Protecting Browsers from DNS Rebinding Attacks

Collin Jackson
Stanford University
collinj@cs.stanford.edu

Adam Barth
Stanford University
abarth@cs.stanford.edu

Andrew Bortz
Stanford University
abortz@cs.stanford.edu

Weidong Shao
Stanford University
wshao@cs.stanford.edu

Dan Boneh
Stanford University
dabo@cs.stanford.edu

ABSTRACT

DNS rebinding attacks subvert the same-origin policy of browsers and convert them into open network proxies. We survey new DNS rebinding attacks that exploit the interaction between browsers and their plug-ins, such as Flash Player and Java. These attacks can be used to circumvent firewalls and are highly cost-effective for sending spam e-mail and defrauding pay-per-click advertisers, requiring less than $100 to temporarily hijack 100,000 IP addresses. We show that the classic defense against these attacks, called "DNS pinning," is ineffective in modern browsers. The primary focus of this work, however, is the design of strong defenses against DNS rebinding attacks that protect modern browsers: we suggest easy-to-deploy patches for plug-ins that prevent large-scale exploitation, provide a defense tool, `dnswall`, that prevents firewall circumvention, and detail two defense options, policy-based pinning and host name authorization.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms

Security, Design, Experimentation

Keywords

Same-Origin Policy, DNS, Firewall, Spam, Click Fraud

1. INTRODUCTION

Users who visit web pages trust their browser to prevent malicious web sites from leveraging their machines to attack others. Organizations that permit JavaScript and other active content through their firewall rely on the browser to protect internal network resources from attack. To achieve these security goals, modern browsers implement the *same-origin policy* that attempts to isolate distinct "origins," protecting sites from each other.

DNS rebinding attacks subvert the same-origin policy by confusing the browser into aggregating network resources controlled by distinct entities into one origin, effectively converting browsers into open proxies. Using DNS rebinding, an attacker can circumvent firewalls to spider corporate intranets, exfiltrate sensitive documents, and compromise unpatched internal machines. An attacker can also hijack the IP address of innocent clients to send spam e-mail, commit click fraud, and frame clients for misdeeds. DNS rebinding vulnerabilities permit the attacker to read and write directly on network sockets, subsuming the attacks possible with existing JavaScript-based botnets [24], which can send HTTP requests but cannot read back the responses.

To mount a DNS rebinding attack, the attacker need only register a domain name, such as `attacker.com`, and attract web traffic, for example by running an advertisement. In the basic DNS rebinding attack, the attacker answers DNS queries for `attacker.com` with the IP address of his or her own server with a short time-to-live (TTL) and serves visiting clients malicious JavaScript. To circumvent a firewall, when the script issues a second request to `attacker.com`, the attacker rebinds the host name to the IP address of a target server that is inaccessible from the public Internet. The browser believes the two servers belong to the same origin because they share a host name, and it allows the script to read back the response. The script can easily exfiltrate the response, enabling the attacker to read arbitrary documents from the internal server, as shown in Figure 1.

To mount this attack, the attacker did not compromise any DNS servers. The attacker simply provided valid, authoritative responses for `attacker.com`, a domain owned by the attacker. This attack is very different from "pharming" [34], where the attacker must compromise a host name owned by the target by subverting a user's DNS cache or server. DNS rebinding requires no such subversion. Consequently, DNSSEC provides no protection against DNS rebinding attacks: the attacker can legitimately sign all DNS records provided by his or her DNS server in the attack.

DNS rebinding attacks have been known for a decade [8, 36]. A common defense implemented in several browsers is *DNS pinning*: once the browser resolves a host name to an IP address, the browser caches the result for a fixed duration, regardless of TTL. As a result, when JavaScript connects to `attacker.com`, the browser will connect back to the attacker's server instead of the internal server.

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

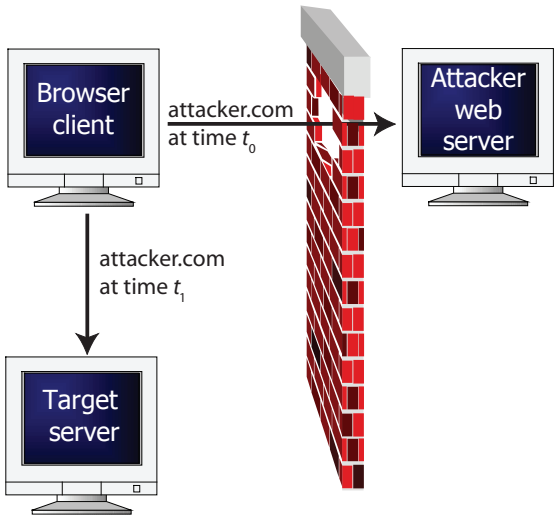

Figure 1: Firewall Circumvention Using Rebinding

Pinning is no longer an effective defense against DNS rebinding attacks in current browsers because of vulnerabilities introduced by plug-ins. These plug-ins provide additional functionality, including socket-level network access, to web pages. The browser and each plug-in maintain separate pin databases, creating a new class of vulnerabilities we call *multi-pin* vulnerabilities that permit an attacker to mount DNS rebinding attacks. We demonstrate, for example, how to exploit the interaction between the browser and Java LiveConnect to pin the browser to one IP address while pinning Java to another IP address, permitting the attacker to read and write data directly on sockets to a host and port of the attacker's choice despite strong pinning by each component.

Our experiments show how an attacker can exploit multi-pin vulnerabilities to cheaply and efficiently assemble a temporary, large-scale bot network. Our findings suggest that nearly 90% of web browsers are vulnerable to rebinding attacks that only require a few hundreds of milliseconds to conduct (see Table 1). These attacks do not require users to click on any malicious links: users need only view an attacker's web advertisement. By spending less than $100 on advertising, an attacker can hijack 100,000 unique IP address to send spam, commit click fraud, or otherwise misuse as open network proxies.

The bulk of our work focuses on designing robust defenses to DNS rebinding attacks that protect current and future browsers and plug-ins:

1. To combat firewall circumvention, we recommend organizations deploy DNS resolvers that prevent *external* names from resolving to *internal* addresses. We provide an open-source implementation of such a resolver in 300 lines of `C` called `dnswall` [15].

2. For Flash Player, Java, and LiveConnect, we suggest specific, easy-to-deploy patches to prevent multi-pin vulnerabilities, mitigating large-scale exploitation of DNS rebinding for firewall circumvention and IP hijacking.

Technology	Attack Time
LiveConnect (JVM loaded)	47.8 ± 10.3 ms
Flash Player 9	192 ± 5.7 ms
Internet Explorer 6 (no plug-ins)	1000 ms
Internet Explorer 7 (no plug-ins)	1000 ms
Firefox 1.5 and 2 (no plug-ins)	1000 ms
Safari 2 (no plug-ins)	1000 ms
LiveConnect	1294 ± 37 ms
Opera 9 (no plug-ins)	4000 ms

Table 1: Time Required for DNS Rebinding Attack by Technology (95% Confidence)

3. We propose two options for protecting browsers from DNS rebinding: smarter pinning that provides better security and robustness, and a backwards-compatible use of the DNS system that fixes rebinding vulnerabilities at their root (which we implemented as a 72-line patch to Firefox 2).

The remainder of the paper is organized as follows. Section 2 describes existing browser policy for network access. Section 3 details DNS rebinding vulnerabilities, including standard DNS rebinding and current multi-pin vulnerabilities. Section 4 explains two classes of attacks that use these vulnerabilities, firewall circumvention and IP hijacking, and contains our experimental results. Section 5 proposes defenses against both classes of attacks. Section 6 describes related work. Section 7 concludes.

2. NETWORK ACCESS IN THE BROWSER

To display web pages, browsers are instructed to make network requests by static content such as HTML and by active content such as JavaScript, Flash Player, Java, and CSS. Browsers restrict this network access in order to to prevent web sites from making malicious network connections.

The same-origin policy provides partial resource isolation by restricting access according to *origin*, specifying when content from one origin can access a resource in another origin. The policy applies to both network access and browser state such as the Document Object Model (DOM) interface, cookies, cache, history, and the password database [20]. The attacks described in this paper circumvent the same origin-policy for network access.

Access Within Same Origin. Within the same origin, both content and browser scripts can read and write network resources using the HTTP protocol. Plug-ins, such as Flash Player and Java, can access network sockets directly, allowing them to make TCP connections and, in some cases, send and receive UDP packets as well. Java does not restrict access based on port number, but Flash Player permits access to port numbers less than 1024 only if the machine authorizes the connection in an XML policy served from a port number less than 1024.

Access Between Different Origins. In general, content from one origin can make HTTP requests to servers in another origin, but it cannot read responses, effectively restricting access to "send-only." Flash Player permits its movies to read back HTTP responses from different origins, provided the remote server responds with an XML policy authorizing the movie's origin. Flash Player also permits

reading and writing data on TCP connections to arbitrary port numbers, again provided the remote server responds with a suitable XML policy on an appropriate port.

By convention, certain types of web content are assumed to be public libraries, such as JavaScript, CSS, Java applets, and SWF movies. These files may be included across domains. For example, one origin can include a CSS file from another origin and read its text. Scripts can also read certain properties of other objects loaded across domains, such as the height and width of an image.

Prohibited Access. Some types of network access are prohibited even within the same origin. Internet Explorer 7 blocks port numbers 19 (chargen), 21 (FTP), 25 (SMTP), 110 (POP3), 119 (NNTP), and 143 (IMAP), Firefox 2 blocks those plus 51 additional port numbers, but Safari 2 does not block any ports. Some of these port restrictions are designed to prevent malicious web site operators from leveraging visiting browsers to launch distributed denial of service or to send spam e-mail, whereas others prevent universal cross-site scripting via the HTML Form Protocol Attack [41].

Origin Definition. Different definitions of "origin" are used by different parts of the browser. For network access, browsers enforce the same-origin policy [38] based on three components of the Uniform Resource Locator (URL) from which it obtained the content. A typical URL is composed of the below components:

```
scheme://hostname:port/path
```

Current browsers treat two objects as belonging to the same origin if, and only if, their URLs contain the same scheme, host name, and port number (e.g., `http://amazon.com/` is a different origin than `http://amazon.co.uk/`, even though the two domains are owned by the same company). Other resources use fewer components of the URL. For example, cookies use only the host name.

Objects on the Internet, however, are not accessed by host name. To connect to a server, the browser must first translate a host name into an IP address and then open a socket to that IP address. If one host name resolves to multiple IP addresses owned by multiple entities, the browser will treat them as if they were the same origin even though they are, from an ownership point-of-view, different.

3. DNS REBINDING VULNERABILITIES

The network access policy in web browsers is based on host names, which are bound by the Domain Name System (DNS) to IP addresses. An attacker mounting a DNS rebinding attack attempts to subvert this security policy by binding his or her host name to both the attack and target server's IP addresses.

3.1 Standard Rebinding Vulnerabilities

A standard rebinding attack uses a single browser technology (e.g. JavaScript, Java, or Flash Player) to connect to multiple IP addresses with the same host name.

Multiple A Records. When a client resolves a host name using DNS, the authoritative server can respond with multiple `A` records indicating the IP addresses of the host. The first attack using DNS rebinding [8] in 1996 leveraged this property to confuse the security policy of the Java Virtual Machine (JVM):

1. A client visits a malicious web site, `attacker.com`, containing a Java applet. The attacker's DNS server binds `attacker.com` to two IP addresses: the attacker's web server and the target's web server.

2. The client executes the attacker's applet, which opens a socket to the target. The JVM permits this connection, because the target's IP address is contained in the DNS record for `attacker.com`.

Current versions of the JVM are not vulnerable to this attack because the Java security policy has been changed. Applets are now restricted to connecting to the IP address from which they were loaded. (Current attacks on Java are described in Section 3.2.)

In the JavaScript version of this attack, the attacker sends some JavaScript to the browser that instructs the browser to connect back to `attacker.com`. The attacker's server refuses this second TCP connection, forcing the browser to switch over to the victim IP address [21]. By using a `RST` packet to refuse the connection, the attacker can cause some browsers to switch to the new IP address after one second. Subsequent XMLHttpRequests issued by the attacker's code will connect to the new IP address.

Time-Varying DNS. In 2001, the original attack on Java was extended [36] to use use time-varying DNS:

1. A client visits a malicious web site, `attacker.com`, containing JavaScript. The attacker's DNS server is configured to bind `attacker.com` to the attacker's IP address with a very short TTL.

2. The attacker rebinds `attacker.com` to the target's IP address.

3. The malicious script uses frames or XMLHttpRequest to connect to `attacker.com`, which now resolves to the IP address of the target's server.

Because the connection in Step 3 has the same host name as the original malicious script, the browser permits the attacker to read the response from the target.

Pinning in Current Browsers. Current browsers defend against the standard rebinding attack by "pinning" host names to IP address, preventing host names from referring to multiple IP addresses.

- *Internet Explorer 7* pins DNS bindings for 30 minutes.[1] Unfortunately, if the attacker's domain has multiple `A` records and the current server becomes unavailable, the browser will try a different IP address within one second.

- *Internet Explorer 6* also pins DNS bindings for 30 minutes, but an attacker can cause the browser to release its pin after one second by forcing a connection to the current IP address to fail, for example by including the element `<img src="http://attacker.com:81/">`.

[1]The duration is set by the registry keys `DnsCacheTimeout` and `ServerInfoTimeOut` in `HKEY_CURRENT_USER\SOFTWARE\Microsoft Windows\CurrentVersion\Internet Settings`

- *Firefox 1.5 and 2* cache DNS entries for between 60 and 120 seconds. DNS entries expire when the value of the current minute increments twice. [2] Using JavaScript, the attacker can read the user's clock and compute when the pin will expire. Using multiple `A` records, an attacker can further reduce this time to one second.
- *Opera 9* behaves similarly to Internet Explorer 6. In our experiments, we found that it pins for approximately 12 minutes but can be tricked into releasing its pin after 4 seconds by connecting to a closed port.
- *Safari 2* pins DNS bindings for one second. Because the pinning time is so low, the attacker may need to send a "`Connection: close`" HTTP header to ensure that the browser does not re-use the existing TCP connection to the attacker.

Flash Player 9. Flash Player 9 permits SWF movies to open TCP sockets to arbitrary hosts, provided the destination serves an XML policy authorizing the movie's origin [2]. According to Adobe, Flash Player 9 is installed on 55.8% of web browsers (as of December 2006) [1]; according to our own experiments, Flash Player 9 was present in 86.9% of browsers. Flash Player is vulnerable to the following rebinding attack:

1. The client's web browser visits a malicious web site that embeds a SWF movie.
2. The SWF movie opens a socket on a port less than 1024 to `attacker.com`, bound to the attacker's IP address. Flash Player sends `<policy-file-request />`.
3. The attacker responds with the following XML:

```
<?xml version="1.0"?>
<cross-domain-policy>
 <allow-access-from domain="*" to-ports="*" />
</cross-domain-policy>
```

4. The SWF movie opens a socket to an arbitrary port number on `attacker.com`, which the attacker has rebound to the target's IP address.

The policy XML provided by the attacker in step 3 instructs Flash Player to permit arbitrary socket access to `attacker.com`. Flash Player permits the socket connections to the target because it does not pin host names to a single IP address. If the attacker were to serve the policy file from a port number ≥ 1024, Flash Player would authorize only ports ≥ 1024.

3.2 Multi-Pin Vulnerabilities

Current browsers use several plug-ins to render web pages, many of which permit direct socket access back to their origins. Another class of rebinding attacks exploit the fact that these multiple technologies maintain separate DNS pin databases. If one technology pins to the attacker's IP address and another pins to the target's IP address, the attacker can make use of inter-technology communication to circumvent the same-origin restrictions on network access. Some of these attacks have been discussed previously in the full-disclosure community [4].

Java. Java, installed on 87.6%[3] of web browsers [1], can also open TCP connections back to their origins. The Java Virtual Machine (JVM) maintains DNS pins separately from the browser, opening up the possibility of DNS rebinding vulnerabilities. Java applets themselves are not vulnerable because the JVM retrieves applets directly from the network, permitting the JVM to pin the origin of the applet to the correct IP address. Java is vulnerable, however, to the following attacks.

- *LiveConnect* bridges JavaScript and the JVM in Firefox and Opera, permitting script access to the Java standard library, including the `Socket` class, without loading an applet. The browser pins to the attacker's IP address, but the JVM spawned by LiveConnect does a second DNS resolve and pins to the target's IP address. The attacker's JavaScript can exploit this pin mismatch to open and communicate on a socket from the client machine to an arbitrary IP address on an arbitrary destination port, including UDP sockets with a source port number ≥ 1024.

- *Applets with Proxies* are also vulnerable to a multi-pin attack, regardless of which browser the client uses. If the client uses an HTTP proxy to access the web, there is yet another DNS resolver involved—the proxy. When the JVM retrieves an applet via a proxy, it requests the applet by host name, not by IP address. If the applet opens a socket, the JVM does a second DNS resolve and pins to the target's IP address.

- *Relative Paths* can cause multi-pin vulnerabilities. If a server hosts an HTML page that embeds an applet using a relative path with the parameter `mayscript` set to `true`, that machine can be the target of a multi-pin attack. The browser pins to the target, retrieves the HTML page, and instructs the JVM to load the applet. The JVM does a second DNS resolve, pins to the attacker, and retrieves a malicious applet. The applet instructs the browser, via JavaScript, to issue XMLHttpRequests to the target's IP address.

Flash Player. Flash Player would still be vulnerable to multi-pin attacks even if it pinned DNS bindings. Flash Player does not retrieve its movies directly from the network. Instead, the browser downloads the movie and spawns Flash Player, transferring the movie's origin by host name. When the attacker's movie attempts to open a socket, Flash Player does a second DNS resolution and would pin to the target's IP address. The `URLLoader` class is not vulnerable to multi-pin attacks because it uses the browser to request the URL and thus uses the browser's DNS pins, but the `Socket` class could still be used to read and write on arbitrary TCP sockets.

Other Plug-ins. Other browser plug-ins permit network access, including Adobe Acrobat and Microsoft Silverlight. Acrobat restricts network communication to the SOAP protocol but does not restrict access by document origin. Often, the Acrobat plug-in will prompt the user before accessing the network. Silverlight permits network access through `BrowserHttpWebRequest`, which uses the browser to make the request (like `URLLoader` in Flash Player) and thus uses the browser's DNS pins.

[2] The duration is set by `network.dnsCacheExpiration`.
[3] We observed 98.1% penetration in our experiment.

4. ATTACKS USING DNS REBINDING

An attacker can exploit the DNS rebinding vulnerabilities described in Section 3 to mount a number of attacks. For some of these attacks, the attacker requires the direct socket access afforded by DNS rebinding with Flash Player and Java, whereas others require only the ability to read HTTP responses from the target. The attacks fall into two broad categories, according to the attacker's goal:

- *Firewall Circumvention.* The attacker can use DNS rebinding to access machines behind firewalls that he or she cannot access directly. With direct socket access, the attacker can interact with a number of internal services besides HTTP.
- *IP Hijacking.* The attacker can also use DNS rebinding to access publicly available servers from the client's IP address. This allows the attacker to take advantage of the target's implicit or explicit trust in the client's IP address.

To mount these attacks, the attacker must first induce the client to load some active content. This can be done by a variety of techniques discussed in Section 4.4. Once loaded onto the client's machine, the attacker's code can communicate with any machine reachable by the client.

4.1 Firewall Circumvention

A firewall restricts traffic between computer networks in different zones of trust. Some examples include blocking connections from the public Internet to internal machines and mediating connections from internal machines to Internet servers with application-level proxies. Firewall circumvention attacks bypass the prohibition on inbound connections, allowing the attacker to connect to internal servers while the user is visiting the attacker's Internet web page (see Figure 1).

Spidering the Intranet. The attacker need not specify the target machine by IP address. Instead, the attacker can guess the internal host name of the target, for example `hr.corp.company.com`, and rebind `attacker.com` to a `CNAME` record pointing to that host name. The client's own recursive DNS resolver will complete the resolution and return the IP address of the target. Intranet host names are often guessable and occasionally disclosed publicly [30, 9]. This technique obviates the need for the attacker to scan IP addresses to find an interesting target but does not work with the multiple `A` record technique described in Section 3.1.

Having found a machine on the intranet, the attacker can connect to the machine over HTTP and request the root document. If the server responds with an HTML page, the attacker can follow links and search forms on that page, eventually spidering the entire intranet. Web servers inside corporate firewalls often host confidential documents, relying on the firewall to prevent untrusted users from accessing the documents. Using a DNS rebinding attack, the attacker can leverage the client's browser to read these documents and exfiltrate them to the attacker, for example by submitting an HTML form to the attacker's web server.

Compromising Unpatched Machines. Network administrators often do not patch internal machines as quickly as Internet-facing machines because the patching process is time-consuming and expensive. The attacker can attempt to exploit known vulnerabilities in machines on the internal network. In particular, the attacker can attempt to exploit the client machine itself. The attacks against the client itself originate from `localhost` and so bypass software firewalls and other security checks, including many designed to protect serious vulnerabilities. If an exploit succeeds, the attacker can establish a presence within the firewall that persists even after clients close their browsers.

Abusing Internal Open Services. Internal networks contain many open services intended for internal use only. For example, network printers often accept print jobs from internal machines without additional authentication. The attacker can use direct socket access to command network printers to exhaust their toner and paper supplies.

Similarly, users inside firewalls often feel comfortable creating file shares or FTP servers accessible to anonymous users under the assumption that the servers will be available only to clients within the network. With the ability to read and write arbitrary sockets, the attacker can exfiltrate the shared documents and use these servers to store illicit information for later retrieval.

Consumer routers are often installed without changing the default password, making them an attractive target for reconfiguration attacks by web pages [40]. Firmware patches have attempted to secure routers against cross-site scripting and cross-site request forgery, in an effort to prevent reconfiguration attacks. DNS rebinding attacks allow the attacker direct socket access to the router, bypassing these defenses.

4.2 IP Hijacking

Attackers can also use DNS rebinding attacks to target machines on the public Internet. For these attacks, the attacker is not leveraging the client's machine to connect to otherwise inaccessible services but instead abusing the implicit or explicit trust public services have in the client's IP address. Once the attacker has hijacked a client's IP address, there are several attacks he or she can perpetrate.

Committing Click Fraud. Web publishers are often paid by web advertisers on a per-click basis. Fraudulent publishers can increase their advertising revenue by generating fake clicks, and advertisers can drain competitors' budgets by clicking on their advertisements. The exact algorithms used by advertising networks to detect these "invalid" clicks are proprietary, but the IP address initiating the click is widely believed to be an essential input. In fact, one common use of bot networks is to generate clicks [7].

Click fraud would appear to require only the ability to send HTTP requests to the advertising network, but advertisers defend against the send-only attacks, permitted by the same-origin policy, by including a unique nonce with every advertising impression. Clicks lacking the correct nonce are rejected as invalid, requiring the attacker to read the nonce from an HTTP response in order to generate a click.

This attack is highly cost-effective, as the attacker can buy advertising *impressions*, which cost tens of cents per thousand, and convert them into *clicks*, worth tens of cents each. The attack is sufficiently cost-effective that the attacker need not convert every purchased impression into a click. Instead, the fraudster can use most of the purchased impressions to generate fake impressions on the site, maintaining a believable click-through rate.

Sending Spam. Many e-mail servers blacklist IP addresses known to send spam e-mail [39]. By hijacking a client's IP address, an attacker can send spam from IP addresses with clean reputations. To send spam e-mail, the attacker need only write content to SMTP servers on port 25, an action blocked by most browsers but permitted by Flash Player and Java. Additionally, an attacker will often be able to use the client's actual mail relay. Even service providers that require successful authentication via POP3 before sending e-mail are not protected, because users typically leave their desktop mail clients open and polling their POP3 servers.

Defeating IP-based Authentication. Although discouraged by security professionals [10], many Internet services still employ IP-based authentication. For example, the ACM Digital Library makes the full text of articles available only to subscribers, who are often authenticated by IP address. After hijacking an authorized IP address, the attacker can access the service, defeating the authentication mechanism. Because the communication originates from an IP address actually authorized to use the service, it can be difficult, or even impossible, for the service provider to recognize the security breach.

Framing Clients. An attacker who hijacks an IP address can perform misdeeds and frame the client. For example, an attacker can attempt to gain unauthorized access to a computer system using a hijacked IP address as a proxy. As the attack originates from the hijacked IP address, the logs will implicate the client, not the attacker, in the crime. Moreover, if the attacker hosts the malicious web site over HTTPS, the browser will not cache the page and no traces will be left on the client's machine.

4.3 Proof-of-Concept Demonstration

We developed proof-of-concept exploits for DNS rebinding vulnerabilities in Flash Player 9, LiveConnect, Java applets with proxy servers, and the browser itself. Our system consists of a custom DNS server authoritative for `dnsrebinding.net`, a custom Flash Player policy server, and a standard Apache web server. The various technologies issue DNS queries that encode the attacker and target host names, together with a nonce, in the subdomain. For each nonce, the DNS server first responds with the attacker's IP address (with a zero TTL) and thereafter with the target's IP address. Our proof-of-concept demo, `http://crypto.stanford.edu/dns`, implements `wget` and `telnet` by mounting a rebinding attack against the browser.

4.4 Experiment: Recruiting Browsers

Methodology. We tested DNS rebinding experimentally by running a Flash Player 9 advertisement on a minor advertising network targeting the keywords "Firefox," "game," "Internet Explorer," "video," and "YouTube." The experiment used two machines in our laboratory, an attacker and a target. The attacker ran a custom authoritative DNS server for `dnsrebinding.net`, a custom Flash Player policy server, and an Apache web server hosting the advertisement. The target ran an Apache web server to log successful attacks. The Flash Player advertisement exploited the vulnerability described in Section 3.1 to load an XML document from the target server in our lab. The attack required only that the client view the ad, not that the user click on the ad.

Vulnerability	Impressions
Flash Player 9	86.9%
LiveConnect	24.4%
Java+Proxy	2.2%
Total Multi-Pin	90.6%

Table 2: Percentage of Impressions by Vulnerability

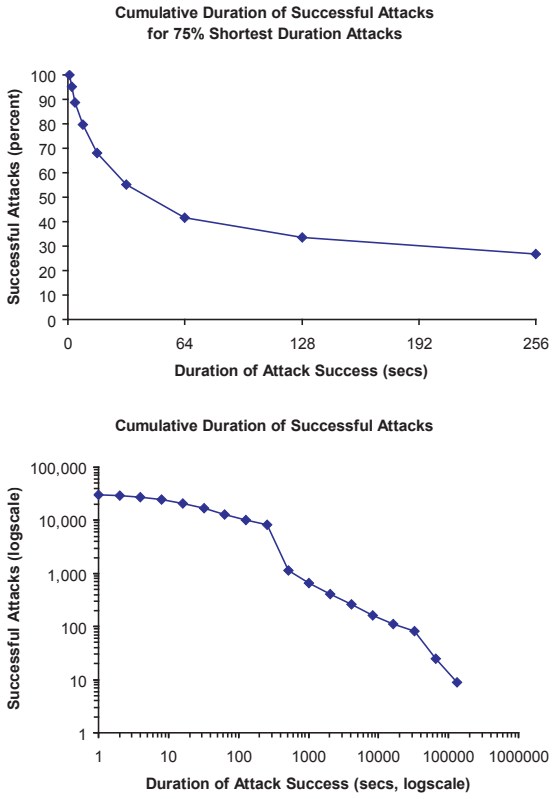

Figure 2: Duration of Successful Attacks

The experiment lasted until the user navigated away from the advertisement, at which time we lost the ability to use the viewer's network connection. For privacy, we collected only properties typically disclosed by browsers when viewing web pages (e.g., plug-in support, user agent, and external IP address). The experiment conformed to the terms of service of the advertising network and to the guidelines of the independent review board at our institution. Every network operation produced by the advertisement could have been produced by a legitimate SWF advertisement, but we produced the operations through the `Socket` interface, demonstrating the ability to make arbitrary TCP connections.

Results. We ran the ad beginning at midnight EDT on three successive nights in late April 2007. We bid $0.50 per 1000 impressions for a variety of keywords. We spent $10 per day, garnering approximately 20,000 impressions per day. Due to a server misconfiguration, we disregarded approximately 10,000 impressions. We also disregarded 19 impressions from our university. We received 50,951 impressions from 44,924 unique IP addresses (40.2% IE7, 32.3% IE6, 23.5% Firefox, 4% Other).

We ran the rebinding experiment on the 44,301 (86.9%) impressions that reported Flash Player 9. We did not attempt to exploit other rebinding vulnerabilities (see Table 2). The experiment was successful on 30,636 (60.1%) impressions and 27,480 unique IP addresses. The attack was less successful on the 1,672 impressions served to Mac OS, succeeding 36.4% of the time, compared to a success rate of 70.0% on the 49,535 (97.2%) Windows impressions.[4] Mac OS is more resistant to this rebinding attack due to some caching of DNS entries despite their zero TTL.

For each successful experiment, we measured how long an attacker could have used the client's network access by loading the target document at exponentially longer intervals, as shown in Figure 2. The median impression duration was 32 seconds, with 25% of the impressions lasting longer than 256 seconds. We observed 9 impressions with a duration of at least 36.4 hours, 25 at least 18.2 hours, and 81 at least 9.1 hours. In aggregate, we obtained 100.3 machine-days of network access. These observations are consistent with those of [24]. The large number of attacks ending between 4.2 and 8.5 minutes suggests that this is a common duration of time for users to spend on a web page.

Discussion. Our experimental results show that DNS rebinding vulnerabilities are widespread and cost-effective to exploit on a large scale. Each impression costs $0.0005 and 54% of the impressions convert to successful attacks from unique IP addresses. To hijack 100,000 IP addresses for a temporary bot network, and attacker would need to spend less than $100. This technique compares favorably to renting a traditional bot network for sending spam e-mail and committing click fraud for two reasons. First, these applications require large numbers of "fresh" IP address for short durations as compromised machines are quickly blacklisted. Second, while estimates of the rental cost of bot networks vary [44, 14, 7], this technique appears to be at least one or two orders of magnitude less expensive.

5. DEFENSES AGAINST REBINDING

Defenses for DNS rebinding attacks can be implemented in browsers, plug-ins, DNS resolvers, firewalls, and servers. These defenses range in complexity of development, difficulty of deployment, and effectiveness against firewall circumvention and IP hijacking. In addition to necessary mitigations for Flash Player, Java LiveConnect, and browsers, we propose three long-term defenses. To protect against firewall circumvention, we propose a solution that can be deployed unilaterally by organizations at their network boundary. To fully defend against rebinding attacks, we propose two defenses: one that requires socket-level network access be authorized explicitly by the destination server and another works even if sockets are allowed by default.

5.1 Fixing Firewall Circumvention

Networks can be protected against firewall circumvention by forbidding external host names from resolving to internal IP addresses, effectively preventing the attacker from naming the target server. Without the ability to name the target, the attacker is unable to aggregate the target server into an origin under his or her control. These malicious bindings can be blocked either by filtering packets at the firewall [5] or by modifying the DNS resolvers used by clients on the network.

- *Enterprise.* By blocking outbound traffic on port 53, a firewall administrator for an organization can force all internal machines, including HTTP proxies and VPN clients, to use a DNS server that is configured not to resolve external names to internal IP addresses. To implement this approach, we developed a 300 line `C` program, `dnswall` [15], that runs alongside BIND and enforces this policy.

- *Consumer.* Many consumer firewalls, such as those produced by Linksys, already expose a caching DNS resolver and can be augmented with `dnswall` to block DNS responses that contain private IP addresses. The vendors of these devices have an incentive to patch their firewalls because these rebinding attacks can be used to reconfigure these routers to mount further attacks on their owners.

- *Software.* Software firewalls, such as the Windows Firewall, can also prevent their own circumvention by blocking DNS resolutions to `127.*.*.*`. This technique does not defend services bound to the external network interface but does protects a large number of services that bind only to the loopback interface.

Blocking external names from resolving to internal addresses prevents firewall circumvention but does not defend against IP hijacking. An attacker can still use internal machines to attack services running on the public Internet.

5.2 Fixing Plug-ins

Plug-ins are a particular source of complexity in defending against DNS rebinding attacks because they enable subsecond attacks, provide socket-level network access, and operate independently from browsers. In order to prevent rebinding attacks, these plug-ins must be patched.

Flash Player. When a SWF movie opens a socket to a new host name, it requests a policy over the socket to determine whether the host accepts socket connections from the origin of the movie. Flash Player could fix most of its rebinding vulnerabilities by considering a policy valid for a socket connection only if it obtained the policy from the *same IP address* in addition to its current requirement that it obtained the policy from the same host name. Using this design, when `attacker.com` is rebound to the target IP address, Flash Player will refuse to open a socket to that address unless the target provides a policy authorizing `attacker.com`. This simple refinement uses existing Flash Player policy deployments and is backwards compatible, as host names expecting Flash Player connections already serve policy documents from all of their IP addresses.

SWF movies can also access ports numbers $\geq$ 1024 on their origin host name without requesting a policy. Although the majority of services an attacker can profitably target (e.g., SMTP, HTTP, HTTPS, SSH, FTP, NNTP) are hosted on low-numbered ports, other services such as MySQL, BitTorrent, IRC, and HTTP proxies are vulnerable. To fully protect against rebinding attacks, Flash Player could request a policy before opening sockets to any port, even back to its origin. However, this modification breaks

[4]We succeeded in opening a socket with 2 of 11 PlayStation 3 impressions (those with Flash Player 9), but none of the 12 Nintendo Wii impressions were vulnerable.

backwards compatibility because those servers might not be already serving policy files.

Java. Many deployed Java applets expect sockets to be allowed by default. If clients are permitted to use these applets from behind HTTP proxies, they will remain vulnerable to multi-pin attacks because proxy requests are made by host name instead of by IP address. A safer approach is to use the `CONNECT` method to obtain a proxied socket connection to an external machine. Typically proxies only allow `CONNECT` on port 443 (HTTPS), making this the only port available for these applets. Alternatively, proxies can use HTTP headers to communicate IP addresses of hosts between the client and the proxy [28, 29], but this approach requires both the client and the proxy to implement the protocol.

Java LiveConnect. LiveConnect introduces additional vulnerabilities, but browsers can fix the LiveConnect multi-pin vulnerability without altering the JVM by installing their own DNS resolver into the JVM using a standard interface. Firefox, in particular, implements LiveConnect through the Java Native Interface (JNI). When Firefox initializes the JVM, it can install a custom `InetAddress` class that will handle DNS resolution for the JVM. This custom class should contain a native method that implements DNS resolution using Firefox's DNS resolver instead of the system resolver. If the browser implements pinning, LiveConnect and the browser will use a common pin database, removing multi-pin vulnerabilities.

5.3 Fixing Browsers (Default-Deny Sockets)

Allowing direct socket access by default precludes many defenses for DNS rebinding attacks. If browser plug-ins defaulted to denying socket access, as a patched Flash Player and the proposed TCPConnection (specified in HTML5 [19]) would, these defenses would become viable. Java and LiveConnect, along with any number of lesser-known plug-ins, expect socket access to be allowed, and fixing these is a challenge.

Checking Host Header. HTTP 1.1 requires that user agents include a `Host` header in HTTP requests that specifies the host name of the server [11]. This feature is used extensively by HTTP proxies and by web servers to host many virtual hosts on one IP address. If sockets are denied by default, the `Host` header reliably indicates the host name being used by the browser to contact the server because XMLHttpRequest [43] and related technologies are restricted from spoofing the `Host` header.[5] One server-side defense for these attacks is therefore to reject incoming HTTP requests with unexpected `Host` headers [28, 37].

Finer-grained Origins. Another defense against DNS rebinding attacks is to refine origins to include additional information, such as the server's IP address [28] or public key [27, 23], so that when the attacker rebinds `attacker.com` to the target, the browser will consider the rebound host name to be a new origin. One challenge to deploying finer-grained origins is that every plug-in would need to revise its security policies and interacting technologies would need to hand-off refined origins correctly.

[5]Lack of integrity of the `Host` header has been a recurring source of security vulnerabilities, most notably in Flash Player 7.

- *IP Addresses.* Refining origins with IP address [28] is more robust than pinning in that a single browsing session can fail-over from one IP address to another. When such a fail-over occurs, however, it will likely break long-lived AJAX applications, such as Gmail, because they will be prevented from making XMLHttpRequests to the new IP address. Users can recover from this by clicking the browser's reload button. Unfortunately, browsers that use a proxy server do not know the actual IP address of the remote server and thus cannot properly refine origins. Also, this defense is vulnerable to an attack using relative paths to script files, similar to the applet relative-path vulnerability described in Section 3.2.

- *Public Keys.* Augmenting origins with public keys [27, 23] prevents two HTTPS pages served from the same domain with different public keys from reading each other's state. This defense is useful when users dismiss HTTPS invalid certificate warnings and chiefly protects HTTPS-only "secure" cookies from network attackers. Many web pages, however, are not served over HTTPS, rendering this defense more appropriate for pharming attacks that compromise victim domains than for rebinding attacks.

Smarter Pinning. To mitigate rebinding attacks, browsers can implement smarter pinning policies. Pinning is a defense for DNS rebinding that trades off robustness for security. RFC 1035 [32] provides for small (and even zero) TTLs to enable dynamic DNS and robust behavior in the case of server failure but respecting these TTLs allows rebinding attacks. Over the last decade, browsers have experimented with different pin durations and release heuristics, leading some vendors to shorten their pin duration to improve robustness [13]. However, duration is not the only parameter that can be varied in a pinning policy.

Browsers can vary the *width* of their pins by permitting host names to be rebound within a set of IP addresses that meet some similarity heuristic. Selecting an optimal width as well as duration enables a better trade-off between security and robustness than optimizing duration alone. One promising policy is to allow rebinding within a class C network. For example, if a host name resolved to `171.64.78.10`, then the client would also accept any IP address beginning with `171.64.78` for that host name. The developers of the NoScript Firefox extension [26] have announced plans [25] to adopt this pinning heuristic.

- *Security.* When browsers use class C network pinning, the attacker must locate the attack server on the same class C network as the target, making the rebinding attack much more difficult to mount. The attack is possible only if the attacker co-locates a server at the same hosting facility or leverages a cross-site scripting vulnerability on a co-located server. This significantly raises the bar for the attacker and provides better recourses for the target.

- *Robustness.* To study the robustness of class C network pinning, we investigated the IP addresses reported by the 100 most visited English-language sites (according to Alexa [3]). We visited the home page of these sites and compiled a list of the 336 host names

used for embedded content (e.g., `www.yahoo.com` embeds images from `us.i1.yimg.com`). We then issued DNS queries for these hosts every 10 minutes for 24 hours, recording the IP addresses reported.

In this experiment, 58% reported a single IP address consistently across all queries. Note that geographic load balancing is not captured in our data because we issued our queries from a single machine, mimicking the behavior of a real client. Averaged over the 42% of hosts reporting multiple IP addresses, if a browser pinned to an IP address at random, the expected fraction of IP addresses available for rebinding under class C network pinning is 81.3% compared with 16.4% under strict IP address pinning, suggesting that class C pinning is significantly more robust to server failure.

Other heuristics for pin width are possible. For example, the browser could prevent rebinding between public IP addresses and the RFC 1918 [35] private IP addresses. This provides greater robustness for fail-overs across data centers and for dynamic DNS. LocalRodeo [22, 45] is a Firefox extension that implements RFC 1918 pinning for JavaScript. As for security, RFC 1918 pinning largely prevents firewall circumvention but does not protect against IP hijacking nor does it prevent firewall circumvention in the case where a firewall protects non-private IP addresses, which is the case for many real-life protected networks and personal software firewalls.

Even the widest possible pinning heuristic prevents some legitimate rebinding of DNS names. For example, public host names controlled by an organization often have two IP addresses, a private IP address used by clients within the firewall and a public IP address used by clients on the Internet. Pinning prevents employees from properly connecting to these severs after joining the organization's Virtual Private Network (VPN) as those host names appear to rebind from public to private IP addresses.

Policy-based Pinning. Instead of using unpinning heuristics, we propose browsers consult server-supplied policies to determine when it is safe to re-pin a host name from one IP address to another, providing robustness without degrading security. To re-pin safely, the browser must obtain a policy from both the old and new IP address (because some attacks first bind to the attacker whereas others first bind to the target). Servers can supply this policy at a well-known location, such as `/crossdomain.xml`, or in reverse DNS (see Section 5.4).

Pinning Pitfalls. Correctly implementing pinning has several subtleties that are critical to its ability to defend against DNS rebinding attacks.

- *Common Pin Database.* To eliminate multi-pin attacks, pinning-based defense require that all browser technologies that access the network share a common pin database. Many plug-ins, including Flash Player and Silverlight, already use the browser's pins when issuing HTTP requests because they issue these requests through the browser. To share DNS pins for other kinds of network access, either the browser could expose an interface to its pin database or the operating system could pin in its DNS resolver. Unfortunately, browser vendors appear reluctant to expose such an interface [12, 33] and pinning in the operating system either changes the semantics of DNS for other applications or requires that the OS treats browsers and their plug-ins differently from other applications.

- *Cache.* The browser's cache and all plug-in caches must be modified to prevent rebinding attacks. Currently, objects stored in the cache are retrieved by URL, irrespective of the originating IP address, creating a rebinding vulnerability: a cached script from the attacker might run later when `attacker.com` is bound to the target. To prevent this attack, objects in the cache must be retrieved by both URL and originating IP address. This degrades performance when the browser pins to a new IP address, which might occur when the host at the first IP address fails, the user starts a new browsing session, or the user's network connectivity changes. These events are uncommon and are unlikely to impact performance significantly.

- *document.domain.* Even with the strictest pinning, a server is vulnerable to rebinding attacks if it hosts a web page that executes the following, seemingly innocuous, JavaScript:

  ```
  document.domain = document.domain;
  ```

  After a page sets its `domain` property, the browser allows cross-origin interactions with other pages that have set their `domain` property to the same value [42, 17]. This idiom, used by a number of JavaScript libraries[6], sets the `domain` property to a value under the control of the attacker: the current host name.

5.4 Fixing Browsers (Default-Allow Sockets)

Instead of trying to prevent a host name from rebinding from one IP address to another—a fairly common event—a different approach to defending against rebinding is to prevent the attacker from naming the target server, essentially generalizing `dnswall` to the Internet. Without the ability to name the target server, the attacker cannot mount a DNS rebinding attack against the target. This approach defends against rebinding, can allow socket access by default, and preserves the robustness of dynamic DNS.

Host Name Authorization. On the Internet, clients require additional information to determine the set of valid host names for an given IP address. We propose that servers advertise the set of host names they consider valid for themselves and clients check these advertisements before binding a host name to an IP address, making explicit which host names can map to which IP addresses. Host name authorization prevents rebinding attacks because honest machines will not advertise host names controlled by attackers.

Reverse DNS already provides a mapping from IP addresses to host names. The owner of an IP address *ip* is delegated naming authority for *ip*`.in-addr.arpa` and typically stores a `PTR` record containing the host name associated with that IP address. These records are insufficient for host name authorization because a single IP address can have many valid host names, and existing `PTR` records do not indicate that other host names are invalid.

[6]For example, "Dojo" AJAX library, Struts servlet/JSP based web application framework, jsMath AJAX Mathematics library, and Sun's "Ultimate client-side JavaScript client sniff" library are vulnerable in this way.

The reverse DNS system can be extended to authorize host names without sacrificing backwards compatibility. To authorize the host `www.example.com` for `171.64.78.146`, the owner of the IP address inserts the following DNS records:

```
auth.146.78.64.171.in-addr.arpa.
    IN A 171.64.78.146
www.example.com.auth.146.78.64.171.in-addr.arpa.
    IN A 171.64.78.146
```

To make a policy-enabled resolution for `www.example.com`, first resolve the host name a set of IP addresses normally and then validate each IP address as follows:

1. Resolve the host name `auth.`*ip*`.in-addr.arpa`.
2. If the host name exists, *ip* is policy-enabled and accepts only authorized host names. Otherwise, *ip* is not policy-enabled and accepts any host name.
3. Finally, if *ip* is policy-enabled, resolve the host name

   `www.example.com.auth.`*ip*`.in-addr.arpa`

   to determine if the host name is authorized.

An IP address *ip* implicitly authorizes every host name of the form `*.auth.`*ip*`.in-addr.arpa`, preventing incorrect recursive policy checks. For host names with multiple IP addresses, only authorized IP addresses should be included in the result. If no IP addresses are authorized, the result should be "not found." If an IP address is not policy enabled, DNS rebinding attacks can be mitigated using the techniques in Section 5.3.

The policy check can be implemented in DNS resolvers[7], such as ones run by organizations and ISPs, transparently protecting large groups of machines from having their IP addresses hijacked. User agents, such as browser and plug-ins, can easily query the policy records because they are stored in `A` records and can issue policy checks in parallel with HTTP requests (provided they do not process the HTTP response before the host name is authorized). Standard DNS caching reduces much of the overhead of redundant policy checks issued by DNS resolvers, browsers, and plug-ins. As a further optimization, policy-enabled resolvers can include policy records in the "additional" section of the DNS response, allowing downstream resolvers to cache complete policies and user-agents to get policy records without a separate request. We have implemented host name authorization as a 72-line patch to Firefox 2.

One disadvantage of this mechanism is that the owner of an IP address, the ISP, might not be the owner of the machine at that IP address. The machine can advertise the correct set of authorized host names only if the ISP is willing to delegate the `auth` subdomain to the owner or insert appropriate DNS records. Instead, machines could advertise authorized host names over HTTP in a well-known location, similar to `crossdomain.xml`, but this has several disadvantages: it requires policy-enabled DNS resolvers to implement HTTP clients, it requires all machines, such as SMTP gateways, to run an HTTP server, and policy queries are not cached, resulting in extra traffic comparable to `favicon.ico`.

[7]To prevent a subtle attack that involves poisoning DNS caches, a policy-enabled DNS resolver must follow the same procedure for `CNAME` queries as for `A` queries, even though responses to the former do not directly include IP addresses.

Trusted Policy Providers. Clients and DNS resolvers can also check policy by querying a trusted policy provider. Much like spam black lists [39] and phishing filters [6, 31, 16], different policy providers can use different heuristics to determine whether a host name is valid for an IP address, but every provider should respect host names authorized in reverse DNS. When correctly configured, host name authorization in reverse DNS has no false negatives (no valid host name is rejected) but many false positives (lack of policy is implicit authorization). Trusted policy providers can greatly reduce the false positive rate, possibly at the cost of increasing the false negative rate. Clients are free to select as aggressive a policy provider as they desire.

6. RELATED WORK

Using Browsers as Bots. The technique of luring web users to an attacker's site and then distracting them while their browsers participate in a coordinated attack is described in [24]. These "puppetnets" can be used for distributed denial of service but cannot be used to mount the attacks described in Section 4 because puppetnets cannot read back responses from different origins or connect to forbidden ports such as 25.

JavaScript can also be misused to scan behind firewalls [18] and reconfigure home routers [40]. These techniques often rely on exploiting default passwords and on underlying cross-site scripting or cross-site request forgery vulnerabilities. DNS rebinding attacks can be used to exploit default passwords without the need for a cross-site scripting or cross-site request forgery hole.

Sender Policy Framework. To fight spam e-mail, the Sender Policy Framework (SPF) [46] stores policy information in DNS. SPF policies are stored as `TXT` records in forward DNS, where host names can advertise the set of IP addresses authorized to send e-mail on their behalf.

7. CONCLUSIONS

An attacker can exploit DNS rebinding vulnerabilities to circumvent firewalls and hijack IP addresses. Basic DNS rebinding attacks have been known for over a decade, but the classic defense, pinning, reduces robustness and fails to protect current browsers that use plug-ins. Modern multi-pin attacks defeat pinning in hundreds of milliseconds, granting the attacker direct socket access from the client's machine. These attacks are a highly cost-effective technique for hijacking hundreds of thousands of IP addresses for sending spam e-mail and committing click fraud.

For network administrators, we provide a tool to prevent DNS rebinding from being used for firewall circumvention by blocking external DNS names from resolving to internal IP addresses. For the vendors of Flash Player, Java, and LiveConnect, we suggest simple patches that mitigate large-scale exploitation by vastly reducing the cost-effectiveness of the attacks for sending spam e-mail and committing click fraud. Finally, we propose two defense options that prevent both firewall circumvention and IP hijacking: policy-based pinning and host name authorization. We hope that vendors and network administrators will deploy these defenses quickly before attackers exploit DNS rebinding on a large scale.

Acknowledgments

We thank Drew Dean, Darin Fisher, Jeremiah Grossman, Martin Johns, Dan Kaminsky, Chris Karlof, Jim Roskind, and Dan Wallach for their helpful suggestions and feedback. This work is supported by grants from the National Science Foundation and the US Department of Homeland Security.

8. REFERENCES

[1] Adobe. Flash Player Penetration. http://www.adobe.com/products/player_census/flashplayer/.

[2] Adobe. Adobe Flash Player 9 Security. http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf, July 2006.

[3] Alexa. Top sites. http://www.alexa.com/site/ds/top_sites?ts_mode=global.

[4] K. Anvil. Anti-DNS pinning + socket in flash. http://www.jumperz.net/, 2007.

[5] W. Cheswick and S. Bellovin. A DNS filter and switch for packet-filtering gateways. In *Proc. Usenix*, 1996.

[6] N. Chou, R. Ledesma, Y. Teraguchi, and J. Mitchell. Client-side defense against web-based identity theft. In *Proc. NDSS*, 2004.

[7] N. Daswani, M. Stoppelman, et al. The anatomy of Clickbot.A. In *Proc. HotBots*, 2007.

[8] D. Dean, E. W. Felten, and D. S. Wallach. Java security: from HotJava to Netscape and beyond. In *IEEE Symposium on Security and Privacy: Oakland, California*, May 1996.

[9] D. Edwards. Your MOMA knows best, December 2005. http://xooglers.blogspot.com/2005/12/your-moma-knows-best.html.

[10] K. Fenzi and D. Wreski. Linux security HOWTO, January 2004.

[11] R. Fielding et al. Hypertext Transfer Protocol—HTTP/1.1. RFC 2616, June 1999.

[12] D. Fisher, 2007. Personal communication.

[13] D. Fisher et al. Problems with new DNS cache ("pinning" forever). https://bugzilla.mozilla.org/show_bug.cgi?id=162871.

[14] D. Goodin. Calif. man pleads guilty to felony hacking. *Associated Press*, Janurary 2005.

[15] Google. dnswall. http://code.google.com/p/google-dnswall/.

[16] Google. Google Safe Browsing for Firefox, 2005. http://www.google.com/tools/firefox/safebrowsing/.

[17] S. Grimm et al. Setting document.domain doesn't match an implicit parent domain. https://bugzilla.mozilla.org/show_bug.cgi?id=183143.

[18] J. Grossman and T. Niedzialkowski. Hacking intranet websites from the outside: JavaScript malware just got a lot more dangerous. In *Blackhat USA*, August 2006. Invited talk.

[19] I. Hickson et al. HTML 5 Working Draft. http://www.whatwg.org/specs/web-apps/current-work/.

[20] C. Jackson, A. Bortz, D. Boneh, and J. Mitchell. Protecting browser state from web privacy attacks. In *Proc. WWW*, 2006.

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

[22] M. Johns and J. Winter. Protecting the Intranet against "JavaScript Malware" and related attacks. In *Proc. DIMVA*, July 2007.

[23] C. K. Karlof, U. Shankar, D. Tygar, and D. Wagner. Dynamic pharming attacks and the locked same-origin policies for web browsers. In *Proc. CCS*, October 2007.

[24] V. T. Lam, S. Antonatos, P. Akritidis, and K. G. Anagnostakis. Puppetnets: Misusing web browsers as a distributed attack infrastructure. In *Proc. CCS*, 2006.

[25] G. Maone. DNS Spoofing/Pinning. http://sla.ckers.org/forum/read.php?6,4511,14500.

[26] G. Maone. NoScript. http://noscript.net/.

[27] C. Masone, K. Baek, and S. Smith. WSKE: web server key enabled cookies. In *Proc. USEC*, 2007.

[28] A. Megacz. XWT Foundation Security Advisory. http://xwt.org/research/papers/sop.txt.

[29] A. Megacz and D. Meketa. X-RequestOrigin. http://www.xwt.org/x-requestorigin.txt.

[30] Microsoft. Microsoft Web Enterprise Portal, January 2004. http://www.microsoft.com/technet/itshowcase/content/MSWebTWP.mspx.

[31] Microsoft. Microsoft phishing filter: A new approach to building trust in e-commerce content, 2005.

[32] P. Mockapetris. Domain Names—Implementation and Specification. IETF RFC 1035, November 1987.

[33] C. Nuuja (Adobe), 2007. Personal communication.

[34] G. Ollmann. The pharming guide. http://www.ngssoftware.com/papers/ThePharmingGuide.pdf, August 2005.

[35] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. IETF RFC 1918, February 1996.

[36] J. Roskind. Attacks against the Netscape browser. In *RSA Conference*, April 2001. Invited talk.

[37] D. Ross. Notes on DNS pinning. http://blogs.msdn.com/dross/archive/2007/07/09/notes-on-dns-pinning.aspx, 2007.

[38] J. Ruderman. JavaScript Security: Same Origin. http://www.mozilla.org/projects/security/components/same-origin.html.

[39] Spamhaus. The spamhaus block list, 2007. http://www.spamhaus.org/sbl/.

[40] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. Technical Report 641, Computer Science, Indiana University, December 2006.

[41] J. Topf. HTML Form Protocol Attack, August 2001. http://www.remote.org/jochen/sec/hfpa/hfpa.pdf.

[42] D. Veditz et al. document.domain abused to access hosts behind firewall. https://bugzilla.mozilla.org/show_bug.cgi?id=154930.

[43] W3C. The XMLHttpRequest Object, February 2007. http://www.w3.org/TR/XMLHttpRequest/.

[44] B. Warner. Home PCs rented out in sabotage-for-hire racket. *Reuters*, July 2004.

[45] J. Winter and M. Johns. LocalRodeo: Client-side protection against JavaScript Malware. http://databasement.net/labs/localrodeo/, 2007.

[46] M. Wong and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail. IETF RFC 4408, April 2006.