# Tamper Data: CSRF examined

By Russ McRee – ISSA member, Puget Sound (Seattle), USA chapter

## Prerequisites

Firefox web browser

## Similar Projects

Fiddler (for Internet Explorer)

I have a list of tools I simply can't function without. Adam Judson's Tamper Data is one of them. An easily installed Firefox add-on, Tamper Data is best described as an extension to view and modify HTTP/HTTPS headers and post parameters that is ideally suited for security testing Web-based applications.

It is strongly suggested that you only download and install the current version of Tamper Data (10.1.0) from Mozdev[1] or Firefox Add-ons site.[2]

It's best that you don't grab tools like Tamper Data from the likes of download.com or softpedia.com, given that installing a Firefox extension is akin to giving the author root access to all Web-based activity. Modifying Tamper Data to send all browser activity to a remote website would be a trivial attack.

Adam advised me that Tamper Data started like most Firefox extensions: he had an itch that needed scratching. While working to integrate an online banking site with a third party, he needed to see the headers in the redirect message that their site would send to the browser. He found the excellent Live Http Headers,[3] but didn't care for the format the results were displayed in. Once he'd created a new GUI, it seemed easier to expand the functionality beyond display of data than to install a local version of WebProxy to accomplish the same thing.

After exchanging email with the author of Liver Http Headers, they agreed they were building different tools, so Adam released Tamper Data to the public with the help of mozdev.org. Adam's main goal for Tamper Data has always been to serve as a quick and useful tool to do light penetration testing on websites he was developing. Tamper Data's been expanded to be a bit more useful when displaying web activity, though not always via Adam; most of the graphing code was sent as a user-submitted patch. Tamper Data has been translated into multiple languages by the great community at babelzilla.org.

According to its author, Tamper Data feels complete. He receives numerous requests to add functionality that would add complexity, but he doubts he'll pursue most of them. As an open source tool, patches/derivative work are always welcome.[4]

## Using Tamper Data

After installing Tamper Data from Mozdev or Mozilla, opening Tamper Data is as easy as clicking on Tools on the Firefox menu bar and choosing Tamper Data. The UI will open and you'll be immediately treated to all HTTP/HTTPS requests complete with timestamp, duration data, size, method (POST, GET), status (200, 404) content type, URL, and Load Flags.

One of my favorite pastimes is identifying vulnerable Web applications and reporting them to the vendor for repair. Every CVE, BID, SA, Xforce, and OSVDB number I've ever drawn was the result of analysis conducted with Tamper Data. As I write this column on March 7, 2009 there is a vulnerability advisory pending release by Secunia on April 8th, 2009 for a photo gallery that shall remain anonymous here in order to avoid disclosure before repair. I'll refer to it hereafter as GalleryApp. That said, by the time you read this, the advisory should be available at both my website[5] and Secunia.[6] The photo gallery in question is described as an easy-to-use, multilingual, flexible image gallery written in PHP. With GalleryApp installed on my test server I discovered that it was vulnerable to both cross-site request forgery (CSRF) and cross-site scripting (XSS).

To exemplify Tamper Data at its best, I'll walk through the GalleryApp analysis for your consideration.

First on the list, as I stepped through form inputs, was the discovery that input passed via `POST` to the `friend_full_name` parameter is not properly sanitized by the *admin.php* script before being returned to the user. A vulnerability of this nature can be exploited to execute persistent arbitrary HTML and script code in a user's browser session in the context of an affected site running GalleryApp.

I clicked *Start Tamper* in the *Ongoing Requests* window. There is extremely useful right-click functionality in the ongoing requests Tamper Data view, where one can graph selected or all results, export XML, view the source of specific request, or replay it in the browser.

1 http://tamperdata.mozdev.org.

2 https://addons.mozilla.org/en-US/firefox/addon/966.

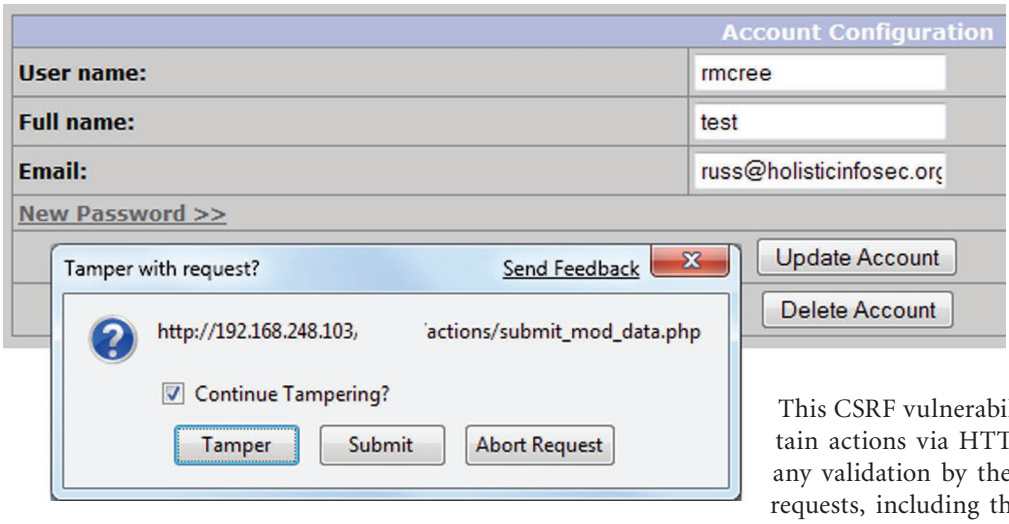
3 https://addons.mozilla.org/en-US/firefox/addon/3829.

4 Interview notes from Tamper Data author Adam Judson.

5 http://holisticinfosec.org/content/view/103/45.

6 http://secunia.com/advisories/34130.

**Figure 1 – Submitting test to GalleryApp**




I submitted the word test to the form via the GalleryApp *Account* configuration view at http://192.168.248.103/GalleryApp/admin.php?redirector=mod_user&page=mysettings.

The Tamper Data prompt opened and asked for confirmation to "tamper" (Figure 1).

Choosing tamper results in the appearance of the Tamper popup, complete with all POST parameter names and values. There is also additional right-click functionality in the Tamper popup view that allows you to populate a specified parameter with included XSS, data, and SQL strings, as well as the ability to add or delete elements and encode/decode the input. You can even add your own strings as you see fit.

Figure 2 displays Javascript pending submittal to the `friend_full_name` parameter.

The results were immediate in the form of a popup containing the document.cookie (PHPSESSID) results (Figure 3).

What makes this vulnerability rather ugly is that, if exploited, it becomes persistent, rather than a typical reflected XSS vulnerability. If an attacker were to opt for an `iframe src=` or `script src=` string, the admin user would immediately be treated to an entirely different web site, were the attack properly crafted.

You're probably asking yourself "How would an attacker make use of this vulnerability, given that it's harder to exploit POST submitted input?"

While that may be true, malicious scripts can be easily created to POST-forward an attack. One need only entice the intended victim to click a malicious URL via some form of social engineering. However, this particular application suffers from an additional vulnerability that allows for a hybrid attack combining CSRF and XSS.

This CSRF vulnerability allows users to perform certain actions via HTTP requests without performing any validation by the *admin.php* script to verify the requests, including the ability to create or delete accounts by tricking an administrative user into visiting a malicious web site.

As such an attacker could utilize the CSRF vulnerability to populate the `friend_full_name` parameter with what can basically be considered XSS defacement, pointing to a malicious website hosting malware or simply collecting credentials.

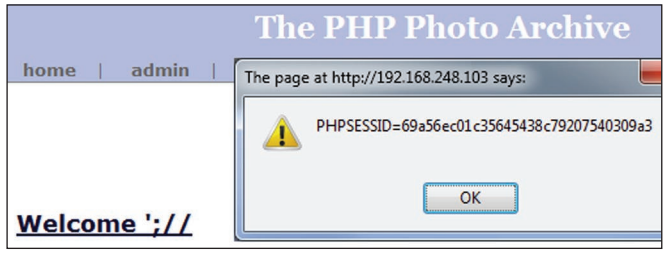



**Figure 3 – POST XSS vulnerability**

Creating CSRF exploits can be as easy as creating a webpage with an image that includes an `img alt=` tag with a GET request crafted to execute commands of the attacker's choosing. If POST parameters are vulnerable in a particular application, it's a bit tougher to analyze, but here's where Tamper Data shines again.

Determining the likelihood of, and validating a CSRF vulnerability is a manual process; automated scanning has limitations when it comes to determining formkey or token use.

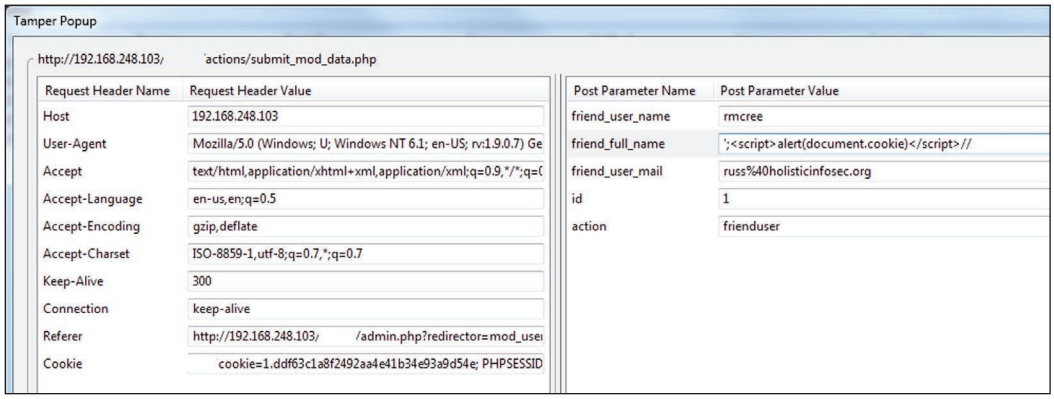



**Figure 2 – Submitting Javascript via Tamper Data**

As GalleryApp exhibits no use of a formkey or token (typical CSRF mitigation), one need only see what POST parameters are submitted to a given script to create a CSRF proof of concept.

Note: You'll see reference to common CSRF mitigations such as requiring checking the HTTP Referer header and, while

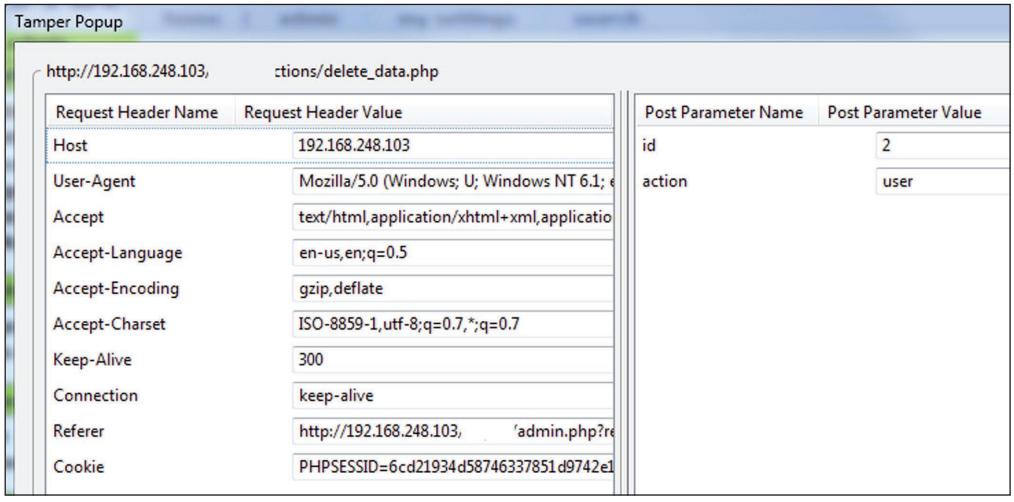



**Figure 4 – Checking for CSRF vulnerability**

this is a worthy step, use of a token is essential to preventing these attacks.

Sadly, in GalleryApp's case CSRF attacks are available throughout the entirety of the *admin.php* script. Again, this allows the ability to create or delete accounts by tricking an administrative user into visiting a malicious web site.

As an example, with Tamper Data running, I visited my GalleryApp test instance.[7] To validate the CSRF vulnerability, I first created a second user, selected *Start Tamper* in Tamper Data, and then choose to click *Delete* to analyze the POST parameters submitted to complete the action. As the application makes us of common logic it assigned the second user an id of 2. Logic like this allows for predictability that an attacker can make quick use of as seen in Figure 4. Again, note the absence of a token of any kind.

My attack proof of concept took the form of an HTML page that included a form with name (*deleteID*) and action parameters, as well as a POST method and hidden inputs. To round out the effort, the page included script for a quick timing delay and `document.deleteID.submit();` to complete the intended task. As seen in Figure 4, to complete my validation with the values seen via Tamper Data, the hidden input included:

```
<input type="hidden"
name="id" value="2">
<input type="hidden"
name="action"
value="user">
```

My final step included stepping on the URL for my PoC script. Figure 5 shows my GalleryApp user management page before the attack.

Figure 6 is the same view after the attack.

A malicious attacker would only need to send a similar URL to the site admin, thus causing user account creation or deletion. With the help of Tamper Data, it becomes immediately evident why CSRF is often referred to as a sleeping giant.[8]

## In conclusion

As I said in the introduction, Tamper Data is one tool I use every day without fail as it supports me in my job duties and what I feel is my civic duty to aid in the repair of broken Web applications.

The value of Tamper Data speaks for itself when reviewing HTTP/HTTPS requests, and given its extraordinary ease of installation, I insist that you make swift and immediate use of it. ;-)

*Cheers…until next month.*

## Acknowledgments

Adam Judson, Tamper Data developer, for his contributions to this month's columns.

## About the author

*Russ McRee, GCIH, GPEN, GCFA, CISSP, is a security analyst on the Security Incident Management team for Microsoft's Online Services. As an advocate of a holistic approach to information security, Russ' website is holisticinfosec.org. Contact him at russ@holisticinfosec.org.*

7 http://192.168.248.103/GalleryApp/admin.php?page=user&job=user.

8 http://jeremiahgrossman.blogspot.com/2006/09/csrf-sleeping-giant.html.

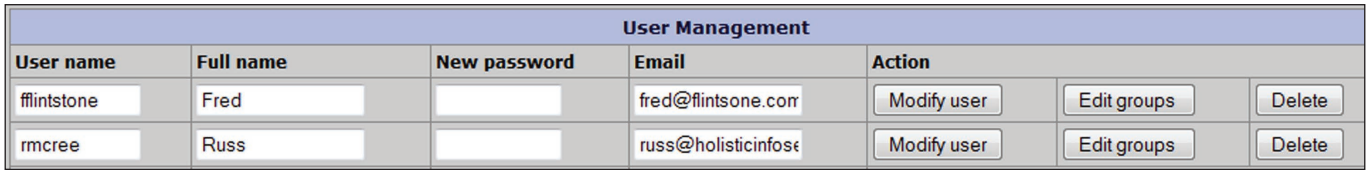



**Figure 5 – GalleryApp before CSRF exploit**

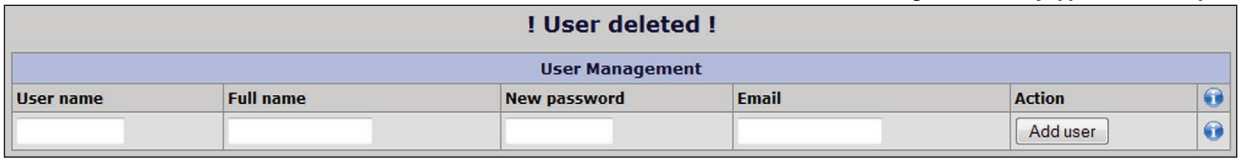



**Figure 6 – GalleryApp after CSRF exploit**