Socket Capable Browser Plugins Result In Transparent Proxy Abuse

By Robert Auger (PayPal Information Risk Management Team)
V1.0

Abstract

Transparent proxies allow organizations to influence and monitor the traffic from its users without their knowledge or participation. Transparent proxies act as intermediaries between a user and end destination, and aren't generally apparent to users sitting behind them. Enterprises, Hotels, and Internet Service Providers often use transparent proxy products to lower bandwidth consumption, speed up page loads for their users, and for monitoring and filtering of web surfing. When certain transparent proxy architectures are in use an attacker can achieve a partial Same Origin Policy Bypass resulting in access to any host reachable by the proxy via the use of client plug-in technologies (such as Flash, Applets, etc) with socket capabilities. This write up will describe this architecture, how it may be abused by Flash, its existence in various network layouts, and mitigations.

Background

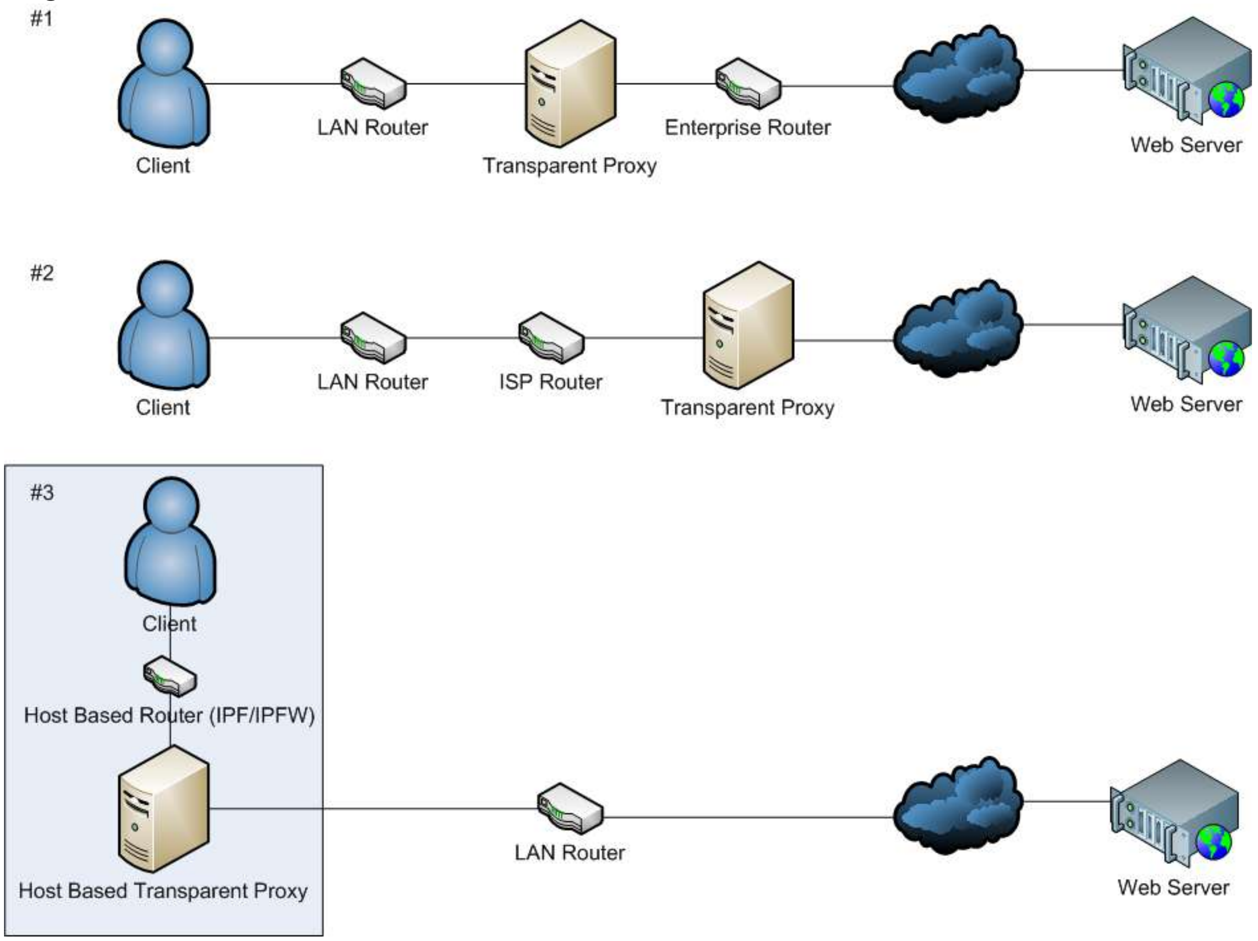
RFC 2616 defines [2] a proxy as "An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients". There are two main types [3] of web proxies in use by client machines, explicit and transparent. Explicit proxies require a host machine or application to specify the address and port of the proxy. Transparent proxies do not require client configuration and typically utilize forwarding mechanisms such as GRE tunneling, Cisco's WCCP Protocol, or MAC rewrites [3] to force users to use them.

The advantages of transparent proxies are that they can be implemented without a machine's knowledge or configuration making them ideal as web accelerators, or web filtering gateways. Various types of software implement transparent proxies as part of their operations including

- Web filtering gateways
- Parental control software
- Web Accelerators/Caching appliances

Proxies are typically placed in one of the following network configurations

Diagram A.



The first configuration is a likely layout for a large company in order to perform web filtering, caching, and employee monitoring. The second configuration is how an ISP may implement a transparent proxy for web acceleration purposes. The third configuration is utilized by some parental control software utilizing local transparent proxy instances to control web surfing. If the proxy is located on an internal network, LAN users may be able to fetch internal resources through the proxy depending on network/proxy ACLs.

Same origin policy

Wikipedia defines [20] the Same Origin Policy (SOP) as "the policy permits scripts running on pages originating from the same site to access each other's methods and properties with no specific restrictions — but prevents access to most methods and properties across pages on different sites". The same origin policy is designed to restrict a browser's access to the site it is currently on. Without the SOP SiteA would be able to make requests to SiteB and see the full response and Cookie data. Until Netscape Navigator 2.0 most browsers lacked formal capabilities to restrict cross site access.

Technical Details

There are two common ways a transparent proxy will relay a user's request to an end destination.

Approach A: Use the destination IP from the client
The first involves receiving a request from a client, inspecting the http payload, then forwarding this HTTP payload to the 'destination IP' specified by the client. In this configuration the transparent proxy routes requests much like a standard router by basing its routing decisions off of the network layer (layer 3).

Approach B: Inspect application layer data
The second involves identifying the end destination based on the HTTP payload instead of the client destination IP by inspecting the HTTP 'Host' header or request URI. In this configuration the transparent proxy is determining IP destinations based on the application protocol (layer 7) instead of IP (layer 3). Due to the socket capabilities of browser plug-ins (flash/etc) this second architecture can be exploited by an attacker to gain access to any destination accessible by the proxy.

Diagram B.

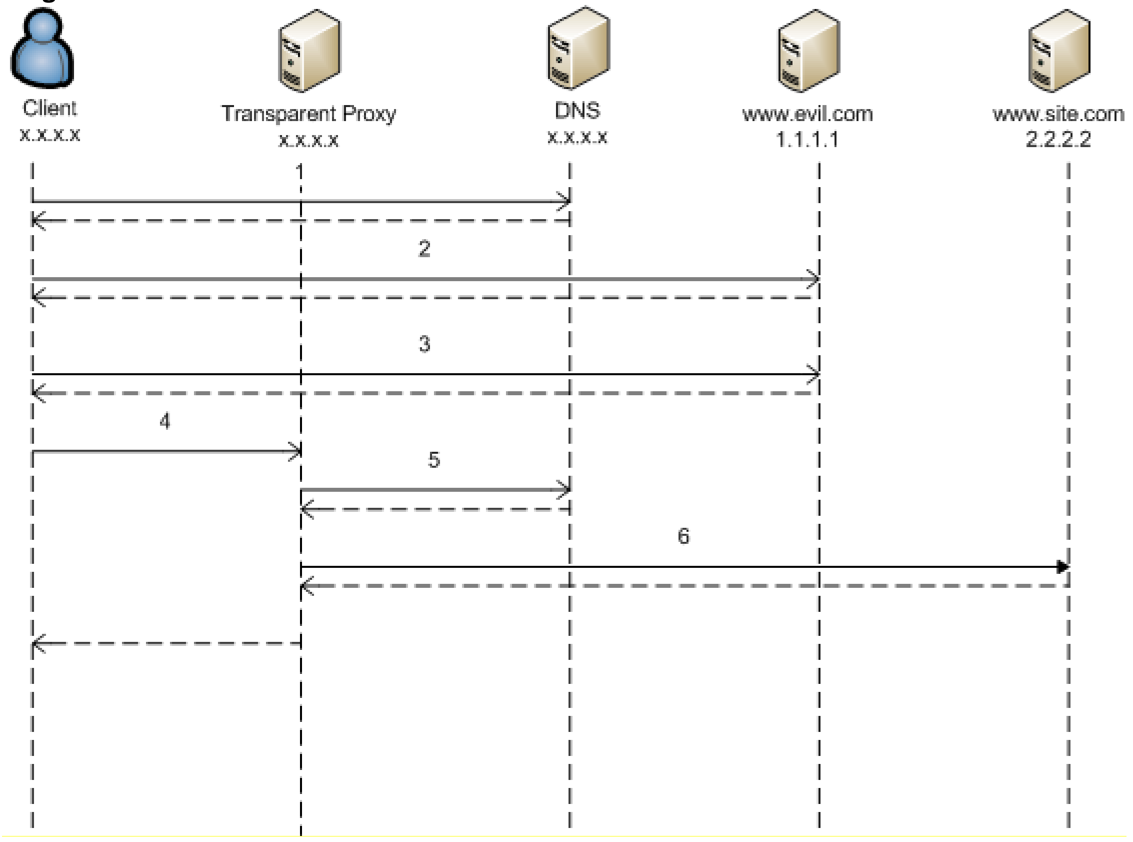


Diagram B visualizes the following attack flow when a transparent proxy using Approach B is located between a client and an evil website. The preface for this would be a user visiting a malicious website, or a website with malicious ad which spawns an iframe to an attacker controlled destination.

1. A client wishes to visit www.evil.com and performs a DNS lookup. The DNS server responds with www.evil.com resolving to 1.1.1.1.
2. The client requests a malicious flash application from www.evil.com (1.1.1.1). www.evil.com (1.1.1.1) replies with the flash application.
3. Before establishing a socket connection to www.evil.com (1.1.1.1) flash first sends a request to see if www.evil.com (1.1.1.1) allows raw sockets access. This is performed by contacting the socket policy server via a Flash Security.loadPolicyFile() call.
4. After parsing the policy file and obtaining permission the malicious flash application establishes a TCP socket connection to www.evil.com (1.1.1.1) with one of the following HTTP payloads.

```
GET http://www.site.com HTTP/1.x
```

```
GET / HTTP/1.x
Host: www.site.com
```

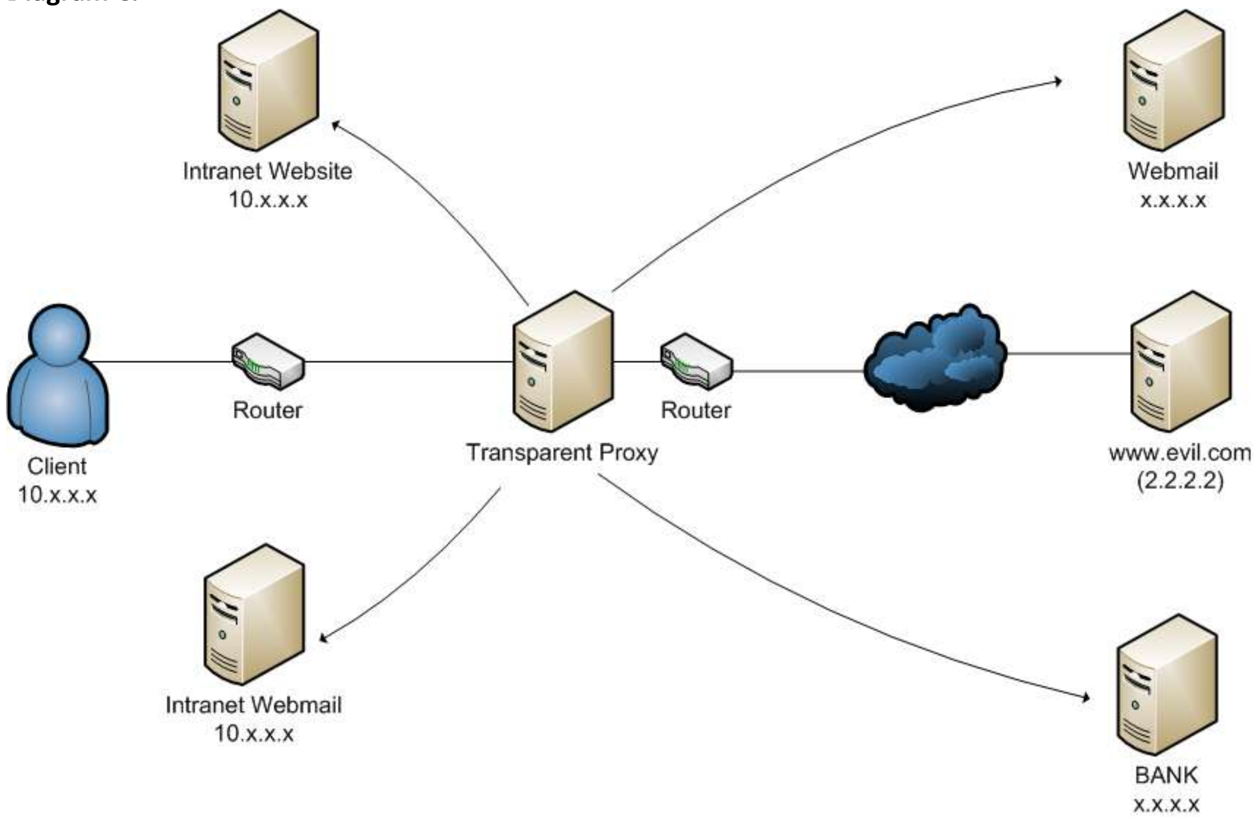
```
CONNECT www.site.com:443 HTTP/1.x
```

   An intercepting transparent proxy receives this request from the client. Upon further inspection it identifies www.site.com in the HTTP request URI or Host header as the end destination host.
5. The proxy performs a DNS lookup for www.site.com. The DNS server replies with 2.2.2.2
6. The proxy retains the connection open from the client to 1.1.1.1 and establishes a connection with 2.2.2.2 forwarding along the client HTTP payload. The proxy receives a reply back from 2.2.2.2, obtains the raw response, and replies to the client from 1.1.1.1 with the content from 2.2.2.2.
7. The malicious Flash application on the client machine has access to this raw response. From here it can send it back to the attacker for further processing or to perform a malicious action.

The proxy is effectively acting as a MITM and routing requests to hosts specified in the HTTP protocol payload instead of the IP destination. This results in a partial breaking of the SOP allowing an attacker to send raw HTTP payloads to any host accessible by the proxy. The remaining SOP still applies and restricts access to cookies and/or HTTP authentication because these would be within the context of www.evil.com. Depending on the location of the transparent proxy this may result in intranet access and likely always will result in relaying requests to internet destined hosts (See Diagram C).

In the case of Flash, sockets require the explicit permission [21] of the site via a socket policy file. An attacker simply needs to run a socket policy server on their own host and after validation will have the ability to send raw HTTP requests. Because the proxy is intercepting the request, Flash has the capability to make requests to any host accessible by the proxy and not just the originating host. While this example utilizes Flash other plug-in technologies supporting Sockets can also be used as a vector.

Diagram C.



Consulting the standards
In order to better understand this behavior I consulted the main proxy RFCs [2] [3] to see if the vendors affected were vulnerable due to a lack of conforming to the RFCs. Upon further examination of RFC1919 (marked Informational) [3] I discovered found the following.

> "In transparent proxy configurations, client systems MUST be
> able to resolve server names belonging to remote networks. This
> is critical since the proxy will determine the target server
> from the destination IP address of the packets arriving from
> the client. Because of this, the "client" internetwork needs to
> have some form of DNS interconnection to the remote network. If
> internal client and name server IP addresses must be hidden
> from the outside, these DNS queries must also be proxied."

RFC 1919 Section 5 also contains a chart and states "resolution of outside names by inside systems is required". This outlines 'Approach A' and indicates vulnerable vendors are not compliant with the behavior outlined in RFC 1919. RFC 2616 does not clearly outline how a transparent proxy must determine an end destination.

The Impact Of Network Topology And Design

The topology of a proxy on an internal network can determine your susceptibility to abuse. Upon further review it was discovered that certain configurations were more vulnerable than others. Networks which disallow internal DNS resolution of external hostnames pose their own unique issues.

Explicit Proxies

A common approach for networks that disallow internal DNS lookups is to utilize an explicit proxy to perform these lookups. These configurations are not known to be affected by the open relay issue.

Transparent Proxies

Another approach is to utilize the transparent proxy to perform all external DNS resolutions. When an explicit proxy is utilized the client knows to always send all traffic to the proxy. With transparent proxies the client must first be able to resolve a given hostname before sending its packet over the wire. In this setup there are three approaches to using a transparent proxy to perform DNS resolution

Option

A. To resolve all hostnames to a single IP address
B. To resolve all hostnames to different internal IP addresses
C. To resolve all hostnames to external IP addresses different than the external result.

Due to the introduction of client side plug-ins supporting sockets, ‘Option A’ fundamentally breaks the same origin policy because every hostname is bound to the same IP address. This allows flash (or another client side plug-in with sockets capabilities) access to every hostname regardless if the proxy is implementing ‘Approach A’ or ‘Approach B’. Networks utilizing ‘Option A’ configurations should attempt to implement either an explicit proxy, or internal DNS resolution of external hostnames to address this issue. Both Option B and C in theory could be done safely assuming each host resolves to a different IP address however it isn't advisable to use either configuration.

The proxy chaining dilemma

It was discovered that if a transparent proxy utilizes an explicit proxy between itself and the end destination, that this vulnerability exists if the transparent proxy utilizes ‘Approach A’ or ‘Approach B’.

Diagram D.

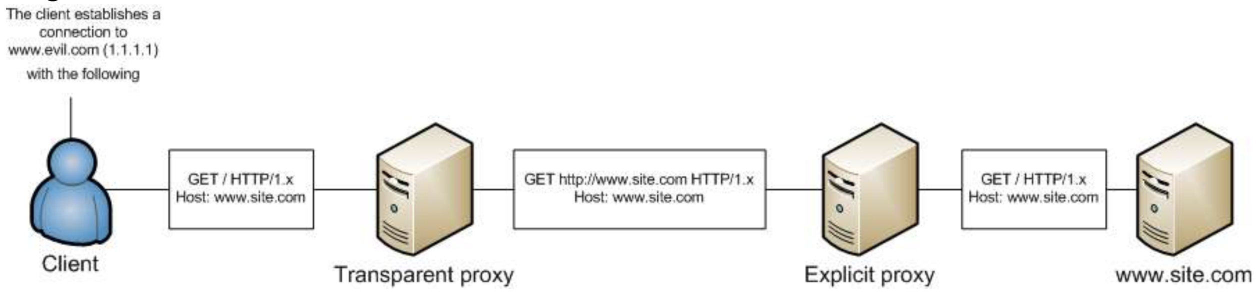
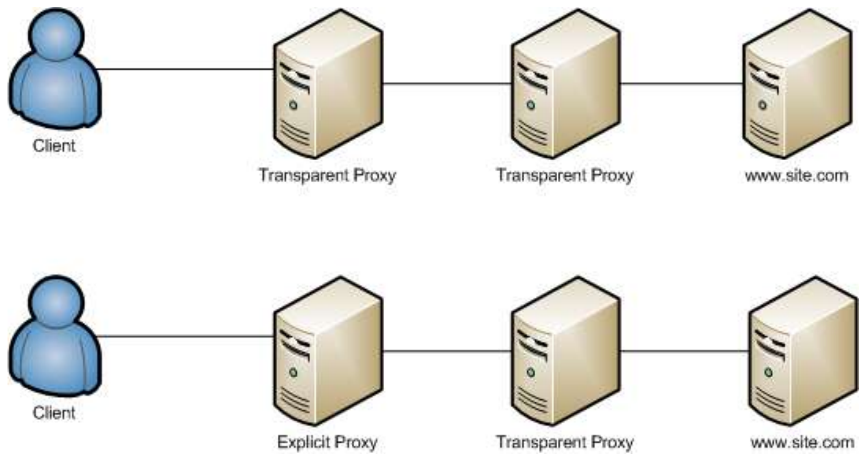

The client establishes a connection to www.evil.com (1.1.1.1) with the following
Client
GET / HTTP/1.x Host: www.site.com
Transparent proxy
GET http://www.site.com HTTP/1.x Host: www.site.com
Explicit proxy
GET / HTTP/1.x Host: www.site.com
www.site.com

Diagram D outlines the following

1. The user performs a DNS lookup to www.evil.com (1.1.1.1) and upon receiving the IP establishes an HTTP connection to 1.1.1.1.
2. The intercepting transparent proxy receives the request and reformats it for use through an explicit proxy that it utilizes. At this stage the transparent proxy must utilize the host specified in either the URI line or the Host Header (Diagram D demonstrates the Host header method).
3. The explicit proxy establishes a connection to www.site.com, replies to the transparent proxy (its client), which in turn replies from 1.1.1.1 to the client.

Diagram E outlines chain configurations which are **not vulnerable** to ‘Approach A’.

Diagram E



In the first approach if both proxies implement ‘Approach A’ then this vulnerability does not exist. In the second approach the client specifies an explicit proxy for web requests which disallows requests to be made by raw sockets resulting in this configuration being safe. For raw sockets to work through an explicit proxy the proxy must host a socket policy file allowing this access.

Further research is needed by the http and proxy community to close this loophole. Unfortunately at this time there is no fix for environments utilizing this configuration. It is recommended to utilize a non-vulnerable configuration if at all possible.

Caching considerations

As was mentioned earlier proxy products often cache responses to cut on load times for the client. While the proxy itself should be basing the end destination on the client supplied IP, proxy caching mechanisms must validate DNS resolution for the given host before caching its content to avoid cache poisoning scenarios. While caching is out of scope for this write-up, it is a related architectural issue that was worth mentioning.

Plausible vectors for exploitation and abuse
Any website hosting a flash application or sockets capable plug-in can be a possible vector. A likely scenario would involve a malicious ad loading a hidden iframe from a malicious website containing an evil flash application. As was mentioned earlier flash does not have access to cookies or credentials to other websites due to the Same Origin Policy. While flash will not automatically negotiate HTTP or form based authentication on raw sockets, the functionality to manually negotiate them exists. This provides a nice vector for an attacker to perform brute force attacks against internal or external targets (depending on the network configuration).

The attack would be limited to the time that a user has the malicious flash application loaded and upon closing their browser the attack would terminate. The kind of access an attacker is granted resembles DNS rebinding [23] in many ways.

Vendor Discussions And Coordination

We originally started contacting vendors that we knew to be affected as well as those that were likely to be affected. As the list began to grow the scope of our participation grew requiring us to work with an organization proficient in these matters. We contacted the Computer Emergency Response Team (CERT) to assist with vendor coordination, notification, and disclosure efforts. We'd like to thank Ryan Giobbi and CERT for their professionalism and assistance with handling this issue.

Discovery And Related Research

I first observed this behavior in 2007 while writing a proxy scanning tool to detect open relays , however didn't have the time to properly research the issue until early 2008. Due to the nature of this flaw I assumed it was widely known, however upon testing additional products I found them to also be exploitable. While researching this flaw other researchers started to publish material and participate in discussions involving ways they could abuse the Host header for malicious purposes [4] [5] [16] including for cache poisoning. Amit Klein was one of those researchers who had previously published material involving cache poisoning, http, and proxy related abuses [8] [10] [12][15]. I had a chance to speak with Amit during Bluehat where he confirmed seeing this behavior in most of the products he had tested during his research [15]. Upon further discussions and some searching online we discovered that Dan Kaminksy had a brief entry in his slides for Blackhat [6] touching on this behavior. In December of 2008 I contacted Dan Kaminksy who was rumored to be working on a related project. After an initial discussion with Dan and Amit we concluded a joint discussion was in order to share information, validate our assumptions, and ensure our parallel works properly factored in necessary considerations. Dan Kaminsky's CanSecWest talk "*Stuck In The Middle: Advanced Manipulation Of Middleware Through Conspiring Endpoints*" discusses abuse cases against other application protocols influencing the network layer.

As we've seen time and time again multiple researchers often discover (or discover portions of) the same issue, independently due to their related interests and technical background. I'd recommend reading the references below for additional information on this behavior/related behaviors as they've posted some great material.

Reproduction Instructions

To identify if your environment is vulnerable you can perform the following manual steps.

1. Perform a DNS lookup against a test website name
2. Telnet to that website's IP on port 80 ($ telnet <host> 80)
3. Paste the following request as the payload

```
GET / HTTP/1.0
Host: <put a different website name here>
```

4. Hit enter twice

It is important to specify a different website name in the 'Host' header. The reply will look similar to the following with HTTP headers followed by HTML.

```
HTTP/1.1 200 OK
Date: Thu, 05 Mar 2009 22:20:41 GMT
Server: Apache
Cache-Control: private
Pragma: no-cache
Content-Type: text/html; charset=UTF-8

<html>
```

If you receive content from the host specified in the host header then you're affected.

Affected Products and Mitigations

To identify if your product is affected, and how to mitigate this issue please review the CERT Advisory [1] located at the end of this document in addition to the product support website for your vendor.

Special Thanks and Participation

I would like to thank Amit Klein (CTO, Trusteer), Dan Kaminksy (Director of Penetration Testing, IOActive Inc.), and Adrian Chadd (Director, Xenion Pty Ltd) for taking the time to discuss this issue in depth. Our joint discussions contributed heavily to this paper and clarified several points. I encourage security researchers to combine efforts with others in their field to enhance their own research, as well as better assist the industry.

Debate

Due to the nature of this behavior and the technologies involved there will likely be differences of opinion in the community as to where this issue lies and where it should be remediated. This write up's intention is to shed light on this abuse case and provide information on currently available mitigations. Future changes to products and/or protocols are likely required to adequately provide a long term solution.

It is clear however that there are security limitations with the same-IP policy implemented for sockets in many RIA frameworks

Conclusions

With the expansion of technologies operating on the framework of older technologies additional research needs to be performed to see how older assumptions might be used and abused in unexpected ways.

References and related reading

Intercepting proxy servers may incorrectly rely on HTTP headers to make connections
[1] http://www.kb.cert.org/vuls/id/435052

RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1
[2] http://www.ietf.org/rfc/rfc2616.txt

RFC 1919: Classical versus Transparent IP Proxies
[3] http://www.faqs.org/rfcs/rfc1919.html

'HTTP cache poisoning via Host header injection' Email Thread
[4] http://www.webappsec.org/lists/websecurity/archive/2008-06/msg00071.html
[5] http://www.webappsec.org/lists/websecurity/archive/2008-06/msg00075.html

Black Ops 2007: Design Reviewing The Web
[6] http://www.doxpara.com/slides/DMK_BO2K7_Web.ppt

DNS Pinning and Web Proxies
[7] http://www.ngssoftware.com/research/papers/DnsPinningAndWebProxies.pdf

Technical Note by Amit Klein: "XST Strikes Back"
[8] http://archive.cert.uni-stuttgart.de/bugtraq/2006/01/msg00395.html

Meanwhile, on the other side of the web server: A survey of new attacks on the less explored parts of the web application
[9] http://www.securityfocus.com/archive/1/401866

Write-up by Amit Klein: "IE + some popular forward proxy servers = XSS, defacement (browser cache poisoning)"
[10] http://www.webappsec.org/lists/websecurity/archive/2006-05/msg00140.html

HTTP Request Smuggling (with Chaim Linhart, Ronen Heled and Steve Orrin)
[11] http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf

Divide and Conquer - HTTP Response Splitting, Web Cache Poisoning Attacks, and Other Topics
[12] http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

Protecting Browsers from DNS Rebinding Attacks
[13] http://crypto.stanford.edu/dns/dns-rebinding.pdf

Wikipedia Proxy Entry
[14] http://en.wikipedia.org/wiki/Proxy_server

Technical note: under some conditions, it's possible to steal HTTP credentials using Flash
[15] http://www.webappsec.org/lists/websecurity/archive/2006-08/msg00047.html

HTTP Cache Poisoning via Host Header Injection
[16] http://carlos.bueno.org/2008/06/host-header-injection.html

SQUID Wiki
[17] http://wiki.squid-cache.org/SquidFaq/InterceptionProxy

Web Cache Communication Protocol
[18] http://en.wikipedia.org/wiki/Web_Cache_Communication_Protocol

Wikipedia GRE
[19] http://en.wikipedia.org/wiki/Generic_Routing_Encapsulation

Wikipedia Same Origin Policy
[20] http://en.wikipedia.org/wiki/Same_origin_policy

Policy file changes in Flash Player 9 and Flash Player 10
[21]
http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7c60.html

Domain Contamination
[22] http://www.webappsec.org/projects/articles/020606.shtml

Wikipedia DNS Rebinding
[23] http://en.wikipedia.org/wiki/DNS_rebinding

Staring Into The Abyss: Revisiting Browser v. Middleware Attacks In The Era of Deep Packet Inspection
[24] http://www.doxpara.com/abyss