

# **There's One In Every Family**

*Exploiting DNS-based Trust Relationships On The Web*

Michael Bailey

Senior Security Researcher

[m Bailey@foregroundsecurity.com](mailto:m Bailey@foregroundsecurity.com)

As Software-as-a-Service becomes an increasingly popular business model, network administrators and application maintainers are left trying to integrate third-party sites with their own. A common convention for doing so is to configure DNS servers, creating A or CNAME records pointing to the third-party site's server. While this may ease the integration process, many of the client-side web technologies we use make trust decisions based on these DNS records, and records pointed at poorly configured systems can be used to leak data and compromise even the strongest of web applications.

These vulnerabilities are remarkably common, and many have not been formally addressed. This paper will include demonstrations of attacks on high-profile websites, as well as a discussion on mitigation methods.

## DNS in security on the Web

Many web technologies rely on DNS as a part of their security architecture. Javascript has its same-origin policy, restricting client-side scripts from accessing resources outside the web page's domain. Flash's policy is similar. Web browser cookies are restricted by a combination of domain name and path, though implementation of path restrictions on cookies is rare.

A Cross-site Request Forgery (CSRF) attack is often considered to be a violation of these policies, but in reality a site can often be used to perform CSRF attacks on itself. Likewise, Cross-Site Scripting (XSS) often involves receiving code from, and sending data to a separate server, but many XSS attacks (particularly worms) can be wholly contained within a single application. These misconceptions about common web attacks can provide an interesting insight- while the client-side security policies often define an application by the domain it lives on (and this will become even more common with the Mozilla's Content Security Policy, NoScript's Application Boundary Enforcer, and other upcoming browser-based protections), the attacks they are intended to stop are not restricted by that definition. These DNS-based protections may help to patch the problem, but they do not, and cannot fix the root issues.

DNS was never intended to be a security feature. This is made explicit by RFC 3467, which specifically dictates DNS's role as a system for looking up addresses of servers, and nothing more. Formulating complex trust relationships based on DNS records may be effective (though this paper should prove otherwise), but it is an abuse of the system. DNS may still be useful as one part of a security policy, but as currently implemented, it does a poor job at best.

DNS is a hierarchical system. A domain (such as example.com) can have virtually unlimited number of subdomains (foo.example.com), which may in turn have up to 127 subdomains (foo.bar.example.com). Within the domain name system, a domain can create records for and modify its own subdomains, but cannot tamper with its parent domain. This system allows an organization to run multiple servers within a namespace, while keeping those servers separated based on technological, network layout, and semantic boundaries. To demonstrate, foo.example.com cannot control the domain bar.example.com, but it can affect bar.foo.example.com.

In a web browser, however, these distinctions are implemented backwards. foo.example.com can both read and write cookies for example.com, but example.com cannot access cookies from foo.example.com. To complicate matters, if a specific cookie does not already exist for foo.example.com, the web browser will instead use cookies from its parent, example.com. To simplify things, consider the following cases.

### **CASE 1 (Normal use of example.com)**

1. Browser requests a page from example.com
2. example.com sets cookie "session=1" for example.com
3. Browser requests a page from example.com
4. Browser sends cookie "session=1" to example.com

### **CASE 2 (Normal use of foo.example.com)**

1. Browser requests a page from foo.example.com
2. foo.example.com sets cookie "session=1" for foo.example.com
3. Browser requests a page from foo.example.com
4. Browser sends cookie "session=1" to foo.example.com

### **CASE 3 (Setting global cookies from foo.example.com)**

1. Browser requests a page from foo.example.com
2. foo.example.com sets cookie "session=1" for example.com
3. Browser requests a page from bar.example.com
4. Browser sends cookie "session=1" to bar.example.com

### **CASE 4 (Reading global cookies from foo.example.com)**

1. Browser requests a page from example.com
2. example.com sets cookie "session=1" for example.com
3. Browser requests a page from foo.example.com
4. Browser sends cookie "session=1" to foo.example.com

### **CASE 5 (Cookie overloading causes browser to send most-specific cookie)**

1. Browser requests a page from foo.example.com
2. foo.example.com sets cookie "session=1" for example.com
2. foo.example.com sets cookie "session=2" for foo.example.com
3. Browser requests a page from foo.example.com
4. Browser sends cookie "session=2" to foo.example.com

## **CASE 6 (Specific cookies prevent data leakage)**

1. Browser requests a page from foo.example.com
2. foo.example.com sets cookie "session=1" for foo.example.com
3. Browser requests a page from bar.example.com
4. Browser sends no cookies to bar.example.com

This basic cookie policy is supported by all major browsers today. While the flaw in the system should be apparent already, it isn't until we start looking at specific examples that we realize just how dangerous (and how common) this policy can be.

### **Locating Vulnerable Servers**

Administrators often point DNS records at development servers, user-supplied content, mirrored content, partner organizations, third party applications, dissimilar applications, internal applications, and untrusted applications.

It is common for these DNS records to be left active after a server has been taken offline. If this server's IP address is returned to an ISP's pool of available IP addresses and leased to a malicious (or an innocent, but careless) system administrator, it can be used to attack administrators.

Upstream internet providers may intercept invalid records and place their own web pages on subdomains. In 2008, Dan Kaminsky found an XSS vulnerability in the advertising provider for Comcast, Earthlink, Cox, Verizon, and Qwest. This was used to demonstrate a simple attack on users' web browsers. Considering that any domain's records can be poisoned in this matter, Kaminsky could have used the techniques described below to wreak havoc on nearly every major financial, ecommerce, social networking, and government website.

To an administrator, all subdomains should be considered valuable, and care should be taken with DNS records. The smallest XSS vulnerability within the namespace can be used to attack other applications both passively and actively. More critical vulnerabilities such as code injection flaws, buffer overflows and header injections may allow the attacker even more access to other applications. Even if the vulnerable server is firewalled and located on a completely different network, the implied relationship in the DNS records can be leveraged in an attack.

If zone transfers are enabled on a targeted DNS server, complete records can be acquired quickly and reviewed for vulnerable web servers. It is common practice to disable zone transfers in a DNS server's configuration, but they are still common enough that they are worth checking.

Vulnerable webservers may be located using a brute force tool to perform lookups on a large quantity of subdomains from a wordlist. Those that are valid will resolve to an IP address. At this point, further attack surface can be located by resolving found DNS records to IP addresses and performing reverse lookups on those addresses (as well as any other addresses known to be related to the target). Finally, the records can be manually reviewed for semantic, numerical, or other patterns. If patterns are found, an attacker may be able to predict the existence of further DNS records.

Finally, undocumented and undiscovered webservers may be found by using the



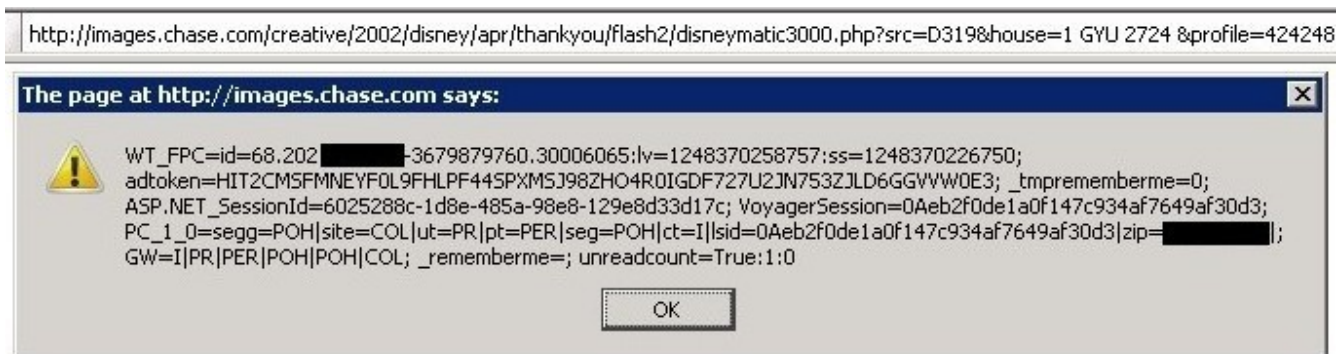
cleared his cookies or the server overwrote them- which may take months.

Other traditional browser-based protections for cookies can be bypassed using a compromised subdomain. The HTTPOnly flag can be set by the server when sending cookies, making them inaccessible to Javascript and other client-side scripts. This can be useful for protecting session cookies, but cannot be relied on. Let's add another case to the above collection:

### CASE 7 (Compromised server leaks HTTPOnly cookies)

1. Browser requests a page from foo.example.com
2. foo.example.com sets cookie "session=1" for example.com, setting the HTTPOnly flag to true
3. Browser requests a page from bar.example.com, which is owned by the attacker
4. Browser sends "session=1" cookie to bar.example.com

When integrating third-party content with their own websites, many organizations simply point a subdomain at the third-party provider's server and forget about them. This is the case with images.chase.com, images.capitalone.com, and online.chasevisasignature.com. All of these domains are pointed at the same server as images.bigfootinteractive.com, which is run by an advertising agency. Whenever this agency sends out email advertising runs, they host images, landing pages, and other content for their clients on this server. Unfortunately, this server is itself vulnerable to XSS, and can be used to compromise personal information and session data from users who do no more than visit a malicious website while logged in.



*An XSS hole in images.chase.com compromises a legitimate user's session*

## Remediation

Remediation of the vulnerabilities described in this paper generally parallel established best practices for web application security. While the attacks outlined above are fairly simple, they are often overlooked by website administrators. Clearly, any vulnerability on an untrusted subdomain can also affect a trusted domain, and no application should be considered so trivial or obscure that it does not deserve care in deployment, integration, and maintenance. Administrators should always be wary of third-party servers as well as third-party applications on their own servers. By extension of the same principle, user-supplied content should be hosted on a completely separate domain name.

High-value DNS records should be audited to locate unused servers and IP addresses. This audit should be performed as part of a regular security review, as well as a part of any penetration test that takes place.

Application logic can and should be modified to set specific domain and path parameters on cookies. While this will prevent data leakage through cookies, it will not prevent poisoning of session data and other cookie-based parameters. Currently, there is no common browser-based solution to prevent sending global cookies when more specific ones are not available, so extreme care must be taken to not allow the attacker to set these cookies.

All applications should be reviewed for common security vulnerabilities such as XSS, CSRF, and session fixation. While remediation of these vulnerabilities is beyond the scope of this paper, many of the exploits described here leverage them as part of complex attacks.

Users should take care to not retain browser settings indefinitely- clearing cookies periodically may prevent data leakage. Better yet, using separate web browsers or browser sessions for high-sensitivity, normal, and risky browsing activity will prevent particularly sensitive data from being compromised.