DNS Pinning and Web Proxies

An NGSSoftware Insight Security Research (NISR) Publication

Abstract

DNS-based attacks can be used to perform a partial breach of browser same origin restrictions in some situations, enabling a malicious web site to perform two-way interaction with a different domain.

The attacks that are normally conceived against browser-based DNS pinning are capable of being resolved through additional safeguards within browsers. However, the same attacks can also be performed against web proxies, where browser DNS pinning does not apply. Corporate web users accessing the Internet via a proxy are at risk from such attacks.

There are various ways in which DNS-based attacks against web proxies could potentially be prevented through changes to proxy and browser software. Each of the fixes considered suffers from important shortcomings. In the meantime, there are other defences that organisations and individuals can employ to prevent attacks against them.

Author

Dafydd Stuttard, Principal Security Consultant – email: daf[at]ngssoftware[dot]com

Background

DNS-based attacks against browsers have been known about for many years. These attacks have become the subject of increased attention recently, following the discovery of defects in browser-based DNS pinning defences.

So far, discussion has focused solely on browser issues and has ignored the fact that web proxies are also vulnerable to the same attacks.

Browser same origin policy

The browser same origin policy is designed to place barriers between different web sites that are being accessed by the browser, to prevent them from interfering with each other. The implementation of this concept involves various subtleties, but for present purposes it may be summarised as follows: a web site may generate a request to a different domain, but it may not retrieve and process the response data returned from that domain.

Cross-site request forgery

Because a web site can generate arbitrary requests to a different domain, there is a category of vulnerability known as cross-site request forgery. For example, if you are logged into your bank's web application and visit a malicious web site in the same browser, the malicious site can generate requests to your bank to carry out arbitrary actions, such as transferring funds to the attacker's account. Because your browser automatically submits your cookies to the banking application (which typically include your session token), the attacker-generated requests occur within the context of your session with that application, exactly as if you had performed them yourself.

The browser same origin policy means that code running on the malicious web site cannot retrieve and process the responses to any of the requests that it generates to the banking application. So, for example, the malicious web site cannot read the contents of your bank statements. For this reason, cross-site request forgery is often described as a "one way" attack.

Quick-swap DNS

To understand what DNS pinning is and why it is implemented, consider the following attack, which uses "quick-swap DNS" to retrieve and process data from a different domain:

1. An unwitting user follows a link to the URL `http://wwww.attacker.com/`.
2. The user's browser resolves the domain name `www.attacker.com`. To do this, it performs a DNS look-up on the attacker's name server. The name server responds with the IP address of the attacker's web server (`1.2.3.4`), with a time to live (TTL) of one second.
3. The user's browser issues the following request to IP address `1.2.3.4`:

   ```
   GET / HTTP/1.1
   Host: www.attacker.com
   ```

4. The attacker's web server returns a page containing a script that waits for two seconds, and then performs two actions. The first action is to use the JavaScript object XMLHttpRequest to make an asynchronous request for `http://www.attacker.com/`. Because this is the same domain which

invoked the script, the script is permitted to retrieve the response from this URL.

5. Because the browser has waited for two seconds, its previous DNS look-up on `www.attacker.com` has now expired, and so the browser performs a second look-up. This time, the attacker's name server responds with the IP address of `www.niceapp.com`, which is `5.6.7.8`.
6. The user's browser issues the following request to IP address `5.6.7.8`:
   ```
   GET / HTTP/1.1
   Host: www.attacker.com
   ```
7. The `www.niceapp.com` server responds with its content, which the attacker's script is able to process via the responseText property of the XMLHttpRequest object.
8. The attacker's script loaded in Step 4 performs its second action, which is to transmit the data retrieved in Step 7 to a location controlled by the attacker. Recall that any web site can issue a request to any other domain, and in this case the attacker's script posts the captured data to `www2.attacker.com`.

This attack succeeds in retrieving data across domains, however it only constitutes a partial breach of the same origin policy. Crucially, in Step 3 the user's browser believes it is submitting a request to the domain `www.attacker.com`, and this is the context in which the request is made. Any cookies that the user has for the domain `www.niceapp.com`, such as session tokens, are not transmitted. Unlike in a cross-site request forgery attack, the attacker-generated request does not occur within the context of the user's session (if any) with `www.niceapp.com`. This means that the content retrieved in the attack will be the same as if the attacker had simply visited `http://www.niceapp.com/` directly himself.

So what does the attack achieve? It is effective in retrieving content from web sites which the user can access but which the attacker cannot:

- If the user is on a corporate LAN, the attacker can browse intranet sites on the LAN.
- If the user is on a home DSL connection, the attacker can communicate with the administrative interface on their router, which only listens on the internal home network.
- The attacker can interact with any web-based services on the user's own computer, even if these are protected by a personal firewall.

In these situations, the attacker can reach web servers that are defended by the network topology rather than by authentication and sessions. A sophisticated attack could turn the user's browser into an open proxy, allowing the attacker to capture data from, and perform arbitrary actions against, any target that the user can access. Clearly, the targets of the attack are at nothing like the level of risk that they would be if they could be accessed directly by malicious users on the Internet. However, in many contexts, the attack described could present a serious threat.

Browser DNS pinning

It is to prevent the attack just described that browser DNS pinning exists.

In the Firefox browser, when a domain name has been resolved to an IP address, the browser caches the IP address for the duration of the current browser session, regardless of the TTL value specified in the response to the look-up. Hence, in Step 5 of the attack, the browser continues to associate `www.attacker.com` with the original IP address `1.2.3.4`, and so does not make any request to the server at `www.niceapp.com`.

Internet Explorer does not specifically implement DNS pinning as a defence, however it caches DNS look-ups regardless of the TTL, and so effectively achieves the same result.

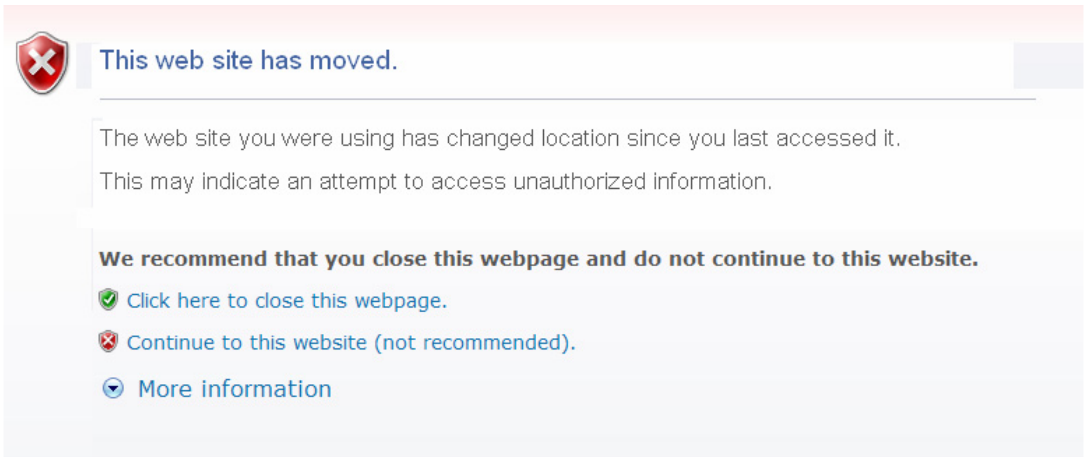
Attacks against browser DNS pinning

In August 2006, Martin Johns discovered that browser DNS pinning can be defeated by refusing HTTP connections. In Step 5 of the attack, the user's browser enforces DNS pinning and so makes the subsequent request to the original IP address `1.2.3.4`. However, if the attacker's server rejects this connection attempt (for example, by firewalling its HTTP port), then the user's browser drops the DNS pinning and performs a fresh look-up on `www.attacker.com`. At this point, the attacker responds with the IP address `5.6.7.8` and the attack proceeds as originally described. This behaviour means that the protection offered by DNS pinning can be trivially defeated by any serious attacker.

A further twist in the DNS pinning story relates to the HTTP Host header. Notice that in Step 6, the request to the `www.niceapp.com` web server contains the domain `www.attacker.com` in its Host header, because the user's browser still believes it is accessing the attacker's domain. This means that web sites could seek to defend against anti-DNS pinning by checking the Host header in all requests, and rejecting those specifying a different domain. However, an attacker can spoof an arbitrary Host header through various means, including older versions of XMLHttpRequest and Flash. Hence, checking the Host header should not be considered a reliable means of thwarting anti-DNS pinning attacks.

Preventing DNS-based attacks in the browser

Today's browsers have not yet implemented any new measures to address the defect discovered in DNS pinning. However, the attack based on refusing HTTP connections could be prevented in various ways. The reason browsers drop DNS pinning when a server refuses connections is that web servers may change location for perfectly legitimate reasons. If the browser did not drop DNS pinning in these circumstances, users would need to open a fresh browser instance to continue using the site. Hence, one possible way to balance the competing requirements of security and usability would be for browsers to present users with a warning message advising them of what has happened and asking whether they trust the web site which has just moved, or whether they are suspicious and wish to stop using it. For example:



This web site has moved.

The web site you were using has changed location since you last accessed it.

This may indicate an attempt to access unauthorized information.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.

Continue to this website (not recommended).

More information

DNS-based attacks against web proxies

Even if DNS-based attacks are completely addressed within browsers, the problem is not going to go away altogether. Browser DNS pinning does not apply when a web proxy is being used, because in this situation DNS resolution is performed by the proxy, not the browser. The browser sends HTTP requests to the proxy containing the full URL for the requested resource, including the domain name of the web site, for example:

```
GET http://www.niceapp.com/foo HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, applic
ation/x-shockwave-flash, application/vnd.ms-excel, application/vnd.
ms-powerpoint, application/msword, */*
Referer: http://www.niceapp.com/
Accept-Language: en-gb,en-us;q=0.5
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)
Host: www.niceapp.com
```

The web proxy parses the domain name from the URL in the first line of the request, and uses this to determine which destination IP address the request should be forwarded on to. There is no way in the HTTP specification for the browser to instruct the proxy which IP address it should connect to, as distinct from the domain name in the URL. In many corporate environments, internal users cannot access external DNS, and yet are able to browse the Internet because the proxy handles external DNS resolution for them.

The defence of DNS pinning cannot be straightforwardly applied to web proxies. That defence involves ignoring the TTL specified in DNS responses, and caching each look-up for the duration of the browser session. But the proxy server does not have any concept of a browser session. It receives HTTP requests from a variety of clients, and it can certainly determine which client is responsible for each request, either via the client IP address or (preferably) using proxy authentication. However, the proxy cannot determine which browser instance on the user's computer created the request. Requests generated by two separate browser processes may be identical in every respect.

Attacks against proxies using DNS trickery do not appear to have been anticipated in today's web proxy software, and no defensive measures are implemented to fill the gap left by the absence of browser DNS pinning in this situation:

- Recent versions of **Microsoft ISA server** implement their own DNS cache for performance reasons. In a default installation, when a look-up returns a TTL of 1 second, the result is cached for approximately six minutes. This presents an obstacle to trivial attacks, but not an insuperable one.
- Recent versions of **Squid proxy server** also cache DNS look-ups, but the TTL is observed, meaning that the attack originally described works against users of squid proxy, just as if browser DNS pinning had never existed.

In the original scenario, an attacker succeeded in gaining two-way interaction with web sites that the user could access but the attacker could not, due to the network topology. The most significant opportunities for such attacks arise within organisations that expose sensitive information and web applications to their internal users. The attack effectively opens those organisations' corporate intranets to the world. Very many such organisations require their internal users to use a proxy server to access the Internet. Hence, a large proportion of the most obvious targets for DNS-based attacks will still be vulnerable even if DNS pinning is properly fixed within the browser.

An important caveat to note here is that DNS-based attacks against web proxies may face a partial limitation in their scope, depending on the network configuration. In attacks against browsers, the attacker gains access to any web site that the user can access from their browser. In attacks against web proxies, the attacker gains access to any web site that the user can access *via the proxy*. If the proxy server does not allow internal users to connect to internal servers, and this restriction is enforced at the network layer rather than by domain name, then the attacker cannot leverage the proxy to access those servers. This point is elaborated on more fully later.

Software solutions to DNS-based attacks against web proxies

How could today's software be modified to prevent the attacks described?

Unfortunately, there is nothing as obvious or conceptually simple as browser DNS pinning that can be applied to web proxies. There are various possible solutions that may be considered for addressing the problem, involving modifications to proxies, browsers or both. However, each of these suffers from important shortcomings.

Solution 1: Validate DNS resolution through reverse look-ups

When the proxy server resolves each domain name to an IP address, the proxy could also perform a reverse look-up on the IP address to confirm that it belongs to that domain, and reject the resolution if it does not.

However, this stringent approach to name resolution would block access to very many legitimate web sites that do not have valid reverse DNS records.

Solution 2: Extend the DNS cache duration

Microsoft ISA server caches DNS look-ups for longer than the period specified in the TTL, and this presents an obstacle to trivial exploitation. Hence, it might be considered that extending the cache duration further would increase the size of this obstacle, making a successful attack highly unlikely.

However, this solution does not address the core problem, and attacks could still succeed regardless of the cache duration chosen. For example, if lookups are cached for six hours, and a single corporate user visits a malicious web site at 9am, this means that all users of the same web proxy will be vulnerable to an attack carried out at 3pm, when the next look-up occurs.

A second issue with this solution is how to deal with rejected HTTP connections, which may be assumed to occur more frequently if look-ups are cached long beyond their TTL. The solution suggested earlier for fixing DNS pinning in the browser involved presenting the user with a warning, and letting them decide whether to continue using a web site that has moved. But applying this solution to the proxy situation is harder. Firstly, the proxy would need to communicate the alert back to the browser, and receive the user's decision. This could possibly be achieved using new HTTP headers and status codes, or by presenting the proxy's own warning as HTML content. Secondly, if a single user makes the decision to accept the newly-located server, then presumably all other users of the same proxy will be affected.

Solution 3: Communicate DNS resolution information to the browser

Proxy servers could add a new HTTP response header indicating to the browser the IP address from which each response was retrieved. For example, if the original attack were to be performed against a proxy server, then in Step 8 the proxy could add the header:

```
X-Originating-IP: 5.6.7.8
```

Browsers could use the IP resolution information to simulate DNS pinning even where a web proxy is being used. The idea here would be that the browser would track within each session the IP address used to retrieve responses from each different domain name that it accesses. If the IP address associated with a domain changed within a single session, then the browser would alert the user in the way already described.

However, this solution will generate false positive alerts with web sites that use round-robin DNS. When the TTL expires on a domain name that has been resolved by the proxy, it will re-perform the resolution, and the look-up will probably return a different IP address, causing an alert within the browsers of all users currently accessing the site. Because many high-volume web sites use round-robin DNS for load balancing, false positives will occur frequently and will quickly lead users to ignore the resulting security alerts.

Round-robin DNS does not cause problems for traditional browser-based DNS pinning, of course, because the TTL is ignored and each rotated resolution occurs within a different browser session.

Solution 4: Perform DNS resolution in the browser

Browsers could be modified to perform their own DNS resolution even when they are using a web proxy. Instead of using the web site’s domain name in the first line of each request, they would use the resolved IP address, ensuring that the proxy forwards the request to the intended (and pinned) destination.

However, many organisations employing web proxies will be reluctant to accept this change in behaviour, because it may diminish their security in other ways. Firstly, allowing internal users to perform external DNS look-ups may result in unauthorised communication channels that bypass the corporate proxy and any other network-layer defences. Any protocol can be tunnelled over DNS to a cooperating server. Secondly, using IP addresses in requests to the web proxy prevents the proxy from performing any filtering or access control on the basis of domain names. Many organisations presently use their web proxies for this purpose.

Solution 5: Track browser sessions on the proxy

As already noted, the reason browser DNS pinning cannot be straightforwardly implemented in web proxies is that the proxy does not have any concept of a browser session. To address this core problem, proxies could be modified to keep track of browser sessions using ephemeral cookies. This would enable proxies to maintain a per-session DNS cache, and carry out per-session DNS pinning in the same way as is currently done by browsers. Proxies would still need a way of alerting the user when DNS pinning is dropped due to rejected connections, but this could now be done on a per-user basis, unlike in solution 2.

This solution appears to avoid the problems affecting the previous options. However, it imposes a considerable performance overhead that is likely to be considered prohibitive.

Solution 6: Leverage the browser to implement a per-session DNS cache

The same functional solution as described in solution 5 could be achieved without incurring the overhead of tracking browser sessions on the proxy or implementing a per-session cache. The proxy could use ephemeral cookies to store within the browser the resolved IP address of each domain name accessed during the session. In this solution, when a request arrives that contains the cookie, the proxy forwards the request to that address. When a request arrives that does not contain the cookie, the proxy uses the address stored in its own cache (or performs a new look-up) and sets the appropriate cookie.

To prevent users (and malicious web sites) tampering with the IP address stored in the cookie, so as to subvert the behaviour of the proxy, the cookie would need to be encrypted.

This solution still entails some performance overhead on the web proxy, and involves a significant re-engineering of its core behaviour that may be considered disproportionate to the scale of the threat.

Preventing DNS-based attacks within infrastructures

It seems likely that attacks against web users based on DNS trickery will not be prevented by either browsers or web proxies for the foreseeable future. In the meantime, how can organisations and individuals protect themselves against these attacks?

There are two main areas in which defences can be implemented, affecting internal web-based resources and the use of proxy servers.

Firstly, web-based resources that appear to be protected from unauthorised access by the network topology should be treated as if they are accessible from the public Internet, if web users who can access them can also access the Internet:

- Sensitive content should be protected by robust authentication that can withstand the range of attacks that exist against authentication mechanisms. Using domain credentials for HTTP-based authentication may be ineffective if these are automatically submitted by browsers on behalf of the logged-in user.
- Internal web applications should be subjected to the same level of security testing and hardening as is desirable for public-facing applications. It is common for organisations to tolerate flaws such as SQL injection in internal applications, on the basis that their employees are trusted and lack the technical knowledge required to exploit them. These assumptions are probably wrong in any case, but the DNS-based attacks described mean that malicious web sites on the Internet can also probe for and exploit those flaws.
- SSL can be used to protect access to internal web servers. If an internal user falls victim to a DNS-based attack, their browser will produce a security alert because the domain name being used to access the server does not match the domain name on the server's certificate.

The second area of defence applies in situations where internal users access the Internet via a web proxy. To defend against DNS-based attacks targeting internal web resources, the proxy can be prevented from accessing those resources. Note that blocking access by domain name within the proxy's own configuration will not be effective here, because the whole point of the attack is that an external domain is pointed at an internal IP address after a user has begun accessing it. The safest place to implement the defence is probably at the network layer, by preventing the proxy server from initiating connections to any internal web servers. Of course, users will need a means of accessing internal web sites – they can do so directly, or via a different web proxy that cannot be used to access the Internet.

Conclusion

The DNS-based attacks described enable a malicious web site to gain two-way interaction with web-based resources that cannot be reached from the Internet due to the network topology. Clearly, those resources are not at the same level of risk as they would be if they could be reached directly from the Internet. Nevertheless, in some situations DNS-based attacks may present a serious threat to organisations and individuals.

As with other areas of security, as general awareness of web security threats matures, attention is slowly moving from the server side to the client side. Attacks against web users by malicious sites are going to be an increasing area of concern, and preventing breaches of the browser same origin policy is going to be a growing objective for software manufacturers and users. Hopefully, the ideas discussed in this paper provide some food for thought as to how some of those breaches may be defended against.