Cross-Site Request Forgery:
The Sleeping Giant

July 2007
Jeremiah Grossman
Founder and CTO, WhiteHat Security

Confused? Every year, for the past several years, the exact same Web attack is discovered, analyzed, and subsequently renamed. Whatever it’s called, it all means the same thing: An attacker is forcing an unsuspecting user's browser to send requests they didn’t intend and potentially compromising their own banking, e-commerce or other website accounts.

Attackers have begun to actively exploit CSRF vulnerabilities across the Web. Why now? Because it’s incredibly easy and the vast majority of websites are vulnerable to it. How do you stop an attack originating from a “real user,” who could be properly logged-in, from making a legitimate request - except the problem is they did not intend to make the request?

For those familiar with Cross-Site Scripting, Chris Shiflett (principal of OmniTI) said it best: “Cross-Site Request Forgeries are an almost opposite style of attack. Rather than exploiting the trust that a user has for a website, they (CSRF attacks) exploit the trust that a website has for a user.”⁵

Here’s an example of how a CSRF attack works:

Let’s say you’re logged-in to your online bank, which has a “Transfer Funds” feature. To transfer money from one account to another, you would fill out a Web-form similar to the one in Figure 1. After specifying the appropriate “From” account, “To” account, and dollar amount, you click the “Continue” button. For our purposes, let’s say the “From” account is “314159265,” the “To” account is “011235813,” and we’re transferring $5,000.

![Figure 1: Transfer Funds Web-form:](image_url)

When the button is pressed, your Web browser issues an HTTP request (Figure 2) to the Web server executing the process. Notice that the form values are located within the POST body of the HTTP request and the session credential (Cookie) in the headers.
POST http://webbank/transfer_funds.cgi HTTP/1.1
Host: webbank
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;)
Firefox/1.4.1
Cookie: JSPSESSIONID=4353DD35694D47990BCDF36271740A0C
from=314159265&to=011235813&amount=5000&date=11072006

Figure 2: Transfer Funds POST Request

If the request was successful, $5,000 would be transferred from account “314159265” to account “01123581.” What’s interesting is that many Web applications, such as transfer_funds.cgi, do not distinguish between parameters sent using GET or POST. The same Transfer Funds request could be updated to use GET and work just fine. In Figure 3, notice the POST method has been replaced by GET and the parameters in the HTTP body have been added to the query string.

GET http://webbank/transfer_funds.cgi?
from=314159265&to=011235813&amount=5000&date=11072006
HTTP/1.1
Host: webbank
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;)
Firefox/1.4.1
Cookie: JSPSESSIONID=4353DD35694D47990BCDF36271740A0C

Diagram 3: Transfer Funds GET Request

When bank customers are still logged-in, they may stumble across a Web page containing the HTML in Figure 4. A customer may find this link in a phishing email, message board post, instant message spam, etc. Notice the SRC attribute of the IMG tag has a similar URL value to that of Figure 3. – similar in that the “To” parameter value has been updated with another account number.

Diagram 4: http://hacker/foo.html

When the Web browser loads this page, the IMG tag forces a “forged” request with the URL specified and, if the customer is still logged-in, $5,000 from account “314159265” will instead to be sent to account “1618,” belonging to the hacker. To the online bank the request completely legitimate. CSRF attacks succeed because the customer is the one who is actually making the request by automatically sending the session credentials (cookies). Just about every important feature of every website has the potential for being exploited in this way.
The Confusion over CSRF Solutions

Well-meaning researchers have posited various solutions to the CSRF problem. However, CSRF is a particularly difficult issue to resolve and it is important to understand what really works and how. Let's take a look at some common suggestions and why they don't work before digging into an effective solution.

1. **Some online guidance says CSRF is solvable by forcing all sensitive requests over POST while denying GETs.** The thinking is you can only automatically force the user’s browser to send GET requests, via the IMG tag for instance, and not POST. But, this is not the case. JavaScript is fully capable of sending POST requests via form submissions. It's not to say that sensitive should not be restricted to POST for other valid security reasons, just not CSRF protection.

2. **XSS vulnerabilities are capable of rendering any CSRF solution as useless.** In fact, the Samy Worm\(^6\) that targeted MySpace managed to circumvent the website’s CSRF protections by using XSS to acquire the tokens ahead of time. The bottom line is if you plan to add CSRF protections, you must address any XSS issues as well to achieve the value.

3. **Another suggested solution to CSRF is the use of HTTP Request Referers.** The rational is that CSRFs have off-domain referers, since they are generated on another website other than the legitimate one. Normally, client-side data should never be trusted for security purposes. But, in the case of CSRF, it's been thought that attackers cannot modify the referer on behalf of the user. However, it has been found this in fact can be done through the use of certain browser\(^7\) and flash exploits\(^8\). For this reason, the method is not reliable, but perhaps can be viewed as another incremental roadblock.

A Real Solution

The best defense against CSRF attacks is unpredictable tokens, a piece of data the server can use to validate the request which an attacker can’t guess. For example, an “important” request could contain a digest of the user’s session credential, which is different for every user. And, for a little extra security, add a timestamp to the token to limit the window of opportunity as specified in the POST body of Figure 5: token = md5(session + timestamp).

```
POST http://webbank/transfer_funds.cgi HTTP/1.1
Host: webbank
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;)
Firefox/1.4.1
Cookie: JSPSESSIONID=4353DD35694D47990BCDF36271740A0C

from=314159265&to=011235813&amount=5000&date=11072006&token=4e0fc1a82c3ca3b9d2a1462570f6c674&timestamp=1184001456
```

Diagram 5: Transfer Funds POST Request with Token
Unless the attacker can obtain the user’s session credential, which they should never be able to do, they can’t manufacture a valid request even if they knew the scheme. This method can also be applied to either GET or POST requests.

There are two other solutions worth mentioning: The first is limiting the time in which a user’s session credential is valid. By enforcing inactivity timeouts, you limit the window of opportunity for a successful CSRF attack. (This is independent of whether the user decides to logout or not.)

The second method is password re-verification, in which the user must type in their password when requesting a particularly critical function.

WhiteHat’s Security Operations team continues to research CSRF and evaluate its impact on the more than 600 sites for which we provide ongoing website vulnerability management. As new solutions and methods of vulnerability identification are discovered, we will continue to share the findings in future whitepapers.

Notes:
1 http://en.wikipedia.org/wiki/Cross-site_request_forgery
2 http://www.securenet.de/papers/Session_Riding.pdf
3 http://www.zope.org/Members/jim/ZopeSecurity/ClientSideTrojan
4 http://www.cap-lore.com/CapTheory/ConfusedDeputy.html
5 http://phpsec.org/projects/guide/2.html
6 http://www.namb.la/
7 http://www.securityfocus.com/archive/1/411823
The WhiteHat Sentinel Service – Complete Website Vulnerability Management

Find Vulnerabilities, Protect Your Website – The WhiteHat Sentinel Service is a unique combination of expert analysis and proprietary automated scanning technology that delivers the most comprehensive website vulnerability coverage available. Worried about the OWASP Top Ten vulnerabilities or the WASC Threat Classification? Scanners alone cannot identify all the vulnerabilities defined by these standards. WhiteHat Sentinel can. Many of the most dangerous vulnerabilities reside in the business logic of an application and are only uncovered through expert human analysis.

Continuous Improvement and Refinement – WhiteHat Sentinel stays one step ahead of the latest website attack vectors with persistent updates and refinements to its service. Updates are continuous – as often as one day to several weeks, versus up to six months or longer for traditional software tools. And, Sentinel uses its unique “Inspector” technology to apply identified vulnerabilities across every website it evaluates. Ultimately, each site benefits from the protection of others.

Virtually Eliminate False Positives – No busy security team has time to deal with false positives. That’s why the WhiteHat Sentinel Security Operations Team verifies the results of all scans. Customers see only real, actionable vulnerabilities, saving time and money.

Total Control – WhiteHat Sentinel runs on the customer’s schedule, not ours. Scans can be manually or automatically scheduled to run daily, weekly, and as often as websites change. Whenever required, WhiteHat Sentinel provides a comprehensive assessment, plus prioritization recommendations based on threat and severity levels, to better arm security professionals with the knowledge needed to secure them.

Unlimited Assessments, Anytime Websites Change – With WhiteHat Sentinel, customers pay a single annual fee, with unlimited assessments per year. And, the more applications under management with WhiteHat Sentinel, the lower the annual cost per application. High volume e-commerce sites may have weekly code changes, while others change monthly. WhiteHat Sentinel offers the flexibility to assess sites as frequent as necessary.

Simplified Management – There is no cumbersome software installation and configuration. Initial vulnerability assessments can often be up-and-running in a matter of hours. With WhiteHat Sentinel’s Web interface, vulnerability data can be easily accessed, scans or print reports can be scheduled at any time from any location. No outlays for software, hardware or an engineer to run the scanner and interpret results. With the WhiteHat Sentinel Service, website vulnerability management is simplified and under control.

About the Author
Jeremiah Grossman is the founder and CTO of WhiteHat Security, a world-renowned expert in website vulnerability management, co-founder of the Web Application Security Consortium, and recently named to InfoWorld's Top 25 CTOs for 2007. Mr. Grossman is a frequent speaker at industry events including the BlackHat Briefings, ISACA, CSI, OWASP, Vanguard, ISSA, OWASP, Defcon, etc. He has authored of dozens of articles and white papers, credited with the discovery of many cutting-edge attack and defensive techniques and is co-author of the book XSS Exploits. Mr. Grossman is frequently quoted in major media publications such as InfoWorld, USA Today, PCWorld, Dark Reading, SC Magazine, SecurityFocus, C-Net, SC Magazine, CSO, and InformationWeek. Prior to WhiteHat he was an information security officer at Yahoo!

About WhiteHat Security, Inc.
Headquartered in Santa Clara, California, WhiteHat Security is a leading provider of website vulnerability management services. WhiteHat delivers turnkey solutions that enable companies to secure valuable customer data, comply with industry standards and maintain brand integrity. WhiteHat Sentinel, the company's flagship service, is the only solution that incorporates expert analysis and industry-leading technology to provide unparalleled coverage to protect critical data from attacks. For more information about WhiteHat Security, please visit www.whitehatsec.com.