Protecting Browsers from DNS Rebinding Attacks

COLLIN JACKSON, ADAM BARTH, ANDREW BORTZ, WEIDONG SHAO, and DAN BONEH
Stanford University

DNS rebinding attacks subvert the same-origin policy of browsers, converting them into open network proxies. Using DNS rebinding, an attacker can circumvent organizational and personal firewalls, send spam email, and defraud pay-per-click advertisers. We evaluate the cost effectiveness of mounting DNS rebinding attacks, finding that an attacker requires less than $100 to hijack 100,000 IP addresses. We analyze defenses to DNS rebinding attacks, including improvements to the classic "DNS pinning," and recommend changes to browser plug-ins, firewalls, and Web servers. Our defenses have been adopted by plug-in vendors and by a number of open-source firewall implementations.

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

General Terms: Security, Design, Experimentation

Additional Key Words and Phrases: Same-origin policy, DNS, firewall, spam, click fraud

ACM Reference Format:

Jackson, C., Barth, A., Bortz, A., Shao, W., and Boneh, D. 2008. Protecting browsers from DNS rebinding attacks. ACM Trans. Web 3, 1, Article 2 (January 2009), 26 pages. DOI = 10.1145/1462148.1462150 http://doi.acm.org/10.1145/1462148.1462150

1. INTRODUCTION

Web users 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. Modern browsers meet these expectations

A preliminary version of this article appeared in *Proceedings of the 14th ACM Conference on Computer and Communication Security (CCS)*, 421–431. This work is supported by grants from the National Science Foundation and the U.S. Department of Homeland Security.
Authors' addresses: C. Jackson (corresponding author), A. Barth, A. Bortz, W. Shao, D. Boneh, Department of Computer Science, Stanford University, 450 Serra Mall, Stanford, CA 94305; email: collinj@cs.stanford.edu.

by implementing a security policy, based on the *same-origin principle*, that protects honest sites from dishonest sites by isolating content from distinct "origins." DNS rebinding attacks subvert the same-origin principle by confusing the browser into mixing content controlled by distinct entities into a single security origin.

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 a user's IP address to send spam email and to defraud pay-per-click advertisers. DNS rebinding vulnerabilities let the attacker read and write directly on network sockets, greatly expanding on the attacks possible with existing JavaScript-based botnets [Lam et al. 2006], which can send HTTP requests to foreign hosts but cannot read the HTTP responses.

Basic DNS rebinding attacks have been known for over a decade [Dean et al. 1996; Roskind 2001]. In the classic attack an attacker need only register a domain name, such as `attacker.com`, and attract Web traffic, for example, by running an advertisement. The attacker answers DNS queries for `attacker.com` with the IP address of his or her own server and a short Time-To-Live (TTL). The user's browser issues an HTTP request to the attacker's server, which responds with a malicious HTML document. The attacker's document then issues a second HTTP request to `attacker.com`. The user's DNS cache has expired, causing the browser to issue another DNS query for `attacker.com`. This time, the attacker's DNS server responds with the IP address of a target server. The browser allows the attacker's script to read HTTP responses from the target server because the two connections share a single host name and therefore belong to the same browser security origin. If the user is behind a firewall, the attacker can use this attack to read sensitive documents hosted on servers behind the firewall, as shown in Figure 1.

To mount a DNS rebinding attack, an attacker need not compromise any DNS servers. The attacker simply provides valid, authoritative responses for `attacker.com`, a domain owned by the attacker. Unlike "pharming" attacks [Ollmann 2005] in which the attacker must compromise an honest host name, DNS rebinding attacks can be mounted without unusual network privileges. DNSSEC [Arends et al. 2005] provides no protection against DNS rebinding attacks: The attacker can legitimately sign all DNS records for `attacker.com` because the attacker owns the domain.

Pinning

A common DNS rebinding defense implemented in several browsers is *DNS pinning*: Once the browser has resolved a host name to an IP address, the browser caches the result for a fixed duration, regardless of TTL. With pinning, all network connections to a given host name will be routed to the same IP address, preventing the attacker from mixing malicious content with content from an honest server.

Pinning is no longer an effective defense against DNS rebinding attacks because of vulnerabilities introduced by browser plug-ins. These plug-ins provide

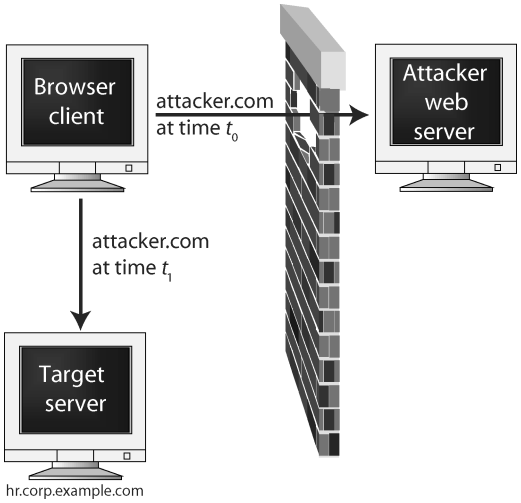

Fig. 1. Firewall circumvention using DNS rebinding.

additional functionality, such as socket-level network access, to Web content. The browser and each plug-in maintain separate pin databases, creating a new class of DNS rebinding vulnerabilities we refer to as *multipin* vulnerabilities. We demonstrate, for example, how to use the interaction between the browser and Java LiveConnect to pin the browser to one IP address while pinning Java to another IP address. An attacker can exploit multipin vulnerabilities to read and write directly on sockets to a host and port of the attacker's choice. Unfortunately, repairing multipin vulnerabilities is not as simple as creating a common pin database for browsers and plug-ins (see Section 5.2).

To assess the severity of these attacks, we experimentally measured the cost and effort required to use multipin vulnerabilities to assemble a temporary, large-scale bot network. Nearly 90% of Web browsers we measured were vulnerable to DNS rebinding attacks that required only a few hundred milliseconds to conduct (see Table I). These attacks do not require users to click on any malicious links: Users need only view an attacker's Web advertisement. Our findings indicate that an attacker can spend less than $100 and hijack 100,000 unique IP addresses for sending spam and committing click fraud.

Defenses. The first priority for defending against DNS rebinding attacks is to prevent a malicious Web site from obtaining socket-level access to an arbitrary IP address. To prevent socket-level attacks, we propose modifications to the socket access policies of Flash Player and Java, the two most widely deployed browser plug-ins. Our proposed defenses have been adopted by Adobe for Flash Player [Mitre 2007c] and by Sun for Java [Mitre 2007a] and LiveConnect [Mitre 2007b]. The vendors have deployed these defenses in patches, preventing large-scale firewall circumvention and IP hijacking.

To combat firewall circumvention, we recommend that organizations deploy DNS resolvers that prevent *external* host names from resolving to *internal*

Table I. Time Required for DNS Rebinding Attack by Technology (95% confidence)

Technology	Attack Time
LiveConnect (JVM loaded)	47.8 ± 10.3 ms
Flash Player 9.0.48.0	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 3 (no plug-ins)	1000 ms
LiveConnect	1294 ± 37 ms
Opera 9 (no plug-ins)	4000 ms

IP addresses. Without the ability to resolve `attacker.com` to an internal IP address, the attacker cannot use DNS rebinding to circumvent firewalls. We provide an open-source implementation of such a resolver, called `dnswall` [Bortz et al. 2007]. Our implementation is included in FreeBSD [Haupt 2008] and is deployed at a number of organizations. Several open-source consumer firewall projects [Kelley 2008; Fainelli 2008; Gottschall et al. 2008] now implement this defense.

Individual servers can defend themselves against DNS rebinding attacks by validating the HTTP `Host` header and rejecting requests that contain an unexpected `Host` header value. Without socket-level access, the attacker Web content is unable to spoof the `Host` header. This defense is appropriate for servers that place trust, either explicitly or implicitly, in the browser's IP address. For example, the server responsible for recording clicks on pay-per-click advertisements ought to validate the `Host` header before accepting an HTTP request as a valid advertisement click.

Organization. The remainder of the article is organized as follows. Section 2 describes the existing browser security policy for network access. Section 3 details DNS rebinding vulnerabilities, including classic DNS rebinding and modern multipin vulnerabilities. Section 4 explains two classes of attack that use these vulnerabilities, firewall circumvention and IP hijacking, and contains our experimental results. Section 5 investigates potential defenses based on pinning. Section 6 recommends defenses against DNS rebinding attacks. Section 7 describes related work. Section 8 concludes.

2. NETWORK ACCESS IN THE BROWSER

Web content, such as HTML, JavaScript, Cascading Style Sheets (CSS), SWF movies, and Java applets, can instruct the browser to issue network requests. Browsers restrict this network access to prevent malicious Web sites from abusing the user's network connectivity. The browser's security policy provides partial resource isolation by restricting access to resources according to *origin*, preventing content from one origin from accessing resources from another origin. The policy applies to both network access and browser state, including the Document Object Model (DOM) interface, cookies, cache, history, and the password database. The attacks described herein circumvent the same-origin policy for network access.

Definition of origin. For network access, browsers enforce the same-origin policy [Ruderman 2001] based on three components of the Uniform Resource Locator (URL). A typical URL contains the following components.

```
scheme://host:port/path?query#fragment
```

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).

Objects on the Internet, however, are not accessed by host name. To retrieve an object at a given URL, the browser must first translate the URL's 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 all as if they belonged to the same origin.

Access in an origin. Within the same origin, Web content can both read and write network resources using the HTTP protocol. Plug-ins, such as Flash Player and Java, let Web content access network sockets directly. Using these plug-ins, sites can make TCP connections and, in some cases, send and receive UDP packets. Java does not restrict access based on port number, but Flash Player 9.0.48.0 permits access to port numbers less than 1024 only if the remote server authorizes the connection in an XML policy served from a port number less than 1024.

Access between origins. In general, Web content from one origin can send HTTP requests to servers in another origin, but Web content cannot read HTTP responses from foreign origins, effectively restricting access to "send-only." Flash Player allows Web content to read HTTP responses from foreign origins, but only if the remote server responds with an XML policy that authorizes the read. Flash Player also allows 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, SWF movies, and Java applets. These content types can be included from foreign origins. For example, one origin can include a CSS file from another origin. Web content can also read certain properties of other types of objects across origins. For example, the height and width of an image are visible to foreign origins.

Prohibited access. Some kinds 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 and Safari 3 block those plus 51 additional port numbers. Some of these port restrictions prevent Web sites from launching denial-of-service attacks or sending spam email, whereas others prevent universal cross-site scripting via the HTML form protocol attack [Topf 2001].

3. DNS REBINDING VULNERABILITIES

The network access security 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 subverts the browser's security policy by binding his or her host name to two IP addresses: one that belongs to the attacker and one the belongs to a target server.

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 browser resolves a host name, the authoritative DNS server can respond with multiple `A` records indicating the IP addresses of the host. The first DNS rebinding attack [Dean et al. 1996] in 1996 leveraged multiple `A` records to confuse the security policy of the Java Virtual Machine (JVM). The attack proceeds as follows.

(1) The user's browser visits a malicious Web site, `http://attacker.com/`, that contains a Java applet. The attacker's DNS server (which is authoritative for `attacker.com`) responds with two `A` records: one that contains the IP address of the attacker's Web server and another that contains the IP address of the target's Web server. The JVM chooses one of these IP addresses, typically the first, opens a socket, and retrieves the applet.
(2) The browser runs the attacker's applet, which requests that the JVM open a socket to the target's IP address. The JVM opens the socket because the target's IP address is contained in the DNS response for `attacker.com`.

Current versions of the JVM are not vulnerable to this attack because the Java security policy has been changed. The JVM now prevents applets from connecting to an IP address other than the IP address from which the JVM loaded the applet. (More recent DNS rebinding attacks on Java are described in Section 3.2.)

In the JavaScript variation of this attack, the attacker responds with an HTML document containing malicious JavaScript that instructs the browser to load another URL from `attacker.com`. The attacker's server refuses this second TCP connection, forcing the browser to fail over to the target IP address [Johns 2006]. By using a `RST` packet to refuse the connection, the attacker can cause some browsers to fail over to the new IP address after one second. Subsequent network requests issued by the attacker's script will connect to the new IP address, which points to the target.

Time-varying DNS. In 2001, security researchers discovered an extension to the original DNS rebinding attack using time-varying DNS [Roskind 2001].

(1) The user's browser visits a malicious Web site, `http://attacker.com/`. The attacker's DNS server responds with a single `A` record binding `attacker.com` to the attacker's IP address with a very short TTL.

(2) The HTML document from the attacker's server contains a script that issues an `XMLHttpRequest` to another URL on `attacker.com`. The DNS cache has expired, causing the browser to issue another DNS query for `attacker.com`. The attacker's DNS server now responds with a single `A` record binding `attacker.com` to the target's IP address.

(3) The browser connects to the target server and retrieves the specified URL. Because the request was issued to the same host name as the original malicious script, the browser delivers the HTTP response to the attacker's script, which then exfiltrates the data back to the attacker's server.

Pinning in current browsers. Current browsers defend against standard rebinding attacks 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 host name resolves to multiple `A` records and the current server becomes unavailable, the browser will fail over to another 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 loading an image from a closed port via the HTML element `<img src="http://attacker.com:81/">`.

—*Firefox 1.5 and 2* cache DNS entries for between 60 and 120 seconds. DNS entries expire after the value of the current minute increments twice.[2] To compute when the pin will expire, the attacker can use JavaScript to read the user's clock. Using multiple `A` records, an attacker can mount an DNS rebinding attack in approximately one second.

—*Opera 9* behaves similarly to Internet Explorer 6, pinning for approximately 12 minutes, but the browser can be tricked into releasing its pin after 4 seconds by connecting to a closed port.

—*Safari 3* 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. Flash Player 9.0.48.0 lets SWF movies open TCP sockets to arbitrary hosts, provided the recipient authorizes the connection by serving an appropriate XML policy [Adobe 2006]. According to Adobe, Flash Player 9 is installed on 55.8% of Web browsers as of December 2006 [Adobe 2008]; our experiment in April 2007 observes Flash Player 9 on 86.9% of browsers. Flash Player 9.0.48.0 is vulnerable to the following rebinding attack, which is shown in Figure 2.

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

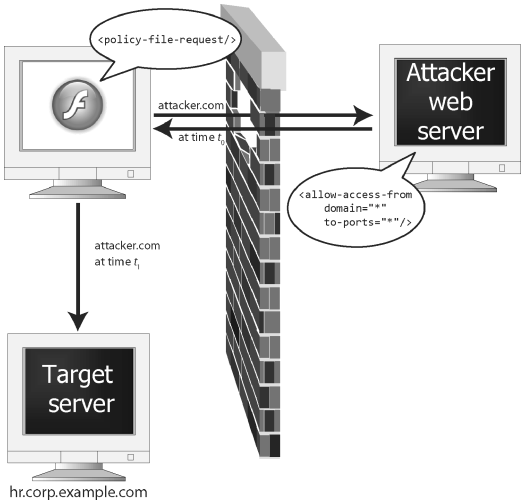

Fig. 2. DNS rebinding attack using the Flash Player policy file.

(1) The user's 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`, which the attacker's authoritative DNS server binds to the IP address of the attacker's server with a short TTL.
(3) Flash Player sends `<policy-file-request />` on the socket.
(4) The attacker responds with the following XML.

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

This XML policy authorizes all domains to make socket connections to all ports.

(5) The SWF movie opens a socket to a port number of the attacker's choice on `attacker.com`. Because the DNS cache has expired, Flash Player makes another DNS query for `attacker.com`, which the attacker's authoritative DNS server now binds to the IP address of the target.

Flash Player 9.0.48.0 permits the socket connection to the target server because it does not pin host names to a single IP address. If the attacker were to serve the policy file from port number 1024 or above, Flash Player would authorize connections only to port numbers 1024 or above.

3.2 Multipin Vulnerabilities

Current browsers use a number of plug-ins to render Web pages, several of which provide Web content some form of direct socket access. These plug-ins restrict socket access based on the origin of the content opening the connection. Even if each plug-in pins host names to IP addresses, there is no guarantee that

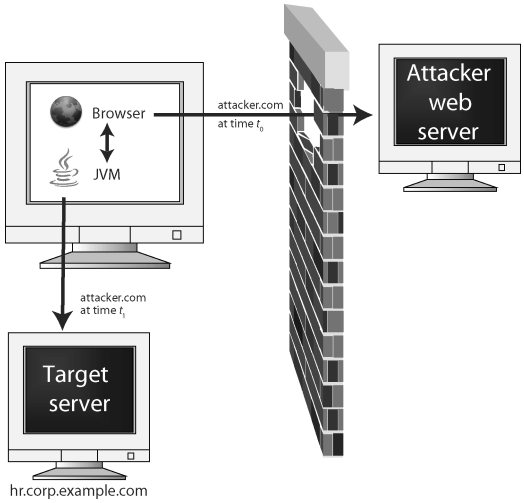

Fig. 3. DNS rebinding using LiveConnect.

all the technologies will pin the attacker's host name to the same IP address. If one component pins `attacker.com` to the attacker's IP address and another pins `attacker.com` to the target's IP address, the attacker can make use of intercomponent communication to circumvent the restrictions on network access. Some of these attacks have been discussed previously in the full-disclosure community [Anvil 2007].

Java. Java, installed on 87.6%[3] of Web browsers [Adobe 2008], lets Web content open TCP connections back to its origin. The Java Virtual Machine (JVM) pins DNS host names to IP addresses using a different pin database than the browser, opening up the possibility of multipin DNS rebinding vulnerabilities. Java applets themselves are not vulnerable because the JVM retrieves applets directly from the network, letting the JVM pin the origin of the applet to the correct IP address. However, the JVM (JRE 6 Update 2 and earlier [Mitre 2007a, 2007b]) is vulnerable 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 an accompanying applet. The browser pins `attacker.com` to the attacker's IP address, but the JVM spawned by LiveConnect does a second DNS resolve and pins `attacker.com` to the target's IP address. As shown in Figure 3, the attacker can exploit this pin mismatch to open and communicate on a socket from the user's machine to an IP address and port of the attacker's choice. Additionally, the attacker can use this technique to communicate on UDP sockets with high-numbered source ports.

—*Applets behind HTTP proxies* are also vulnerable to a multipin attack. If the user accesses the Web via an HTTP proxy, there is yet another DNS resolver

[3]We observed 98.1% penetration in our experiment.

involved: the proxy. When the JVM retrieves an applet via a proxy, the JVM requests the applet by host name, not by IP address. If the applet opens a socket, the JVM does a second DNS resolve and the attacker can cause the JVM to pin to the target's IP address.

—*Relative paths* can cause multipin vulnerabilities. If a server hosts an HTML document that embeds an applet using a relative path, that server can be the target of a multipin attack. The attacker first binds `attacker.com` to the target server's IP address, pinning the browser to the target. The browser retrieves the target's HTML document, which embeds an applet via a relative path. The browser completes the relative path to an absolute path on `attacker.com` and instructs the JVM to load the applet. The JVM does a second DNS resolve, pins to the attacker, and retrieves an applet from the attacker. The malicious applet then instructs the browser, using a `javascript:` URL, to issue `XMLHttpRequests` to `attacker.com`, which the browser has pinned to the target's IP address.

Flash Player. Flash Player would still be vulnerable to multipin attacks, even if it strictly pinned host names to IP addresses. In some embeddings, Flash Player does not retrieve SWF movies directly from the network. Instead, the browser downloads the movie and spawns the Flash Player, transferring the movie's origin by URL. When the attacker's movie attempts to open a socket, Flash Player does a second DNS resolution and could receive a different IP address than the browser.

In addition to sockets, Flash Player provides a facility for retrieving URL, the `URLLoader` class. `URLLoader` is not vulnerable to multipin attacks because Flash Player fulfills URL requests using the browser's network stack, providing the attacker little additional leverage over the browser's native `XMLHttpRequest`.

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. Other attacks 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.* Using DNS rebinding, the attacker can connect to otherwise inaccessible machines behind firewalls. 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 user's IP address. This allows the attacker to take advantage of the target server's implicit or explicit trust in the user's IP address.

To mount these attacks, the attacker must first attract the user to `attacker.com`, causing the browser to render the attacker's active content. Section 4.3 discusses a number of techniques an attacker can use to drive traffic to `attacker.com`. Once loaded onto the client's machine, the attacker's code can

exploit DNS rebinding vulnerabilities to communicate with any machine reachable by the user's machine.

4.1 Firewall Circumvention

A firewall restricts traffic between computer networks in different zones of trust. Typically, organizations deploy firewalls to block connections from the public Internet to internal machines and to mediate 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 `attacker.com` (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 browser'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 [Microsoft 2004; Edwards 2005]. Using this technique, the attacker need not scan IP addresses to find an interesting target. However, this technique is incompatible with vulnerabilities that require the attacker's DNS server to return multiple `A` records.

Having found a machine on the intranet, the attacker can connect to the machine over HTTP and request the / document. If the server responds with an HTML document, the attacker can follow the document's hyperlinks, eventually spidering the organization's intranet. Web servers behind 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 user'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, expensive, and can have unpredictable effects. Using DNS rebinding, an attacker can attempt to exploit known vulnerabilities in machines on the internal network. In particular, the attacker can attempt to exploit the user's machine itself. The attacks against the client itself originate from `localhost` and so bypass software firewalls and other security checks, including many designed to mitigate serious vulnerabilities. If an exploit succeeds, the attacker can establish a presence within the firewall that persists even after the user closes his or her browser.

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 behind 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 users 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. Additionally, consumer routers are often installed without changing the default password, making them an attractive target for reconfiguration attacks by malicious Web sites [Stamm et al. 2006]. 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 bypass these defenses and let an attacker access the router's configuration interface as if the attacker was behind the firewall.

4.2 IP Hijacking

Attackers can also use DNS rebinding attacks against machines on the public Internet. In these attacks, the attacker does not leverage the user's machine to connect to otherwise inaccessible services. Instead, the attacker abuses the implicit or explicit trust that public services place in the user's IP address. After hijacking the user's IP address, the attacker can perpetrate a number of attacks.

Click fraud. Web publishers are often paid by Web advertisers per advertisement click. Fraudulent publishers can increase their advertising revenue by generating fake advertisement 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 [Daswani et al. 2007]. Although generating a fraudulent click appears require only the ability to send an HTTP request to the advertisement network, many networks defend against send-only attacks 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 register a click. Using DNS rebinding to commit click fraud is highly cost effective because the attacker can buy advertising *impressions*, which cost tens of cents per thousand, and convert them into *clicks*, worth tens of cents each. In fact, the attack is sufficiently cost effective that the attacker need not convert every purchased impression into a click. Instead, the fraudster can use the vast majority of the purchased impressions to generate fake impressions on the publisher's site, maintaining a believable click-through rate.

Spam. Many email servers blacklist IP addresses that are known to send spam email [Spamhaus 2007]. By hijacking a user's IP address, an attacker can send spam email from an IP address with a clean reputation. To send spam email, the attacker must send packets to SMTP servers on port 25, an action blocked by most browsers but permitted by Flash Player and Java. Additionally, an attacker can often send spam email using the user's actual mail relay.

Even service providers that require successful authentication via POP3 before sending email are not protected because users typically leave their desktop mail clients open and polling their POP3 servers.

IP-based authentication. Although discouraged by security professionals [Fenzi and Wreski 2004], 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 protected service, defeating the authentication mechanism. Because the communication originates from an IP address actually authorized to use the service, the attack can be difficult for the service provider to detect.

Framing users. An attacker who hijacks a user's IP address can perform misdeeds and frame the user. For example, an attacker can attempt to gain unauthorized access to another computer system using a hijacked IP address as a proxy. As the attack originates from the hijacked IP address, the target system's logs will implicate the user, not the attacker, in the crime. Moreover, if the attacker hosts the malicious Web site over HTTPS, the user's browser will not cache the responses and the user might be left without exculpatory evidence.

4.3 Proof-of-Concept Experiment

We developed proof-of-concept exploits for DNS rebinding vulnerabilities in Flash Player 9.0.48.0, LiveConnect, Java applets with proxy servers, and the browser itself. Our system consists of a custom DNS server (authoritative for a domain we control), a custom Flash Player policy server, and a standard Apache Web server. The various technologies issue DNS queries that encode the attacker and target IP addresses, 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.

Methodology. We used our proof-of-concept to test experimentally the practicality of mounting DNS rebinding attacks. We ran a SWF advertisement on an advertising network targeting the keywords "Firefox," "game," "Internet Explorer," "video," and "YouTube." One machine in our laboratory played the role of the attacker and another played the role of a target. The attack server ran a custom DNS server, a custom Flash Player policy server, and a standard Apache Web server hosting the advertisement. The target ran a standard Apache Web server to log successful attacks. The SWF advertisement exploited the DNS rebinding vulnerability described in Section 3.1 to retrieve an XML document from the target server in our lab. To succeed, the attack requires only the user's browser view our advertisement. No user interaction is required.

The experiment lasted until the user navigated away from the advertisement, at which time we lost the ability to use the user'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).

Table II. Percentage of Impressions by Vulnerability

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

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` API, 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 and 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 in total counted 50,951 impressions from 44,924 unique IP addresses (40.2% IE7, 32.3% IE6, 23.5% Firefox, 4% Other).

We ran the DNS rebinding experiment on the 44,301 (86.9%) impressions that reported support for 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 DNS rebinding attack because the Mac OS caches DNS entries despite their zero TTL.

For each successful experiment, we measured how long an attacker could have used the user's network access by loading the target document at exponentially longer intervals, as shown in Figure 4. 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 Lam et al. [2006]. 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 viewing 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. For the impressions that reported a vulnerable version of

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

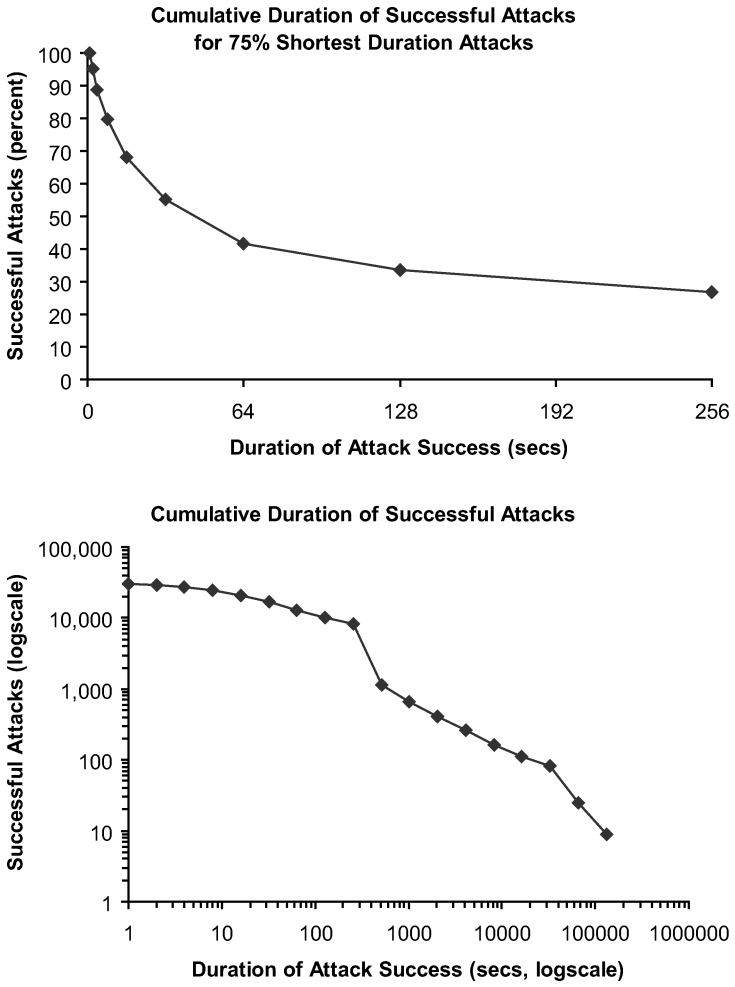

Fig. 4. Duration of successful attacks.

Flash Player but did not lead to a successful attack, we speculate the attack was unsuccessful due to firewalls that blocked outbound connections to our policy server, insufficient impression duration, or users who have disabled Flash Player.

To temporarily hijack 100,000 IP addresses an attacker would need to spend less than $100. This technique compares favorably to renting a traditional bot network for sending spam email and committing click fraud, for two reasons. First, these applications require large numbers of "fresh" IP addresses for short durations because compromised machines are quickly blacklisted. Second, although estimates of the rental cost of bot networks vary [Warner 2004; Goodin 2005; Daswani et al. 2007], this technique appears to be at least one or two orders of magnitude less expensive.

5. PINNING

Historically, browsers have used pinning to defend against DNS rebinding attacks. In this section, we explore improvements to pinning and discuss several disadvantages to the pinning approach.

5.1 Smarter Pinning

Pinning trades-off availability for security. RFC 1035 [Mockapetris 1987] 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 pin release heuristics, leading some vendors to shorten their pin duration to improve robustness [Fisher et al. 2003]. However, duration is not the only parameter that can be varied in a pinning policy.

Pin width. 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 `/24` network. For example, if a host name resolved to `171.64.78.10`, then the browser could let the host name resolve to any IP address beginning with `171.64.78`. The developers of the NoScript Firefox extension [Maone 2007b] announced plans [Maone 2007a] to adopt this pinning heuristic.

Security. When browsers use `/24` network pinning, the attacker must locate the attack server on the same `/24` network as the target, making the rebinding attack much more difficult to mount. The attack is possible only if the attacker colocates a server at the same hosting facility or leverages a cross-site scripting vulnerability on a colocated server. This significantly raises the bar for the attacker and provides better recourses for the target because the target typically has a business relationship with the operator of the `/24` network.

Robustness. To study the robustness of `/24` network pinning, we investigated the IP addresses reported by the 100 most visited English-language sites according to Alexa [2007]. We visited the homepage of these sites and compiled a list of the 336 host names used for embedded content (e.g., `http://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 user. 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 `/24` network pinning is 81.3% compared with 16.4% under strict IP address pinning, suggesting that `/24` pinning is significantly more robust to server failure.

Width heuristics. Other heuristics for pin width are possible. The browser could prevent rebinding between public IP addresses and the RFC 1918 [Rekhter et al. 1996] private IP addresses. This provides greater robustness for fail-overs across data centers and for dynamic DNS. LocalRodeo [Johns and Winter 2007; Winter and Johns 2007] is a Firefox extension that implements RFC 1918 pinning. 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 nonprivate 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 host names. For example, public host names controlled by an organization often have two IP addresses: a private IP address used by machines within the firewall and a public IP address used by browsers on the Internet. Pinning prevents employees from properly connecting to these servers after joining the organization's Virtual Private Network (VPN) because these host names appear to rebind from public to private IP addresses.

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

5.2 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 multipin attacks, pinning-based defenses 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 [Fisher 2007; Nuuja 2007] because changes to the plug-in interface are fragile and difficult to modify without introducing forwards or backwards compatibility issues. Pinning in the operating system either changes the semantics of DNS for other applications or requires that the OS treat browsers and their plug-ins differently from other applications.

Cache. The browser's cache and all plug-in caches must be modified to prevent DNS rebinding attacks. Currently, objects stored in the cache are retrieved by URL, irrespective of the originating IP address, creating a DNS rebinding vulnerability: A cached script from the attacker might run later when

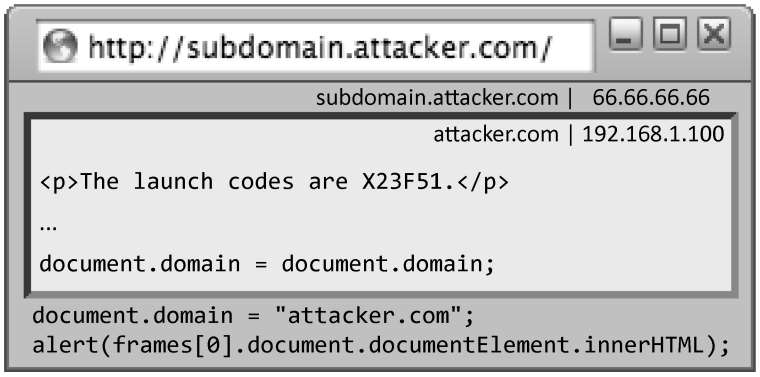

Fig. 5. The attacker's Web page can read confidential information from the inner frame, which points to an intranet site.

`attacker.com` is bound to the target [Soref 2003]. To prevent this attack, objects in the cache must be retrieved by both URL and 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.

document.domain. Even with the strictest pinning, a server is vulnerable to rebinding attacks if the sever hosts an HTML document that executes the following, seemingly innocuous, JavaScript.

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

Setting `document.domain` to `document.domain` is not an NOP. Instead, this loosens the browser's enforcement of the same-origin policy for the current document because the document has invited cross-origin script interactions from any other document that has set its `document.domain` to the same value [Veditz et al. 2002; Grimm et al. 2002]. This idiom is used by an number of JavaScript libraries, such as the Struts servlet/JSP-based Web application framework, the jsMath AJAX Mathematics library, and Sun's "Ultimate client-side JavaScript client sniff" library. Using the idiom, however, renders a site vulnerable to the following attack, as shown in Figure 5. The attacker points `subdomain.attacker.com` at the attacker's Web server (e.g., `66.66.66.66`) and points `attacker.com` at the target server (e.g., `196.168.1.100`). The target server, if it uses the aforesaid idiom, sets its `document.domain` to the value `attacker.com`. The attacker's server also sets its `document.domain` to `attacker.com` and can read documents on the target server. The "Dojo" AJAX library was also affected by this issue, but was fixed in version 0.9.0 after we explained the vulnerability to a Dojo developer.

6. RECOMMENDED DEFENSES

To defend against DNS rebinding attacks, we recommend changes to plug-ins, firewalls, and servers. Plug-ins can repair their socket-level DNS rebinding vulnerabilities by adding additional policy checks before opening sockets. Firewalls can unilaterally defend against their own circumvention by preventing external host names from resolving to internal IP addresses. After the plug-ins have repaired the socket-level vulnerabilities, individual servers can protect themselves from DNS rebinding by validating the HTTP `Host` header.

6.1 Securing Socket Access

Plug-ins comprise 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. Plug-ins that grant socket-level access to Web content either default to allowing socket connection or they default to denying socket-level access. Different DNS rebinding defenses are appropriate for these different socket access paradigms.

Default allow. Java defaults to allowing socket connections back the content's origin server. In situations in which the JVM did not retrieve the content directly from an IP address, the JVM can defend against DNS rebinding attacks by validating that the destination server is willing to accept socket connections from the content's host name. Essentially, the defense requires an oracle that determines, for a given IP address and host name, whether the server at that IP addresses authorizes that host name. Unlike forward DNS queries in which the host name is authoritative for responding to queries, DNS rebinding requires that the IP address is authoritative for authorizing host names.

The reverse DNS system can be extended to authorize host names without sacrificing backwards compatibility. In the simple case, the owner of an IP address, such as `171.64.78.146`, can authorize a host name, such as `www.example.com`, by including the following `PTR` record in DNS.

```
146.78.64.171.in-addr.arpa. IN PTR www.example.com
```

This record follows the format of existing reverse DNS deployments, leveraging the existing deployment of reverse DNS to bootstrap policy deployment. Unfortunately, this record format can encode only one host name per IP address. Many large-scale deployments host a Web server for more than one host name on a single IP address using virtual hosting. We also propose that the JVM consult reverse DNS records of the form given next.

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

This scheme uses the `auth` subdomain to authorize a set of host names for an IP address.

One disadvantage of storing policy information in reverse DNS is that the owner of the reverse delegation 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 is willing to insert appropriate DNS records. Alternatively, machines could advertise authorized host names over HTTP in a well-known location, similar to `crossdomain.xml`, but this has several disadvantages: Servers accepting socket connections must run an HTTP server and policy files served over HTTP will not be cached by the DNS infrastructure, resulting in extra traffic comparable to `favicon.ico`.

Default deny. By default, Flash Player denies SWF movies socket-level access. The destination server can opt into accepting socket connections by providing an XML policy. Flash Player can defend against DNS rebinding by considering these XML policies valid only for the IP address from which they were obtained. If a SWF movie attempts to connect to another IP address, even one with the same host name, Flash Player should request another policy file. If an attacker attempts to rebind `attacker.com` to a target server, Flash Player will request another XML policy from the target server to determine whether to open the socket. This modification to the policy behavior is backwards compatible with existing policy deployments because all honest servers that share a host name are expected to serve the same XML policy.

For port numbers greater than or equal to 1024, Flash Player 9.0.48.0 defaults to allowing socket access to the SWF movies origin server. 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, typically listen on high-numbered ports. We propose that Flash Player default to denying socket-level access in these cases and use the XML policy mechanism to prevent DNS rebinding attacks. This change breaks backwards compatibility with existing deployments who are not hosting XML policies on high-numbered ports.

Adoption. Sun has adopted our proposal in the JVM, both for applets behind proxies [Mitre 2007a] and for LiveConnect [Mitre 2007b]. In both cases, the updated JVM blocks network requests if the host name is not explicitly authorized in reverse DNS. Adobe has adopted both of our proposals and has patched Flash Player to prevent socket-level DNS rebinding vulnerabilities [Mitre 2007c]. Microsoft also adopted our proposed defense for Flash Player to secure Silverlight's `Socket` API [Microsoft 2008].

6.2 Firewall Defenses

Networks can protect themselves against firewall circumvention by forbidding external host names from resolving to internal IP addresses, preventing the attacker from naming an internal target server `attacker.com`. Without the ability to name the target, the attacker cannot aggregate the target server into a browser security origin under his or her control. These malicious DNS bindings can be blocked unilaterally, either by filtering packets at the firewall [Cheswick and Bellovin 1996] or by modifying the DNS resolvers used by machines on the network.

Internal IP addresses. Which IP addresses are "internal" varies by network configuration. Although some internal networks use publicly routable IP addresses, most use RFC 1918 [Rekhter et al. 1996] private IP addresses. In addition to the RFC 1918 addresses, many machines will respond to other IP addresses. In some deployments, an attacker can mount a DNS rebinding attack by returning a DNS record as described next.

—*CNAME.* Unless the filter blocks `CNAME` records that refer to internal host names, an attacker can use the following DNS record to bind to an internal IP address: `attacker.com. IN CNAME www.corp.example.com`. By examining DNS responses after the external DNS resolver completes a recursive DNS resolution, the filter can block all `CNAME` records.

—*Localhost.* In addition to `127.0.0.1`, many machines listen on the entire class A subnet `127.*.*.*`.

—*Zero Configuration.* In the absence of DHCP, many machines will allocate themselves an unused address in the class B subnet `169.254.*.*` [Cheshire et al. 2005]. These addresses can remain valid even after the machine receives an address from DHCP.

—*Multicast.* Although not commonly used, multicast lets many machines receive data sent to a single IP address. Internal machines could respond to IP addresses in the subnets `224-239.*.*.*`, which are reserved for multicast [Reynolds and Postel 1994; Meyer 1998].

—*IPv6.* Some machines respond to IPv6 addresses, contained in `AAAA` records. The filter blocks loopback, `::1/128`, link-local, `fe80::/10`, site-local, `fec0::/10`, multicast, `ff00::/8`, and "globally unique local." `fc00::/7`, addresses [Hinden and Deering 2003; Hinden and Haberman 2005]. To assist backwards compatibility, the IPv4 address space is embedded into the IPv6 address space twice [Hinden and Deering 2003]; specifically, once at `::/96` and once at `::ffff:00:00:00:00/96`. The filter should block undesirable IPv4 addresses from both embeddings because some machines listen for their IPv4 addresses at both of their embedded IPv6 addresses.

Deployment. There are three classes of firewall which can defend against DNS rebinding by preventing external host names from resolving to internal IP addresses.

—*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 hardened DNS server that prevents external names from resolving to internal IP addresses. We implemented this approach in a 300-line `C` program, called `dnswall` [Bortz et al. 2007]. Running alongside BIND, `dnswall` modifies DNS responses that attempt to bind external names to internal IP addresses. A number of organizations have deployed `dnswall` to protect their corporate networks, and `dnswall` is included in FreeBSD [Haupt 2008].

—*Consumer.* Consumer firewalls can also defend their private networks from firewall circumvention by using `dnswall` to block DNS responses that contain

private IP addresses. These firewalls, such as those produced by Linksys, can implement this defense without user configuration because these devices often manage the allocation of private IP addresses. Moreover, the vendors of these devices have an incentive to patch their products because DNS rebinding attacks can be used to access the private configuration interface of these devices and potentially reconfigure them to mount further attacks on their owners. A number of open-source consumer firewall projects have adopted `dnswall`-like protection, including Dnsmasq [Kelley 2008], OpenWrt [Fainelli 2008], and DD-WRT [Gottschall et al. 2008].

—*Software.* Software firewalls can defend the local loopback interface from DNS rebinding attacks by blocking DNS resolutions to `127.*.*.*`. This technique does not protect services that bind to the external network interface, even if the software firewall blocks inbound packets. The technique does, however, defend services that bind to the local loopback interface. Window and Linux both have a number of sensitive services, such as RPC, that listen on ports on the local loopback interface.

6.3 Server Defense: Host Header Checking

HTTP 1.1 requires that user agents include a `Host` header in HTTP requests that specifies the host name of the server [Fielding et al. 1999]. This feature is used extensively by HTTP proxies and by Web servers to host many virtual hosts on one IP address. Now that plug-in vendors have repaired the socket-level vulnerabilities, servers can use the `Host` header to protect themselves against DNS rebinding attacks.

During a DNS rebinding attack, the browser sends a `Host` header with the attacker's host name to the target server. Using browser APIs such as `XMLHttpRequest`, Web content can specify HTTP headers but cannot modify the `Host` header. A server can protect itself from DNS rebinding attacks by rejecting HTTP requests that contain an unrecognized or unexpected `Host` header [Megacz 2002; Ross 2007]. For example, Apache servers can defend themselves using the ModSecurity following rule.

```
SecRule REQUEST_HEADERS:Host !^www\.example\.com(:\d+)?$ deny,
 status:403
```

This rule validates that the `Host` header contains the expected value, rejecting requests with unexpected or absent `Host` headers.

Limitations. Checking the `Host` header is inconvenient for servers that do not know their host name. For example, most home routers to not know the host name of their Web configuration interface. Because these servers are typically behind firewalls, the firewall defenses described in Section 6.2 are more appropriate. Older versions of Flash Player and the JVM have DNS rebinding vulnerabilities that allow an attacker to spoof the `Host` header [Klein 2006]. These outdated plug-ins also have other vulnerabilities that can be used to bypass the browser's same-origin policy entirely [Mitre 2008, 2007d], preventing any of defenses presented in this article from protecting servers or firewalls. We recommend that users of these older versions update immediately.

7. RELATED WORK

IP-based origins. Another approach to repairing DNS rebinding vulnerabilities is to refine the browser's definition of a security origin to include the remote server's IP address [Megacz 2002]. In this approach, an attacker's script, retrieved when `attacker.com` was bound to the IP address of the attacker's server, will be unable to access content retrieved when `attacker.com` is bound to the IP address of the target. IP-based origins are more robust than pinning because a single browsing session can fail-over from one IP address to another. Such a fail-over, however, will likely break long-lived AJAX applications, such as Gmail, because the browser will prevent the application from issuing `XMLHttpRequests` to the new IP address.

There are a number of challenges to implementing IP-based origins.

(1) *Proxies.* Browsers behind a proxy server do not know the actual IP address of the remote server and thus cannot properly refine origins. There are two proposals [Megacz 2002; Megacz and Meketa 2003] for including IP addresses in proxy responses, but neither has been adopted.

(2) *Cache.* Browsers typically do not store the IP address of cached resources. IP-based origins require augmenting the browser cache with IP addresses, as described in Section 5.1.

(3) *Plug-Ins.* All browser technologies, including plug-ins, must implement IP-based origins and interacting technologies must hand-off refined origins correctly.

(4) *Relative Paths.* IP-based origins do not prevent DNS rebinding attacks if the target server embeds active content, such as a script, a SWF movie, or a Java applet, via a relative path [Jackson and Barth 2008].

Using browsers as bots. To create a Puppetnet [Lam et al. 2006], an attacker lures Web users to an malicious site and then distracts them while their browsers participate in a coordinated attack. 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 HTTP responses from different origins nor can they connect to forbidden ports such as 25. Browsers can also be misused to scan behind firewalls [Grossman and Niedzialkowski 2006] and reconfigure home routers [Stamm et al. 2006]. These techniques often rely on guessing a device's default passwords or on an underlying cross-site scripting or cross-site request forgery vulnerability.

Dynamic pharming. Some attacks [Karlof et al. 2007; Gajek et al. 2008] combine DNS rebinding with compromised DNS servers. These "dynamic" pharming attacks use DNS rebinding to rebind a trusted host name from the IP address of the attacker's server to the IP address of the target server. If the site employs HTTPS, this attack typically requires the user to dismiss common-name mismatch certificate errors. However, DNS rebinding is not required for these attacks because the attacker can simply forward the TLS session to the target server at the TCP layer. Instead of rebinding the trusted host name, the attacker can simply cease to forward TCP packets.

8. CONCLUSIONS

DNS rebinding has been a recurring source of browser vulnerabilities for the past decade. The classic defense, pinning, reduces robustness and has difficulty scaling to modern browsers, which use a number of plug-ins to render Web content. If each plug-in maintains a separate pin database, an attacker can use intratechnology communication to mount a multipin DNS rebinding attack. Plug-ins create additional security challenges because they expose additional functionality to Web content, including the ability to communicate directly on sockets. Using DNS rebinding, an attacker can use this functionality to obtain socket-level access to an arbitrary host from the user's machine, circumventing firewalls and hijacking the user's IP address.

Because DNS rebinding attacks do not require the attacker to compromise DNS, these vulnerabilities can be exploited on a large scale at low cost. By running a Web advertisement, an attacker can supply content to a large number of browsers at a fraction of a cent per impression. The simple act of rendering the attacker's rich-media advertisement is sufficient to mount a DNS rebinding attack. Our experiments indicate that this technique is an order of magnitude more cost effective than creating and operating a traditional bot network.

To prevent these attacks, we recommend specific changes to the security policies of plug-ins such as Flash Player and Java. The vendors of these plug-ins have adopted our suggestions and deployed patches to their users, preventing widespread exploitation of socket-level DNS rebinding vulnerabilities. To prevent firewall circumvention, we recommend that organizations deploy a circumvention-resistant DNS resolver, such as `dnswall`, that prevents external host names from resolving to internal IP addresses. These resolvers are now part of several open-source distributions, including FreeBSD, and have been deployed at a number of organizations.

Even with these defenses deployed, servers on the public Internet can still be targeted by DNS rebinding attacks. A server can defend itself against these attacks by validating the HTTP `Host` header because the header contains the attacker's host name during an attack. Upon receiving an unexpected `Host` header, a server should respond with a redirect to a URL containing the correct host name. This defense is especially important for servers that place trust, either explicitly or implicitly, in the browser's IP address.

ACKNOWLEDGMENTS

We thank D. Dean, D. Fisher, J. Grossman, M. Johns, D. Kaminsky, C. Karlof, J. Roskind, and D. Wallach for their helpful suggestions and feedback. We thank Google for providing the resources to implement `dnswall`.

REFERENCES

ADOBE. 2006. Adobe Flash Player 9 security. http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf.

ADOBE. 2008. Flash Player penetration. http://www.adobe.com/products/player_census/flashplayer/.

ALEXA. 2007. Top sites. http://www.alexa.com/site/ds/top_sites?ts_mode=global.
ANVIL, K. 2007. Anti-DNS pinning + socket in flash. http://www.jumperz.net/.
ARENDS, R., AUSTEIN, R., LARSON, M., MASSEY, D., AND ROSE, S. 2005. DNS security introduction and requirements. RFC 4033.
BORTZ, A., BARTH, A., AND JACKSON, C. 2007. Google `dnswall`. http://code.google.com/p/google-dnswall/.
CHESHIRE, S., ABOBA, B., AND GUTTMAN, E. 2005. Dynamic configuration of IPv4 link-local addresses. IETF RFC 3927.
CHESWICK, W. AND BELLOVIN, S. 1996. A DNS filter and switch for packet-filtering gateways. In *Proceedings of the USENIX Annual Technical Conference*.
DASWANI, N. AND STOPPELMAN, M. 2007. The anatomy of Clickbot.A. In *Proceedings of 1st Workshop on Hot Topics in Understanding Botnets (HotBots)*.
DEAN, D., FELTEN, E. W., AND WALLACH, D. S. 1996. Java security: From HotJava to Netscape and beyond. In *IEEE Symposium on Security and Privacy*.
EDWARDS, D. 2005. Your MOMA knows best. http://xooglers.blogspot.com/2005/12/your-moma-knows-best.html.
FAINELLI, F. 2008. The OpenWrt embedded development framework. In *Free and Open Source Software Developers' European Meeting*.
FENZI, K. AND WRESKI, D. 2004. Linux security HOWTO.
FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. 1999. Hypertext Transfer Protocol—HTTP/1.1. RFC 2616.
FISHER, D. 2007. Personal communication.
FISHER, D. ET AL. 2003. Problems with new DNS cache ("pinning" forever). https://bugzilla.mozilla.org/show_bug.cgi?id=162871.
GAJEK, S., SCHWENK, J., AND XUAN, C. 2008. On the insecurity of Microsoft's identity metasystem. Tech. Rep. HGI-TR-2008-003, Horst Görtz Institute for IT Security, Ruhr University Bochum. May. http://demo.nds.rub.de/cardspace/.
GOODIN, D. 2005. Calif. man pleads guilty to felony hacking. *Assoc. Press*.
GOTTSCHALL, S. ET AL. 2008. DD-WRT (version 24). http://www.dd-wrt.com/.
GRIMM, S. ET AL. 2002. Setting document.domain doesn't match an implicit parent domain. https://bugzilla.mozilla.org/show_bug.cgi?id=183143.
GROSSMAN, J. AND NIEDZIALKOWSKI, T. 2006. Hacking intranet Websites from the outside: JavaScript malware just got a lot more dangerous. In *Blackhat USA*. Invited talk.
HAUPT, E. 2008. dnswall FreeBSD port. http://www.freebsd.org/cgi/cvsweb.cgi/ports/dns/dnswall/.
HINDEN, R. AND DEERING, S. 2003. Internet protocol version 6 (IPv6) addressing architecture. IETF RFC 3513.
HINDEN, R. AND HABERMAN, B. 2005. Unique local IPv6 unicast addresses. IETF RFC 4193.
JACKSON, C. AND BARTH, A. 2008. Beware of finer-grained origins. In *Web 2.0 Security and Privacy*.
JOHNS, M. 2006. (Somewhat) breaking the same-origin policy by undermining DNS pinning. http://shampoo.antville.org/stories/1451301/.
JOHNS, M. AND WINTER, J. 2007. Protecting the Intranet against "JavaScript Malware" and related attacks. In *Proceedings of the GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.
KARLOF, C. K., SHANKAR, U., TYGAR, D., AND WAGNER, D. 2007. Dynamic pharming attacks and the locked same-origin policies for Web browsers. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.
KELLEY, S. 2008. Dnsmasq (version 2.41). http://www.thekelleys.org.uk/dnsmasq/doc.html.
KLEIN, A. 2006. Host header cannot be trusted as an anti anti DNS-pinning measure. http://www.securityfocus.com/archive/1/445490.
LAM, V. T., ANTONATOS, S., AKRITIDIS, P., AND ANAGNOSTAKIS, K. G. 2006. Puppetnets: Misusing Web browsers as a distributed attack infrastructure. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.
MAONE, G. 2007a. DNS spoofing/pinning. http://sla.ckers.org/forum/read.php?6,4511,14500.
MAONE, G. 2007b. NoScript. http://noscript.net/.
MEGACZ, A. 2002. XWT Foundation security advisory. http://www.megacz.com/research/sop.txt.
MEGACZ, A. AND MEKETA, D. 2003. X-RequestOrigin. http://www.xwt.org/x-requestorigin.txt.

MEYER, D. 1998. Administratively scoped IP multicast. IETF RFC 2365.
MICROSOFT. 2004. Microsoft Web enterprise portal. http://www.microsoft.com/technet/itshowcase/content/MSWebTWP.mspx.
MICROSOFT. 2008. Socket class (System.Net.Sockets). http://msdn.microsoft.com/en-us/library/system.net.sockets.socket(VS.95).aspx.
MITRE. 2007a. CVE-2007-5273.
MITRE. 2007b. CVE-2007-5274.
MITRE. 2007c. CVE-2007-5275.
MITRE. 2007d. CVE-2007-6244.
MITRE. 2008. CVE-2008-1192.
MOCKAPETRIS, P. 1987. Domain names—Implementation and specification. IETF RFC 1035.
NUUJA, C. 2007. Personal communication.
OLLMANN, G. 2005. The pharming guide. http://www.ngssoftware.com/papers/ThePharmingGuide.pdf.
REKHTER, Y., MOSKOWITZ, B., KARRENBERG, D., DE GROOT, G. J., AND LEAR, E. 1996. Address allocation for private Internets. IETF RFC 1918.
REYNOLDS, J. AND POSTEL, J. 1994. Assigned numbers. IETF RFC 1700.
ROSKIND, J. 2001. Attacks against the Netscape browser. In *RSA Conference*. Invited talk.
ROSS, D. 2007. Notes on DNS pinning. http://blogs.msdn.com/dross/archive/2007/07/09/notes-on-dns-pinning.aspx.
RUDERMAN, J. 2001. JavaScript security: Same origin. http://www.mozilla.org/projects/security/components/same-origin.html.
SOREF, J. 2003. DNS: Spoofing and pinning. http://viper.haque.net/ timeless/blog/11/.
SPAMHAUS. 2007. The Spamhaus block list. http://www.spamhaus.org/sbl/.
STAMM, S., RAMZAN, Z., AND JAKOBSSON, M. 2006. Drive-By pharming. Tech. Rep. 641, Computer Science Department, Indiana University. December.
TOPF, J. 2001. HTML form protocol attack. http://www.remote.org/jochen/sec/hfpa/hfpa.pdf.
VEDITZ, D. ET AL. 2002. Document.domain abused to access hosts behind firewall. https://bugzilla.mozilla.org/show_bug.cgi?id=154930.
WARNER, B. 2004. Home PCs rented out in sabotage-for-hire racket. *Reuters*.
WINTER, J. AND JOHNS, M. 2007. LocalRodeo: Client-Side protection against JavaScript Malware. http://databasement.net/labs/localrodeo/.

Received June 2008; revised September 2008; accepted October 2008