In Proceedings of the Fourteenth ACM Conference on Computer and Communications Security , November 2007, pp. 58-71

Dynamic Pharming Attacks and Locked Same-origin Policies for Web Browsers

Chris Karlof, J.D. Tygar, David Wagner
UC Berkeley
Computer Science
Berkeley, CA, USA
{ckarlof,tygar,daw}@cs.berkeley.edu

Umesh Shankar
Google, Inc.
New York, NY, USA
ushankar@google.com

ABSTRACT

We describe a new attack against web authentication, which we call *dynamic pharming*. Dynamic pharming works by hijacking DNS and sending the victim's browser malicious Javascript, which then exploits DNS rebinding vulnerabilities and the name-based same-origin policy to hijack a legitimate session after authentication has taken place. As a result, the attack works regardless of the authentication scheme used. Dynamic pharming enables the adversary to eavesdrop on sensitive content, forge transactions, sniff secondary passwords, etc. To counter dynamic pharming attacks, we propose two *locked same-origin policies* for web browsers. In contrast to the legacy same-origin policy, which regulates cross-object access control in browsers using domain names, the locked same-origin policies enforce access using servers' X.509 certificates and public keys. We show how our policies help two existing web authentication mechanisms, client-side SSL and SSL-only cookies, resist both pharming and stronger active attacks. Also, we present a deployability analysis of our policies based on a study of 14651 SSL domains. Our results suggest one of our policies can be deployed today and interoperate seamlessly with the vast majority of legacy web servers. For our other policy, we present a simple incrementally deployable opt-in mechanism for legacy servers using policy files, and show how web sites can use policy files to support self-signed and untrusted certificates, shared subdomain objects, and key updates.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General – Security and protection; H.4.3 [**Information Systems Applications**]: Communication Applications – Information Browsers

General Terms

Security

Keywords

Pharming, Web Authentication, Same-origin Policy

1. INTRODUCTION

Phishing is a social engineering attack in which an adversary lures an unsuspecting Internet user to a web site posing as a trustworthy business with which the user has a relationship [3]. 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. In a more advanced phishing attack known as *pharming* [53], the adversary subverts the domain-name lookup system (DNS), which is used to resolve domain names to IP addresses. In this 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.

Although pharming attacks have been relatively rare in practice, evidence suggests they may become a more serious threat in the near future. Recent research has exposed complex and subtle dependencies between names and name servers [59], suggesting the DNS infrastructure is more vulnerable to DNS poisoning attacks than previously thought. The ubiquity of public wireless access points and wireless home routers introduces new pharming threats. Users are becoming accustomed to accessing wireless routers in airports, restaurants, conferences, libraries, and other public spaces. Adversaries can set up malicious wireless routers in these areas that offer free Internet access but redirect users to spoofed web sites [2]. Also, many users leave the default password and security settings on their home wireless home routers unchanged [66]. This enables *warkitting* attacks [72, 73], a combination of wardriving and rootkitting, where an adversary maliciously alters a router's configuration over a wireless connection. A related attack is *drive-by pharming* [70], where a malicious web site serves content which scans a visitor's internal network and compromises home routers with default passwords. After the adversary has compromised the victim's router, she can change the DNS settings or overwrite the firmware to redirect the victim's requests.

We describe a new type of DNS attack against web authentication we call *dynamic pharming*. In a dynamic pharming attack, the adversary initially delivers a web document containing malicious Javascript code to the victim, and then exploits *DNS rebinding* vulnerabilities in browsers to force the victim's browser to connect to the legitimate server in a separate window or frame. The adversary waits for the victim to authenticate herself to the legitimate server, and then uses the malicious Javascript to hijack the victim's authenticated session.

Policy	Information used to enforce access	Strongest threat model protected against for: Locked web objects	Shared locked web objects	Untrusted certs
Legacy SOP	(protocol,domain,port)	phishers	phishers	phishers
Weak locked SOP	(protocol,domain,port, validity of cert chain)	active attackers	phishers	phishers
Strong locked SOP (w/ policy files)	(protocol,domain,port, server public key)	active attackers	active attackers	active attackers

Table 1: Comparison of our locked same-origin policies with the legacy same-origin policy. This table shows the strongest threat model under which each policy can isolate a legitimate server's web objects (e.g., cookies, HTML documents, etc.) from adversaries. *Locked web objects* refer to objects retrieved over SSL. *Shared locked web objects* refer to objects retrieved over SSL which are intended to be shared among subdomains of a higher-level domain (e.g., domain cookies). *Untrusted certs* refer to a legitimate server using a self-signed certificate or a certificate issued by a root CA untrusted by browsers.

Dynamic pharming takes advantage of how browsers currently implement the *same-origin policy*. The same-origin policy prohibits a web object from one site from accessing web objects served from a different site. Browsers currently enforce this by checking that the two objects' originating domain names, ports, and protocols match. However, when an adversary controls the domain name mapping, the legacy same-origin policy does not provide strong isolation between web objects co-executing in a user's browser. In a dynamic pharming attack, malicious Javascript from the pharmer and content from the legitimate server both appear to have the same "origin" (i.e., same domain, port, and protocol), and the browser allows the Javascript to access to the user's authenticated session. As a result, the attacker can gain complete control of the session, enabling her to eavesdrop on sensitive content, forge transactions, sniff secondary passwords, etc. Since dynamic pharming hijacks users' sessions after authentication completes, irrespective of the authentication mechanism, it can be used to compromise even the strongest web authentication schemes currently known, including passwords, authentication cookies, and client-side SSL. We present dynamic pharming in more detail in Section 3.

1.1 The locked same-origin policies

Since dynamic pharming hijacks a user's session after initial authentication completes, it is unlikely any future web authentication protocol developed for currently deployed browsers will resist dynamic pharming either. To resist dynamic pharming, we must address the root of the problem: we must upgrade the browser's same-origin policy. We propose two *locked same-origin policies*: instead of comparing domain names to enforce access control, our policies enforce access control for web objects retrieved over SSL by using servers' public keys and X.509 certificates. We refer to web objects retrieved over SSL as *locked web objects* because the browser can clearly associate the public key and X.509 certificate of server hosting the object with the object. Our first proposal, the *weak locked same-origin policy*, isolates a domain's locked web objects with valid certificate chains from objects with invalid chains. This enables browsers to distinguish a legitimate server using a valid certificate from pharmers using invalid certificates, such as self-signed certificates or certificates with CN/domain mismatches. Our second proposal, the *strong locked same-origin policy*, enforces access control using cryptographic identity, namely web sites' public SSL keys. In the strong locked same-origin policy, the browser compares the public keys it associates with locked web objects; access in granted only if they match. We present our locked same-origin policies in more detail in Section 4 and show a comparison with the legacy same-origin policy in Table 1.

We show how our policies substantially increase browsers' resistance to pharming attacks and provide both a solid and necessary foundation for developing pharming resistant authentication. We show how our policies help two existing web authentication mechanisms, client-side SSL and SSL-only cookies, resist pharming and stronger attacks. In addition, we evaluate our policies in terms of deployability, meaning how well they interoperate with existing web servers. Based on the results of a study of 14651 SSL domains, we found strong evidence that the weak locked same-origin policy can replace the legacy same-origin policy today with minimal risk of breaking existing web sites (Section 4.4).

Although we did not find similar evidence for the strong locked same-origin policy, we propose a simple, incrementally deployable and backwards compatible mechanism for web sites to opt-in using policy files (Section 4.5). To opt-in, we propose a web site posts a policy file at a static well-known file name, say `pk.txt`, which enables a web site to specify how it would like the browser to enforce the strong locked same-origin policy. Policy files also support flexible server configurations and key updates. In contrast to the weak locked same-origin policy, the strong locked same-origin policy has better security properties, is compatible with sites using self-signed or untrusted certificates, and supports subdomain object sharing.

The locked same-origin policies are similar to work done independently and concurrently by Masone et al. on Web Server Key Enabled Cookies, a new browser policy for protecting authentication cookies against pharming attacks [41]. However, their proposal falls short of protecting cookies against dynamic pharming attacks. Also, they do not address pharming attacks against other web objects or other web authentication mechanisms, e.g., client-side SSL, nor do they address subdomain object sharing or key updates.

2. PRELIMINARIES

2.1 Threat models

We consider three broad classes of adversaries, classified according to their capabilities.

Phishers. We assume a phisher has the following capabilities:

- Complete control of a web server with a public IP address. We assume a phisher uses a different domain name from the target domain.
- The ability to send communications such as emails and instant messages to potential victims.
- Mount 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.

There have been relatively few documented cases of application-layer man-in-the-middle attacks [21, 56, 57], most likely because of the extra effort required to implement the attack. However, researchers have recently discovered a "Universal Man-in-the-Middle Phishing Kit" [40], a hacker toolkit which enables a phisher to easily set up a MITM proxy attack against any site she wishes.

Pharmers. 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, modifying a user's `/etc/hosts` file, tricking a user to modify her DNS settings, or by social engineering attacks against a domain name registry. We assume the server under the pharmer's control does not have the same IP address as the victim and cannot receive packets destined to the victim's IP address.

Active attackers. An active attacker has all the abilities of a pharmer, plus

- The ability to control the Internet routing infrastructure and re-route traffic destined to particular IP addresses.
- Eavesdrop on all traffic.
- Mount active, network-layer, man-in-the-middle attacks.

To date, phishers have been by far the most prevalent class of attacker; however, looking to the future, pharmers and active attackers are a growing threat [2, 59, 70, 72, 73], and it seems prudent to defend against these more powerful attackers as well, to the extent possible.

2.2 The legacy same-origin policy

The same-origin policy (SOP) 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 [51]. By web objects, we mean HTTP cookies, HTML documents, images, Javascript, CSS files, XML files, etc. A common example of "access" is Javascript referencing another object. There are a few situations our work does not address, and we discuss these in Section 4.10. In the remainder of this paper, we will use "SOP" as an abbreviation for "same-origin policy".

Browsers currently consider two objects to have the same origin if the originating host, port, and protocol are the same for both web objects. For example, Javascript executing on `http://www.foo.com/index.html` is allowed to access `http://www.foo.com/other.html`, but is not allowed to access `https://www.foo.com/secure.html` (different protocol) or `http://www.xyz.com/index.html` (different host). Other examples of "web object accesses" subject to the SOP include determining which cookies to append to an HTTP request, Javascript `document.cookie` references, and XMLHTTPRequest.

Note there is a distinction between "access" and "causing to load". After the browser receives an HTML page, it processes dependent requests necessary to render the page, such as images, style sheets, etc. These requests can cause the browser to fetch and load a web object from a different domain. However, these requests are not considered violations of the SOP. The document can read certain metaproperties of the object (e.g., height, width), but the SOP prevents the document from reading or modifying the content of the loaded object.

2.3 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 Internet traffic [13, 71]. HTTP over SSL is also known as HTTPS.

SSL uses X.509 certificates [26] 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 typically a certificate authority (CA); web browsers ship with the certificates of some CAs which are deemed to be trusted.

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 a high level the browser must:

- Verify that every certificate in the chain has a valid signature from its predecessor, using the public key of the predecessor, and that the last certificate in the chain is from a trusted CA.
- Verify that the `CN` field of the first certificate in the chain matches the domain name of the web site the browser intended to visit.
- Verify every certificate has not expired.

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. The reason is to ensure compatibility with misconfigured certificates and SSL servers; a periodic survey by Security Space shows that approximately 63% of SSL certificates have such problems [65]. 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, 10, 77]. In fact, users have become so ambivalent to security warnings, one vendor has developed "mouse auto-clicker" software, aptly titled "Press the Freakin Button" [58], to automatically click through dialogs like these. Instead of a dialog box, IE 7.0 uses a full page warning within the browser window offering similar options (i.e., ignore and continue, or cancel connection). Unfortunately, studies suggest that users will ignore a full page warning as well [64]. Accordingly, we consider a certificate that does not generate any warnings as *valid*. Otherwise, we consider it *invalid*.

After the browser validates the server's certificate, it participates in a cryptographic protocol with the server where: 1) the server proves knowledge of the private key corresponding to the public key in the certificate, and 2) they negotiate a session key to encrypt and authenticate subsequent traffic between them. Unlike the certificate validation step, if there are any errors in this protocol, the browser closes the connection with no chance of user override.

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.

2.4 Assumptions

We assume that attackers do not have access to the target site's server machines or any secrets, such as private keys, contained thereon. We also assume that many users will ignore certificate warnings, as researchers have shown that users routinely ignore and dismiss such warnings [5, 10, 64, 77].

3. DYNAMIC PHARMING ATTACKS

In this section, we show a new attack against web authentication we call *dynamic pharming*. In a simple, static pharming attack, the adversary arranges for the victim's DNS queries for the target domain to always return the adversary's IP address. In contrast, in a dynamic pharming attack, the adversary causes DNS queries to return either the legitimate server's IP or its own IP, depending on the situation.

We show how an adversary can use dynamic pharming to infect the victim's browser with malicious Javascript and use this Javascript to hijack the victim's session with the target domain's legitimate server. Dynamic pharming enables an adversary to compromise all known authentication schemes for existing browsers, including passwords, authentication cookies, and client-side SSL. In addition, the adversary can eavesdrop on sensitive content, forge transactions, sniff secondary passwords, and so on.

We now describe how dynamic pharming works. Suppose the pharmer can control the results of DNS queries for `www.vanguard.com`, and users authenticate themselves to `www.vanguard.com` using a strong authentication mechanism, say client-side SSL. We assume users' machines have been initialized with client-side SSL certificates, and `www.vanguard.com` knows the public keys of its users' certificates.

First, the pharmer initializes the DNS entry for `www.vanguard.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. Now suppose a user Alice visits `https://www.vanguard.com/index.html` with the intention of authenticating herself. 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 trusted CAs in the browser or the certificate's `CN` does not match the server's domain (i.e., `www.vanguard.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 attack fails. If the user accepts the pharmer's certificate—and there is substantial evidence that the user would (see Section 2.3)—Alice's browser will establish an SSL connection to the pharmer at 6.6.6.6 and request `index.html`.

In response, the pharmer returns a "trojan" `index.html` document. The purpose of this trojan document is to monitor and influence Alice's subsequent interactions with the legitimate `www.vanguard.com`. The trojan document has the following general structure:

```
<html>
<body>
<script>
  ---MALICIOUS JAVASCRIPT CODE---
</script>
<iframe
src="https://www.vanguard.com/index.html">
</iframe>
</body>
</html>
```

After the pharmer returns the trojan document to Alice, it updates the DNS entry for `www.vanguard.com` to the IP address of the legitimate server for `www.vanguard.com`, say 1.2.3.4. This forces the browser to load the legitimate `https://www.vanguard.com/index.html` document into the `<iframe>` and display it to the user. [1] Since this request is over SSL, the legitimate server for `www.vanguard.com` will request client authentication, and the user's browser will authenticate her using her private key and certificate.[2]

After authentication completes, the malicious Javascript in the outer document takes control and monitors the user's interactions in the `<iframe>` with the legitimate server for `www.vanguard.com`. Since the outer document and the `<iframe>` both have the same domain (`www.vanguard.com`) and same protocol (https), the SOP will allow the malicious Javascript running in the outer 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, sniff secondary passwords, etc. We show an example of a dynamic pharming attack in Figure 1.

3.1 Defeating DNS pinning

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 a fixed period of time, regardless of the DNS entry's specified lifetime. Browsers implement DNS pinning to defend against variants of the "Princeton attack" [22], also known as *DNS rebinding* attacks. 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. The SOP 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 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 continue to use the IP address 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.

As a result, Martin Johns discovered a technique for circumventing DNS pinning [32]. 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 in response to subsequent attempts to connect to the server. The browser reacts by refreshing its DNS entry for the domain.

In the basic dynamic pharming attack, we exploit Johns's technique. After the pharmer delivers the trojan document to the user,

[1]Iframes are HTML elements which enable embedded documents. To prevent infinite recursion, most browsers disallow nesting where the URL of the framed document is the same as an ancestor. To address this issue, the attack could redirect the victim's first request to `https://www.vanguard.com/index2.html` or arrange so that the legitimate home page from `www.vanguard.com` loads in a separate window.

[2]The authentication process could be more complicated, say with a supplementary password or explicit login button, but the presence of any additional login mechanisms does not affect our attack.

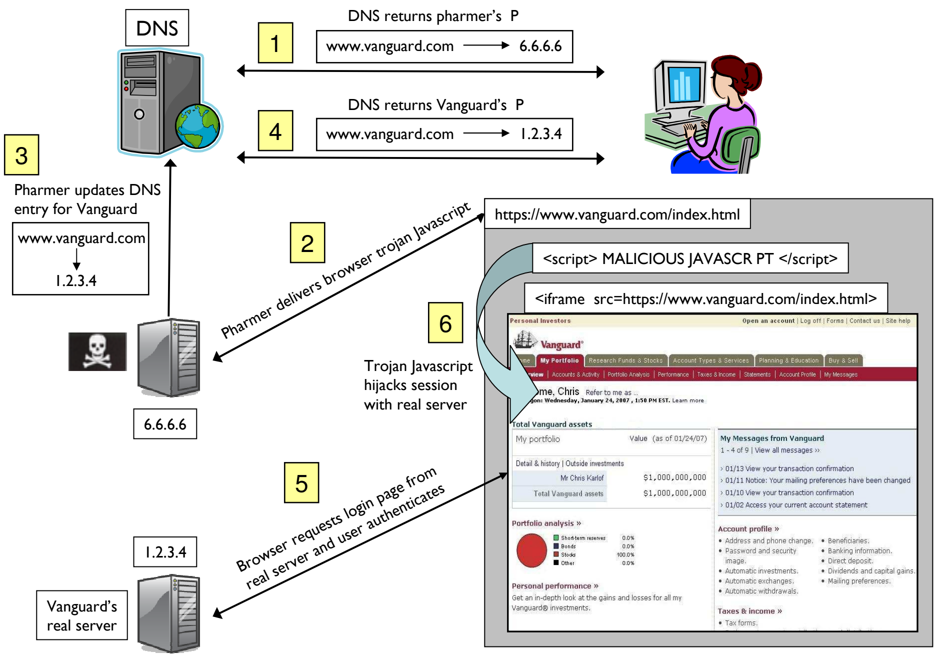
DNS
1
DNS returns pharmer's P
www.vanguard.com → 6.6.6.6
4
DNS returns Vanguard's P
www.vanguard.com → 1.2.3.4
3
Pharmer updates DNS entry for Vanguard
www.vanguard.com → 1.2.3.4
2
Pharmer delivers browser trojan Javascript
https://www.vanguard.com/index.html
<script> MALICIOUS JAVASCR PT </script>
<iframe src=https://www.vanguard.com/index.html>
6
Trojan Javascript hijacks session with real server
6.6.6.6
5
Browser requests login page from real server and user authenticates
1.2.3.4
Vanguard's real server

Figure 1: An example of a dynamic pharming attack against www.vanguard.com. (1) Initially, the pharmer arranges for the victim's DNS queries for www.vanguard.com to resolve to the pharmer's IP address, 6.6.6.6. (2) Then, when the victim visits www.vanguard.com, the pharmer returns a trojan document containing malicious Javascript and a iframe referencing Vanguard's home page. (3) The pharmer then updates the DNS entry for www.vanguard.com to the IP address of Vanguard's legitimate server and denies subsequent connections from the victim. (4) This causes the victim's browser to renew its DNS entry for www.vanguard.com, and (5) load Vanguard's legitimate home page in the iframe. (6) After the user authenticates herself, the malicious Javascript in the trojan document hijacks her session with the legitimate server.

it rejects subsequent requests from user's machine and updates the DNS entry for `www.vanguard.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 continues as before.

3.2 Using round robin DNS

To parallelize dynamic pharming attacks against multiple concurrent users, it is inefficient to repeatedly update the DNS entry for `www.vanguard.com`. If the adversary has compromised a local, root, or authoritative DNS server, or changed the authoritative server of record for `www.vanguard.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 `www.vanguard.com` on a DNS server (e.g., by cache poisoning), 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.

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. The DNS server returns an ordered list of the IP addresses in response to a query, but rotates the order for each response. Web sites typically use round robin DNS to implement load balancing or automatic failover. 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 the pharmer will deliver the trojan document. 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. For the other half of responses, the user will be delivered directly to the legitimate server and the pharming attack will silently fail. This shows how an attacker with the ability to replace a single record, once, (e.g., by cache poisoning) can still attack thousands or millions of users.

3.3 Discussion

Dynamic pharming attacks do not leverage vulnerabilities in any particular authentication mechanism; rather, they exploit how browsers currently enforce the SOP. Since dynamic pharming hijacks the victim's session after she authenticates herself to the legitimate server, this attack most likely affects all known authentication mechanisms for current browsers, and probably all future ones as well.

In some cases, pharming attacks can also steal users' authentication credentials, e.g., passwords and authentication cookies. Since the users' URL bar will show the correct domain name, even the most meticulous user might be fooled into revealing her password. Also, since browsers enforce the SOP based on domain names, pharmers can steal user's authentication cookies for the target site. Although dynamic pharming attacks against client-side SSL authentication do not enable pharmers to steal users' authentication credentials (i.e., their private keys), as we have seen, they can compromise users' sessions in real time.

Some web sites use Javascript to detect and prevent framing, e.g.,

```
if (parent.frames.length > 0)
   top.location.replace(document.location);
```

However, Javascript anti-framing techniques are not sufficient to resist dynamic pharming. Our attack does not depend on the use of iframes to be successful. For instance, the attacker could load the legitimate `index.html` in another tab or window. The SOP still allows malicious Javascript access to the second window, and this situation is much harder for the legitimate site to detect.

3.4 Proof of concept implementation

We implemented a proof of concept dynamic pharming attack using a pair of Apache SSL web servers (i.e, a pharmer and a target) and round robin DNS. We tested the attack against two browsers: Firefox 2.0 running on Debian GNU/Linux 3.1 and Microsoft Internet Explorer 7.0 running on Windows XP Professional SP2. After the adversary delivers the trojan document, she refuses further connections from the client. This causes the browser to renew its DNS entry for the target domain and connect to the legitimate server, after which the adversary hijacks the session with the malicious Javascript in the trojan document. We found both browsers to be vulnerable to this dynamic pharming attack.

4. THE LOCKED SAME-ORIGIN POLICY

Since dynamic pharming hijacks a user's session after initial authentication completes, this attack is independent of the authentication mechanism and affects all known authentication schemes for current browsers, including passwords, authentication cookies, and client-side SSL. It is therefore unlikely that any future web authentication protocol developed for existing browsers will resist dynamic pharming either. Although dynamic pharming attacks leverage the implementation details of DNS pinning, "fixing" DNS pinning to resist DNS rebinding attacks is challenging. DNS pinning has a lengthy and controversial history in Firefox and Mozilla [47], and the current implementation is an explicit compromise to support dynamic DNS and round robin DNS for failover [46, 48]. From the browser's point of view, a dynamic pharming attack is indistinguishable from a failure of a site and DNS round robin recovery. Lastly, it is unlikely web sites can resist dynamic pharming attacks effectively. The adversary has the advantage of loading her document first; she can read and modify all of the legitimate server's documents in the victim's browser, as well as control their execution environment.

To resist dynamic pharming, we must address the root of the problem: we must upgrade browsers' SOP. A SOP based on domain names will fail because pharmers control the mapping from domain name to subject. For web objects retrieved over insecure HTTP, it is unclear how the browser can distinguish a pharmer from the legitimate server. However, for objects retrieved over SSL, we argue browsers should enforce the SOP using cryptographic identity. We refer to web objects retrieved over SSL as *locked web objects*, and we propose two *locked same-origin policies* to resist dynamic pharming attacks against them.

We first present the *weak locked same-origin policy*, which isolates a domain's locked web objects with valid certificate chains from objects with invalid chains. We then present the *strong locked same-origin policy*, which is based on cryptographic identity, namely web sites' SSL public keys.

Both policies only apply new restrictions to locked web objects. For non-SSL web objects, the legacy SOP (namely, using domain names) still applies. Like the legacy SOP, both locked SOPs deny unlocked web objects (that is, objects not retrieved over SSL) access to locked web objects. We summarize our policies in comparison to the legacy SOP in Table 1.

4.1 The weak locked same-origin policy

The legacy SOP currently allows access to locked web objects only from other locked web objects originating from the same domain.[3] However, the legacy SOP does not distinguish between locked web objects retrieved from a legitimate server and those from a pharmer spoofing the server's domain name, and will allow access if the user ignores any certificate warnings. To resist pharming attacks, the weak locked SOP augments the legacy SOP by tagging each locked web object with a validity bit indicating whether the certificate chain corresponding to the SSL connection over which the object was retrieved contained any errors (e.g., self-signed certificate, CN/domain mismatch), irrespective of how the user responded to any certificate warnings. Then, the browser allows a locked web object to access another locked web object if and only if 1) the legacy SOP would allow access and 2) the validity bits match.

4.2 The strong locked same-origin policy

With the strong locked SOP, we propose browsers augment the legacy SOP by tagging each locked web object with the public key of the other endpoint of the SSL connection (i.e., the web server). Then, the browser allows a locked web object to access another locked web object if and only if 1) the legacy SOP would allow access and 2) the associated public keys match. The strong locked SOP was inspired by Key Continuity Management [19, 23, 60, 84], a technique for associating public keys with subjects and taking defensive action when a subject's public key unexpectedly changes in a future interaction.

4.3 Security analysis

Weak locked same-origin policy. If a web server hosting domain D (i.e., the target domain) uses a valid X.509 certificate signed by a trusted root CA, the weak locked SOP resists phishing, pharming, and active attacks against D's locked web objects (i.e., illegitimate access by the adversary's web objects) as long as the adversary is unable to obtain a valid certificate for D. The weak locked SOP resists phishing attacks because a phisher has a different domain name. For pharming and active attacks, the adversary can arrange for her web objects to have the same name as the target domain, but if she does not have a valid certificate for the target domain, the validity bit will be `false`, while the validity bit of the web server's locked objects will be `true`. Thus, the adversary is denied access.

If the legitimate target site uses an invalid X.509 certificate (e.g., expired, CN/domain mismatch, or self-signed), the weak locked SOP provides no additional protection over the legacy SOP. It resists phishing attacks, but does not protect against pharmers or active attackers.

In contrast to the legacy SOP, the weak locked SOP does not depend on users correctly answering prompts in response to certificate errors (e.g., if an adversary presents a self-signed certificate with a spoofed domain name). The browser tags locked web objects according to the validity of the server's certificate and its domain name, and nothing else. However, the weak locked SOP does as-

[3]Exception: if a web site sets a non SSL-only cookie (i.e., without the `secure` attribute) over an SSL connection, then this policy allows the same domain to access the cookie over non-SSL connections as well. Essentially, a non SSL-only cookie set over an SSL connection gets downgraded to an unlocked web object.

sume that the trusted root CAs do not issue valid certificates for D to unauthorized parties. Although CAs take measures to prevent this, mistakes have been made in the past [45].

Strong locked same-origin policy. If a web server hosting domain D uses an X.509 certificate with public key PK, the strong locked SOP resists phishing, pharming, and active attacks against D's locked web objects as long as the adversary does not know the corresponding private key for PK. As with the weak locked SOP, the strong locked SOP resists phishing attacks because a phisher has a different domain name. In order to access D's locked web objects, the adversary must pharm D and also arrange for its own objects to be tagged with PK. However, the browser will only do this if 1) the adversary presents a X.509 certificate with PK, and 2) the browser and adversary can successfully establish an SSL connection. If the adversary tries to present a certificate for PK and she does not know the private key corresponding to PK, she will not be able to successfully complete the SSL handshake; the browser will automatically cancel the connection with no option of user override. Thus, since the browser will only tag the adversary's locked web objects with a public key different from PK, the browser will deny the adversary access to D's locked web objects. For the same reason, the strong locked SOP protects D's locked web objects against active attackers as well.

As with the weak locked SOP, the strong locked SOP does not depend on users correctly answering prompts in response to certificate errors. Furthermore, in contrast to the weak locked SOP, the strong locked SOP does not require a web site to trust root CAs not to issue certificates to unauthorized parties for its domain. Enforcement relies only on servers' public keys.

4.4 Deployability analysis

If our locked same-origin policies are to be successful, they need to be easy to deploy and backwards compatible; they should not "break the web" because of problems with deployment or interoperability with existing web servers. Since no browser developer is likely to embrace a policy that makes her browser incompatible with existing web sites, legacy web servers had better continue to work even when visited with locked SOP enabled browsers.

Our policies are more restrictive than the legacy SOP, but we only want to deny access to an attacker – never the legitimate server. We will "break" a web site if there is a situation where our policy would deny a legitimate server access to one of its locked web objects, but the legacy SOP would allow access.

There are a few situations where our policies could potentially break a web site. For example, suppose server A for `xyz.com` has an valid certificate, but server B for `xyz.com` has an invalid certificate (or vice versa). Then the weak locked SOP would deny Javascript from server A from accessing an HTML document from server B, but the legacy SOP would allow access. This situation might arise if `xyz.com` uses round robin DNS for load balancing and a browser request objects from both servers during a session. Note that the weak locked SOP will not break a domain which uses invalid certificates on all its servers (e.g., it uses self-signed certificates) since objects from these servers have equivalent validity: they are all invalid.

If server A and server B use different public keys, then the strong locked SOP would also deny access. However, the strong locked SOP does not necessarily require the domain to use certificates issued by a root CA trusted by browsers. As long as all servers use the same public key, the web site can use certificates issued by a root CA untrusted by browsers or self-signed certificates.

Browser	Session caching of CA certs	Persistent caching of CA certs	Uses AIA
IE	✓	✓	✓
Firefox	✓		

Table 2: Summary of browser mechanisms used to address missing and expired intermediate CA certificates. AIA refers to the optional Authority Information Access X.509 extension.

4.4.1 An SSL server survey

To evaluate the deployability of our policies, we must determine how many sites we could potentially break; in other words, how often the above configurations actually arise in practice. To measure this, we surveyed SSL servers in the real world to determine how many servers may not currently interoperate with our policies. We constructed a sample of SSL servers by first crawling the web, starting from a list of major news, portal, and financial sites. Whenever we found an HTTPS link, we added the domain in the link to our sample. 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 14651 fully qualified SSL domains from 6192 second-level domains.[4] This corresponds to roughly 6.5% of the number of SSL domains found by the more extensive monthly SSL survey conducted by E-Soft and `securityspace.com` [65].

We are primarily interested in finding domains hosted by multiple servers, since it is these domains that our policies could potentially break. We can discover some servers by looking for use of round robin DNS; if a server uses round robin DNS, a DNS query returns a list of IP address. However, Akamai-style load balancing often considers the physical location of the DNS querier and may only return the IP address of most appropriate server. To take Akamai-style load balancing into account, we constructed a list of servers for each domain by requesting recursive DNS queries to 15 geographically distributed public DNS servers [76].[5] Of the 14651 domain names, we found 1464 that resolved to multiple IP addresses. For each of these domains, we established an SSL connection to each of the domain's servers and recorded each server's certificate chain and public key.

4.4.2 Certificate chain validation: Firefox and IE

The next step was to validate the certificates we collected. To maximize the practical relevance of our study, we simulated the validation procedures of Firefox 2.0 and Internet Explorer 7.0. The validation procedures of Firefox and IE are close to the process we described in Section 2.3, but there are some differences in how each browser handles missing and expired intermediate CA certificates. Intermediate CA certificates are certificates issued by a CA's root certificate which it uses to directly issue certificates to web servers. This results in certificate chains of length 3 or more. Since most browsers only ship with root CA certificates, to guarantee a client can verify its chain, a server must also send any intermediate CA certificates in addition to its own.

Unfortunately, many servers are not configured to send intermediate CA certificates. Also, there are several widely used intermediate CA certificates which have expired, and although the CA has reissued a replacement with the same name (and often, the same

[4]For our study, a second-level domain means the last two components of a non-international fully qualified domain name, e.g., `yahoo.com`.

[5]A limitation of this approach is that we cannot discover multiple servers behind a front-end load balancer with a single IP address.

public key), many servers have not updated them and are still sending the expired version. We found that Firefox and Internet Explorer handle these situations slightly differently. We determined each browser's validation procedure through source code analysis, empirical testing, and various public sources [20, 49, 50].

First, both Firefox and Internet Explorer cache the intermediate CA certificates they encounter during a user's browsing session and use this cache to help verify certificate chains. This means if the user visits a site with a missing intermediate CA certificate, and previously in the session, the user visited a different site using the same intermediate certificate, the browser uses the cached copy to verify the chain. In addition, if the user visits a site which sends an expired intermediate CA certificate, both Firefox and Internet Explorer will automatically replace it with the more recent version if they have seen it previously in the session.

Internet Explorer takes some additional measures to address missing intermediate CA certificates that Firefox does not. First, in addition to caching intermediate CA certificates within a session, Internet Explorer caches these certificates persistently, across sessions. Second, Internet Explorer takes advantage of the Authority Information Access (AIA) extension included in some X.509 certificates. The AIA extension "indicates how to access CA information for the issuer of the certificate in which the extension appears" [63]. We found for many server certificates issued by an intermediate CA certificate, they include the AIA extension with a URL for the intermediate CA certificate, and Internet Explorer automatically downloads and uses it to verify the chain. Firefox does not use the AIA extension. It is unclear exactly why not, but discussions on Mozilla Bugzilla suggest it might be because some of the Mozilla developers believe the AIA standard is not well specified [49]. As a result of these additional mechanisms in Internet Explorer, Firefox generates more certificate warnings on average for sites with missing or expired intermediate CA certificates. We summarize these differences in Table 2.

4.4.3 Evaluation results

Weak-locked same-origin policy. To evaluate the deployability of the weak locked SOP, we validated the servers' certificate chains in our survey using two procedures: Pessimistic Validation and Optimistic Validation. Pessimistic Validation models the worse case scenario: a Firefox user visits a web site with a missing or expired intermediate CA certificate at the start of a session, or a user freshly installs IE and visits the same site, and the server certificate does not support AIA. Through empirical analysis we identified 18 widely used intermediate CA certificates, and for Optimistic Validation, we assume the user's browser has cached valid versions of these certificates. We intend Optimistic Validation to model a long Firefox session or a "broken in" Internet Explorer installation with a substantial intermediate CA certificate cache.

Then, for the 1464 fully qualified SSL domains which use multiple servers, we counted the number of domains which had servers with both valid and invalid certificate chains, since it is these domains that the weak locked SOP may break. Using Pessimistic Validation, we found 8 such domains, and for Optimistic Validation we found 4 domains. For each of the other 1456 domains, its servers either had all valid certificates or all invalid certificates.

The difference between the Optimistic and Pessimistic Validation results means we found 4 domains that contained a mix of servers with missing or expired intermediate certificates and correctly configured servers. Of the 4 remaining domains which still cause problems with Optimistic Validation, 3 are probably the result of virtual hosting issues. For example, of the 3 servers we found for `signin.half.ebay.com`, one had a valid certificate, and the other two had CN/domain name mismatch problems. These two servers presented certificates for `signin.ebay.com`. The remaining problem domain was the result of an expired certificate on one of its servers. When the domain's administrators updated their certificates, they probably overlooked this server.

Policy	**Percentage of potentially non-interoperating domains**
Weak locked SOP	0.05%
Strong locked SOP	0.6%

Table 3: Summary of our deployability analysis of the locked same-origin policies using a sample of 14651 SSL domains. This table shows the percentage of domains in our survey which may not correctly interoperate with the locked same-origin policies.

This means the weak locked SOP would potentially break at most 0.05% of the SSL domains we found in our survey. These results are strong evidence that browsers could enforce the weak locked SOP today and still interoperate with the vast majority of web sites while providing increased protection against pharming attacks. Furthermore, since the number of problem domains is relatively small, browser developers can conceivably work with these domains' administrators to make their servers consistent. In conclusion, we can safely deploy the weak locked SOP in a way which requires minor browser changes, but does not require changes to the HTTP specification, SSL, or web servers.

Strong locked same-origin policy. To evaluate the deployability of the strong locked SOP, we counted the number of fully qualified SSL domains with multiple servers that do not use the same public key on all of the servers. We found 83 such domains, representing 0.6% of the total number of SSL domains in our survey. This is problematic for two reasons. First, it represents an order of magnitude more servers than that are affected by the weak locked SOP. Second, unlike before, these servers are not necessarily misconfigured, so browser developers cannot work with the domain's administrators to "fix" the problem. Using a different key on each server is good security practice, since it limits the scope of key compromise. In fact, VeriSign explicitly recommends customers use different public keys on each server [75].

Another problem concerns certificate expiration. 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. When web sites renew their certificates they often follow good security practice and generate a new public key. Since the strong locked SOP applies to all locked web objects, if a web site uses persistent SSL-only cookies to authenticate users (see Section 5), every user's cookie will simultaneously "expire" (i.e., become inaccessible by the server) when the site starts using the new public key, regardless of the value of the cookie's `expires` attribute.

Based on this evidence, we conclude browsers cannot currently enforce the strong locked SOP without potentially breaking a significant number of web sites. However, this does not mean that deploying the strong locked SOP is hopeless; it only means a browser must first get the site's explicit cooperation and approval to enforce it. In the next section, we describe a simple incrementally deployable solution using policy files for the strong locked SOP which supports multiple public keys and key updates.

4.5 Policy files for supporting multiple keys and key updates

We propose an incrementally deployable solution where a web site can opt-in to the strong locked SOP; then, browsers which support the policy can safely enforce it without the risk of unintentionally breaking the site. To opt in, we propose a site's servers post a policy file with a static well-known file name, say `pk.txt`, which would be periodically retrieved by web clients (over SSL), similar to `robots.txt` or `favicon.ico`. If a browser finds a `pk.txt` file, it parses the file and starts enforcing the strong locked SOP for that domain. If all the domain's servers use the same public key and persistent objects are not an issue, a site can simply post an empty `pk.txt`, since it will already interoperate with the strong locked SOP.

If the site uses multiple servers, labeled $i = 1 \ldots n$, with different public keys, then `pk.txt` on server k contains a list of the form:

$$(pk_1, \{pk_k\}_{sk_1}),\ (pk_2, \{pk_k\}_{sk_2}),\ \ldots,\ (pk_n, \{pk_k\}_{sk_n})$$

where pk_k is the public key of the server hosting this `pk.txt` file and $\{pk_k\}_{sk_i}$ represents a signature of pk_k by the secret key corresponding to pk_i. The browser then verifies each of the signatures, and for $i = 1 \ldots n$, if the i^{th} signature is valid, then it considers pk_k to "speak for" pk_i. We then extend the strong locked SOP with the following rule: a browser allows a locked web object tagged with (D, pk_j) to access another locked web object tagged with (D, pk_l) if a policy file attests that pk_j speaks for pk_l.

Note that `pk.txt` cannot simply list the public keys $(pk_1, pk_2, \ldots, pk_n)$; otherwise a pharmer can serve the same file to a victim, and the victim's browser will infer that the pharmer's public key speaks for each of the keys of the legitimate servers. However, since the pharmer does not know the legitimate servers' private keys, it will not be able to generate any valid signatures required for the victim's browser to infer the "speaks for" relation.

Policy files also address the problem of public key updates discussed in Section 4.4.3. For example, suppose a web site wants to renew its certificate with a new public key pk_{new}. Then, several months before the certificate expires, the site can include $(pk_i, \{pk_{\text{new}}\}_{sk_i})$ in its `pk.txt` files for each server i, $i = 1 \ldots n$. Then, users that retrieve `pk.txt` during this transition period will not "lose" persistent objects tagged with an old public key.

In conclusion, policy files address the deployability problems with the strong locked SOP we identified in Section 4.4.3. They enable us to enforce the strong locked SOP in current browsers in a way which is incrementally deployable and backwards compatible with legacy servers. The strong locked SOP in conjunction with policy files requires browser changes and server configuration changes for sites wishing to take advantage of the policy, but does not require changes to the HTTP specification or SSL.

4.6 Support for subdomain object sharing

Up until now we have implicitly assumed a web site consists of a single fully qualified domain name, e.g., `www.xyz.com`. More generally, a web site might be composed of several domain names, e.g., `mail.xyz.com`, `www.xyz.com`, `login.xyz.com`, and the legacy SOP supports some exceptions which enable these sites to share information among subdomains through certain web objects. For example, a user might authenticate herself to the server for `login.xyz.com`, and this server will set a domain cookie with `domain=xyz.com` for authenticating the user to the other subdomain servers. The user's browser will allow any subdomain of `xyz.com` to access this cookie. Another way subdomains can share information is by setting the DOM property `document.domain`. For example, if a document from `www.xyz.com` sets `document.domain=xyz.com`, the browser permits any object from a subdomain of `xyz.com` to access the document.

However, these domain sharing mechanisms are vulnerable to pharming attacks. For example, if an adversary pharms any host name in `xyz.com`, she can steal users' domain cookies for `xyz.com`. Ideally, we would like to enforce our locked same-origin policies in these situations as well. Fortunately, extending the strong locked SOP to support subdomain sharing is straightforward with policy files. The site simply adds the servers' public keys to its policy files and we extend the strong locked SOP with the following rule: if S is locked web object hosted by server l and is designated to be shared among subdomains of a higher-level domain TD (e.g., `xyz.com`), a browser allows a locked web object tagged with (D, pk_j) to access S if D is a subdomain of TD and a policy file attests that pk_j speaks for pk_l.

Unfortunately, it is not clear how to extend the weak locked SOP to support shared domain objects without any server cooperation. An natural candidate extension would be to allow access if both subdomain servers have valid certificates or invalid certificates. However, we must have confidence this policy will not "break the web" and not deny access to a legitimate server when the legacy SOP would allow access. Roughly, this would require for each higher-level domain, either all its subdomain servers have valid certificates or all its subdomain servers have invalid certificates. Unfortunately, our survey survey shows this is far from the case. Of the 6192 second-level SSL domains we found, over 1000 did not satisfy this property. This means for browsers enforcing the weak locked SOP, they must default back to the legacy SOP for shared domain objects, which provides no protection against pharming.

4.7 Support for key revocation

To address key compromise, our locked same-origin policies and policy files can be extended to support expiration times, certificate revocation lists (CRLs), and the Online Certificate Status Protocol (OCSP), but for space reasons, we do not go into the details here.

4.8 Active objects

We refer to *active objects* as web objects containing scripts or executable code. Examples of active objects include Javascript files, Flash movies, and Java applets. Similar to images and style sheets, browsers allow web pages to include active objects from other domains. We refer to active objects from domains different from the enclosing page as *cross-domain active objects*, and those from the same domain as the enclosing page as *same-domain active objects*.

Subtle pharming vulnerabilities can arise even if a web site only uses same-domain active objects. For example, suppose `xyz.com` includes a Javascript file in its home page:

```
<script type="text/javascript"
  src="https://xyz.com/login.js">
```

Javascript files run directly in the context the page including them. In the above example, an adversary could use dynamic pharming against `xyz.com` to first cache a copy of `login.js` in the victim's browser, and then cause the browser to retrieve the legitimate home page for `xyz.com`. As the browser renders the page, it will use the adversary's cached copy of `login.js`, enabling the adversary to hijack the user's session with the legitimate server for `xyz.com` [68]. Similar vulnerabilities exist for Flash movies and Java applets.

Cross-domain active objects are commonly used in web mashups, such as `www.housingmaps.com`, but other web sites also use them. For example, the home page for `www.paypal.com` includes the following element:

```
<script type="text/javascript"
  src="https://www.paypalobjects.com/...">
```

In this case, the Javascript file from `www.paypalobjects.com` will execute in the context of `www.paypal.com` and the browser gives it full access to `www.paypal.com`. An adversary capable of pharming `www.paypalobjects.com` will effectively be able to insert Javascript on the PayPal homepage and hijack users' sessions. Notice that the adversary does not need to pharm `www.paypal.com` for this attack to succeed. Browsers also allow cross-domain Java and Flash, and there are mechanisms which enable Java and Flash to interact with the enclosing document.

In light of these vulnerabilities, a vigilant web site should forgo the use of Java and Flash, and include all necessary Javascript explicitly within its web pages. However, this significantly restricts flexibility and functionality. Unfortunately, we can do little to protect active objects requested over HTTP. For active objects requested over HTTPS, we discuss two solutions: 1) YURLs, and 2) extending the locked SOPs.

4.8.1 YURLs

A YURL [8] consists of a URL and a public key hash, and the browser can use the hash to authenticate the server. For example, if browser requests a YURL for `xyz.com`, the browser uses DNS to resolve `xyz.com` to an IP address, and then establishes an SSL connection with the server. After a connection is established, the browser compares the public key hash in the YURL against the public key of the server. If the hash is consistent with the server's public key, the browser proceeds with the request; otherwise it cancels the connection with no opportunity for user override. YURLs help active objects resist pharming attacks because they enable web sites to explicitly specify the server hosting an object by its public key. Browsers do not currently support YURLs, but the necessary modifications should be relatively simple and straightforward.

4.8.2 Extending the locked same-origin policies

We can also extend the locked SOPs to help protect same-domain requests for active objects against pharming attacks. Since in the strong locked SOP, a web site explicitly specifies the list of valid public keys for its domain, a browser can safely refuse to load active objects from a alleged server for that domain with a different public key. We can similarly extend the weak locked SOP. If the server for the enclosing page has valid certificate, then a browser should only accept a same-domain active object from a server with a valid certificate. Based on the results of our deployability analysis, this additional restriction for the weak locked SOP should be safe.

Unfortunately, this approach will not work for cross-domain active objects. Since the enclosing page and the object are from different domains, their certificates are incomparable. Currently, if the server hosting the object has an invalid certificate, most browsers and plugins will issue a warning, but as we have seen, users often ignore and override such warnings.

4.9 Caching and dependent elements

Browsers do not by default persistently cache HTTPS web objects (except SSL-only cookies), but they do use session caching. As we demonstrated in Section 4.8, adversaries can use the cache to launch subtle dynamic pharming attacks. Also, vulnerabilities may arise if adversaries can control dependent HTML elements, such as images and style sheets. To resist attacks involving the cache and dependent elements, browsers and web sites can use the countermeasures in Section 4.8 to protect these objects as well.

4.10 Limitations

Our locked same-origin policies do not address attacks where the adversary tricks a victim into installing malicious software such as executable malware, an ActiveX plugin, or a browser extension. These objects usually execute with elevated privilege and are not governed by the SOP.

Security researchers have found DNS rebinding weaknesses in third-party browser plugin implementations such as Flash, Java, and Adobe Reader. Some of these weaknesses can be exploited to launch dynamic pharming attacks, and the locked SOPs do not address them. Jackson et al. address some of these vulnerabilities [29], and the locked SOPs can work in conjunction with their solutions to resist dynamic pharming attacks.

The locked same-origin policies do not address problems in the Javascript language or implementation (e.g., Javascript Prototype Hijacking) [54], cross-side scripting (XSS) vulnerabilities in servers, and cross-site request forgery (XSRF) attacks. Other research efforts address XSS vulnerabilities [27, 39, 44, 79, 80] and XSRF attacks [31, 34, 35], and these techniques complement our work. We also do not address browser-side cross-site scripting vulnerabilities, such as Universal XSS [54].

5. APPLICATIONS TO WEB AUTHENTICATION

In this section, we discuss how the locked same-origin policies can help protect two existing browser authentication mechanisms, client side-SSL and SSL-only cookies, against pharmers and active attackers. However, the web authentication problem is actually two distinct subproblems: the initialization of users' authentication credentials and the use of those credentials to authenticate users to web sites. Our discussion in this section focuses primarily on the latter, but we discuss initialization briefly.

Client-side SSL. Intuitively, since client-side SSL authenticates users with end-to-end cryptography, one might expect it would protect sensitive web sessions against pharming and active attacks, but unfortunately, the presence of dynamic pharming vulnerabilities proves this is not the case. However, using one of our locked same-origin policies in conjunction with client-side SSL results in an authentication scheme with strong security properties. The user is not required to memorize her private key. After the user imports her private key, her browser uses it automatically. Although an adversary may be able to trick a user into participating in mutual authentication using SSL, the adversary cannot use this interaction to impersonate the user at another web site. Authentication requires knowledge of the private key, which the user's browser always keeps secret. As a result, the browser authenticates the user's requests cryptographically and the locked SOP isolates the user's authenticated sessions from malicious subjects – even if the adversary is a pharmer or active attacker.

SSL-only cookies. Many web sites use cookies for authentication [16]. 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 [28], Vanguard [74], and Yahoo [81]. 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. In current browsers, cookie authentication resists phishing attacks but is vulnerable to pharming attacks. Our locked same-origin policies protect SSL-only cookies against pharmers and active attackers. Thus, in conjunction with browsers enforcing a locked SOP, web sites can use SSL-only persistent cookies to authenticate users and resist phishing, pharming, and active attacks.

Other authentication mechanisms. The locked same-origin policies nicely complement other authentication mechanisms designed to resist pharming, such as Phoolproof phishing prevention [55] and Passpet [83]. Our policies help these schemes resist dynamic pharming attacks.

The registration problem. The initialization of a user's authentication credentials is commonly known as the *registration problem*. The registration problem is a critical element of any web authentication scheme, and a challenging problem on its own right. For space reasons, we do not attempt to explore potential solutions here. One key challenge in the registration problem is resisting *registration attacks*. Bank of America's SiteKey and similar anti-phishing mechanisms have registration attack vulnerabilities [67, 85]. 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. We are exploring registration protocols utilizing "out of band" channels to distribute authentication credentials.

6. RELATED WORK

Several anti-phishing mechanisms help provide information to users regarding the trustworthiness of web sites. Since studies have shown that users can be fooled by misleading domain names and do not understand browser security indicators [10, 14, 15, 18], several researchers and security vendors have developed browser extensions to make it easier for users to interpret relevant security information [7, 25, 43, 69], use a blacklist to help identify known phishing sites [11, 52], or establish trusted paths with sites users have a relationship with [9, 78, 82]. Recent versions Firefox and Internet Explorer have adopted similar mechanisms. However, these approaches still expect some degree of diligence from users to reliably observe security warnings and indicators to operate securely, and studies have shown that users still have troubling interpreting improved security indicators and warnings [30, 64, 77, 78]. In addition, studies have also shown that many browser extensions which try to automatically detect phishing sites are often wrong and inconsistent [86].

Another approach to resisting phishing attacks is better password management. Passwords are still the dominant method of web authentication. Password databases included with most modern web browsers automatically fill in passwords for users. However, users might still manually disclose their passwords to phishing sites or use the same password for multiple sites. Password hashing addresses these problems by hashing the user's secret password together with a variable, non-secret string (e.g., each site's domain name) to produce per-site passwords [1, 17, 24, 37, 38, 62, 83]. Recent work in this area has made usability one of the primary goals [24, 62, 83], but studies have shown some users still have trouble using them correctly and securely [6]. Also, if password hashing scheme generates passwords based on the site's domain [24, 62], it is vulnerable to pharming attacks. Passpet [83] provides some resistance to pharming attacks, but is still vulnerable to dynamic pharming.

The Phoolproof phishing prevention system uses cell phones to manage client-side SSL certificates for authentication on behalf of users [55]. Phoolproof's designers also noted the importance of disclosure resistant authentication credentials. In Phoolproof, a user logs in using a secure bookmark on her cell phone. The cell phone then 1) initiates an SSL connection to the web site via a Bluetooth connection with a web browser on the user's computer; 2) checks the site's X.509 certificate against the one stored in the bookmark; and 3) authenticates the user via client-side SSL. Although Phoolproof verifies the site's certificate in step 2, this protocol is still vulnerable to a dynamic pharming attack if the adversary is able to pharm the user (i.e., serve the user a web page which appears to come from the target domain) before she activates the login process.

The locked same-origin policies are similar to work done independently and concurrently by Masone et al. on Web Server Key Enabled Cookies, a new cookie policy which tags SSL-only cookies with the server's public key and allows access only to a server which can authenticate itself to the same key [41]. However, their proposal falls short of protecting cookies against dynamic pharming attacks. Also, they do not address pharming attacks against other web objects or other web authentication mechanisms, e.g., client-side SSL, nor do they address subdomain object sharing or key updates.

The locked SOP was inspired by the concept of Key Continuity Management (KCM), a model for key management first proven successful by SSH [60, 84] and made more explicit by Gutmann [23]. KCM associates public keys with subjects and takes defensive action when a subject's public key unexpectedly changes. Garfinkel expands on KCM further, and applies it to S/MIME [19].

Security researchers and browser developers have been aware of DNS rebinding vulnerabilities since as early as 1996 [22]. In 2001 and 2002, Jim Roskind and Adam Megacz, resp., described firewall circumvention DNS rebinding attacks using DNS records with short TTLs [42, 61]. DNS pinning was adopted by browsers to defend against these kinds of attacks, but pinning has a lengthy and controversial history in Firefox and Mozilla [46, 47, 48]. The current implementation is an explicit compromise to support dynamic DNS and round robin DNS for failover. In August 2006, Martin Johns discovered a reliable technique for circumventing DNS pinning completely [32], and in early 2007, Johns and Kanatoko found additional DNS rebinding vulnerabilities with Flash and Java [33, 36].

Jackson et al. present a excellent analysis of DNS rebinding vulnerabilities, including issues with Flash, Java, VPNs, caching, and proxies [29]. They discuss several countermeasures, including host name authorization, a technique based on a variant of reverse DNS lookups. With cooperation from web sites' DNS servers, host name authorization enables clients to determine the valid set of domain names for a particular IP address. Host name authorization is promising approach, but since it relies on DNS, it is ineffective against adversaries capable of subverting DNS.

7. CONCLUSION

We demonstrated how adversaries can use dynamic pharming attacks to hijack users' authenticated web sessions, irrespective of the authentication mechanism. Dynamic pharming enables an adversary to eavesdrop on sensitive content, forge transactions, sniff secondary passwords, etc. To address dynamic pharming attacks, we introduced two locked same-origin policies, which regulate cross-object access control using servers' X.509 certificates and public keys, rather than domain names. We evaluated the security and deployability of our approaches and showed how browsers can deploy these policies today to substantially increase their resistance to pharming attacks and provide both a solid and necessary foundation for developing pharming resistant authentication.

8. ACKNOWLEDGEMENTS

This work was supported in part by National Science Foundation award number CCF-0424422 (The TRUST Center). We thank Dan Boneh and Adam Barth for helpful suggestions with policy files. We thank Ben Adida, Naveen Sastry, Ka-Ping Yee, and Zulfikar Ramzan for providing useful comments on previous drafts.

9. REFERENCES

[1] Martin Abadi, T. Mark A. Lomas, and Roger Needham. Strengthening passwords. Technical Report 1997-033, SRC, September 1997.

[2] P. Akritidis, W.Y. Chin, V.T. Lam, S. Sidiroglou, and K.G. Anagnostakis. Proximity Breeds Danger: Emerging Threats in Metro-area Wireless Networks. In *Proceedings of the* 16^{th} *USENIX Security Symposium*, pages 323–338, August 2007.

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

[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] Sonia Chiasson, P.C. van Oorschot, and Robert Biddle. A usability study and critique of two password managers. In *Proceedings of the 15th USENIX Security Symposium*, pages 1–16, August 2006.

[7] Tyler Close. Petname tool. `http://petname.mozdev.org/`.

[8] Tyler Close. Waterken YURL. `http://www.waterken.com/dev/YURL/httpsy/`.

[9] Rachna Dhamija and J. D. Tygar. The Battle Against Phishing: Dynamic Security Skins. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 77–88, July 2005.

[10] 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.

[11] Earthlink Toolbar Featuring ScamBlocker for Windows Users. `http://www.earthlink.net/software/free/toolbar/`.

[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] Eran Gabber, Phillip B. Gibbons, Yossi Matias, and Alain J. Mayer. How to make personalized web browsing simple, secure, and anonymous. In *Proceedings of Financial Cryptography (FC '97)*, pages 17–32, 1997.

[18] Evgeniy Gabrilovich and Alex Gontmakher. The homograph attack. *Communications of ACM*, 45(2):128, February 2002.

[19] Simson Garfinkel. *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, Massachusetts Institute of Technology, 2005.

[20] David Goldsmith. How a 'Catch-22' Turns into a 'Shame on You'. `http://isc.sans.org/diary.html?storyid=1230`, March 2006.

[21] Anti-Phishing Working Group. Ebay - Update Your Account MITM attack. `http://www.antiphishing.org/phishing_archive/05-03-05_Ebay/05-03-05_Eba%y.html`.

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

[23] Peter Gutmann. Why isn't the Internet secure yet, dammit. In *AusCERT Asia Pacific Information Technology Security Conference 2004*, May 2004.

[24] J. Alex Halderman, Brent Waters, and Edward W. Felten. A convenient method for securely managing passwords. In *Proceedings of the 14th International World Wide Web Conference*, May 2005.

[25] Amir Herzberg and Ahmad Gbara. Security and Identification Indicators for Browsers against Spoofing and Phishing Attacks. Cryptology ePrint Archive, Report 2004/155, 2004.

[26] 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.

[27] 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.

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

[29] Collin Jackson, Adam Barth, Andrew Bortz, Weidong Shao, and Dan Boneh. Protecting Browsers from DNS Rebinding Attacks. In *14th ACM Conference on Computer and Communications Security (CCS '07)*, November 2007.

[30] Collin Jackson, Daniel R. Simon, Desney S. Tan, and Adam Barth. An Evaluation of Extended Validation and Picture-in-Picture Phishing Attacks. In *Proceedings of Usable Security (USEC'07)*, February 2007.

[31] 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.

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

[33] Martin Johns. Using Java in anti DNS-pinning attacks. `http://shampoo.antville.org/stories/1566124/`, February 2007.

[34] Martin Johns and Justus Winter. RequestRodeo: Client Side Protection against Session Riding. In *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, pages 5 – 17. Departement Computerwetenschappen, Katholieke Universiteit Leuven, May 2006.

[35] 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.

[36] Kanatoko. Anti-DNS Pinning (DNS Rebinding) + Socket in FLASH. `http://www.jumperz.net/index.php?i=2&a=3&b=3`, January 2007.

[37] Alan H. Karp. Site-specific passwords. Technical Report HPL-2002-39R1, HP Labs, 2002.

[38] John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. *Lecture Notes in Computer Science*, 1396:121–134, 1998.

[39] 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.

[40] Uriel Maimon. Universal Man-in-the-Middle Phishing Kit – why is this even news? `http://www.rsa.com/blog/entry.asp?id=1160`.

[41] Chris Masone, Kwang-Hyun Baek, and Sean Smith. WSKE: Web Server Key Enabled Cookies. In *Proceedings of Usable Security (USEC)*, February 2007.

[42] Adam Megacz. XWT Foundation Advisory: Firewall circumvention possible with all browsers. `http://www.megacz.com/research/papers/sop.txt`, July 2002.

[43] Microsoft. Better Website Identification and Extended Validation Certificates in IE7 and Other Browsers. `http://blogs.msdn.com/ie/archive/2005/11/21/495507.aspx`.

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

[45] Microsoft. Microsoft security bulletin MS01-017: Erroneous VeriSign-issued digital certificates pose spoofing hazard. `http://www.microsoft.com/technet/security/Bulletin/MS01-017.mspx`, March 2001.

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

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

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

[49] Mozilla Bugzilla bug 245609 – Mozilla not getting certificate issuer from Authority Information Access CA Issuers, June 2004.

[50] mozilla.dev.security. VeriSign Class 3 Secure Server CA? `http://groups.google.com/group/mozilla.dev.security/browse_thread/threa%d/6830a8566de24547/0be9dea1c274d0c5`, March 2007.

[51] mozilla.org. The same-origin policy. `http://www.mozilla.org/projects/security/components/same-origin.html`.

[52] Netcraft anti-phishing toolbar. `http://toolbar.netcraft.com/`.

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

[54] Stefano Di Paola and Giorgio Fedon. Subverting Ajax. In *23rd Chaos Communication Congress*, December 2006.

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

[56] Washington Post. Citibank Phish Spoofs 2-Factor Authentication. `http://blog.washingtonpost.com/securityfix/2006/07/citibank_phish_spoof%s_2factor_1.html`.

[57] Washington Post. Not Your Average Phishing Scam. `http://blog.washingtonpost.com/securityfix/2007/01/not_your_average_ama%zon_phishi.html`.

[58] PTFB Pro. `http://www.ptfbpro.com/`.

[59] 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.

[60] Nicholas Rosasco and David Larochelle. How and why more secure technologies succeed in legacy markets: Lessons from the success of SSH. In *Proceedings of the Second Annual Workshop on Economics and Information Security*, May 2003.

[61] Jim Roskind. Attacks against the netscape browser. Invited talk, RSA conference, April 2001.

[62] Blake Ross, Collin Jackson, Nicholas Miyake, Dan Boneh, and John C. Mitchell. Stronger password authentication using browser extensions. In *Proceedings of the 14th USENIX Security Symposium*, pages 17–32, August 2005.

[63] Stefan Santesson and Russell Housley. Internet X.509 Public Key Infrastructure Authority Information Access Certificate Revocation List (CRL) Extension. http://www.ietf.org/rfc/rfc4325.txt, December 2005.

[64] Stuart Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. Emperor's new security indicators: An evaluation of website authentication and the effect of role playing on usability studies. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, May 2007.

[65] Security Space and E-Soft. Secure Server Survey. `http://www.securityspace.com/s_survey/sdata/200704/certca.html`, May 2007.

[66] Rajiv Shah and Christian Sandvig. Software Defaults as De Facto Regulation: The Case of the Wireless Internet. In *The 33rd Research Conference on Communication, Information, and Internet Policy*, September 2005.

[67] Christopher Soghoian and Markus Jakobsson. A Deceit-Augmented Man In The Middle Attack Against Bank of America's SiteKey Service. `http://paranoia.dubfire.net/2007/04/deceit-augmented-man-in-middle-atta%ck.html`, April 2007.

[68] Josh Soref. DNS: Spoofing and Pinning. `http://viper.haque.net/~timeless/blog/11/`.

[69] Spoofstick. `http://www.spoofstick.com/`.

[70] Sid Stamm, Zulfikar Ramzan, and Markus Jakobsson. Drive-by pharming. Technical Report 641, Indiana University Computer Science, December 2006.

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

[72] 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.

[73] 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.

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

[75] VeriSign. Licensing VeriSign Certificates Securing Multiple Web Server and Domain Configurations. `http://www.verisign.com/static/001496.pdf`, June 2005.

[76] VivilProject. List of public DNS servers.

[77] 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.

[78] 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.

[79] 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.

[80] 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.

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

[82] Eileen Ye and Sean Smith. Trusted paths for browsers. In *Proceedings of the 11th USENIX Security Symposium*, pages 263–279, August 2002.

[83] Ka-Ping Yee and Kragen Sitaker. Passpet: Convenient password management and phishing protection. In *Proceedings of the Symposium on Usable Privacy and Security (SOUPS)*, pages 32–43, July 2006.

[84] Tatu Ylonen. SSH – secure login connections over the Internet. In *Proceedings of the 6th USENIX Security Symposium*, pages 37–42, 1996.

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

[86] Yue Zhang, Serge Egelman, Lorrie Faith Cranor, and Jason Hong. Phinding phish: Evaluating anti-phishing tools. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, February 2007.