Identifying Robust Defenses for Login CSRF

Balamurugan Prabakaran, Gurumurthy Athisenbagam and Karthik Thotta Ganesh
Department of Computer Science
University of Illinois at Chicago

Abstract

CSRF is a widely exploited vulnerability in websites. We show that many web sites that have fixed their CSRF vulnerabilities are still vulnerable to login CSRF attacks, which lets an attacker log-in a user with his credentials. Based on our analysis on the top 50 sites, we have identified that 64% of the websites were vulnerable to this attack. We suggest a browser-side solution of blocking cross-domain cookies and implemented a plugin to Firefox. Our test results show that we can effectively prevent login CSRF attacks. We also discuss the costs of this defense in terms of user web experience. We suggest some simple heuristics on how to improve this scenario, and also discuss an alternative of uniquely identifying login cookies. We also discuss the issue of how to notify the server on the removal of the corresponding cookie at the browser. We were not able to come up with a solution but we will demonstrate how even the industry standards are lacking in this aspect.

Key Words: Same Origin Policy, Cross Site Request Forgery, Session Cookie.

1. Introduction

Cross-site request forgery is a type of attack in which malicious web requests are sent to websites via trusted users. The attacker in fact exploits the trust the website has in the user's browser, because for every request to the website the browser automatically sends a cookie that was earlier set by the website. This attack is possible because of the fact that the website fails to verify if the request indeed originated from one of its valid pages which it serviced to its trusted user.

This attack can be evaded by inserting action tokens into the pages that the website provides to its trusted users. These tokens are embedded into the web pages and are sent with every request to the server, originating from that page. The action tokens are generated using a function that involves the user's credentials (usually a cookie), a server secret and the URI. Since these details are also known on the server side, the server can recompute the action-token and can verify if the request indeed came from a legitimate page serviced to a trusted user. This attack can also be evaded by strict “Referrer” validation [1]. But it does not provide a robust defense because several organization infrastructures filter this header at their perimeter citing privacy concerns [1].

Many sites which were vulnerable to this attack have deployed appropriate defenses as mentioned above, but fail to defend themselves against a new variation of this attack, namely login CSRF [1]. This attack involves logging the victim into the honest website as an attacker. This attack can be used to collect information like search history, credit card information and can also be used to execute cross site scripts. This attack is possible because the web server fails to maintain state information before a user actually logs into their service. The attacker exploits this no-state condition to log-in the victim with his credentials.

The authors of [1] have suggested use of strict referrer validation as a defense for this attack. Their experimental study indicates that the referrer header is not filtered over a majority of HTTPS connections, and since login requests are generally sent over HTTPS they suggest that this could be an effective defense.

The login CSRF can also be defended with the use of “login” cookies [3]. This involves setting a special 'login cookie' whenever a login form is requested and embedding an action token in the login form. So when a login request is sent, the action token and login cookie are sent to the server along with the other credentials. The server can then recompute the action token to verify if the request indeed came from a legitimate login form.

The solutions for preventing login CSRF discussed above have some limitations. The strict referrer validation requires that all the users' infrastructures and browsers implement this header properly [2]. In cases where it is not implemented or implemented incorrectly it will result in service inaccessibility to a legitimate user. In the login cookie method, the bots can flood the server with bogus login form requests. This will require the server to

generate unique session-ids to each of these requests, which can be a serious overhead.

The origin header method improves on the referrer header method by eliminating privacy concerns that lead to referrer blocking, and eliminates the need for secret tokens over both HTTP and HTTPS [1]. But CSRF is a vulnerability introduced by the browser and hence we should attempt to fix it at the browser. The proposed method does not discuss a way in which the origin header can be secured from spoofing, so effectively if the origin header is spoofed the attacker can still execute a CSRF. Their draft to the IETF also discusses about backward-compatibility which involves maintaining a table of compliant user-agents, and this method has several limitations[2].

As part of this project we wanted to demonstrate the prevalence of these vulnerabilities even today by analyzing the 50 most popular websites for this vulnerability. We discuss our attack vectors in section (2) and our survey results in section (2.2).

The login CSRF is a vulnerability introduced by the browser because they allow automatic redirection to third party sites via scripts and allows corresponding responses to set cookies without verification, and hence we attempted to fix this at the browser. We discuss more about our approach in section (3). And we provide implementation details in section (3.1). We identified several usability issues as a result of this approach which we document in section (3.2).

Though this prevents the attacker's cookie from being set in the browser, we will also need to notify the server of the failure to set the cookie in the users' browser. This will enable the server to reset the state it had created for this connection. We would like to explore on how to this server notification can be achieved and on how to reset the bogus connection's state. Though we were not able to come up with a solution to this problem, we have recorded our observations on this issue in section (4).

We discuss related work in section (5.1) and our conclusions in section (6).

2. CSRF Vulnerabilities

In this section we describe three vulnerabilities that we discovered during our analysis to determine Login CSRF vulnerabilities.

2.1 CSRF

2.1.1 eBay

The CSRF bug found on eBay site would let an attacker to modify the shipping address of the victim's account to any other postal address. To exploit this vulnerability, an attacker causes a logged-in user's browser to send a request to eBay.com (User Address page). Since User Address page does not protect against CSRF attacks, the request would update the shipping address. If the attacker changes the shipping address to his own, he will receive the items purchased by the victim. In order to exploit this vulnerability attacker just need to fill the form inputs and send a post request and there is no secret authentication value or IDs that the attacker has to guess, hence attack will succeed.

We verified this attack in Firefox 3.0.10. We are in the process of reporting this vulnerability to eBay.

2.1.2 Craigslist

Nearly every action a user can perform in settings page has CSRF vulnerabilities. An attacker can simply take over the account by making use of "change email-address" feature which prompts for new email address. The attacker can send a request to send an email to an address of the attacker's choice and later activate the account by providing a new password. Thus an attacker can successfully hijack the victim's account. The other vulnerabilities include setting the default site and the duration to stay logged in.

We verified this attack in Firefox 3.0.10. We are in the process of reporting this vulnerability to Craigslist.

2.1.3 Dailymotion

The CSRF vulnerability discovered in dailymotion.com personal info page would allow an attacker to modify the personal information of a victim. The attacker could use this vulnerability to violate user privacy by modifying birth date, first name, last name and even gender. The other vulnerability found in profile page lets an attacker to update the home page and description field.

We verified this attack in Firefox 3.0.10. We are in the process of reporting this vulnerability to dailymotion.

2.2 Login CSRF

2.2.1 Prevalence of Login CSRF

We analyzed the 50 most popular websites for login CSRF vulnerability [5]. Out of which 64% of the sites were vulnerable to this attack. 26% of the sites had secured themselves and remaining 10% of the sites did

not have a login page. Out of the 32 sites that were vulnerable, we found that 25 of them do not check for a session cookie on logout request. Hence this can be subject to a timing based attack, in which attacker can logout the user and log the victim with the attacker's credentials. We also identified cases where the sites attempt to fix login CSRF but does it incorrectly. Many of the sites had either login CSRF vulnerabilities or a history of vulnerabilities. The fact that so many sites are vulnerable to login CSRF attacks shows that many web administrators are unaware about the risk of login CSRF vulnerabilities.

.

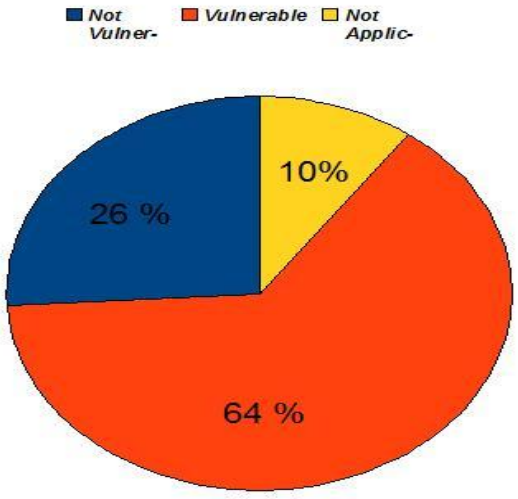

Figure 1. Survey on Top 50 sites for Login CSRF Vulnerability

2.2.2 Case Study: Amazon

Amazon lets its user to add their credit card details to the account. A web attacker can use login CSRF to mount the following attack.

1. The victim visits a malicious web site.
2. The attacker silently logs the victim into his account.
3. To fund the purchase, the victim may add a credit card to the account, but the details have actually been added to the attacker's Amazon account.

<form id="menubar_login" name="menubar_login" action="https://www.amazon.com/gp/flex/sign-in/help-select.html/ref=ya_sign_in_" method="post">

<input type="hidden" value="1" name="useRedirectOnSuccess"/>

<input type="hidden" value="/gp/css/homepage.html" name="path"/>

<input type="hidden" value="sign-in" name="action"/>

<input type="hidden" value="https" name="protocol"/>

<input type="text" value="tgkarthik@XYZ.com" name="email"/>

<input type="password" name="password" value="XXXXXX"/>

<input type="hidden" value="Firefox 3.0.8 Windows" name="metadata1"/>

<input type="hidden" value="timezone: 6 execution time: 14" name="metadata3"/>

<input type="hidden" value="0" name="x"/>

<input type="hidden" value="0" name="y"/>

2.2.3 Case Study: ask.com

Many search engines provide search history feature where user can keep track of his/her search queries. Search queries contain sensitive details about the user's activities and interest [6] and could be used by an attacker to steal user's identity or to spy on the user. This can be done by logging the user into the search engine as the attacker and the search strings are stored in the attacker's search history, and the attacker can retrieve the queries by logging into his or her own account. In Ask.com once if the user logs in, the user's nick name is displayed in all subsequent pages instead of the userid. This feature can be used by the attacker to trick the user by displaying keywords such as “Guest”, “Ask”.

<form id="menubar_login" name="menubar_login" action="http://mystuff.ask.com/service/api/login" method="post">

<input type="hidden" alue="4QrcOUm6Wau+VuBX8g+IPg==" name="crep"/>

<input type="hidden" value="" name="random"/>

<input type="hidden" value="" name="prod"/>

<input type="hidden" value="ask.nav.SigninDialogController.signInCallback" name="fn"/> <input id="password" type="password" style="width: 95%;" autocomplete="off" maxlength="32" value="xxxxxxxxx" name="password"/>

<input id="email" type="text" style="width: 95%;" maxlength="64" value="admin.webhistory@XYZ.com" name="email"/><input type="hidden" value="1" name="permchk"/></form>

Figure 2. Facebook Cookie snapshot

2.2.4 Case Study: Facebook

Facebook is an exception where the site attempt to fix login CSRF but does it incorrectly. In Figure 2, you can observe that the “login” cookie thats used has a static value. Facebook's login form, however, requires cookie to be sent thats set when user visits login page along with the login request. This approach is inefficient has an attacker can place code to render facebook.com and then issue cross site request.

3. Our Approach

In our approach we would like to block the cookie being set by the target website, whenever a illegitimate login is attempted. This is done by checking if the top level domain of the frame to which the illegitimate login response is directed to, does not have the same domain as the cookie being set. This will prevent the attacker from logging-in into the target site using the victim's browser via cross-domain request. The only way the user can log into a particular website is by visiting a legitimate login form of the website. In a sense we are trying to extend the SOP with regard to login cookies to a page and not individual pages contained in it.

Traditionally the notion on SOP has been confined to that of a frame. We attempt to extend this principle to the entire window (i.e. all the frames the window is comprised of) as shown in figure [ ] with respect to cookies . Also criteria to determine same-origin has been restricted to domain-names instead of domain, protocol and port. This way a response to a cross-domain login request will not be able to set the cookie on the browser on behalf of some other domain.

3.1 Implementation

We implemented the above approach as an add-on to Firefox. This add-on constantly monitors cookie activity in the browser. It first calculates the top-level domain of the page (eg: uic.edu) to which the response is directed. Then the add-on examines the host/domain attribute in the cookie being set, and calculates the top-level-domain of the cookie (eg: google.com). If there is a mismatch in these calculated attributes then the response is not allowed to set the cookie. It should be noted that is such cases the page is rendered though the cookie it attempted to set was rejected by the plugin. Our initial tests with this implementation show that we are able to avert login CSRF attacks with this approach. But as we see in the next section it is done at the cost of the user's web experience.

3.2 Impact on user's web experience

Since the current implementation blocks cross-domain cookies in all scenarios, there is a significant impact to the users' web experience. Our testing revealed that mashups that puts together applications that require cookies failed to work. And with the plugin installed, users cannot create their own html pages that put together multiple frames rendering URLs from different domains. Also websites that have multiple frames rendering cross-domain content might not work, if it very reliant on the information that is being contained in the cookies.

4. Server side notification

In our proposed solution, it is important to notify the server that the cookie is not set. This should be done so that the server can destroy the corresponding session. This is crucial because it can be a potential vector to carry

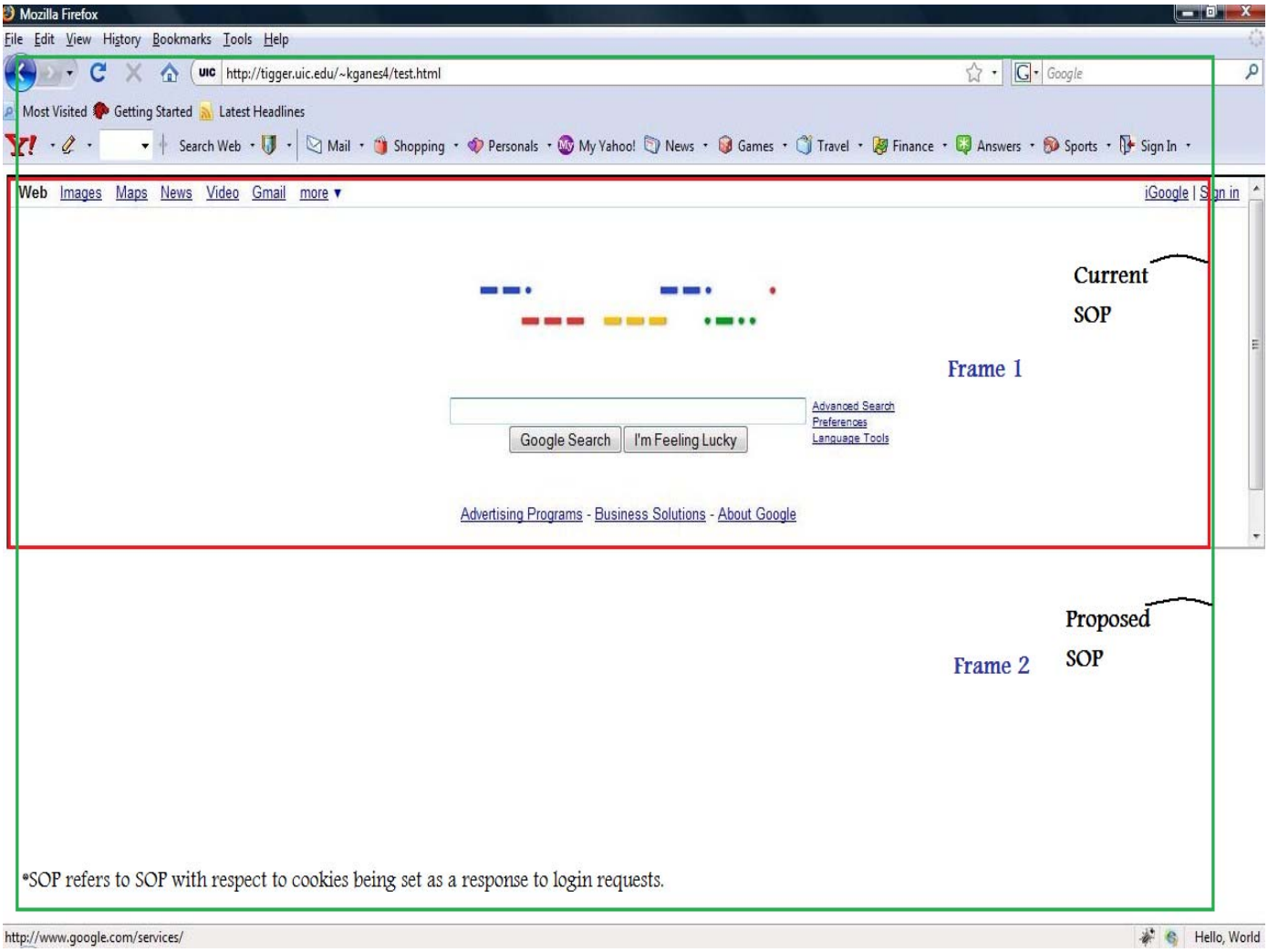

Figure 3. Window showing the proposed SOP

out denial of service attack. The common attack scenario taking advantage of asymmetry being, the attacker effortlessly sends out Login CSRF request continually to the same server but the server has to allocate new session for each request.

An analysis of the existing solution for handling this scenario revealed that solutions are more a server side solution. Random deletion and timeout at server are the more prominent solution in practice. In Random deletion, the server deletes the sessions randomly when it reaches the maximum number of connections. Alternatively the server times out the session at server by tracking inactivity i.e. a garbage collector to server to delete inactive sessions.

It was found that the top websites handled this issue by running a script at the client that checks for cookie deletion periodically. And if it identifies that the cookie is deleted it will redirect to a login page.

To sum it, it is difficult to identify an ideal value for timeout. Also notifying the server which session got invalidated is a hard problem. We suggest a collaborative solution where the browser will notify the server with details of the session it invalidates.

5. Future Work

Our current implementation simply blocks cross-domain cookies without validating if it was intended for a login form. This affects the current web experience of the user

to a great extent. We suggest the following possible approaches to this problem.

To identify if the page is a login form we can employ some simple heuristics; we can parse the document's DOM object and see if the page has a password field. Then if it does have a password field we can check if it has an action associated with the same form object or we can depend on the server to set cookies for login responses with a special attribute [1] to indicate that they are login cookies.

6. Related Works

6.1 Third party cookie blocking

The same origin policy (SOP) of current browsers causes privacy concerns. The chief concern with SOP was that it permitted tracking of users extensively across any number of collaborating domains without permission (in the simplest form, by simply including tracking code in an IFRAME pointing to a common attacker.com resource on any number of web pages, so that the same attacker.com cookie can be correlated across all properties). Widespread criticism eventually resulted in many browsers enabling restrictions on any included content on a page setting cookies for any domain other than that displayed in the URL bar. This mechanism is called Third party cookie blocking. This was to prevent third-party content (advertisements, etc) included via <IMG>, <IFRAME>, <SCRIPT>, and similar tags, from setting tracking cookies that could identify the user across unrelated sites relying on the same ad technology.

Third party cookie blocking was intended to handle the privacy issues. But we found that this is an effective mechanism to avert the Login CSRF attack. But most commercial browsers today like Mozilla Firefox, Safari etc have this feature disabled by default. Microsoft Internet Explorer is a particularly interesting case; it rejects third-party cookies with the default "automatic" setting but permits sites to override this behavior by declaring a proper, user-friendly intent through compact P3P privacy policy. If a site specifies a privacy policy and the policy implies that personally identifiable information is not collected (e.g., P3P: CP=NOI NID NOR), with default security settings, session cookies are permitted to go through regardless of third-party cookie security settings.

7. Acknowledgments

Thanks to Dr. V.N. Venkatakrishnan and peers for their comments and suggestions.

8. References

[1] Adam Barth, Collin Jackson, John C. Mitchell, “Robust Defenses for Cross-Site Request Forgery”. 15th ACM Conference on Computer and Communications Security (CCS 2008), October 2008.

[2] HTTP Origin Header – IETF mailing list http://lists.w3.org/Archives/Public/ietf-http-wg/2009JanMar/0037.html

[3] Django-developers Google Groups (Login Cookie Implementation) http://markmail.org/thread/mqf5yiocte6jchp6

[4] Michal Zalewski, *Browser Security Handbook, part 2*, Copyright 2008 Google Inc, rights reserved.

[5] http://www.alexa.com/

[6] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In InfoScale ’06: Proceedings of the 1st International Conference on Scalable Information Systems, 2006.