Locked cookies: Web authentication security against phishing, pharming, and active attacks

Chris K. Karlof
Umesh Shankar
Doug Tygar
David Wagner

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2007-25
http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-25.html

February 7, 2007

Acknowledgement

This work was supported in part by National Science Foundation award number CCF-0424422 (Trust).

Locked cookies: Web authentication security against phishing, pharming, and active attacks

Chris Karlof
ckarlof@cs.berkeley.edu
UC Berkeley

Umesh Shankar
ushankar@google.com
Google

Doug Tygar
tygar@cs.berkeley.edu
UC Berkeley

David Wagner
daw@cs.berkeley.edu
UC Berkeley

Abstract

This paper proposes new methods for web authentication that are secure against phishing and pharming attacks. We explore the use of browser cookies as authenticators that cannot inadvertently be given away by users, and introduce *locked cookies*, which are cookies that are bound to the originating server's public key. Locked cookies defeat phishing, pharming, and network-controlling active attacks, since the user's browser can verify that the attacker's public key is different from that of the server that set the cookie in the first place, even though the domain names may be the same. Locked cookies are transparent to the user and do not require any server-side changes. We evaluate and compare authentication schemes based on conventional cookies, IP cookies, and locked cookies.

1 Introduction

Phishing is a social engineering attack in which a criminal lures an unsuspecting Internet user to a web site posing as a trustworthy business with which the user has a relationship [2]. The broad goal is identity theft; phishers try to fool web visitors into revealing their login credentials, sensitive personal information, or credit card numbers with the intent of impersonating their victims for financial gain. Phishers commonly lure victims by sending an email containing a warning about a "problem" which requires immediate attention, along with a link the user can click to take action. If a user clicks on the link, however, she will in fact reach the phishing site. Typically the user is prompted to enter some personal information, such as a login name, password, or social security number, before the "problem" with her account can be addressed.

In a more advanced phishing attack known as *pharming* [40], the adversary subverts the domain-name lookup system (DNS), which is used to resolve domain names to IP addresses. In the attack, the DNS infrastructure is compromised so that DNS queries for the victim site's domain (say, `google.com`) return an attacker-controlled IP address. This can be accomplished via several techniques, including DNS cache poisoning and DNS response forgery. Pharming attacks are particularly devious because the browser's URL bar will display the domain name of the legitimate site, potentially fooling even the most meticulous users.

Recent research has exposed complex and subtle dependencies between names and name servers [46], suggesting the DNS infrastructure is more vulnerable to DNS poisoning attacks than previously thought. In March 2005, the security organization SANS issued a warning about a DNS poisoning attack which

affected at least 1300 domain names, 8 million HTTP GET requests, and 75,000 email messages [48, 54]. More focused DNS poisoning attacks have also affected Hushmail [24], panix.com [42], and Ebay [11].

The ubiquity of public wireless access points and wireless home routers introduce new pharming threats. Users are becoming more and more accustomed to accessing wireless routers in airports, restaurants, conferences, libraries, and other public spaces. Adversaries can set up malicious wireless routers in these areas which offer free Internet access but redirect users to spoofed web sites. Also, many users leave the default settings on their home wireless home routers unchanged. Since the default settings on home routers are optimized for ease of use rather than security, these settings often disable encryption and access control, permit wireless access to router administration, and use published, weak passwords to guard access to administrative functions. These vulnerabilities enable *warkitting* attacks [56, 57], a combination of wardriving and rootkitting, where an adversary maliciously alters wireless a router's configuration over a wireless connection. In the worst case, the adversary might be able to completely overwrite the router's firmware. Warkitting attacks enable *active attacks*, giving an adversary complete control of users' network connections.

Phishing attacks have become a serious threat. In 2006, Gartner Research estimates 109 million Americans received phishing emails, and approximately 24.4 million Americans clicked on a phishing email [17]. Financial losses stemming from phishing attacks reached $2.8 billion. Phishing sites have become more elusive and harder to shut down. Over the last few years, the average lifetime of phishing sites has shrunk from a week to only a few hours. Some phishing sites use distributed, fault-tolerant architectures based on botnets which serve phishing content from compromised machines to evade authorities [45]. We anticipate these trends will continue.

Most web sites currently authenticate users with a simple password submitted via an HTML form. Unfortunately, information submitted via HTML forms has proven to be highly susceptible to phishing. Evidence suggests somewhere between 3–5% of phishing targets disclose sensitive information and passwords to spoofed web sites [18, 33], and phishing attacks are becoming more and more sophisticated [45]. These problems are not easy to fix: good phishing techniques can fool even the most vigilant users [8, 26]; trusted paths in browsers for entering sensitive information are susceptible to spoofing [39, 62]; and multi-factor authentication schemes are also vulnerable [50].

The problem with passwords is that it is too easy for users to reveal their passwords to parties who should not receive them. Phishing works by tricking users into voluntarily sharing their passwords in dangerous ways. Our insight is that if it is impossible for the user to give away her authenticator (even if she wanted to), then attackers will no longer be able to use social engineering to steal the user's authentication credentials. In particular, computers are not fooled by social engineering attacks, so they are perfect candidates to store and release authenticators on behalf of their users. We refer to authentication credentials that users can not easily give away as *disclosure resistant*. Other researchers have also noted the benefits of disclosure resistant authentication credentials [43].

In this paper, we tackle the problem of designing a web authentication scheme that resists phishing and pharming attacks. In one standard classification of authentication schemes, passwords are "something you know"; but the problem with using "something you know" for authentication is that anything the user knows, she can—and in a nontrivial fraction of cases, will—reveal to a phisher. We instead suggest relying on "something your computer knows." This is roughly equivalent to "something you have," except that we do it in software, without requiring the physical hardware tokens normally used to fill that role.

HTTP cookies are a natural candidate to be used as web authentication credentials (§ 4.1). In this regime, a site should require a particular cookie be sent from the client for authentication; additional authentication methods, such as passwords, may also be implemented. As we will show later, since the browser cannot be "tricked" into giving a useful cookie to an attacker, social engineering is eliminated.

We will also show how web sites can authenticate users with cookies without requiring major changes to the infrastructure of the web. We first analyze cookie authentication schemes compatible with legacy browsers, using *SSL-only cookies* (§ 4.1.1) and *IP cookies* (§ 4.1.2), which provide security against phishing and pharming, respectively.

Although IP cookies resist pharming attacks, they can be cumbersome and tricky for web sites to deploy. If an anti-pharming solution is too expensive, complicated, or restricts scalability, web sites will choose efficiency, simplicity, and functionality over security. To address the drawbacks of IP cookies, Juels et al. propose *active cookies* (§ 4.1.3), an extension of IP cookies designed to resist pharming attacks. However, we show active cookies are vulnerable to *dynamic pharming*, a new pharming attack against web authentication which enables pharmers to hijack users' web sessions. We also show authentication via client-side SSL, currently considered the most secure option for web authentication by many researchers and application developers, is vulnerable to dynamic pharming as well.

1.1 The locked same-origin policy

The same-origin policy in web browsers governs access control among different web objects and prohibits a web object from one origin from accessing web objects from a different origin. Browsers currently enforce the same-origin policy using the domain name from which the object originated. However, enforcing same-origin based on domain name is problematic in the presence of pharming attacks because pharmers can influence the mapping from domain name to subject.

To resist pharming attacks, we propose the *locked same-origin policy*, a new same-origin policy for browsers based on cryptography (§ 5). The locked same-origin policy enforces access control for SSL web objects based on servers' public keys, which cannot be spoofed by attackers. The locked same-origin policy permits enforcement of a more robust and consistent same-origin policy: name-based enforcement allows pharmers to access objects that they did not originate. Instead of comparing domain names to enforce access control, the browser compares the public key stored with the web object to the public key sent with a new connection; access in granted only if they match. It is this key-matching requirement that gave rise to the term "locked."

Applying the locked same-origin policy to SSL-only cookies yields *locked cookies* (§ 5.1), an extension to SSL-only cookies which binds them to the public key of the originating server. Locked cookies are an attractive solution for pharming resistant authentication. Locked cookies require no changes to the HTTP cookie specification, SSL, or web servers; web sites set and receive cookies in the same manner as before. In addition to phishing and pharming attacks, locked cookies resist active attacks, where adversaries have complete control of the network, including the routing infrastructure. Locked cookies also protect legacy applications of cookies for authentication. Finally, for web sites needing to authenticate clients to multiple servers within the same domain, we propose locked domain cookies, which do the same thing for domain cookies that locked cookies do for host cookies (§ 5.2). We summarize our results in Table 2.

We emphasize that in this paper, we are concerned only with attacks that target web authentication credentials. We do not consider the problem of protecting other kinds of credentials; for instance, our methods do not help to protect users' social security numbers, credit card numbers, birth dates, bank account numbers, or other personal information from phishing.

Auth cookie type	Scope	New?	How verified
SSL-only host cookies	Host	No	Match server's domain name against host field of cookie
SSL-only IP cookies	IP address	No	Match server's IP address against host field of cookie
SSL-only domain cookies	Domain suffix	No	Match server's domain name against wildcard in `domain` field of cookie
Locked cookies	Host	Yes	Match server's public key against public key stored in locked cookie
Locked domain cookies	Domain suffix	Yes	Match server's domain name against wildcard certificate sent in server's cert chain

Table 1: **Types of authentication cookies.** *Scope*: A scope of "Host" does not mean a particular physical machine, but rather a fully-qualified domain name, such as `mail.google.com`. A scope of "Domain suffix" means a set of hosts characterized by a common suffix, for example `*.google.com`.

Server authentication mechanism	Strongest threat model protected against	
	With legacy browsers	With locked same-origin policy
Passwords	(no protection)	(no protection)
SSL-only cookies with domain names	phishing	active attacks
IP cookies	pharming	active attacks
Client-side SSL	phishing	active attacks

Table 2: **Security of cookie authentication mechanisms using legacy browsers and locked cookie browsers.** Each cell reports the strongest threat model resisted by each combination of authentication mechanism and browser type. We consider phishing, pharming, and active attacks (Section 2.1).

2 Threat models and goals

2.1 Threat models

We consider three broad classes of threats, classified according to the capabilities of the attacker.

Phishing attacks. To implement phishing attacks, no capabilities are needed beyond control of a single internet node. In this scenario, an attacker can:

- Have complete control over some web server with a public IP address
- Send communications such as emails and instant messages to potential victims
- Effect application-layer man-in-the-middle attacks, representing a legitimate server to the victim and proxying input from the victim to the real server as needed.

Pharming attacks. An attacker with pharming capability has all the abilities of a phisher, plus

- The ability to change DNS records for the target site, such that the victim will resolve the target site's name to the attacker's IP address.

In practice, such an attack might work through DNS poisoning, spoofed DNS responses, or by social engineering attacks against a domain name registry. We assume pharmers do not have the same IP address as the victim and cannot receive packets destined to the victim's IP address.

Active attacks. An active attack is the most powerful threat we consider here. In addition to pharming capability, an active attacker can:

- Control the internet routing infrastructure and can re-route traffic destined to particular IP addresses
- Eavesdrop on all traffic
- Mount active, network-layer, man-in-the-middle attacks

To date, phishing has been by far the most prevalent class of attack; however, looking to the future, it seems prudent to defend against more powerful attackers as well, to the extent possible.

Significantly, we assume that an active attacker does *not* have access to the target site's server machines or any secrets, such as private keys, contained thereon.

We also do not address cross-side scripting (XSS) attacks, and we assume that the target web site is free of XSS vulnerabilities. Other research efforts address XSS vulnerabilities [23, 32, 34, 63, 64].

Initialization of authentication credentials. We separate the web authentication problem into two distinct subproblems: the initialization of users' authentication cookies and the use of those cookies to authenticate users to web sites. Our primary focus is on the latter, but there are several options available for initialization. We discuss the initialization problem further in Section 8.

2.2 Goals

There are three metrics by which we measure our authentication mechanisms: security, usability, and deployability.

2.3 Security

Our primary security goal is to authenticate a user to a web site and create an authenticated, trustworthy channel between the user and web site, via a web browser. Section 2.1 is the basis for our security evaluation, and we classify each mechanism according to the strongest threat it can resist: phishing, pharming, or active attacks, in order from weakest to strongest.

2.4 Usability

Users' psychological acceptance of an authentication mechanism is vital to its success [47]. Psychological acceptance means that a mechanism's behavior must closely match users' expectations. To meet users' expectations, we must understand users' *mental models*. Essentially, a user's mental model is her understanding of a mechanism's goals, interface, assumptions, risks, guarantees, and operation.

Unfortunately, studies have shown many users have an incomplete or inaccurate mental model of current web security mechanisms. Users' interpretations of "secure" web connections vary significantly, and many users have trouble accurately interpreting browser security indicators and cues, such as the URL bar, the "lock" icon, certificate dialogs, and the myriad of security warnings [8, 14, 15, 61]. In addition, users' awareness of risks on the web are only loosely correlated to the spectrum of vulnerabilities and necessary countermeasures, and studies show users can manage risks they are familiar with, but have trouble extrapolating to unfamiliar risks [9].

This evidence suggests web authentication must be robust to inaccuracies in users' mental models; otherwise, adversaries could leverage users' misconceptions, misunderstandings, and mistakes in an attack. We identify four design principles to minimize the effects of imperfect mental models:

Authentication should be nearly invisible to users. Since security is generally not users' primary goal, a security mechanism should not make unnecessary demands which interfere with their ability to complete tasks or interrupt their workflow [49]. If given the choice between complying with security requirements or working towards their goals, users will circumvent or disable the mechanism if it is annoying or too hard to use [19, 35, 66]. This problem is exacerbated if users do not understand the necessity of these security mechanisms.

Users should be able to understand how to use web authentication securely. When users do not understand what they need to do to be secure, they often make poor decisions [7]. For instance, users will take the "path of least resistance" and ignore confusing security alerts they do not comprehend [8]. Users should not have to understand the technical details of how an authentication scheme works to use it securely.

It should be difficult or impossible for users to take actions that defeat security. Security protections should be mandatory, and not up to the user's discretion; there should be no way for the user to bypass or disengage the security mechanisms. If there is a way for users to switch the system to an insecure mode, then phishers will try to trick users into flipping this switch.

Web authentication should use disclosure-resistant credentials. One usability problem with passwords is that users must decide whether or not it is safe to disclose her password to a web site when prompted to do so. Unfortunately, this imposes an impossible burden; users must thoroughly analyze and understand the contents of the URL bar and site's X.509 certificate, and interpret any browser messages and errors. Phishing exploits this flaw. There are limits to what we can do about this [1]: we cannot prevent users from visiting phishing sites (because of the problem of false positives), and warning messages do not seem to be effective [8, 61]. Therefore, we need authentication schemes that remain secure even if users should happen to visit phishing sites. Since we cannot fully anticipate the tricks phishers may use to solicit users' passwords and how users will behave in these situations, we conclude that authentication credentials should be *disclosure-resistant*. In other words, since it seems likely that any authentication credential the user can readily give away will be vulnerable to phishing, we propose that credentials should be designed so that users cannot easily disclose them.

2.5 Deployability

If a new solution is to be successful, it should be easy to deploy and backwards-compatible: New authentication mechanisms should not "break the web" because of problems in deployment or interoperability. Naturally, any security benefits of the new scheme may not be gained when interoperating with existing systems.

Another goal is to preserve web sites' ability to control the user experience. We prefer, when possible, not to impose restrictions on how users must interact with the site at login or thereafter. We hope this will increase the chances that our schemes will be adopted. For the same reasons, we also prefer schemes that have few configuration parameters, especially those that can jeopardize compatibility with other implementations.

3 Background

3.1 HTTP cookies.

HTTP cookies are a general mechanism for web servers to store and retrieve persistent state on web clients [44]. Since HTTP is a stateless protocol, cookies enable web applications to store persistent state over multiple HTTP requests. For example, web shopping applications can use cookies to track which items a user adds to her shopping cart.

When a client makes an HTTP request to a server, the server has the option of including one or more `Set-Cookie` headers in its response. The `Set-Cookie` header requests clients to store a cookie—a simple name-value pair of strings—in the client's cookie jar and to return it on subsequent visits to this web site by providing it in the `Cookie` header associated with the subsequent HTTP requests.

Additional attributes may be specified. The `expires` attribute indicates when the cookie should be deleted by the browser. If the `expires` attribute is omitted, then the cookie is called a *session cookie* and should be deleted when user closes the web browser. Cookies with an `expires` attribute are called *persistent cookies*.

The `domain` and `path` attributes are used to qualify the set of HTTP requests for which clients should send back cookies. The client searches its cookie jar for cookies whose domains which suffix-match the domain of the request and whose paths prefix-match the path of the request. For example, if the user requests the URL `http://online.foobar.com/store/index.html`, then a cookie with `domain=.foobar.com` and `path=/store` would be included with this request, but a cookie with `domain=pics.foobar.com` would not. The default values of the `domain` and `path` attributes are the host name of the server which generated the cookie response and the full path of the document described by the HTTP header, respectively. We refer to a cookie with an explicit `domain` attribute as a *domain cookie* and a cookie which omits it as a *host cookie*.

Web browsers use the `domain` and `path` attributes to enforce a *same-origin policy* for cookies. The same-origin policy in web browsers prohibits a web object from one origin from accessing web objects from a different origin. In particular, a browser will only append a cookie to an HTTP request if the domain attribute of the cookie tail-matches the domain of the request. Cookies are also accessible through the Javascript property `document.cookie`. For this mode of access, web browsers use the URL of the document executing the Javascript to determine the appropriate cookies.

The final optional cookie attribute, `secure`, indicates that the cookie should only be sent over SSL connections. We refer to a cookie including the `secure` attribute as an *SSL-only cookie*. We now give some background on SSL.

3.2 Secure Sockets Layer (SSL) and X.509 certificates.

The Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are cryptographic protocols for establishing end-to-end secure channels for HTTP and other Internet traffic [13, 55]. HTTP over SSL is also known as HTTPS.

SSL uses X.509 certificates [22] to identify the server participating in the SSL connection. An X.509 certificate contains the server's public key, the domain name of the web site (specified in the `CN` subfield of the certificate), the public key of the issuer of the certificate, the time period for which the certificate is valid, and the issuer's signature over these fields. The private key corresponding to a X.509 certificate can be used to sign another certificate, and so on, creating a chain of trust.

The root of this trust chain is a certificate authority (CA). Web browsers ship with the certificates of

some number of CAs which are deemed to be trusted; these are called *root certificates*. Most commonly, a web site's certificate is signed directly ("issued") by a CA, and this length-two chain is sent at the start of an SSL connection. For example, if the CA Verisign issues a certificate to `www.foobar.com`, the resulting chain is of length two: Verisign's root certificate is sent, followed by the Verisign-signed certificate for `www.foobar.com`.

Longer certificate chains can be constructed if a CA issues a *signing certificate* to an entity. Signing certificates can sign other certificates. For example, if `foobar.com` has several domains for which it needs certificates (e.g., `www.foobar.com`, `accounts.foobar.com`, `mail.foobar.com`, etc.), it could request a signing certificate from a CA and then issue its own certificates for each its subdomains using its public key in the signing certificate. In this case, the issuer for `foobar.com`'s signing certificate would be the CA, but the issuer for each of the `foobar.com`'s subdomain certificates would be `foobar.com`'s signing certificate.

When the client's web browser makes a connection to an SSL enabled web server over HTTPS, the browser must verify the server's certificate is valid. This involves numerous checks, but at the high level the browser must:

1. Verify that the first certificate in the chain is from a trusted CA.
2. Verify that (a) every certificate in the chain has a valid signature from its predecessor, using the public key of the predecessor and (b) that no certificate has expired.
3. Verify that the `CN` field of the last certificate in the chain matches the domain name of the web site the browser intended to visit.

If any of these checks fail, the browser warns the user and asks the user if it is safe to continue. If the user chooses, the user may permit the SSL connection to continue even though any or all of these checks have failed.

Note that browsers treat failure of these checks as a "soft" error and prompt the user, rather than preemptively blocking access to the affected web site. The reason is to ensure compatibility with misconfigured certificates and SSL servers; a periodic survey by Security Space shows that approximately 60% of SSL certificates have such problems [52]. Also, this behavior by browsers allows web sites to use self-signed certificates if they choose, instead of paying a CA for a certificate.

Unfortunately, asking users whether to continue anyway in such cases is a serious security vulnerability. Researchers have shown that users routinely ignore such security warnings and just click "OK" [5, 8, 61].

4 Phishing and pharming resistant authentication for legacy browsers

In this section, we examine what web site administrators can do to protect users' authentication credentials from phishing and pharming, given the constraints imposed by existing browsers. We discuss two general approaches: *authentication cookies* and *client-side SSL certificates*.

4.1 Using HTTP cookies for authentication

HTTP cookies are an excellent candidate for web authentication credentials, since they meet many of the goals of § 2.2 well. HTTP cookies have several advantages:

- Excellent usability: they are *easy to use* for authentication and *disclosure-resistant*. Browsers automatically determine the appropriate cookies for web requests and require little to no user involvement to make security decisions, reducing the chance of human error. Also, because it is the browser and not the user who makes decisions about when to disclose cookies, cookies are inherently disclosure-resistant. Since cookies operate below the human-computer interaction level, users do not usually interact with cookies and many users are not even aware of their existence or how to find them on their computers. Phishers could try to solicit cookies from users by giving instructions on how to navigate the file system, find the cookie file and appropriate cookie, and manually type in the cookie data into a web form; but since the process of manually finding a cookie in the browser or file system deviates significantly from the normal authentication and interaction experience, asking users to do so is more likely to be met with suspicion and distrust.

- Ease of deployment: Cookies are well supported in production web servers and major web browsers, and web sites already use cookies for authenticating users' requests after they login. Since cookies operate at the HTTP protocol level, they impose no restrictions on the type of user experience or branding that web sites present to users.

In short, cookies are attractive because they meet our usability and deployability goals. We will now discuss how to build on cookies to provide the desired security properties.

Our solutions use *authentication cookies* – persistent cookies that authenticate users. First, we show how web sites can use SSL-only persistent cookies to defend against phishing. Then, we show how to defend against pharming as well by using SSL-only persistent IP cookies. Finally, we analyze *active cookies* [31], a variant of IP cookies.

4.1.1 Using SSL-only cookies

In the simplest scheme, when a user creates a new account with the web site, the site sets an SSL-only persistent cookie on the user's computer. Possession of a valid cookie is sufficient for authentication: users only need to present a valid cookie to authenticate themselves. The authentication cookie itself should contain a unique value to identify the user and an expiration time with cryptographic integrity protection [16].

Setting the `secure` flag on this cookie ensures that the cookie will only be sent over SSL connections, so eavesdroppers and web proxies cannot learn this secret. The browser's same-origin policy ensures that the cookie will only be sent back to the site that set the cookie, so third parties and phishers cannot learn the secret. When the user returns to the site, the browser will send this cookie and the site can use that to authenticate the user. After the user is authenticated, the user and web site can use the SSL connection between the browser and the server to communicate, and the browser's same-origin policy prevents content from other domains from eavesdropping on or tampering with this connection. Because SSL-only persistent cookies are part of the HTTP specification, sites can deploy this mechanism today without requiring browser extensions. This scheme could be used as a replacement for passwords.

Of course, SSL-only persistent cookies can be combined with other authenticators. For instance, a security-critical site might require the user to present both the authentication cookie and a password to gain access to their account. The cookie authenticates the user's computer and defends against phishing, while the password might be useful to ensure that even if the computer is shared among several family members, other family members cannot gain access to the user's account.

Existing implementations. Some web sites already use persistent authentication cookies. For example, some web sites offer a "remember me" option, which sets a persistent cookie on a user's machine. The

browser will present this cookie during subsequent visits to the web site, enabling the user to bypass the initial login process.

Some existing anti-phishing solutions also use authentication cookies to complement regular password authentication. Examples include Bank of America's Sitekey [4] and similar approaches by ING Direct [25], Vanguard [58], and Yahoo [65]. Before a user is permitted to login from a particular computer, she must "register" it. The registration process sets a SSL-only persistent authentication cookie on the user's computer, and only computers with authentication cookies are permitted to access the user's account.

Security and usability analysis. Phishing sites may be able to fool users into revealing their passwords, but the browser's same-origin policy denies phishers access to the users' authentication cookies. However, this enforcement mechanism is vulnerable to DNS attacks, enabling pharmers to steal users' authentication cookies and gain unauthorized access to users' accounts. Authentication cookies usually include the `secure` attribute, but the `secure` attribute does little to protect cookies against DNS poisoning attacks. The `secure` attribute specifies that a cookie should only be sent only over SSL connections, but it does not specify *which* SSL connection. Suppose a pharmer uses DNS poisoning to hijack `www.xyz.com` and a user subsequently visits `https://www.xyz.com/index.html`. The user's browser will attempt to establish an SSL connection, requiring the pharmer to present an X.509 certificate. If the server certificate is not signed by one of the root CAs in the browser or the certificate's `CN` does not match the server's domain (i.e., `www.xyz.com`), the browser will warn the user and ask her if it is safe to proceed. If the user heeds the warning and answers "no", the browser will cancel the connection and the user's cookies for `www.xyz.com` will remain safe. Unfortunately, most users ignore these warnings and click "OK" [5, 8, 61]. In this case, the browser will establish an SSL connection to the pharmer and append all cookies (secure and not) for `www.xyz.com` to the request.

Because the user has the option to override browser warnings and mistakenly connect to pharming sites, legacy SSL-only persistent cookies do not satisfy our usability goals (§ 2.2). To prevent pharmers from stealing these cookies, users must understand certificate error prompts and respond to them correctly, which is not something we can count on.

Deployability analysis. SSL-only persistent cookies are well supported in web servers and browsers. As evidenced by Sitekey and others, some web sites are already deploying SSL-only persistent cookies as part of an anti-phishing mechanism.

4.1.2 IP cookies

We use the term *IP cookie* to refer to a SSL-only host cookie whose `domain` attribute is an IP address. IP cookies can be used for web authentication in the same way described above. When a user creates a new account, a web site sets an IP authentication cookie on the user's machine. Then for subsequent requests, only computers which present a valid IP cookie are permitted to access the user's account.

Security analysis. Like regular cookies, IP cookies resist phishing attacks, since the browser's same-origin policy denies phishers access to users' authentication cookies. IP authentication cookies also resist pharming attacks, because an IP cookie will only be sent over connections to the IP address listed in the cookie's `domain` attribute: the DNS infrastructure is irrelevant.

Schemes using IP cookies are, however, vulnerable to active attackers that control some of the routing infrastructure. Since these attackers can intercept a client's HTTP requests to the site's IP address, they may

be able to steal its IP cookies, if users are willing to click through certificate error dialogs.

Usability analysis. Browsers automatically and transparently regulate access to IP cookies based on the server's IP address. With respect to pharming attacks, users cannot take actions which bypass or override this policy. However, an active attacker may be able to pilfer a user's IP cookie for a web site if she responds incorrectly to certificate error prompts.

Deployability analysis. Users typically access a web site through its domain name, but this usage is incompatible with IP cookies. For instance, if the browser has an IP cookie for the IP address `1.2.3.4`, this cookie will be sent with a request for the URL `https://1.2.3.4/index.html`, but it will not be sent with a request for `https://www.xyz.com/index.html`, even if the DNS name `www.xyz.com` resolves to IP address `1.2.3.4` for this HTTP connection.

The consequence is that if we want to use IP cookies, all URLs must identify the server using its IP address instead of its domain name. However, we cannot expect users to start accessing web sites using IP addresses. To address this problem, web sites can allow users to initially access content using URLs containing domain names, but when the user logs in, the server redirects her browser to a URL containing the server's IP address (e.g., `https://1.2.3.4/login.html`). The browser will include her IP cookie with this request, enabling the server to authenticate the user.

After authentication, navigating via URLs with the site's domain name may be unsafe, since these requests may be intercepted by the pharmer. This has unfortunate consequences for the user experience, because it means that the URL shown in the browser's address bar will contain an IP address instead of the site's human-recognizable site name. The presence of URLs containing IP addresses in the URL bar or links might be psychologically unacceptable to some users, causing suspicion or confusion. Since a common phishing modus operandi is to lure users with emails containing URLs with IP addresses, anti-phishing education efforts have urged users to be suspicious of URLs containing IP addresses. Also, sites may be reluctant to deploy a security solution that affects the branding of their site in this way.

IP cookies have other deployment problems as well. IP cookies make it harder to do DNS-based load balancing, requiring web sites to use a long-lived static IP address, and are less flexible. More seriously, they may require substantial changes to the web site's architecture, because every page requiring authentication has to redirect to the IP address of the web server if the user is not already authenticated. Also, for sensitive content, such as banking information, it is desirable to use SSL, but it is unclear whether certificate authorities issue SSL certificates for IP addresses – for example, Verisign does not [59]. These factors may make sites reluctant to embrace IP cookies.

4.1.3 Active cookies

To address these drawbacks of IP cookies, Juels, Jakobsson, and Stamm propose *active cookies* [31], a repurposing of IP cookies within a dynamic authentication protocol designed to resist pharming attacks. Authentication using active cookies requires no additional user involvement and is supported on existing browsers. Active cookies use IP cookies to authenticate users, but after authentication completes, the server conducts the remainder of the web session with URLs containing the site's domain name. This approach allows the web site to use SSL after authentication and helps avoid any usability and branding problems caused by IP addresses in the URL bar.

To resist pharming attacks after authentication, active cookies use *client IP binding* [16]. With client IP binding, the web server traces the IP address of the user's machine during authentication and binds this

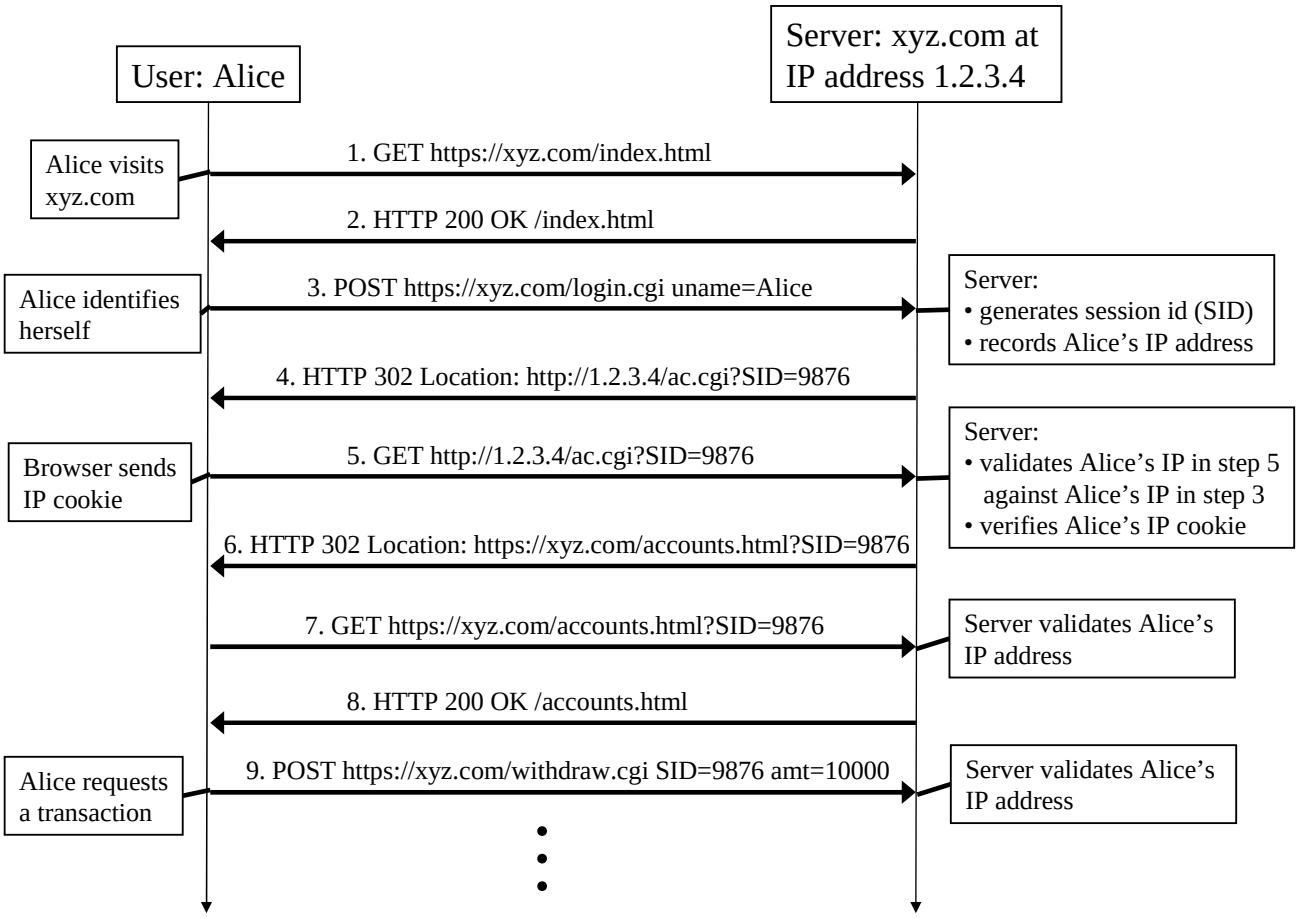

Figure 1: **Active cookies authentication protocol.**

IP address to the session. After the client authenticates herself with her IP cookie, the server only accepts requests from the traced IP address. The server determines the client's IP address from the connection over which it received client's IP cookie. Since this request must use an explicit IP in the URL, it is outside the influence of pharming attacks and enables the server to reliably determine the client's IP address.

We now present the active cookies protocol in more detail. We illustrate an instantiation of this protocol in Figure 1, where a user, Alice, authenticates herself to a server hosting `xyz.com` at IP address 1.2.3.4. We assume Alice previously obtained an IP cookie for 1.2.3.4. Alice first visits the web site and identifies herself (steps 1-3), for example, with a username. This interaction occurs using URLs containing domain names, which rely on the DNS system. Juels et al. refer to requests for URLs which rely on DNS as occurring over the *soft channel*, since a pharmer can intercept these requests.

After the server receives the user's login request in step 3, the server verifies the username and password, creates a unique session id (SID), and traces the user's IP address over the soft channel. Then in step 4, the web server redirects the user (e.g., using the HTTP 302 status code) to a URL containing the server's IP address and the SID (`http://1.2.3.4/ac.cgi?SID=9876`). Juels et al. refer to requests for URLs containing the server's IP address as occurring over the *hard channel*, since these requests do not use the DNS system and cannot be intercepted by pharmers. In step 5, Alice's browser fetches this URL over the hard channel and attaches Alice's IP cookie for 1.2.3.4.

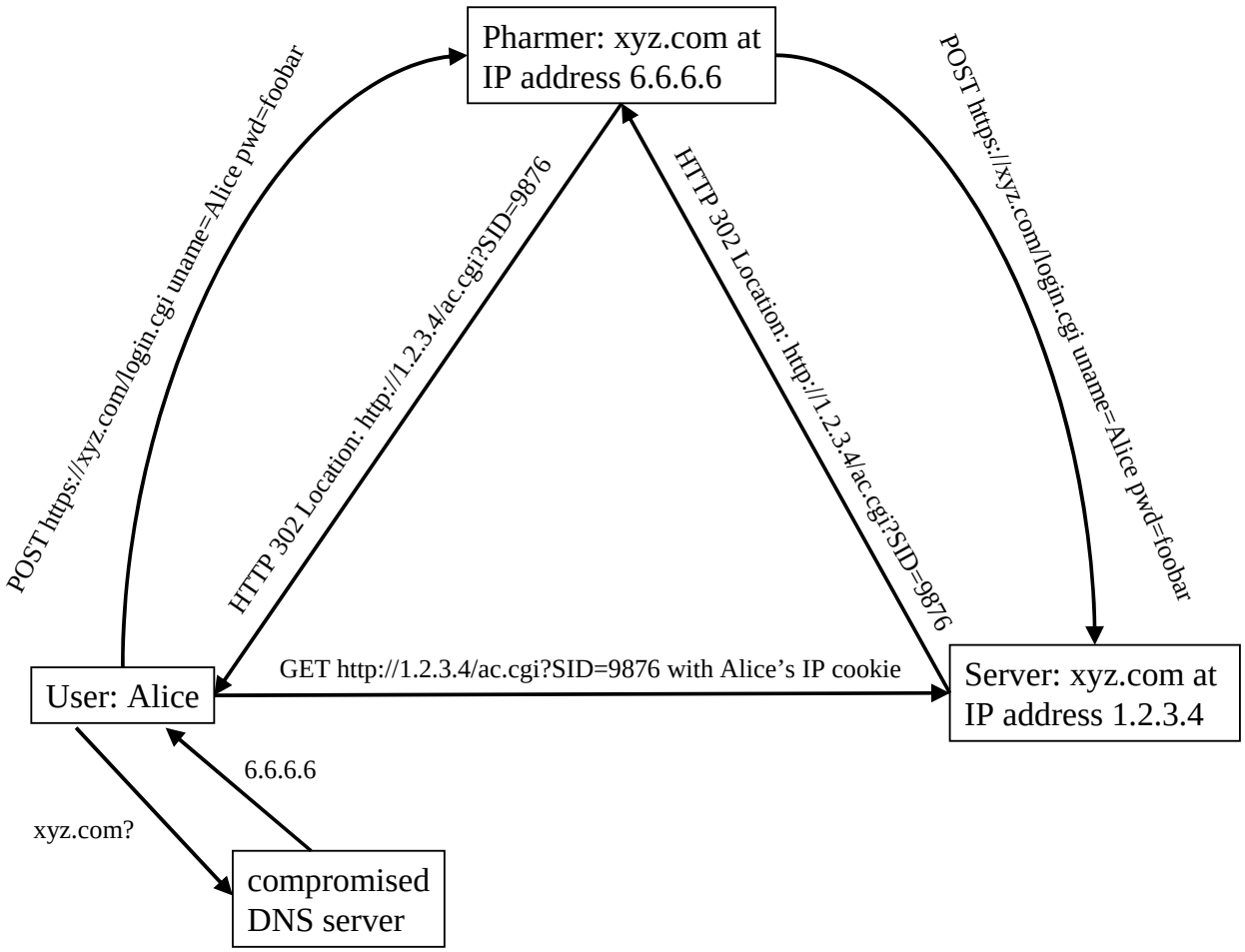

Figure 2: **Active cookies in the presence of a MITM pharming attack.**

The server verifies the IP cookie and compares client's IP address in step 5 against the client's IP address it previously traced in step 3. If these IP addresses are the same, Juels et al. argue there is no pharming attack and the server should allow Alice access to her account, but if they are different, then a MITM pharming attack is likely in progress and the server should deny access. In addition, after the server authenticates Alice, it uses client IP binding to bind Alice's IP address to session and only accepts requests from the traced IP address from steps 3 and 5.

Juels et al. argue that if a pharmer attacking `xyz.com` is present, then it will man-in-the-middle Alice's requests and the server's responses, and in step 3, attempt to log in as Alice. The server responds with a redirect to the hard channel. Since the server will not successfully authenticate Alice without her IP cookie, the pharmer must forward the redirect to Alice. Alice's browser then connects to the server over the hard channel and sends her IP authentication cookie. Since from the server's perspective, the soft channel request (in step 3, from the pharmer) and the hard channel request (in step 5, from Alice) originate from different IP addresses, the server denies access and warns Alice that she may be under attack. We illustrate this scenario in Figure 2. Based on this analysis, Juels et al. conclude that authentication using the active cookie protocol resists pharming attacks.

Security analysis: Dynamic pharming attacks. We show active cookies are vulnerable to pharming. To help motivate our attack, we first make a few observations about the active cookies protocol. Alice's IP cookie is necessary to gain access to Alice's account, and Alice's machine will only release Alice's IP cookie over the hard channel to the legitimate server. Since the pharmer cannot influence the outcome of IP tracing for the hard channel, the pharmer's only option is to also originate soft channel requests from Alice's machine, but maintain control of the session.

To trigger this chain of events while maintaining control of Alice's session, the adversary uses a *dynamic pharming attack*. A dynamic pharming attack is a pharming attack where the adversary changes the DNS entry (i.e, the domain name/IP address mapping) for the target site during the course of the attack. First, the pharmer initializes the DNS entry for `xyz.com` to the pharmer's IP address, say 6.6.6.6. The pharmer also indicates in the DNS record that requesters should not cache this result, i.e., it sets the TTL=0. Then, when Alice requests `https://xyz.com/index.html` in step 1 of Figure 1, her browser sends this request to 6.6.6.6, and the pharmer returns a "trojan" `index.html` document. This trojan document will monitor and influence Alice's subsequent interactions with the legitimate `xyz.com`.[1] The trojan document has following general structure:

```
<html>
<body>
<script>
---MALICOUS JAVASCRIPT CODE---
</script>
<iframe
 src=''https://xyz.com/index.html''>
<\body>
<\html>
```

After the pharmer returns the trojan document to Alice, it updates the DNS entry for `xyz.com` to the IP address of the legitimate server for `xyz.com`, say 1.2.3.4. The goal is to force the browser load the legitimate

[1]This request may use SSL, which we assume here. Since it is unlikely the pharmer can obtain a valid certificate for `xyz.com`, the user will probably receive a certificate warning dialog. Unfortunately, researchers have shown that users routinely ignore such security warnings and just click "OK" [5, 8, 61], allowing the attack to proceed.

`https://xyz.com/index.html` document into the `<iframe>`, which the browser displays to the user. The adversary then waits for the user to login and the active cookie authentication protocol completes, after which the server switches the session back to the soft channel.

At this point the trojan takes control and begins to monitor the user's interactions in the `<iframe>`. Since the parent document and the `<iframe>` both "originated" from `xyz.com`, the browser's same-origin policy will allow the malicious Javascript running the parent document to access the content in the `<iframe>`. The trojan effectively hijacks control of Alice's session – it can eavesdrop on sensitive content, forge transactions, keylog secondary passwords, etc.

One complication to mounting this attack is web browsers' use of *DNS pinning*. With DNS pinning, a web browser caches the result of a DNS query for the user's entire browsing session (i.e., until the user closes the browser), regardless of DNS entry's specified lifetime. Browsers implement DNS pinning to defend against variants of the "Princeton attack" [20]. In the "Princeton attack", a malicious web server first lures a victim who resides within a firewalled network containing privileged web servers. We assume these servers are accessible only to machines behind the firewall. After the victim connects to the malicious server, the adversary changes its DNS entry to the IP address of a sensitive web server located on the victim's internal network. Same-origin policy restricts malicious code from accessing other domains, but since the adversary's domain now resolves to an internal IP address, this attack enables Javascript served by the adversary to the victim to access internal web servers.

DNS pinning poses a problem for dynamic pharming attacks because once a browser resolves a domain name using DNS, it will use the resolved IP address for the entire browsing session and ignore any subsequent changes the pharmer makes in the DNS system. However, since DNS pinning "breaks the web" in certain scenarios, e.g., dual homed IPv6/IPv4 servers, dynamic DNS, and automatic failover, browsers implementors have recently relaxed their DNS pinning policies. For example, the results of DNS queries are now only pinned for a fixed amount time.

However, Martin Johns discovered a technique for circumventing DNS pinning completely [28]. Johns discovered that a pharmer can force a victim to renew its DNS entry for a given domain on demand by rejecting connections from the victim, e.g., by sending an ICMP "host not reachable" message. The browser reacts by refreshing its DNS entry for the domain. Put in the context of our dynamic pharming attack, after the pharmer delivers the trojan page to the user, it rejects subsequent requests from user's machine and updates the DNS entry for `xyz.com` to the IP address of the legitimate server. Now, when the user's browser loads the `<iframe>`, it will first attempt to contact the pharmer, fail, refresh its DNS entry, receive the IP address of the legitimate server, and load the legitimate `index.html` document into the `<iframe>`. The attack now proceeds as before.

To parallelize this attack against multiple concurrent users, it is inefficient to repeatedly update the DNS entry for `xyz.com`. If the adversary has compromised a local, root, or authoritative DNS server, or changed the authoritative server of record for `xyz.com`, the adversary can selectively respond with the pharmers IP or the legitimate server's IP depending on the stage of attack. However, if the adversary only has the ability to change DNS entries for `xyz.com`, say at a local or root DNS server, this attack is unscalable because the pharmer must update the DNS entry for each instance of the attack and reset it after the attack completes.

In the latter scenario, the pharmer can use round robin DNS entries to make this attack scalable. A round robin DNS entry consists of multiple IP addresses for a single domain name. Web sites use round robin DNS to implement load balancing. The DNS server returns an ordered list of the IP addresses in response to a query, but rotates the order for each response. Browsers usually connect to the first IP address in the list, and this achieves some degree of load balancing among clients. When the connection fails, the browser tries the next IP address on the list, until it successfully makes a connection.

To leverage round robin DNS entries in a dynamic pharming attack, the pharmers creates a round robin DNS entry containing two IP address: the pharmer's IP and the legitimate server's IP. Roughly half the DNS responses will be in the order: pharmer's IP, server's IP. In this case, the user will connect to the pharmer first, and pharmer delivers the trojan page. The pharmer rejects subsequent connections from the user, and the user's browser will automatically failover to the legitimate server, after which the attack proceeds as before.

Although dynamic pharming attacks against active cookies do not enable pharmers to steal users' authentication credentials (i.e., their IP cookies), they enable pharmers to compromise users' sessions in real time. Since this is a significant vulnerability, we conclude active cookies are insecure in the presence of pharming.

4.2 Client-side SSL

The most common usage of SSL is for server authentication, but in the SSL specification, a server can also request *client-side authentication*, where the client also presents an X.509 certificate and proves knowledge of the corresponding private key. Using client-side SSL, servers can identify a user with her SSL public key and authenticate her using the SSL protocol. Since client-side SSL authentication relies on end-to-end public key cryptography, currently, it is generally considered the most secure option for web authentication.

Security analysis. Client-side SSL authentication resists phishing attacks. Although a phisher may be able to trick a user into participating in mutual authentication using SSL, the phisher cannot use this interaction to impersonate the user at another web site. Authentication requires knowledge of the private key, which the users' browser always keeps secret.

However, client-side SSL is vulnerable to the dynamic pharming attack described in Section 4.1.3. The attack is nearly identical. After returning the trojan page, the pharmer switches the DNS entry (or uses round robin DNS entries), and denies subsequent connections from the client. The browser will then load content from legitimate server into the `<iframe>` and mutually authenticate itself to the server using its client-side certificate. After authentication completes, the same-origin policy will allow the trojan Javascript in the outer document to affect the authenticated session with legitimate server in the `<iframe>`.

Usability and deployability analysis. An advantage of client-side SSL is that the user's authentication credential, i.e., her private key, is disclosure resistant. The user is not required to memorize her private key, and after the user imports her private key, her browser uses it (almost) automatically. It is difficult for attackers to trick a user into inadvertently disclosing her private key.

However, research studies have shown there are significant usability problems with client-side SSL [3, 10, 21]. To use client-side SSL, a user first must import a certificate and corresponding key pair into her browser. Web sites may provide users these certificates or user may be required to obtain her own certificate signed by a certificate authority. Regardless, studies have shown that obtaining and installing a certificate and key pair is a cumbersome and confusing process for users, and when users make mistakes, they are hard to correct. Also, most users do not understand cryptography, why need they certificates, and do not understand their connection with authentication [60].

5 Resisting pharming and active attacks with the locked same-origin policy

Dynamic pharming attacks pose a significant challenge to the development of deployable pharming resistant authentication for legacy browsers. Although dynamic pharming attacks leverage the implementation details DNS pinning, "fixing" DNS pinning is unlikely. DNS pinning has a lengthy and controversial history in Firefox and Mozilla [37], and the current implementation is an explicit compromise to support dynamic DNS and round robin DNS for failover [36, 38]. Developing better authentication protocols is not the right approach either; since dynamic pharming hijacks a user's session after authentication completes by exploiting weaknesses in browsers' same-origin policy, a better authentication protocol is unlikely to help.

To resist dynamic pharming, we must upgrade browsers' same-origin policy. Enforcing same-origin based on domain name is problematic in the presence of pharming attacks because pharmers can influence the mapping from domain name to subject. We argue browsers should enforce same-origin policy cryptographically. Instead of identifying web objects by name (i.e., domain name), we propose browsers should identify web objects by public key. We refer to web objects identified by public key as *locked web objects*. For locked web objects, we propose browsers enforce a *locked same-origin policy*: the browser allows a web object to access another web object only if their public keys match.

Since the majority of web objects on the Internet are not associated with any public key, enforcing the locked same-origin policy for all objects is impractical. However, browsers already associate web objects retrieved over SSL with a public key: the key of the server hosting the object. We propose browsers should enforce the locked same-origin policy only for SSL web objects, and use the legacy same-origin policy (i.e., using domain names) for non-SSL objects.

To use the locked same-origin policy browsers must enforce three properties: 1) unlocked web objects are prohibited from accessing locked web objects, 2) the owner of a locked web object (i.e., the server from which is was retrieved) must prove knowledge of the associated private key, and 3) a locked web object is allowed to access another locked web object only if their public keys match. Legacy browsers already enforce the first property: objects not retrieved over SSL are not allowed to access SSL objects, even if the domains of the objects would otherwise allow it. Browsers already enforce the second property as well: retrieving a web object over SSL requires the browser to successfully establish an SSL connection to the server, and establishing an SSL connection requires the server to prove knowledge of the private key associated with its advertised public key. Upgrading browsers to enforce the locked same-origin policy requires changing browsers to support the third property: regulating inter-object accesses between locked objects.

Our locked same-origin policy requires no changes to the HTTP cookie specification, SSL, or web servers; web sites operate in the same manner as before. To take advantage of the locked same-origin policy and protect its users from pharming attacks, a web site only needs to use SSL.

5.1 Locked cookies

In Section 4.1, we argued HTTP cookies are an excellent candidate for web authentication credentials, but they have some limitations with legacy browsers. SSL-only cookies are a highly deployable and usable option, but are vulnerable to pharming attacks. Although IP cookies resist pharming attacks, they can be cumbersome and tricky for web sites to deploy. To take advantage of IP cookies a web site must use URLs containing the IP address of the server. With IP based URLs, we lose many advantages of using URLs containing domain names, e.g., SSL, load balancing, branding, and usability. If an anti-pharming solution is too expensive, complicated, or restricts scalability, it will likely be passed over in favor of simplicity and functionality.

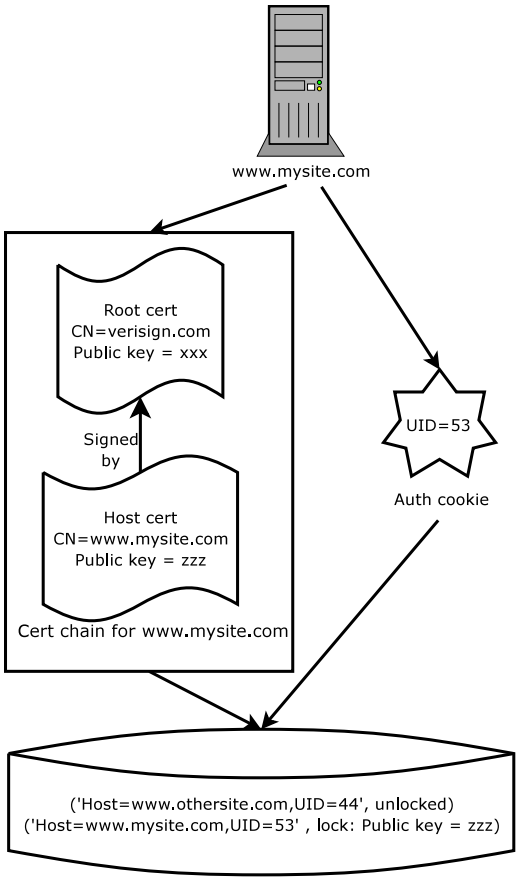

Figure 3: **A locked host cookie.** We show how a site `www.mysite.com` sets a locked cookie over a SSL connection. The rectangle depicts the X.509 certificate chain associated with the SSL connection, and the cylinder at the bottom represents the browser's cookie jar after the cookie is accepted.

However, with browsers enforcing the locked same-origin policy, web sites can now use SSL-only persistent cookies to authenticate users and resist phishing, pharming, and active attacks. We refer to SSL-only cookies under a locked-same-origin policy as *locked cookies*. Browsers enforce the same-origin policy for locked cookies by binding SSL-only cookies to the public key from the originating server's X.509 certificate when the cookies are set. Then, instead of using domain names to enforce access control, the browser uses servers' public keys. Locked cookies require no changes to the HTTP cookie specification, SSL, or web servers; web sites set and receive cookies in the same manner as before.

The design of locked cookies is straightforward in concept. When a web site at domain D sets an SSL-only host cookie C, using either the `Set-Cookie` header or the `document.cookie` Javascript interface, the browser stores the server's public key PK from its X.509 certificate along with the cookie. Then, for subsequent HTTPS requests to D, the browser will append C to the request only if it has successfully established an SSL connection with D, and D's certificate has public key PK. For cookie accesses via the `document.cookie` Javascript interface, the browser grants access to C if the requesting script is executing in a document requested over an SSL connection from D, and D's certificate has public key PK. If the public keys do not match, the browser does not send the cookie and does not prompt the user to override the decision. See Figure 3.

Security analysis. Locked cookies protect authentication cookies against all threat models in Section 2.1: phishing, pharming, and active attacks. With locked cookies, a browser only allows a server access to a host cookie tagged with public key PK if 1) the server presents a certificate for PK, and 2) the browser and server can successfully establish an SSL connection. In contrast to current browser policy, locked cookies do not depend on users correctly answering prompts in response to certificate errors (e.g., an adversary presenting a self-signed certificate with a spoofed domain name) to prevent cookie theft; locked cookies rely solely on the public key in the server's certificate and do not depend on how users respond to any errors. If the adversary tries to present a certificate for PK and does not know the private key corresponding to PK, she will not be able to successfully complete the SSL handshake; the browser will then automatically cancel the connection with no option of user override.

Usability analysis. Locked cookies satisfy all of our usability goals. As we discussed in Section 4.1, cookies are easy to use and disclosure-resistant. In contrast to existing SSL-only cookies, a user cannot take any actions that will cause her browser to send her authentication cookies to attackers, e.g., by ignoring certificate warnings. Locked cookies cryptographically enforce the same-origin policy for authentication cookies automatically; users do not need to understand SSL certificates or how locked cookies work to securely authenticate to web sites and protect their authentication credentials.

Deployability analysis. Implementing locked cookies in browsers would require only minimal changes. Browsers only need to extract and store the public key from the SSL session when a server sets a cookie, and perform an equality check on each cookie access. See Section 7 for details about our implementation of locked cookies in Firefox.

Since we are proposing a change in browser policy, we must make sure this new cookie policy does not "break the web": no browser developer is likely to embrace an extension that makes their browser incompatible with existing web sites, so legacy web servers had better continue to work even when visited with locked cookie-enabled browsers. Locked cookies seem safe from this point of view. Locked cookies are host cookies, and browsers currently enforce the same-origin policy for host cookies by allowing access only to the same fully qualified domain name which set the cookie. Since the binding between a fully qualified domain name and its public key is relatively static and long-lived, in the absence of malicious activity, the locked cookie policy should be for the most part equivalent to today's cookie policy.

One exception is that the relationship between a domain and its public key may change when its certificate expires. The business model of many CAs is to issue certificates that are valid only for a relatively modest period of time, e.g., one or two years, and require customers to renew their certificates when they expire. The idea is that a subscription type service will generate a continuous revenue stream and CAs can adjust their prices on a yearly basis. Unfortunately, when web sites renew their certificates they often generate a server certificate from scratch using a new public key. If server uses a new public key every time its certificate changes, this means its SSL-only persistent cookies on users' machines will expire when the certificate expires, regardless of the value of the cookie's `expires` attribute. At first glance, this might appear to pose a risk of "breaking the web." However, we believe that this risk is minimal. Since clients occasionally delete persistent cookies stored in their browsers, web sites cannot rely on long-term access to persistent cookies, and they already have a strong incentive to structure their applications accordingly. Therefore, we expect that expiring persistent cookies prematurely will not break legacy web sites, though it may inconvenience users slightly. To avoid this inconvenience, web sites aware of locked cookies can arrange for their renewed server certificates to use the same public keys as their old ones.

Any web site that already uses secure host cookies will automatically gain protection from pharming

and active attacks as users upgrade to locked cookie-enabled browsers. Some current anti-phishing schemes (e.g., Sitekey [4]) already utilize secure host cookies; locked cookies would transparently strengthen the security that these schemes provide, without any work from the web site operator.

5.2 Locked domain cookies: Authenticating to multiple servers in a domain

In Section 5.1, we showed how web sites can take advantage of locked cookies to authenticate users to a single web server and resist pharming and active attacks. More generally, a web site might be composed of several host names, e.g., `mail.xyz.com`, `www.xyz.com`, `login.xyz.com`, and the web site must authenticate users to all these servers. We assume these servers share the same second-level domain name, e.g., `xyz.com`. Legacy servers commonly solve this problem using domain cookies. To authenticate clients to multiple servers at `xyz.com`, the web site can set a domain authentication cookie on clients with `domain` attribute `xyz.com`. Then clients will include this cookie with requests to any with `xyz.com` as part of its domain name.

However, domain cookies are vulnerable to pharming attacks. If an adversary pharms any host name in `xyz.com`, she can steal users' domain authentication cookies for `xyz.com`. To address pharming attacks against domain cookies, we propose *locked domain cookies*, a cryptographic same-origin policy for secure domain cookies which enables web sites to specify access rights for domain cookies in an incrementally deployable and backward compatible way. Applying a cryptographic same-origin policy to domain cookies is challenging. We cannot simply extend the same rules for locked host cookies to domain cookies. Since the public key of the server setting the domain authentication cookie is probably different from the public keys of the other servers in the domain, such a policy would deny those servers access to the cookie. To address this problem, our policy tags cookies with a public key, like locked host cookies; however, where locked cookies associate a key with a particular host, SSL domain cookies associate a key with a subdomain range.

To infer this association, locked domain cookies utilize wildcard certificates. A wildcard certificate is an X.509 certificate with a wildcard expression in the `CN` field (e.g., `*.xyz.com`). CAs currently offer wildcard certificates as an option for deploying multiple SSL servers within a subdomain range. Customers purchase a single wildcard certificate and install it (and the corresponding private key) on all their servers in the subdomain range. Modern browsers already support wildcard certificates, and they require no additional user interaction.

To specify access rights for a secure domain cookie with `domain` attribute D (e.g., `xyz.com`), servers install a wildcard certificate in their certificate chain with `CN`=*.D. Unlike the common usage and for more detailed reasons we discuss later, we require this wildcard certificate not be the leaf (i.e., server) certificate, but instead be a signing certificate part of the server's certificate chain (i.e., an internal node). When a server sets a secure domain cookie C with `domain` attribute D, the browser searches the server's certificate chain for a wildcard signing certificate with `CN`=*.D. If it finds such a certificate, it tags C with the wildcard certificate's public key PK. Then, for a future HTTPS request to a server in subdomain range D, the browser will append C to the request only if it can successfully establish an SSL connection to the server, and server's certificate chain includes a wildcard signing certificate with `CN`=*.D and public key PK. A similar policy applies to cookies accesses via `document.cookie`. If a server sets a secure domain cookie and its server chain does not contain such a wildcard signing certificate, the browser applies the legacy same-origin policy to the cookie (i.e., using domain names). See Figure 4.

Security analysis. Locked domain cookies protect secure domain cookies against all threat models in Section 2.1: phishing, pharming, and active attacks. With locked domain cookies, a browser only allows

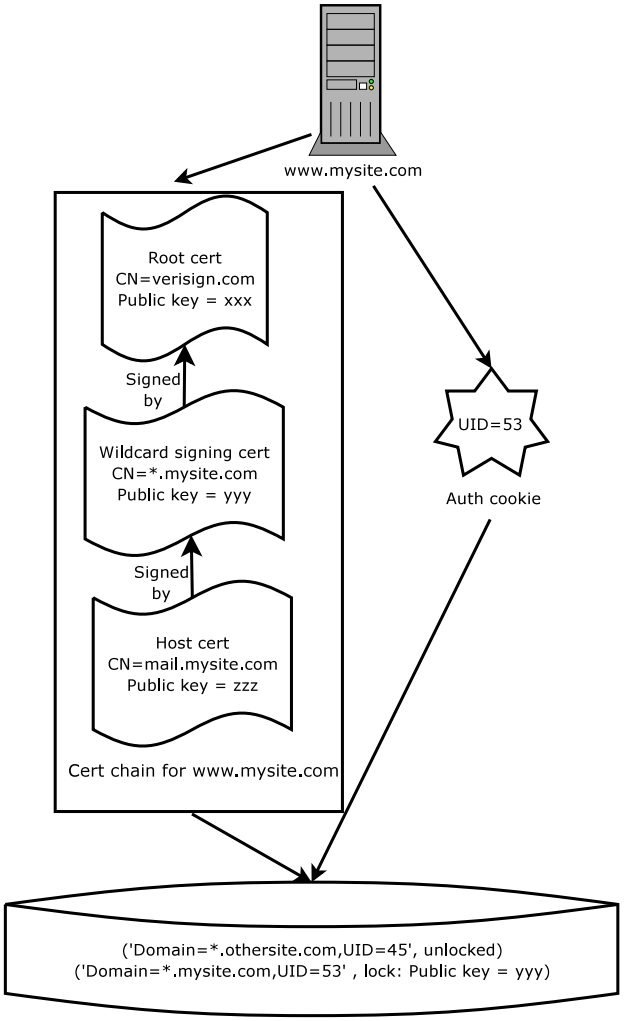

Figure 4: **A locked domain cookie.** We show how a site `www.mysite.com` sets a locked cookie over a SSL connection. The rectangle depicts the X.509 certificate chain associated with the SSL connection, and the cylinder at the bottom represents the browser's cookie jar after the cookie is accepted.

a server access to a domain cookie tagged with public key PK if 1) the server presents a certificate chain containing a non-leaf wildcard signing certificate with PK, and 2) the browser and server can successfully establish an SSL connection. Since the private key of a signing certificate is required to construct a valid certificate chain containing it (as an interior node), an adversary without knowledge of the private key cannot construct a valid certificate chain containing the wildcard certificate. If the certificate chain is invalid, the browser will cancel the SSL connection with no option of user override.

Usability analysis. Like locked host cookies, locked domain cookies satisfy all of our usability goals. The enforcement of the same-origin policy for locked domain cookies does not depend on whether users understand locked domain cookies or how users respond to any certificate errors.

Deployability analysis. For web browser developers to adopt locked domain cookies, and for web sites to take advantage of them we must verify two properties: 1) browsers enforcing the stronger same-origin policy for locked domain cookies will not "break the web" when users visit legacy sites, and 2) users of legacy browsers must still be able to interact with web sites that use wildcard signing certificates to protect domain cookies (even though legacy browsers will not enforce the stronger security policy).

For the first property to hold, no legacy web server must use wildcard signing certificates as a non-leaf node in its certificate chain. Otherwise, there is a chance a browser using locked domain cookies might misinterpret a wildcard certificate, enforce the stronger same-origin policy, and wrongly deny a legitimate server access to a domain cookie.

To check whether servers use wildcard certificates as non-leaf nodes in their certificate chains, we conducted a survey of SSL servers. In our survey, we crawled the web on May 26, 2006 for SSL servers, starting from a list of major news, portal, and financial sites. For the sake of simplicity, we restricted our study to the following top-level domains: `com`, `org`, `net`, `gov`, `edu`, `biz`, `info`, and `name`. We excluded international top-level domains. We found 10,814 fully qualified SSL domains from 4,878 second-level domains. This corresponds to roughly 6% of the SSL servers found by the more extensive monthly SSL survey conducted by E-Soft and `securityspace.com` [53]. In our SSL server survey, we found no evidence of web sites using wildcard signing certificates as non-leaf nodes in certificate chains. This is strong evidence that new browsers which enforce our stronger same-origin policy will not break legacy web sites[2].

The second property also holds. Since modern browsers already accept wildcard certificates, servers which use them will continue to interoperate with legacy browsers. Although legacy browsers will not enforce the stronger same-origin policy of locked domain cookies and thus will be vulnerable to pharming and active attacks, legacy browsers will still enforce the current policy based on domain names.

Locked domain cookies require a few additional changes to the locked same-origin policy. Browsers need to extract and store the public key from the wildcard certificate when a server sets a cookie, and perform an equality check on each cookie access. Server changes are minimal. Web sites can continue to use domain cookies as before; the only required change is for operators to purchase and install wildcard signing certificates in the chains on the servers requiring access to secure domain cookies. Web sites are already starting to use wildcard certificates for other purposes; 6% (305) of the sites in our survey already use wildcard certificates. Also, installing wildcard signing certificates as internal nodes in certificate chains does not require the distribution of any additional private keys to servers—only the distribution of new certificate chains. Web sites can safely store the private keys for their signing certificates in protected storage,

[2]Note that if we had instead used leaf wildcard certificates in our policy, then we would risk breaking legacy web sites. In our survey, we found 88 web sites that used wildcard leaf certificates for some servers in a domain range, but used individual certificates for other servers in the same range, and these sites might not work with new browsers implementing our stronger same-origin policy.

disconnected from the Internet. Finally, locked domain cookies do not restrict how web sites organize server (i.e., leaf) certificates for subdomains. Web sites could use a single wildcard certificate for all subdomains or deploy a certificate with a unique public key for each subdomain; the structure of the servers' leaf certificates is inconsequential to locked domain cookies.

5.2.1 Server configuration changes are necessary to support locked domain cookies

To maximize chances of deployment, an ideal same-origin policy for SSL-only domain cookies would ideally only require modifications to browsers, but no changes to server cookie handling, web site organization, or protocols. Without help from servers to help indicate authorized servers, browsers must infer whether the X.509 certificates for two different subdomains represent the same real world subject. We found developing a heuristic for this to be challenging. The heuristic must not be too conservative; for it to be embraced by browser developers, it must never "break the web" and deny a domain cookie to legitimate server. However, if the heuristic is too liberal, a clever pharmer might be able to purchase certificates which enable him to steal sites' domain cookies.

Unfortunately, due to the large number of misconfigured servers and various inconsistencies in the way some web sites deploy certificates, achieving this with only browser changes in a way which does not "break the web" is unlikely. Our SSL survey (see Section 5.2) found that certificate hierarchies in subdomains are both diverse and disconnected. Of the 4,878 second-level domains in our survey, 9% had subdomains with SSL certificates issued by more than one CA, or used a mixture of self-signed and CA issued certificates. For example, for `wellsfargo.com`, we found 21 different SSL subdomains, with 16 certificates issued by GTE CyberTrust and 5 issued by Verisign. To further complicate matters, the Verisign issued certificates were directly signed by Verisign, but the server certificates in the GTE CyberTrust tree were issued by a Wells Fargo Certification Authority certificate (presumably an internal CA for Wells Fargo), which in turn was issued by GTE CyberTrust. The subject names and auxiliary information of the server certificates also varied significantly.

Requiring the `CN` of the certificate to match the domain name of the server seems promising, since it is more unlikely a major CA would issue such a certificate to a pharmer targeting a major web site. However, a significant number of legitimate servers are misconfigured and we cannot impose this policy without risking "breaking the web". The monthly SSL server survey by E-Soft and `securityspace.com` shows approximately 11% of SSL servers have `CN`/domain mismatches [52]. If a misconfigured web site uses a SSL-only domain session cookie to authenticate users' requests, and we incorrectly deny legitimate servers access to those cookies, the site may become unusable. Relying on users to manually verify mismatches will likely be ineffective; users routinely ignore these warnings and just click "OK" [5, 8, 61].

Since we cannot anticipate all the ways sites may use SSL-only cookies, and we cannot distinguish between 1) a well-organized web site under a pharming attack using a self-signed certificate, and 2) a misconfigured and unstructured web site, we must liberally send SSL-only domain cookies to avoid "breaking the web". Unfortunately, this means the current same-origin policy for SSL-only domain cookies is likely the best we can do without any server or protocol changes.

6 Cross-site request forgeries (XSRFs)

Cross-site request forgery (XSRF) vulnerabilities [6, 51] can arise when a user's web browser uses implicit authentication. An implicit authentication mechanism is one which is automatically executed by the user's browser and requires little or no user interaction. Examples of implicit authentication mechanisms include

cookies, client-side SSL, and HTTP authentication. In a XSRF, an adversary causes a victim's web browser to unintentionally send an HTTP request for a resource which uses the victim's implicit authentication credentials. For example, if a user visits `attacker.org` and uses implicit authentication for `bank.com`, the attacker could return an page with an hidden `<iframe>` containing a request to withdraw money from the user's account at `bank.com`, and the victim's browser will automatically authenticate this request, without the victim's knowledge.

One approach to resist XSRFs is use session tokens embedded in URLs and hidden form fields which are associated with a user's implicit authentication credentials [27, 30]. Since the same-origin policy denies adversaries access to these tokens, a user must explicitly interact with the site to generate valid requests. Researchers have also proposed client-side solutions which remove implicit authentication information from requests which may be unintentional [29].

7 Implementation

We have implemented locked host cookies for Firefox 1.5 in approximately 500 lines of C++ and Javascript. We plan on extending this implementation to enforce Javascript accesses to HTML documents, images, XMLHttpRequest, and other web objects. We have used our prototype locked cookie browser in daily browsing without encountering any problems with any web sites, including those which use secure cookies.

We slightly modified the storage mechanism for cookies in Firefox. Browsers currently manage cookies as a map $(\mathit{domain}, \mathit{name}) \mapsto \mathit{value}$. The cookie interface in browsers has no notion of "modify"; setting a cookie overwrites any previously stored cookie with the same $(\mathit{domain}, \mathit{name})$ pair. This approach is insufficient for locked cookies, because it allows pharmers to overwrite the cookies of legitimate web sites. Note that it does not work to deny the overwrite if the public key in the pharmer's certificate does not match the one in the stored cookie; legitimate web sites may change the public key in the server certificates from time to time, and we must allow these sites to set new cookies. For this reason, our implementation manages cookies as a map $(\mathit{domain}, \mathit{name}, PK) \mapsto \mathit{value}$, where PK is the public key in the certificate of the server which set the cookie. Consequently, we might have multiple cookies associated with a single $(\mathit{domain}, \mathit{name})$ pair, but only one will be sent back on any particular SSL connection.

8 The registration problem

We have thus far ignored one crucial aspect of using cookies for authentication: how does a user's machine initially receives an authentication cookie from the web site? This is commonly known as the *registration problem*. The registration problem is an interesting problem in its own right, and we do not attempt to solve it here.

One key challenge is that the registration procedure must be secure against *registration attacks*. For instance, a phisher or pharmer might trick the user into believing her computer must be unnecessarily re-registered and then somehow attack the re-registration procedure. Bank of America's Sitekey and similar anti-phishing mechanisms have registration attack vulnerabilities [67]. When a Sitekey user initially registers, she gives answers to several "personal entropy" questions [12], questions to which a phisher is unlikely to be able to guess the answers, e.g., "What is the name of your high school mascot?". Users who need to register another computer must correctly answer these questions before receiving an authentication cookie. However, phishers and pharmers can use a man-in-the-middle registration attack to solicit the correct answers from unsuspecting victims and obtain valid authentication cookies, making Sitekey insecure against

registration attacks. The problem is that "in-band" (HTTP-based) registration procedures are normally vulnerable to the same attacks we are trying to prevent. In future work, we plan to explore solutions utilizing email, cell phones, and other "out of band" channels to distribute authentication cookies to users.

9 Future work

Recent studies have found that about 58% of users have deleted their cookies at least once [41]. This is problematic for web authentication cookies, since frequent cookie deletion requires frequent re-registration, potentially frustrating users. It would be useful to develop interfaces for protecting authentication cookies from regular deletions. We must be careful these interfaces do not create opportunities for abuse, e.g., enabling advertisers to set "undeleteable" tracking cookies.

10 Conclusion

We have seen that web browser cookies are a good choice for persistent web authentication: they are transparent to the user and, thus, hard for the user to reveal to an attacker; the contents can be signed and client IP-bound by the server to provide integrity and non-transferability; and they are already supported by existing web server software. Solutions available at present, SSL-only cookies and IP cookies, provide, respectively, defense against phishing or pharming attacks that try to steal authentication credentials. Our extension, locked cookies, adds client-side binding of host cookies to the originating server's public key and in the process hardens cookies against more powerful active attackers. We also describe locked domain cookies, which perform a similar hardening operation for domain cookies, and allow cookies to be shared across many subdomains. Locked cookies and locked domain cookies do not require any code modifications on servers, but in some cases may require changes to the distribution and type of certificates that are used. In short, we feel that locked cookies and locked domain cookies represent a practical, secure approach to authentication and urge their adoption as a component of future authentication schemes.

11 Acknowledgements

This work was supported in part by National Science Foundation award number CCF-0424422 (Trust).

References

[1] Password rescue: A new approach to phishing prevention. In *1st USENIX Workshop on Hot Topics in Security*, 2006.

[2] Anti-phishing working group. `http://www.antiphishing.org/`.

[3] Dirk Balfanz, Glenn Durfee, and D.K. Smetters. Making the impossible easy: Usable PKI. In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, chapter 16, pages 319–334. O'Reilly, 2005.

[4] Bank of America Sitekey: Online banking security. `http://www.bankofamerica/privacy/sitekey/`.

[5] Stephen Bell. Invalid banking cert spooks only one user in 300. ComputerWorld New Zealand, http://www.computerworld.co.nz/news.nsf/NL/-FCC8B6B48B24CDF2CC257002001%8FF73, May 2005.

[6] Jesse Burns. Cross site reference forgery – An introduction to a common web application weakness. http://www.isecpartners.com/files/XSRF_Paper_0.pdf, 2005.

[7] Don Davis. Compliance Defects in Public-Key Cryptography. In *Proceedings of the 6th USENIX Security Symposium*, July 1996.

[8] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 581–590, 2006.

[9] Julie S. Downs, Mandy B. Holbrook, and Lorrie Faith Cranor. Decision strategies and susceptibility to phishing. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 79–90, July 2006.

[10] Peter Doyle and Steve Hanna. Analysis of june 2003 survey on obstables to PKI deployment and usage. http://www.oasis-open.org/committees/pki/pkiobstaclesjune2003surveyrepo%rt.pdf, August 2003.

[11] Teenager admits eBay domain hijack. http://news.com.com/Teenager+admits+eBay+domain+hijack/2100-1029_3-5355%785.html, September 2004.

[12] Carl Ellison, Chris Hall, Randy Milbert, and Bruce Schneier. Protecting secret keys with personal entropy. *Future Generation Computer Systems*, 16(4):311–318, 2000.

[13] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. http://wp.netscape.com/eng/ssl3/, 1996.

[14] Batya Friedman, David Hurley, Daniel C. Howe, Edward Felten, and Helen Nissenbaum. Users' conceptions of web security: A comparative study. In *Proceedings of the Conference on Human Factors in Computing Systems – CHI '02 extended abstracts*, pages 746–747, 2002.

[15] Batya Friedman, David Hurley, Daniel C. Howe, Helen Nissenbaum, and Edward Felten. Users' conceptions of risks and harms on the web: A comparative study. In *Proceedings of the Conference on Human Factors in Computing Systems – CHI '02 extended abstracts*, pages 614–615, 2002.

[16] Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster. Dos and Don'ts of client authentication on the web. In *10th USENIX Security Symposium*, pages 251–268, August 2001.

[17] Gartner Says Number of Phishing E-Mails Sent to U.S. Adults Nearly Doubles in Just Two Years. http://www.gartner.com/it/page.jsp?id=498245.

[18] Gartner study finds significant increase in e-mail phishing attacks. http://www.gartner.com/press_releases/asset_71087_11.html.

[19] Nathan Good, Rachna Dhamija, Jens Grossklags, David Thaw, Steven Aronowitz, Deirdre Mulligan, and Jospeh Konstan. Stopping spyware at the gate: A user study of notice, privacy and spyware. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 43–52, July 2005.

[20] Princeton Secure Internet Programming Group. Dns attack scenario. http://www.cs.princeton.edu/sip/news/dns-scenario.html, February 1996.

[21] Peter Gutmann. Plug-and-Play PKI: A PKI your Mother can Use. In *Proceedings of the* 11^{th} *USENIX Security Symposium*, pages 45–58, August 2003.

[22] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile. http://tools.ietf.org/html/rfc3280, 2002.

[23] Yao-Wen Huang, Fang Yu, Christian Hang, Chung-Hung Tsai, D. T. Lee, and Sy-Yen Kuo. Securing web application code by static analysis and runtime protection. In *Proceedings of 13th international conference on World Wide Web (WWW'06)*, pages 40–52, 2006.

[24] Hushmail DNS incident. https://www.hushmail.com/login-news_dns?PHPSESSID=21af0aea8094713f5fa08%58b9e61148a&, April 2004.

[25] ING direct privacy center. https://home.ingdirect.com/privacy/privacy_security.asp?s=newsecurityfe%ature.

[26] Tom Jagatic, Nathaniel Johnson, Markus Jakobsson, and Filippo Menczer. Social phishing. *To appear in the Communications of the ACM*, 2006.

[27] Martin Johns. On XSRF and Why You Should Care. talk at the PacSec 2006 conference, http://www.informatik.uni-hamburg.de/SVS/personnel/martin/psj06johns-e.%pdf, November 2006.

[28] Martin Johns. (Somewhat) breaking the same-origin policy by undermining DNS pinning. http://shampoo.antville.org/stories/1451301/, August 2006.

[29] Martin Johns and Justus Winter. Requestrodeo: Client side protection against session riding. In Frank Piessens, editor, *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, pages 5 – 17. Departement Computerwetenschappen, Katholieke Universiteit Leuven, May 2006.

[30] Nenad Jovanovic, Engin Kirda, and Christopher Kruegel. Preventing cross site request forgery attacks. In *Proceedings of the Second IEEE Conference on Security and Privacy in Communications Networks (SecureComm)*, August 2006.

[31] Ari Juels, Markus Jakobsson, and Sid Stamm. Active cookies for browser authentication. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS '07)*, 2007.

[32] V. Benjamin Livshits and Monica S. Lam. Finding security vulnerabilities in java applications using static analysis. In *Proceedings of the 14th USENIX Security Symposium*, pages 271–286, August 2005.

[33] Scott Loftesness. Responding to phishing attacks. http://www.glenbrook.com/opinions/phishing.htm.

[34] Mitigating cross-site scripting with HTTP-only cookies. http://msdn.microsoft.com/workshop/author/dhtml/httponly_cookies.asp.

[35] Lynette Millett, Batya Friedman, and Edward Felten. Cookies and web browser design: Toward realizing informed consent online. In *Proceedings of the CHI 2001 Conference on Human Factors in Computing Systems*, pages 46–52, April 2001.

[36] Mozilla Bugzilla bug 149943 – Princeton-like exploit may be possible. `https://bugzilla.mozilla.org/show_bug.cgi?id=149943`.

[37] Mozilla Bugzilla bug 162871 – DNS: problems with new DNS cache ("pinning" forever). `https://bugzilla.mozilla.org/show_bug.cgi?id=162871`.

[38] Mozilla Bugzilla bug 205726 – nsDnsService rewrite. `https://bugzilla.mozilla.org/show_bug.cgi?id=205726`.

[39] Mozilla Bugzilla bug 22183 - UI spoofing can cause user to mistake content for chrome (bug reported 12/20/1999, publicly reported 7/21/2004). `https://bugzilla.mozilla.org/show_bug.cgi?id=22183`.

[40] Gunter Ollmann. The pharming guide. `http://www.ngssoftware.com/papers/ThePharmingGuide.pdf`.

[41] Gavin O'Malley. Jupiter analyst: Nielsen research confirms users delete cookies. `http://publications.mediapost.com/index.cfm?fuseaction=Articles.san&s=28883&Nid=12855&p=297686`.

[42] Panix recovers from domain hijack. `http://www.theregister.co.uk/2005/01/17/panix_domain_hijack/`, January 2005.

[43] Bryan Parno, Cynthia Kuo, and Adrian Perrig. Phoolproof phishing prevention. In *Proceedings of Financial Cryptography (FC'06)*, February 2006.

[44] Persistent client state: HTTP cookies, Preliminary specification. `http://wp.netscape.com/newsref/std/cookie_spec.html`.

[45] Honeynet Project and Research Alliance. Know your enemy: Phishing. `http://www.honeynet.org/papers/phishing/`.

[46] Venugopalan Ramasubramanian and Emin Gun Sirer. Perils of transitive trust in the Domain Name System. In *Proceedings of the Internet Measurement Conference (IMC)*, October 2005.

[47] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[48] SANS March 2005 DNS Poisoning Summary. `http://isc.sans.org/presentations/dnspoisoning.php`.

[49] M. Angela Sasse and Ivan Flechais. Usable Security: Why do we need it? How do we get it? In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, chapter 2, pages 13–30. O'Reilly, 2005.

[50] Bruce Schneier. Two-factor authentication: too little, too late. *Communications of the ACM*, 48(4):136, April 2005.

[51] Thomas Schreiber. Session Riding – A Widespread Vulnerability in Web Applications. http://www.securenet.de/papers/Session_Riding.pdf, December 2004.

[52] Secure Server Survey by Security Space and E-Soft. http://www.securityspace.com/s_survey/sdata/200608/certca.html, September 2006.

[53] Secure Server Survey by Security Space and E-Soft. http://www.securityspace.com/s_survey/sdata/200605/domain.html, June 2006.

[54] Symantec gateway security products – DNS cache poisoning vulnerability. http://securityresponse.symantec.com/avcenter/security/Content/2004.06.%21.html, June 2004.

[55] Win Treese and Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. http://tools.ietf.org/html/rfc4346, 2006.

[56] Alex Tsow. Phishing with consumer electronics – malicious home routers. In *Models of Trust for the Web Workshop at the 15th International World Wide Web Conference (WWW2006)*, May 2006.

[57] Alex Tsow, Markus Jakobsson, Liu Yang, and Susanne Wetzel. Warkitting: the drive-by subversion of wireless home routers. *Journal of Digital Forensic Practice*, 1(3), November 2006.

[58] Vanguard security center. https://flagship.vanguard.com/VGApp/hnw/content/UtilityBar/SiteHelp/Sit%eHelp/SecurityCenterOverviewContent.jsp.

[59] Verisign. Knowledge base: Can an IP address be used as a common name? http://tinyurl.com/yapbk5.

[60] Alma Whitten and J.D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Proceedings of the 8th USENIX Security Symposium*, pages 169–184, August 1999.

[61] Min Wu, Robert C. Miller, and Simson Garfinkel. Do security toolbars actually prevent phishing attacks? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 601–610, 2006.

[62] Min Wu, Robert C. Miller, and Greg Little. Web wallet: Preventing phishing attacks by revealing user intentions. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 102–113, July 2006.

[63] Yichen Xie and Alex Aiken. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th USENIX Security Symposium*, pages 179–192, August 2006.

[64] Wei Xu, Sandeep Bhatkar, and R. Sekar. Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In *Proceedings of the 15th USENIX Security Symposium*, pages 121–136, August 2006.

[65] Yahoo sign-in seal. http://security.yahoo.com/.

[66] Ka-Ping Yee. Guidelines and strategies for secure interaction design. In Lorrie Faith Cranor and Simson Garfinkel, editors, *Security and Usability: Designing Secure Systems That People Can Use*, chapter 13, pages 247–273. O'Reilly, 2005.

[67] Jim Youll. Fraud vulnerabilities in SiteKey security at Bank of America. cr-labs.com/publications/SiteKey-20060718.pdf, July 2006.