



Dynamic Cross-Site Request Forgery

*A Per-Request Approach
to Session Riding*

Shawn Moyer

FishNet Security

<http://www.fishnetsecurity.com>

Nathan Hamiel

Idea Information Security

<http://www.ideainfosec.com>



Abstract

Cross-Site Request Forgery (“CSRF”) has typically been approached as a “replay” or static type of attack, where a bad actor uses markup, JavaScript, or another method to force a client browser to perform an in-session transaction on the affected site without the user’s knowledge.

Typical defensive measures against CSRF address this by creating unique, per-session or per-request tokens not typically available to an attacker, and by checking for other browser behaviors such as referring URL.

In this paper we describe an approach to construct “dynamic” CSRF attacks, forging unique requests on an automated, per-target basis.

“Classical” Cross-Site Request Forgery

Pete Watkins coined the term Cross-Site Request Forgery in a post to Bugtraq in 2001¹, for what had been described in 1988 by Norm Hardy as the “Confused Deputy”², in reference to similar problems with compilers and timesharing systems.

The fundamental premise of CSRF is this: In the course of web browsing, a target user encounters a request from a malicious site or location which makes a request on behalf of the user to a site the user is already authenticated to. This request could be in the form of an HTML tag such as an IMG, IFRAME, or META refresh, or embedded in JavaScript or Flash, or sent as 300-series HTTP redirect from a location under the attacker’s control. If the user is authenticated, the relevant HTTP headers and any authentication data are sent to the site along with the malicious request.

¹ <http://www.tux.org/~peterw/csrf.txt>
<http://www.cis.upenn.edu/~KeyKOS/ConfusedDeputy.html>

²

From Watkins' 2001 Bugtraq post:

CSRF does not in any way rely on client-side active scripting, and its aim is to take unwanted, unapproved actions on a site where the victim has some prior relationship and authority.

Where XSS sought to steal your online trading cookies so an attacker could manipulate your portfolio, CSRF seeks to use your cookies to force you to execute a trade without your knowledge or consent (or, in many cases, the attacker's knowledge, for that matter).

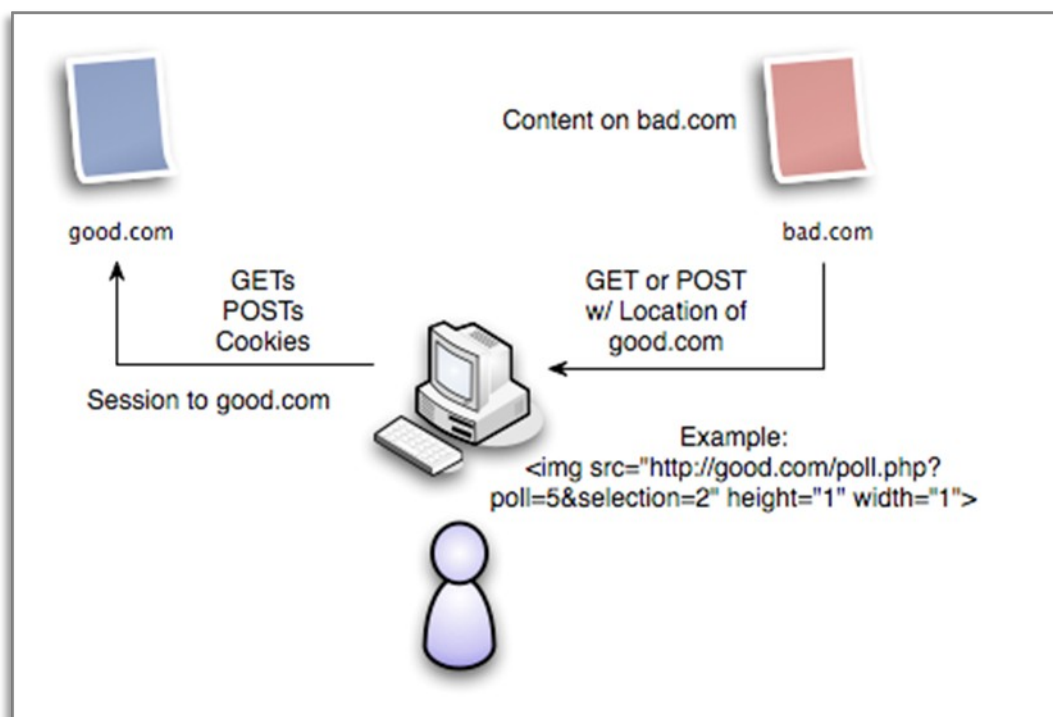


Figure 1: An example of "Classic" CSRF. Hidden markup on bad.com forges a client request to an authenticated session on good.com.

In 2004, Thomas Schreiber's paper "Session Riding"³ further expanded on Watkins post, and went on to describe a number of attack scenarios where CSRF was possible and valuable to an attacker, providing the most detailed accounting of the problem at the time.

³ http://www.securenet.de/papers/Session_Riding.pdf

Schreiber went on to recommend tokenization as a solution, in addition to URL rewriting:

To make a Web application safe against Session Riding, introduce a secret (aka hash, token) and put it into every link and every form, at least into every form that is sensitive. The secret is generated after the user has logged in. It is stored in the server-side session. When receiving an http request that has been tagged with this information, compare the sent secret with the one stored in the session. If it is missing in the request or the two are not identical, stop processing the request and invalidate the session.

Schreiber's recommendation is the most common approach to CSRF protection, and is the typical mitigation approach in place today. While in the case of some newer frameworks, a CSRF token may be unique to a given generated request and recreated per-pageview, often the token value is specific to a session, or valid for a given lifetime.

Interestingly, Watkins' original 2001 post saw the flaw in this approach:

A combination of cookie + URL mangling might not be bad, though in the message board case, a CSRF attacker could use an intermediate redirect (as described earlier) to get the URL mangling (from the Referer), and redirect back to the messageboard with the proper mangling as well as all cookies that might be expected/needed. So in your example case, URL mangling would buy nothing.

This point, that tokens or "URL mangling" buy nothing for sites where offsite linking and referrer leakage is possible, remains valid today, and with the web's increasing emphasis on offsite content, shared APIs, and user-generated content, a number of opportunities exist to prove Watkins' point.

“Dynamic” Cross-Site Request Forgery

In a “dynamic” CSRF scenario, expanding on the approach described by Watkins, an attacker creates a customized, per-request forgery, based on elements obtained through referrer. In order for this approach to succeed, a user would need to make a request to a site under attacker control in the context of the target site.

This scenario is more likely than it would first appear, though. Clicking on a link in the context of a webmail application or message forum may leak the necessary information through referrer, or if the target site permits the inclusion of offsite content, such as offsite images (typical on many web forums and social networks), an attacker could silently obtain the necessary information to construct a custom forged request, and send a response to the target browser to call the “dynamic” CSRF.

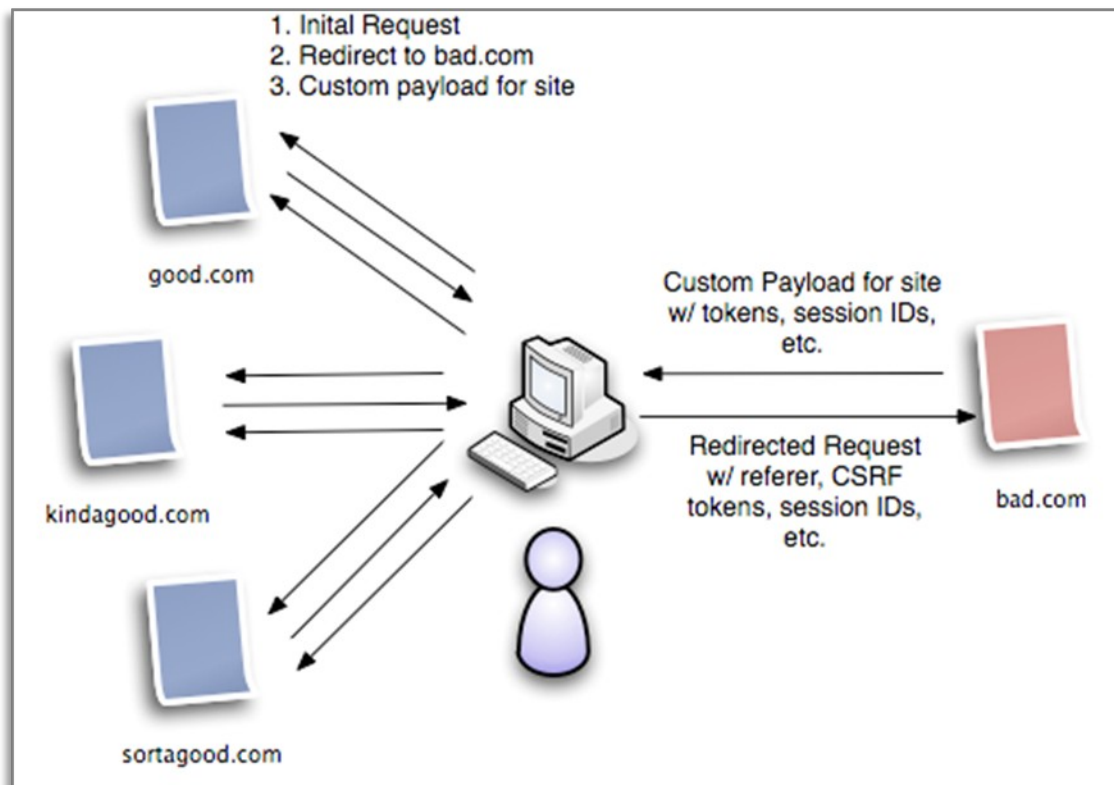


Figure 2: “Dynamic” CSRF scenario: User request is redirected to a CSRF with per-user session information included, such as CSRF token and session ID.

In short, if it's possible to obtain session specific information from a target browser in some fashion, "dynamic" CSRF is possible. To take advantage of this problem on an automated basis, an application controlled by the attacker could package different payloads based on domain from which the request came, and include any relevant "secret" URL-based information in the forged request, such as Session IDs or CSRF tokens leaked to the attacker, .

As an example, if the following value in the referrer were obtained from a request to an attacker-controlled server, originating from a target browser:

<http://good.com/function.php?val1=foo&val2=bar&token=765234>

In "dynamic" CSRF, the application under attacker control could repackage the request, sending a new request to the browser with a custom payload including the browser specific token, such as:

<http://good.com/anotherfunction.php?val4=morefoo&val5=morebar&token=765234>

The user's browser encounters the custom request, as a redirect, a hidden form POST, or some other method, and sends the request along with any relevant HTTP authentication headers for good.com.

"Dynamic" CSRF Payloads

An application that constructs a dynamic payload using cross-domain referrer leakage to construct a custom payload can take a number of approaches. Each scenario will vary depending on what type of data the attacker has the ability to put in the victims path, and the specific function being targeted on the site in question.

HTTP Redirect Payload

A trivial payload, where possible, is the use of a simple HTTP 302 redirect. In this case the attacker would take leaked session data and construct a redirect with the location value of the redirect, populated with the relevant session data obtained from the target browser.

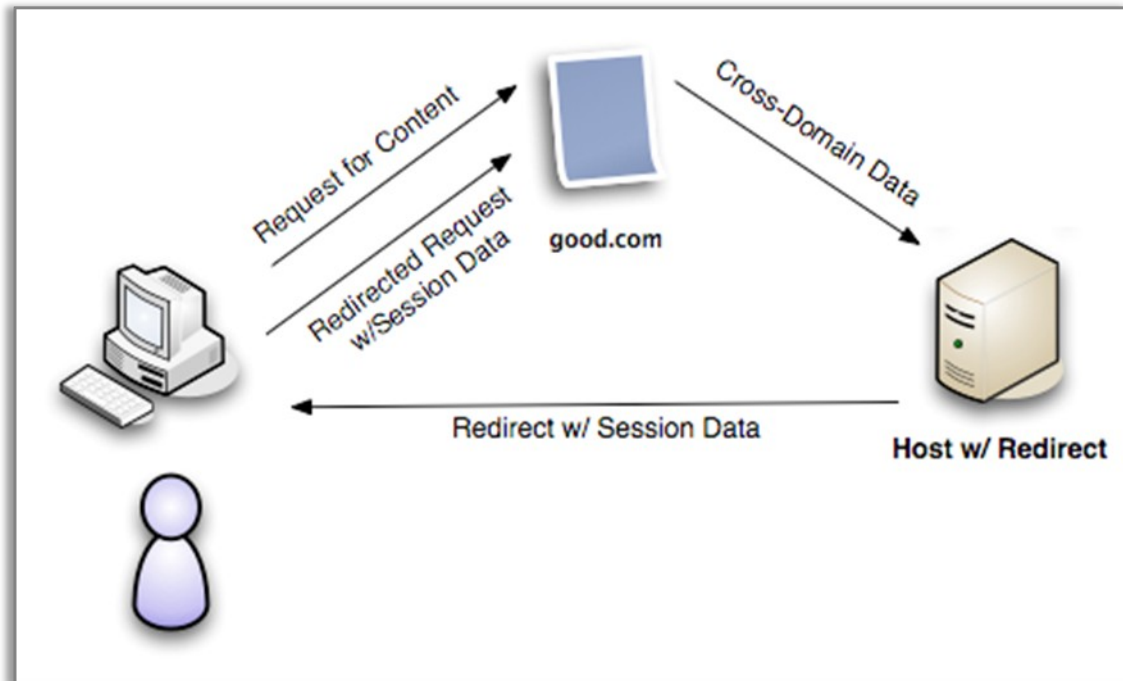


Figure 3: Redirect payload, sending custom redirect to client browser via HTTP 302.

A dynamic payload is created, and a redirect is sent back to the user's browser with the associated token value applied to the payload. This would allow an attacker to potentially bypass CSRF protections in place and execute a request of the attacker's choice, and if the redirect is sent from a link in the context of a target site, referrer checks wouldn't apply - the referring site would remain the source of the request, since browsers leave typically leave the original referrer intact when following a 302 redirect.

HTML-based redirect

An application that creates "Dynamic" CSRF payloads could also create a custom page based on referrer content and redirect the user to another site, performing the CSRF in the background.

An interesting scenario here would be the use of a URL shortening service such as TinyURL, Is.gd, or others, sending a redirect to an ultimate "legitimate" destination, but embedding a custom CSRF payload in as part of an interstitial HTML page that includes a meta refresh to another location that appears legitimate. The generated HTML could construct a GET request with session data taken from cross-domain data, or even a self-submitting form that constructs a POST request.

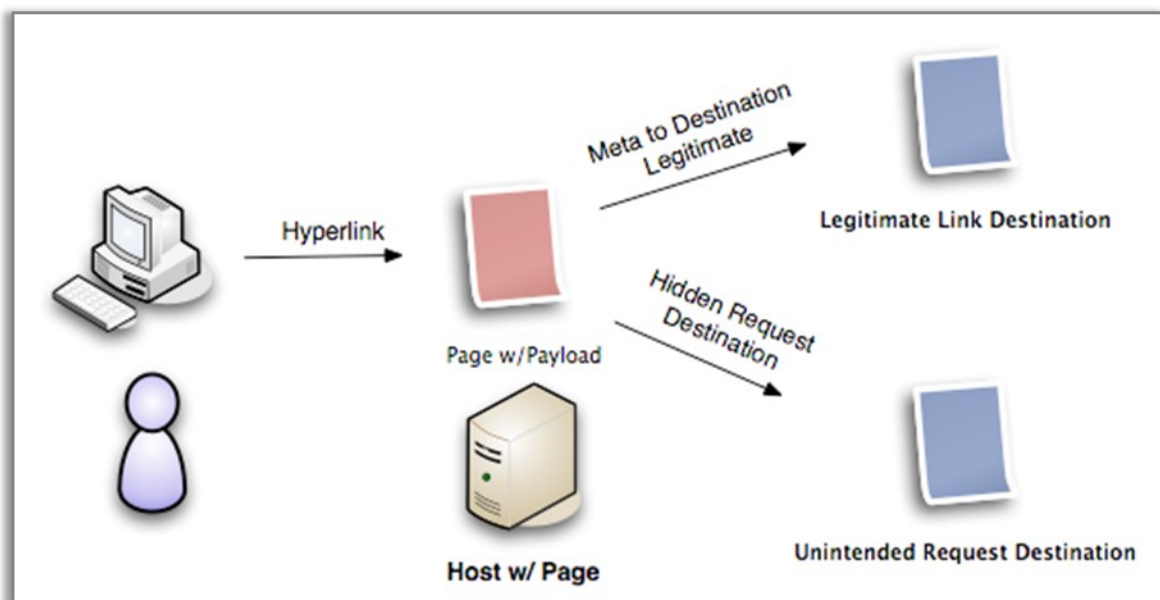


Figure 4: META refresh which includes hidden "dynamic" CSRF constructed from referrer leakage.

POST-based "dynamic" CSRF

It would also be possible to construct a POST "dynamic" CSRF request that requires no interaction from the user's browser. POSTs are by definition a bit more complex, and this approach could only succeed if necessary session data is leaked in a cross-domain fashion in order to

completely construct the POST, and a scenario existed where it was possible to insert HTML into the context of a user session on the target site.

Still, this could be possible in a multi-stage scenario, where for example CSRF tokens and other relevant information that are valid for the duration of a session are leaked via an offsite image or offsite content, and could then be combined into a customized forged POST request.

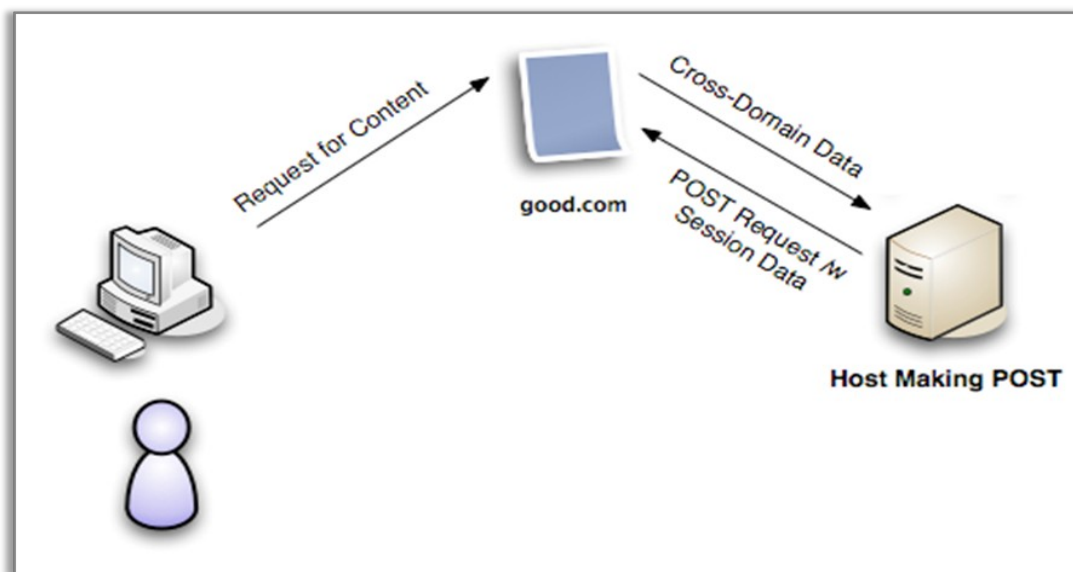


Figure 5: POST-based "dynamic" CSRF scenario.

Further Thoughts

An additional interesting use for dynamically constructing CSRF would be in JavaScript, as a payload for Cross-Site-Scripting or JavaScript Hijacking, in cases where this is feasible. Additionally, theft of CSRF tokens isn't the only use case for this approach; even for non-tokenized CSRF there are cases where dynamically constructing the URL per request may be useful. The larger point here is simply to remember the point Watkins made eight years ago, which seems to have been missed: CSRF is not purely a static attack based only on known, repeatable parameters, it can be fluid, dynamic, and tailored to each target.

References

1. Norm Hardy, "The Confused Deputy", 1988
<http://www.cis.upenn.edu/~KeyKOS/ConfusedDeputy.html>
2. Pete Watkins, "Cross-Site Request Forgery", Bugtraq, June 13, 2001
<http://www.tux.org/~peterw/csrf.txt>
3. Thomas Schreiber, "Session Riding", December, 2004
http://www.securenet.de/papers/Session_Riding.pdf

Shout outs

The authors would like to thank anyone who read this far, and also The Caine-Kronenbergs for being awesome, our bosses for not firing us, Jax Beach's nutty professor, the Internet, Sun Pharma, and the Archduke Ferdinand.