Stripping Referrer for fun and profit

Stripping Referrer for fun and profit - Author not stated, blog.kotowicz.net.

- Published: date not stated
- Original: <http://blog.kotowicz.net/2011/10/stripping-referrer-for-fun-and-profit.html>
- Preserved from: http://blog.kotowicz.net/2011/10/stripping-referrer-for-fun-and-profit.html (live) on 2026-08-09
- 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.

**tldr: **New methods for client side only (no server side script) referrer stripping in POST & GET requests. Code at the end.

Referer is that tiny bit of information that browser sends to servers while you click your way through interwebs, always carrying the URL of the webpage you've clicked the link at (more or less). It's useful for webdevelopers. For example, if they know you've reached their page from Google search results they can tailor the webpage especially for you. Of course, it's a privacy leak, so users can **turn off referrer sending** in current browsers. All in all, Referer is usually spoken in SEO circles, which is not my pair of shoes. However, at least one thing makes Referer very interesting from security point of view.

Are you me?

Sometimes it's used as an **access control** mechanism. It all began with hotliking. Spammy websites started republishing content (e.g. images) from other websites, simply using . While the spammy page looked ok for the viewers, it was stealing bandwidth (and content) from the original website. So to prevent this, websites started checking the Referer header that was added by the browser to image requests. If the referrer URL was from the original website, server would return the image. If it was from a 3rd party (http://spammy-website.info), it would return 403 or something else, so the spammy websites stopped looking good anymore.

Do no evil!

Then came Cross Site Resource Forgery). Visiting a malicious website when being simultaneously logged in to some vulnerable application caused havoc. This malicious website could send

dangerous requests to the original vulnerable app and e.g. delete your account, change your e-mail address or set up a mail filter. Some web developers started adding simple defense mechanism: **just check the referrer**!

- if it's our original application URL, process the request
- in other cases, deny

There - case solved. Only pages from legitimate location could make successful requests. But this quickly gave some problems - what about users that *disabled* referrer sending? They wouldn't be able to use the application at all. For many websites, this was quickly corrected - for missing referrer, give the benefit of doubt and allow.

Stripping for the client!

[[image: image]](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgmhdl-HNhU3OVKLLwTM4UNaALzg7oMbjhX7yFAxcBhJVb9aMoLeL9b-XQoN07ddhKBu4Hpn69-pL90a8Z4OFFHc38NYQIMSttTykmndJgxpI86RMHQkl0JGDa8UKiAvvSPglhiqfO-9VA/s1600/5459038-beautiful-silhouette-of-young-women-dancing-a-striptease.jpg)

Of course, it's a very weak spot and attackers quickly tried to use it to their advantage. The goal was simple: **Strip referrer, keep the cookies**. There are many ways for attackers to lose referrer using some server side redirect etc. You can even fake that header. How? Just check how referer.us does it. But I wanted something different. I wanted to be able to make arbitrary cross origin POST / GET requests with **stripping referrer header using no server-side script **whatsoever****.* Only client-side techniques. I couldn't find any previous tries for this and only found old mentions about meta redirects. There of course are several known attempts of server-side techniques e.g. Jeremiah Grossman directed me to an example from 2006 of doing this with Flash.

Why client side only? Sometimes, as an attacker, you don't have control over the server at all, you can only inject some HTML (e.g. XSS) in a secondary host the victim user will be directed to. Client side only has fewer requirements to plant an attack.

Was it good?

Pretty good actually. Dealing with multiple browser quirks I was able to strip the referrer in **every major browser** for GET requests and in Firefox / WebKit for POST requests. Here's how:

```
function lose_in_webkit(url) {
 // chrome loses it in data uris
 location = "data:text/html,<script>location='" + url + '&_=' + Math.random() + "'</scr"+"ipt>";
 return false;
}

function lose_in_ie(url) {
 // ie loses referer in window.open()
 window.open(url + '&_='+Math.random());
}

function lose_in_ff(url) {
      // ff needs data:uri  AND meta refresh. Firefox, WebKit and Opera
 location = 'data:text/html,<html><meta http-equiv="refresh" content="0; url='+ url + '"></html>';
}

function post_and_lose(url) {
      // POST request, WebKit & Firefox. Data, meta & form submit trinity
 location = 'data:text/html,<html><meta http-equiv="refresh" content="0; url=data:text/html,<form id=f method=post a
}
```

Demo and source.

Update: **@websterprodigy topped that with a nice way to lose the referrer in POST & GET in all browsers using <iframe src=about:blank>. Good job!

Why do I care?

**Pentester: **Let's imagine you encounter a CSRF flaw on a website, but this website does referrer checking. Now, without relying on server side techniques you can use these snippets to quickly prepare a CSRF proof of concept for your client. How often do websites rely on referrer? Pretty often, just wait for the next post (here it is) on a neat Alexa Top 50 example.

**Developer: **Now you know not to rely on referrer checking as CSRF protection. The only way is to use tokens_Prevention_Cheat_Sheet)!