe three approaches to modifying the browser's security policy to grant additional privileges to documents safely.

1. **Embrace.** Grant the privilege to all documents in an origin or to none of the documents in an origin. This

		Circumvented by		
Feature	Sub-Origin Privilege	Contaminated Origin	Library Import	Data Export
Cookie Paths	Read Cookie	✓		
WSKE	Read Cookie	✓	✓	✓
Mixed Content	Show Lock	✓		N/A
Certificate Errors (IE7)	Show Lock	✓	✓	✓
Extended Validation	Show Organization	✓	✓	✓
Petname Toolbar	Show Petname	✓	✓	✓
Passpet	Obtain Password	✓	✓	✓
Signed JARs	Install Software	✓	✓	N/A
Locked SOP	Read Cookie		✓	✓
IP-based Origins	Network Requests		✓	✓

Table 1. Undesirable interactions between browser features

approach harmonizes the new security feature with the current browser security policy.

2. **Extend.** Extend URLs to contain the relevant security information and refine the browser's notion of an origin to distinguish URLs based on this information. This approach lets authors specify the relevant security information when importing libraries and exporting data.

3. **Destroy.** Prevent the browser from rendering documents that would not be granted the privilege. This approach prevents documents that lack the privilege from obtaining the privilege because the browser acts as if those documents did not exist.

Organization. The rest of this paper is organized as follows. In Section 2, we provide examples of origin contamination. In Section 3, we describe existing finer-grained origin proposals and show their library import and data export interactions. In Section 4, we show how these attacks can be prevented. Section 5 concludes.

2 Origin Contamination

There are numerous browser security features that grant privileges to a subset of documents in an origin. Many of these features fail to account for the browser's scripting policy and contain the same poor interaction: a document that lacks the privilege can escalate its privileges by injecting script into a document that has the privilege.

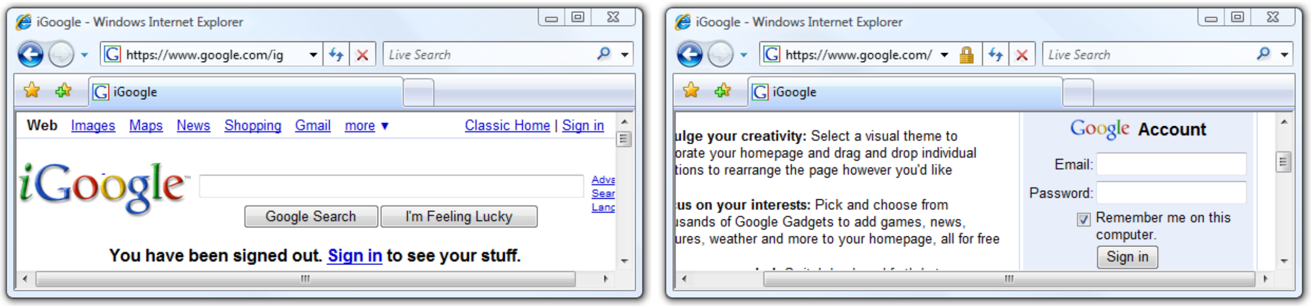
- **Cookie Paths.** One classic example of a sub-origin privilege is the ability to read cookies with "`path`" attributes. In order to read such a cookie, the path of the document's URL must extend the path of the cookie. However, the ability to read these cookies leaks to all documents in the origin because a same-origin document can inject script into a document with the appropriate path (even a 404 "not found" document) and read the cookies. This "vulnerability" has been known for a number of years [10]. This vulnerability was "fixed" by declaring the path attribute to be a convenience feature rather than a security feature.

- **Web Server Key-Enabled Cookies.** A modern example of the same phenomenon is Web Server Key-Enabled (WSKE) cookies [9]. A WSKE cookie can be read only by documents retrieved over an HTTPS session that uses the same public key as the session that stored the cookie. This sub-origin privilege, the ability to read a WSKE cookie for a specific key, has the same vulnerability as the `path` attribute. A document from the same origin (but retrieved using a different key) can inject script into a document with the appropriate key and read the cookie [8].

- **Mixed Content.** Browsers typically indicate whether an HTTPS document imports script libraries over HTTP. These scripts lack the protection afforded by HTTPS and can be replaced by an active network attacker. Browsers, however, fail to indicate that other documents in the same origin are contaminated by mixed content, as shown in Figure 1. Once an HTTPS document has imported a malicious script, the script can can inject malicious scripts into every reachable[1] document in the same origin, including those currently displaying a lock icon. Thus, a document without the privilege to show a lock icon can obtain that privilege due to origin contamination.

- **Certificate Errors.** In Internet Explorer 7, documents obtained from HTTPS connections with certificate errors are displayed with a red address bar and the text

[1] An attacker can reach a document either via a JavaScript pointer or by designating the document's window by name.


iGoogle - Windows Internet Explorer
https://www.google.com/ig
Live Search
iGoogle
Web Images Maps News Shopping Gmail more ▾
Classic Home | Sign in
Google Search
I'm Feeling Lucky
iGoogle - Windows Internet Explorer
https://www.google.com/
Live Search
iGoogle
Google Account
Email:
Password:
Remember me on this computer.
Sign in

You have been signed out. Sign in to see your stuff.

ulge your creativity: Select a visual theme to
orate your homepage and drag and drop individual
tions to rearrange the page however you'd like

us on your interests: Pick and choose from
usands of Google Gadgets to add games, news,
ures, weather and more to your homepage, all for free

Figure 1. A sub-origin privilege. Both documents have the same scheme, host, and port, yet only one of them is permitted to display a lock icon.

"certificate error" because such documents could have been supplied by an active network attacker. The browser does not degrade the security indicators for other documents in the same origin even though the potentially compromised document can inject script and display malicious content in documents without the certificate error warning. (Firefox 2 does not degrade a document's security indicators after the user clicks through a certificate error.)

- **Extended Validation.** The browser's scripting policy does not distinguish between HTTPS connections that use an Extended Validation (EV) certificates from those that use non-EV certificates. For example, PayPal serves `https://www.paypal.com/` using an EV certificate, but a principal who has a non-EV certificate for `www.paypal.com` can inject script into the PayPal login page without disrupting the browser's Extended Validation security indicators; see Figure 2.

- **Petname Toolbar.** The Petname Toolbar [3] allows a user to associate a "pet name" (a short, user-specified string) with an HTTPS certificate. If the browser retrieves a document over an HTTPS connection that uses that certificate, the browser displays the certificate's pet name, highlighted in green, as a security indicator. Displaying the user's pet name is a sub-origin privilege granted only to those documents with the appropriate certificate. If the attacker has contaminated the `https://bank.com/` origin (say by tricking the user into clicking through a certificate warning), the attacker can inject script into a document retrieved over a legitimate HTTPS connection and control the contents of a document decorated the user's pet name indicator. Passpet [11], a password manager based on pet names, also exhibits this behavior.

- **Signed JARs.** In Firefox, documents contained in signed archive files (JARs) are granted the privilege to prompt the user for additional privileges (such as the privilege to write to the user's hard drive and to inject script into any web site). The security user interface for this prompt contains the common name of the certificate that served the document; see Figure 3. This sub-origin privilege can be obtained by all documents in the JAR's origin because those documents can inject script into a signed document once it resides in the browser's memory. This vulnerability is particularly troubling because many well known brands, such as Yahoo!, Google, and eBay have published signed JAR files. A web attacker can mount this attack if the JAR contains at least one file (such as `LICENSE.txt`) into which to inject script.

3 Library Import and Data Export

The most natural way to repair the poor interactions in the preceding section is refine the browser's notion of a security origin to isolate documents that have the privilege from documents that lack the privilege. This approach is often ineffective because browsers provide a number of facilities for endorsing and declassifying information. The most commonly used endorsement facility is importing a library (used on virtually every web site), which lets a document import scripts, style sheets, SWF movies, Java applets, and other resources that contain script. The most commonly used declassification feature is form submission, which lets a document send information to a network endpoint addressable by a URL. Newer browser features, such as cross-site `XMLHttpRequest`, `XDomainRequest`, and `postMessage`, provide both endorsement and declassification facilities because they can be used as two-way communication channels.

Documents that import libraries and export data using relative URLs are especially problematic. The authors of these document typically intend to refer to URLs under their

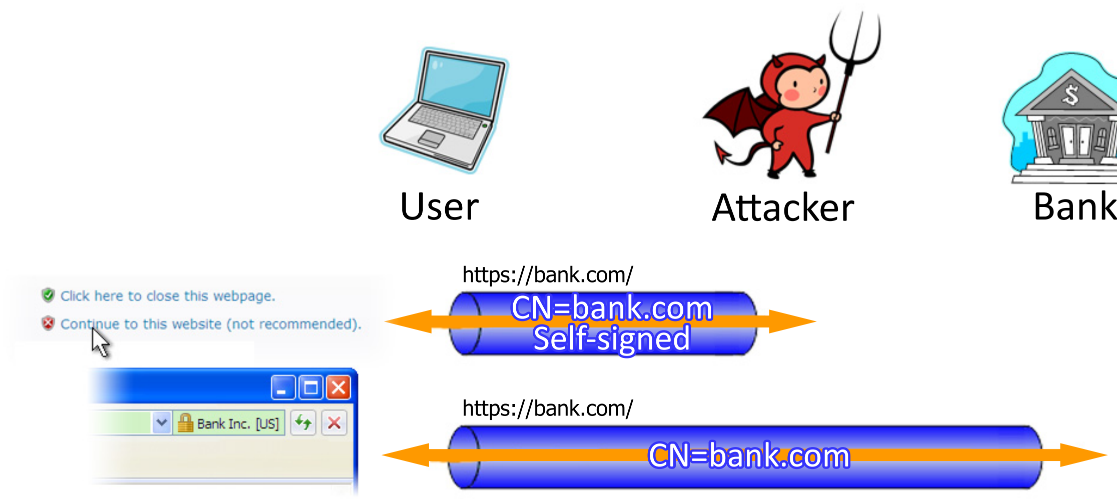


Figure 2. Extended Validation origin contamination. An active network attacker contaminates the bank's origin. When the user visits the real bank, and the attacker injects script into the bank's document without disturbing the Extended Validation indicator.

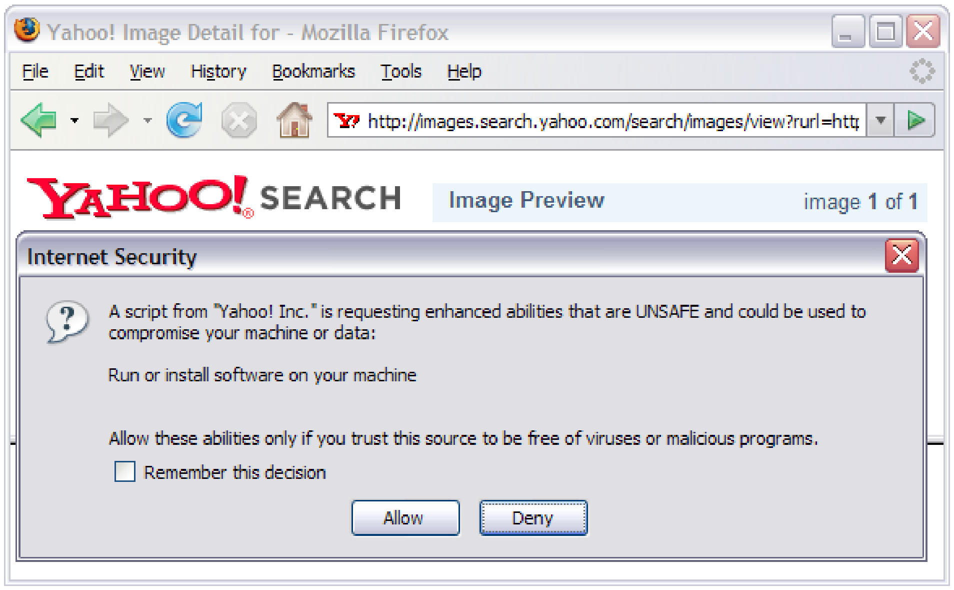
own control, but the browser resolves these URLs relative to the URL of the current document. If the attacker has altered the URL of the document, for example by naming the server containing the document `evil.com`, relative URLs are resolved to an absolute URLs under the control of the attacker.

- **Locked Same-Origin Policy.** The locked same-origin policy [8] augments WSKE [9] by refining origins for HTTPS documents to include the server's public key. The goal of this policy is to prevent a pharming attacker (who controls DNS and can trick the user into clicking through certificate warnings) from reading the `Secure` cookies from a legitimate HTTPS session. If the legitimate document imports a library, a pharming attacker can circumvent the locked same-origin policy and read the forbidden `Secure` cookies. Instead of interposing on the loading of the document, the pharmer waits for the document to import a library and then supplies a malicious response. The malicious library is executed with the document's privileges and can read the `Secure` cookies.

- **IP-based Origins.** The HTML 5 specification proposes that browsers refine origins to include IP addresses in order to prevent DNS rebinding attacks [7], preventing the attacker's script from accesssing content host at the target's IP address. In a DNS rebinding attack, the attacker's DNS server points `evil.com` to both the attacker's server and the victim's server. If a document hosted by the victim server uses a relative path to import a library, the browser will resolve the path relative to `evil.com`. When the browser retrieves the library, the attacker can rebind `evil.com` to the attacker's server, reply with a malicious library, and run script in the origin for the victim's IP address, circumventing the IP-based restrictions.

- **Passpet.** Passpet [11] is an extension of the Petname Toolbar that generates passwords based pet names. When the user clicks on his or her "pet," Passpet injects the generated password into the site's password field. The goal of this feature is to prevent phishing attacks by training users to click their pet rather than entering passwords into web pages. If the user is willing to accept a common name mismatch (and the bank exports data to a relative URL), a phishing attacker can steal the user's bank password (see Figure 4):

  1. When the user visits `https://evil.com/`, the attacker forwards each packet of the HTTPS session to `bank.com`.
  2. If the user clicks through the HTTPS warning dialog box, the browser establishes a TLS session with the real bank, who responds with a login form that submits to a relative path, `/login`. The Passpet indicator considers only the certificate and shows the bank's pet name, even though the location bar reads `https://evil.com`.
  3. The user clicks his or her pet to log in. Based on the certificate, Passpet injects the user's bank password into the password field.


Yahoo! Image Detail for - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://images.search.yahoo.com/search/images/view?rurl=htt
YAHOO! SEARCH
Image Preview
image 1 of 1
Internet Security

A script from "Yahoo! Inc." is requesting enhanced abilities that are UNSAFE and could be used to compromise your machine or data:

Run or install software on your machine

Allow these abilities only if you trust this source to be free of viruses or malicious programs.

☐ Remember this decision

Figure 3. Signed JAR origin contamination. This Yahoo! Image Search document contains a frame to evil.com, which is hosting a JAR signed by Yahoo!. Clicking "Allow" lets the attacker install software on the user's machine. If the user had previously granted Yahoo! this privilege, and checked "Remember my decision," the browser would let the attacker install software without prompting.

4. When the user submits the login form, the attacker ceases to forward the session and establishes a new TLS session directly with the user, presenting a valid certificate for `evil.com`.
5. The browser accepts this TLS session as valid and sends the password to the attacker, who can now log in as the user at the real bank site.

An attacker can also use this technique to steal passwords from users of the Petname Toolbar if the user enters their password based on the displayed pet name, as instructed by the documentation [3].

4 Solutions

Designers of new browser features must consider both the implicit and explicit trust relations between documents. We propose three approaches to improving the browser's security policy that interact securely with the browser's scripting policy and the browser's architecture for importing libraries and exporting data.

Embrace. The simplest approach to designing new privileges is to embrace the current scripting policy and explicitly grant the privilege either to all, or to none, of the documents in an origin. This is the most common approach and is used by the vast majority of new browser features.

- **Frame Navigation.** The frame navigation policy included in Internet Explorer 6, Firefox 2, and Opera 9 grants navigation privileges to documents based on their location in the frame hierarchy. The modern frame navigation policy, adopted in Internet Explorer 7, Firefox 3, and Safari 3 [1], explicitly propagates navigation privileges based on origin. This policy is more convenient for web developers because it is less restrictive, but is no less secure because an attacker could always have injected scripts into the appropriate documents to achieve the same results.

- **Phishing Filters.** The phishing filters included in Internet Explorer and Firefox consider an origin to contain either entirely phishing or entirely non-phishing documents. Had the designers of the phishing filter attempted to classify individual documents, a phishing page could have suppressed the phishing warning by injecting script into a document classified as non-phishing. The current design avoids this pitfall by embracing origin contamination.

Extend. Finer-grained origins can be made secure by extending URLs to contain enough information to designate a fine-grained origin fully. This prevents library import and data export vulnerabilities because a document can fully specify its trust relationships using URLs.

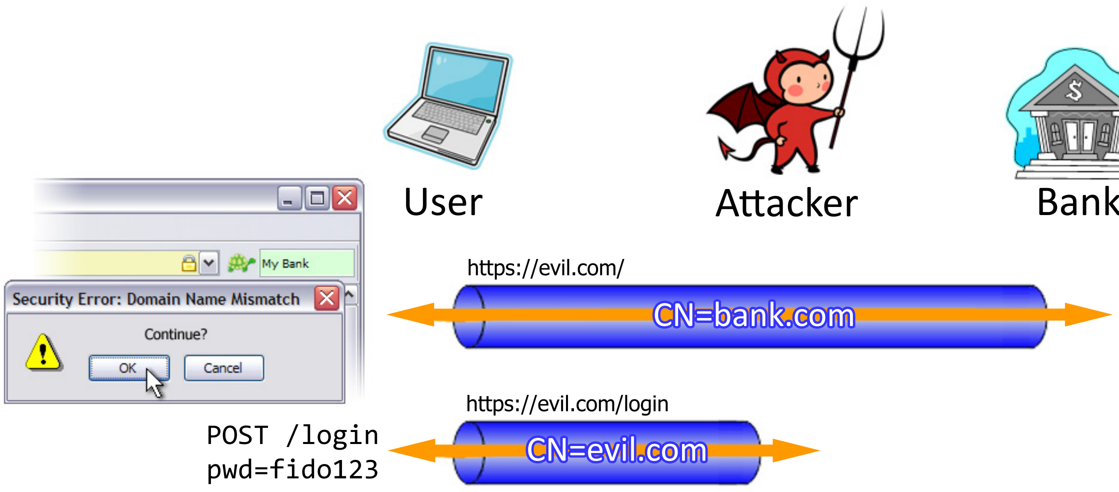


Figure 4. Passpet data export. The phishing web site forwards traffic between the user and the real bank, causing a common name mismatch warning. Because the bank's certificate was used, Passpet injects the user's bank password into the login form. When the login form is submitted to a relative path, the browser establishes a valid TLS session with the attacker and sends the password.

- **HTTPEV.** The Extended Validation behavior described in Section 2 could be addressed by creating an HTTPEV scheme that is similar to the HTTPS scheme, except that only Extended Validation certificates are permitted. A HTTPS document with a domain-validated certificate would be unable to script an HTTPEV document because their schemes would not match. When a document includes a library, the document's author can use HTTPEV to require an EV certificate, preventing domain-validated content from being displayed with EV security indicators.

- **YURL.** A YURL [2] is a URL that includes a public key before its host name. YURLs are retrieved using a TLS session whose public key must match the key embedded in the URL. Connections that do not use the specified public key must be blocked without giving the user an opportunity to proceed insecurely. A browser that supports YURLs would not consider two documents to be in the same origin if their public keys did not match. YURLs are not vulnerable to library import or data export attacks because the URLs used for import or export specify the public key for the intended source of the library or recipient of the data [8]. In particular, the public key remains unchanged when a relative URL is resolved to an absolute URL.

Destroy. Another approach to securing finer-grained origins is to destroy contaminated origins. The browser can eliminate an origin by refusing to display or execute any of its content. In some cases, the browser can revoke the privilege from the entire origin if the browser displays a document that would not be granted the privilege.

- **ForceHTTPS.** When an origin enables ForceHTTPS [5], the browser refuses to ignore certificate errors for that origin and refuses to import non-HTTPS libraries into that origin's documents. ForceHTTPS protects an origin from being contaminated by insecure content by preventing that content from entering the browser.

- **SafeLock.** When a document imports a non-HTTPS library, the browser should revoke the privilege to display a lock icon from all documents in the same origin as the contaminated document, preventing the non-HTTPS content from displaying a lock icon. We have implemented this behavior as a 286-line extension to Firefox, which can be downloaded from our web site [6].

- **ForceCertificate.** We collaborated with Mozilla to deploy a finer-grained origin in Firefox 2.0.0.15 that prevent unsigned documents from tampering with signed JARs [4]. The goal of this security policy is to ensure that a document signed with a particular public key certificate can be modified only both other documents signed with the same certificate. To prevent library import attacks, the browser blocks signed JARs from importing scripts that are unsigned or signed by another certificate.

5 Conclusion

Many current and proposed browser features extend the browser's security policy by granting additional privileges to documents. If designed incorrectly, these features interact poorly with the browser's existing security policy, resulting in the unexpected privilege escalation. Most commonly, a privilege is granted to some, but not all, of the documents in a origin. The documents without this sub-origin privilege can often escalate their privileges by injecting script into documents in their origin that do possess the privilege. We describe a number of browser features that are affected by such origin contamination.

The most natural response to origin contamination is to refine the browser's scripting policy to prohibit documents that lack the privilege from scripting documents that have the privilege. The design of these finer-grained origins is complicated by the built-in browser facilities for explicit endorsement and declassification. These features can be exploited by an attacker to bypass the stricter scripting policy.

We suggest three approaches for addressing the limitations of finer-grained origins. In the embrace approach, the privilege is granted explicitly to entire origins. In the extend approach, URLs are extended with additional security information that is used to refine the browser's notion of origin and restrict which documents have the privilege. In the destroy approach, the browser refuses to interact with resources that lack the desired security property, preventing them from escalating their privileges.

Acknowledgments

We thank Dan Boneh, Tyler Close, Chris Karloff, John Mitchell, Ka-Ping Yee, and Boris Zbarsky for their helpful suggestions and feedback. This work is supported by grants from the National Science Foundation and the US Department of Homeland Security.

References

[1] Adam Barth, Collin Jackson, and John C. Mitchell. Securing frame communication in browsers. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[2] Tyler Close. Decentralized identification. `http://www.waterken.com/dev/YURL/`.

[3] Tyler Close. Petname tool. `http://www.waterken.com/user/PetnameTool/`.

[4] Mozilla Foundation. Signed JAR tampering, July 2008. `http://www.mozilla.org/security/announce/2008/mfsa2008-23.html`.

[5] Collin Jackson and Adam Barth. ForceHTTPS: Protecting high-security web sites from network attacks. In *Proceedings of the 17th International World Wide Web Conference*, 2008.

[6] Collin Jackson and Adam Barth. SafeLock: Preventing origin contamination by mixed content, 2008. `http://crypto.stanford.edu/websec/safelock`.

[7] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting browsers from DNS rebinding attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, November 2007.

[8] Chris Karlof, Umesh Shankar, J. D. Tygar, and David Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, November 2007.

[9] Chris Masone, Kwang-Hyun Baek, and Sean Smith. Wske: Web server key enabled cookies. In *Proceedings of Usable Security 2007 (USEC '07).*

[10] Martin O'Neal. Cookie path best practice. `http://research.corsaire.com/whitepapers/040323-cookie-path-best-practi%ce.pdf`.

[11] Ka-Ping Yee and Kragen Sitaker. Passpet: Convenient password management and phishing protection. In *Proceedings of the 2006 Symposium on Usable Privacy and Security (SOUPS).*