IMPROVING CROSS-SITE REQUEST PRIVACY AND SECURITY:

CLIENT-SIDE CROSS-SITE REQUEST WHITELISTS

By

JUSTIN CLAYTON SAMUEL

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelor's degree
With Honors in

Computer Science

THE UNIVERSITY OF ARIZONA

May 2009

Approved by:

_____

Dr. Beichuan Zhang
Department of Computer Science

Abstract

It is common for websites to instruct browsers to make requests to third-party sites for content, advertisements, as well as for purely user-tracking purposes. Additionally, malicious or compromised websites can perform attacks on users and websites by leveraging their ability to initiate cross-site requests in a user's browser. There are currently only limited protections against these threats to user privacy and security. We design and implement RequestPolicy, an extension for Mozilla browsers that provides a client-side whitelist for controlling cross-site requests.

1 Introduction

Many websites today include content from domains other than the origin domain from which the page was retrieved. This is accomplished by the originally-requested webpage including instructions for the user's web browser to make further requests to other websites for the additional content. The web browser dutifully retrieves the additional content. Many users are not aware of these cross-site requests happening.

There are, however, privacy and security concerns that stem from the ability of a website to instruct the browser to make requests to any other website.

1.1 Privacy Concerns

Any website that receives cross-site requests obtains, as a result of the request, information about the user. This includes information about the pages a user has viewed and when the pages were viewed. In some cases, this is the purpose of the cross-site request. For example, many website traffic analytics services work by having the website owner include instructions in their website's HTML that tell browsers to make requests to the analytics service.

In other cases, the intention of a cross-site request is only to include cross-site content as part of a webpage the user has requested. For example, images or videos may be included in a webpage but the actual images and videos are hosted at a different website than that which the user has requested the page from. Even though the intention in this case is not to provide information to third parties, such cross-site requests have as a side effect the ability for

a third party website to be able to collect information about the browsing habits of users.

In either case, whether intended to disclose browsing information or not, users are often revealing more about their browsing habits than they wanted to. This information and the potential for third parties to link a user's separate website browsing sessions not only decreases the privacy of regular users browsing the web, but also puts users of anonymizing networks and proxies at risk of de-anonymization.

1.2 Security Concerns

Separate from privacy concerns of non-malicious cross-site requests, malicious cross-site requests pose a variety of threats to users of modern web browsers. The attacks that make use of cross-site requests range from impersonating the user for the purpose of server-side data manipulation to theft of user data, network attacks, and even social attacks.

The most well known of attacks related to cross-site requests are Cross-Site Request Forgery (CSRF) attacks. CSRF attacks begin when a website (either malicious or compromised) instructs a web browser to make requests that the user did not intend to make. The request becomes a CSRF attack when the destination website believes the user meant to perform the request. Potential ways that such an attack can be used include requests to a user's bank website telling the bank to transfer funds as well as requests to the Google search engine for searches involving illegal activities that Google then maintains in the user's search history.

CSRF attacks are sometimes referred to as "session riding" attacks because they often require that a user be logged into a website where they have an account. This term, though, is not general enough to account for all types of CSRF attacks. Another category of CSRF attacks includes "session fixation" attacks [17]. Rather than exploiting the trust that a website has that a request was made by the user they expect, session fixation attacks involve an attacker using CSRF to make a user's browser join a website session that belongs to the attacker. Such a situation can result in the attacker being logged in as the user if the user logs into a website after the session fixation attack.

There are many other attacks besides CSRF attacks that can be performed through the use of malicious cross-site requests. One such malicious use of cross-site requests is that of drive-by downloads, which are often per-

formed by compromised websites that do not themselves host malicious files but rather only direct the browser to download the exploit code from a different website. Social attacks can also be performed with cross-site requests. Such attacks include causing a browser to request illegal content so that it would appear that the user had been trying to download the content.

Cross-site requests are also used as part of data theft, often in conjunction with cross-site scripting (XSS) attacks. In order to steal data from a user's browser, code from an XSS attack running in a browser will often need to make cross-site requests to send the stolen data back to the attacker's website where it can be collected. Cross-site requests can also be used as part of distributed denial-of-service (DDoS) attacks and to perform network scanning within a user's local network. Recently, cross-site requests gained further attention for their use in UI redress ("clickjacking") attacks.

1.3 Problem and Goal

Due to the integral nature of cross-site requests as part of many popular websites, disabling cross-site requests completely in the browser is not an option in terms of usable solutions to the security risks posed by cross-site requests. On the other hand, many users desire protection against the privacy and security risks posed by cross-site requests.

Until now, there has been no ability for users of modern browsers to maintain complete control over cross-site requests made by their browser. Therefore, there has been no way for users to maintain complete privacy and security with respect to cross-site requests. In this work, we focus on remedying this situation by designing and developing a highly usable tool to provide users with control over cross-site requests. This has taken the form of RequestPolicy [13], a cross-site request whitelist implemented as an extension for Mozilla browsers such as Firefox.

The rest of this paper proceeds as follows: In Section 2, we first look at the existing partial protections available against the threats posed by cross-site requests. In Section 3, we discuss the design and implementation of RequestPolicy. Section 4 then discusses the results of our work and Section 5 concludes.

2 Related Work

There are a handful of tools and techniques users currently have for improving their privacy with respect to cross-site requests, though none are complete. Proxies and browser extensions exist to suppress sending `Referer` headers in cross-site requests [10, 12]. The risk of privacy loss from cross-site requests, however, is not only due to `Referer` headers. Other sources of information about the user include cookies, the user's IP address and even the URL being requested. The requested URL is a privacy risk as it can contain information such as the user's session ID from the originating site.

Most browsers have options to block third-party cookies. There are also browser extensions [5] that allow greater control over allowed cookies. However, as with `Referer` headers, blocking cookies does not eliminate the risk of privacy loss from cross-site requests.

A few tools do exist that can block some cross-site requests. Such tools include the KarmaBlocker [7] browser extension that uses predictive analysis for cross-site request blocking and blacklist-based advertisement blocking browser extensions such as Adblock Plus [1]. Predictive analysis has the potential to block many undesirable cross-site requests given the correct rules and history, but privacy cannot be guaranteed and many undesired cross-site requests may still be allowed. Advertisement blocking is a very partial solution for the obvious reason that it only prevents privacy lost to advertising companies. Further, because advertisement blocking systems are blacklist-based, they have a delay time between detection of false negatives and corrective updates to the blacklist, which is often updated automatically. The idea of manual rather than subscription blacklisting has been applied in the BlockSite extension [2], but extensions such as this have the same inadequacies as subscription-based advertisement blocking tools in addition to having user interfaces not intended for fine-grained cross-site request control.

Users may also seek anonymity in their browsing as a method of maintaining privacy. To this end, they may use proxies that hide their true IP address from destination websites. Such proxies include Tor [15] and Psiphon [11]. However, cross-site requests should be of concern to users of anonymizing proxies due to their potential for de-anonymization. This potential exists due to the fact that, through cross-site requests, third parties have greater ability to correlate separate user sessions and therefore de-anonymize users.

Traditional focus on the security threats posed by cross-site requests has been largely limited to the threat of Cross Site Request Forgery (CSRF).

Solutions have mostly been on the server side [4, 17, 8], leaving clients at risk with any site not implementing CSRF protections. The most common server-side protection is the use of nonces sent to the client in an earlier request that must be included in later requests to the same website. The inability for an attacker to guess or otherwise obtain these nonces (assuming, for example, that a website is not vulnerable to XSS attacks) allows the server to differentiate between legitimate and illegitimate requests from the client. Nonces only prevent CSRF attacks against the specific websites using them.

Other server-side protections include checking the `Origin` header that is being introduced into browsers [17]. The `Origin` header, like nonces, provides websites that make use of header the ability to detect CSRF attacks. There have been other attempts at server-side protections against CSRF attacks [8], but all such protections share the same limitations: they only protect against CSRF attacks, not all attacks using cross-site request, and only protect against CSRF attacks destined for websites implementing specific protections.

Client-side attempts at protecting users from cross-site request attacks exist but are very incomplete. The RequestRodeo proxy [18] exists to protect users against some CSRF attacks, but the protection provided is very partial, being limited to only certain HTTP requests and no HTTPS requests. NoScript [9] provides the ability for users to control the running of JavaScript, which is often used to initiate malicious cross-site requests. However, there are still many attacks that use cross-site requests which NoScript is unable to protect against, most importantly those that do not use JavaScript but instead rely solely upon HTML-initiated requests. NoScript does specifically provide protection against certain cross-site POST requests, but CSRF dangers are posed by GET requests and, as mentioned, CSRF attacks are not the only danger presented by cross-site requests.

The risk of network scanning is mitigated by the use of NoScript, though it has been shown that network scanning attacks can be done without JavaScript [3]. Another tool, LocalRodeo, provides more complete protection against network scanning attacks [20].

3 Design and Implementation

The solutions discussed in Section 2 all have one or more inadequacies in common with respect to their ability to fully protect users from the privacy

and security threats posed by cross-site requests. Either they only provide protection against some issues or they only apply to some requests. In order to provide users with a tool that gives them full protection against these threats in all cases, we will need a client-side solution rather than a server-side solution. Not only do server-side solutions need to be implemented on all servers to completely protect the client from some classes of attacks, but more importantly server-side solutions do not help against other classes of attacks such as those whose only goal is to make the client request illegal content. There are also privacy aspects in addition to the security aspects, and server-side protections will not help with respect to privacy.

Our solution will therefore need to be on the client-side. However, it will need to offer complete and reliable privacy and security protection, unlike what is offered by existing blacklist-based or predictive blocking tools. The solution is thus to use a client-side whitelist with a default-deny policy. This means that all cross-site requests will be blocked by default and only allowed if the user has approved the cross-site request. The best place to implement such a whitelist is within the browser as a browser extension, as that is the place where the most information about the user's intentions will be available. Additionally, when the user encounters problems while browsing, they will be looking in the browser for the solution.

The major challenges with this approach are to develop an intuitive user interface and maintain a high degree of usability within the tool. We will recognize that our target audience will not be the average Internet user, but rather users who are concerned enough with their online security and privacy to use a tool that does add some degree of interference with their browsing experience. Our goal, however, will be to minimize the extent of this interference and make the tool as usable to the largest user base possible.

We implemented a client-side cross-site request whitelist as a Mozilla browser extension called RequestPolicy. Implementation as an extension for Mozilla browsers as opposed to an extension for other browsers was chosen because of the ease of implementation of Mozilla browser extensions, the extensive API allowing large amounts of access from extensions, as well as the wide reach of Firefox, the most popular Mozilla browser. Additionally, the use of browser extensions is very popular among Firefox users. Firefox even provides a built-in feature that allows users to search for and install extensions hosted by the Mozilla project [6].

Method	Execution
Images	`<img>` tag, CSS styles
Script files	`<script>` tag
Stylesheets	`<link rel="stylesheet">` tag
Frames	`<frame>` and `<iframe>` tags
HTML-based redirects	`<meta http-equiv="refresh">` tag
Header-based redirects	`Location` header, `Refresh` header
Prefetched webpages	`<link rel="prefetch">` tag
Cross-site XMLHttpRequest	New feature in Firefox 3.5
Favicons	`<link rel="icon">` tag
Plugin-initiated requests	Flash, QuickTime, Java

Table 1: Methods of initiating cross-site requests within a browser.

3.1 Blocking Cross-Site Requests

All cross-site requests that are not intended by the user should be blocked by default. A partial list of the many ways cross-site requests may be initiated in a browser is shown in Table 1.

In order to attain the most accurate behavior possible, the extension was implemented to block as much as possible without requiring special cases for different types of content. This minimized the chance that an oversight of a type of cross-site request could result in holes in the privacy or security the extension provides. The Mozilla XPCOM [16] nsIContentPolicy interface provided our extension the ability to make per-request blocking decisions for the majority of requests based on the URL of the originating document and the requested URL.

When URLs use IP addresses rather than domain names as the URL host, IP addresses are treated as distinct from different IP addresses as well as any domain names. The actual classification and comparison of origin and requested URLs is assisted by various XPCOM interfaces, including nsIURI and nsIEffectiveTLDService.

Additional logic was used to detect special cases such as when a user clicks a cross-site link or submits a cross-site form. These are cases where the user intended to make a cross-site request so the request needs to be allowed and not be subject to the whitelist. Additionally, there are some types of cross-site requests such as those performed through header-based redirects that are

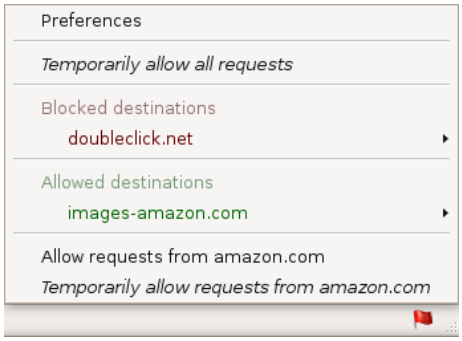

Figure 1: The RequestPolicy menu while visiting `amazon.com`. The destination `amazon-images.com` has been white whitelisted for requests originating from `amazon.com`.

not handled by the nsIContentPolicy interface. For these, we used separate observer interfaces within Mozilla to detect the redirect before it occurs and block the redirect if it is not allowed by the user's whitelist rules.

3.2 User Interface

In order to notify users of blocked content on the current page, a RequestPolicy icon was added to the browser's status bar, the bar that runs along the bottom of the browser window. When cross-site requests are blocked, the RequestPolicy icon changes to indicate that there is blocked content.

Ideally, blocked content would be immediately made apparent to users within the webpage itself so that users know what specific items were blocked. For most types of content, though, it is difficult to indicate to users exactly what is blocked. For example, JavaScript and CSS stylesheets do not occupy a specific visual portion of a web page, though they can greatly impact the appearance of a site when blocked. Blocked images, though, can be intuitively indicated within the webpage. RequestPolicy therefore indicates blocked images with a special graphic and also displays the image's blocked destination host when the cursor is hovered over the graphic representing the blocked image.

Once a user becomes aware of blocked cross-site content, in many cases they will want to determine which requests were blocked as well as which were allowed. Further, they will often want to add or remove items from their whitelist at that time. One type of interface for per-site whitelisting

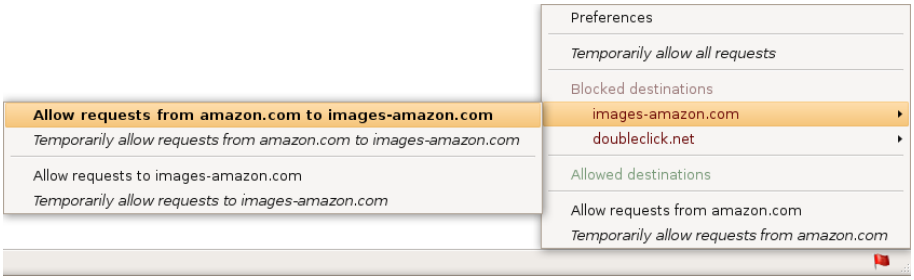

Figure 2: The RequestPolicy menu while visiting `amazon.com`. No requests have been whitelisted.

has been shown to be popular by the NoScript extension. In NoScript, each domain that provides scripts is listed in the menu users see when clicking on the NoScript icon. RequestPolicy used this interface concept as a starting point and improved upon it for greater clarity, granularity, and ease of use with cross-site request whitelisting.

The RequestPolicy menu groups destination domains by whether requests to those domains were blocked or allowed from the current site (Figure 1). Each destination domain entry has a submenu associated with it which allows adding or removing the destination from the whitelist (Figure 2). Any item added to the whitelist can be added temporarily if the user chooses. Temporarily whitelisted items are removed from the whitelist at the end of the browser session.

Importantly, users have much more granularity than just being able to whitelist destination domains. Users can whitelist by origin, destination, or origin-to-destination. For example, a user can allow all requests originating from `bbc.co.uk`, but from `amazon.com` only allow requests to `amazon-images.com`. Users can also allow requests to a specific destination from any origin, such as allowing all requests to `recaptcha.net`. RequestPolicy offers multiple domain classification policies (e.g. full domain name rather than registered domain name). The default is to consider a request's origin and destination to be the same site if their registered domain name is the same. Stricter settings of the classification policy add greater privacy and security at the expense of more interaction with the whitelist being required to make some websites function properly.

3.3 Development Timeline

Proof-of-concept work began on RequestPolicy in early 2008. That work entailed determining the feasibility of selectively blocking certain cross-site requests within a Mozilla browser extension.

Actual development of RequestPolicy began in September 2008. The first public alpha version of RequestPolicy was made available as an experimental extension on the official Mozilla addons site on November 23. Very heavy development continued through January 27. In that time, RequestPolicy had become a stable extension fulfilling its core privacy and security objectives with a clean user interface. On December 30, RequestPolicy was declared by the author as stable and ready for general public usage.

RequestPolicy has been released under the GNU General Public License version 3.

4 Results

Since the time of being declared ready for public usage, RequestPolicy has quickly gained attention among privacy and security concerned individuals including influential web application security experts [14]. As of May 6, 2009, RequestPolicy has been downloaded 5,225 times from the Mozilla addons site and has more than 1,000 active daily users. These numbers are expected to rapidly increase when the extension has its "experimental" status removed at the Mozilla addons site. This is expected to happen in late May 2009.

A paper focusing on the privacy aspects of RequestPolicy has recently been accepted for publication [19]. Further work on the security aspects of client-side cross-site request whitelists is in progress.

Improvements still remain to be made in RequestPolicy. One important area planned for improvement is the ease of whitelisting many items when using the stricter domain classification policies. Additional security work planned for RequestPolicy includes adding detection of requests that are destined for a local network that originated from pages outside of the local network. This is similar to some of the functionality provided by the LocalRodeo extension.

Criticism of RequestPolicy has largely focused on the added user interaction required for many websites to function correctly. This situation is no different than the impact that other popular security extensions such as

NoScript have on the browsing experience.

An additional area for potential future usability improvements is in providing more comprehensive optional initial whitelists or subscription-based whitelists. This would reduce the need for users to manually whitelist cross-site requests that do not impact privacy or security, such as some cross-site requests between sites run by the same organization.

5 Conclusion

Having recognized a deficiency in users' control over their cross-site request privacy and security, we set out to find and implement a solution. We designed and implemented RequestPolicy, a Mozilla browser extension that provides a default-deny policy for cross-site requests and a user-maintained whitelist with an intuitive interface for updating the whitelist.

RequestPolicy has filled a gap in the ability for users of modern web browsers to maintain high levels of privacy and security with respect to cross-site requests. It has been developed to be production-quality software with the intention of being used by a wide audience. The positive attention that RequestPolicy has received and the quick growth of its user base shows that the needs it has fulfilled were not only theoretical. Rather, they were practical concerns users desired more control over.

Acknowledgments

I thank Beichuan Zhang for his work and support on this research and as my honors thesis advisor. We thank John Hartman and Justin Cappos for their feedback on RequestPolicy. We are very grateful for the feedback and support of many RequestPolicy users.

References

[1] Adblock Plus. `http://adblockplus.org/`.

[2] BlockSite - Firefox Add-ons. `https://addons.mozilla.org/en-US/firefox/addon/3145`.

[3] Browser Port Scanning without JavaScript. http://jeremiahgrossman.blogspot.com/2006/11/browser-port-scanning-without.html.

[4] Cross-site request forgery. http://en.wikipedia.org/wiki/Cross-site_request_forgery.

[5] Extended Cookie Manager. http://www.defector.de/blog/category/firefox-extensions/extended-cookie-manager/.

[6] Firefox Add-ons. https://addons.mozilla.org/.

[7] Karma Blocker. http://trac.arantius.com/wiki/Extensions/KarmaBlocker.

[8] NoForge. http://www.seclab.tuwien.ac.at/projects/noforge/.

[9] NoScript - Firefox Add-ons. https://addons.mozilla.org/en-US/firefox/addon/722.

[10] Privoxy. http://www.privoxy.org/.

[11] Psiphon. http://psiphon.ca/.

[12] RefControl. http://www.stardrifter.org/refcontrol/.

[13] RequestPolicy - Firefox addon for privacy and security. http://www.requestpolicy.com/.

[14] RequestPolicy Firefox Extension. http://ha.ckers.org/blog/20090117/request-policy-firefox-extention/.

[15] Tor: anonymity online. http://www.torproject.org/.

[16] XPCOM. http://www.mozilla.org/projects/xpcom/.

[17] A. Barth, C. Jackson, and J.C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM New York, NY, USA, 2008.

[18] M. Johns and J. Winter. RequestRodeo: Client Side Protection Against Session Riding. In *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, pages 5–17.

[19] Justin Samuel and Beichuan Zhang. RequestPolicy: Increasing Web Browsing Privacy through Control of Cross-Site Requests. In *The 9th Privacy Enhancing Technologies Symposium (PETS 2009) [To appear].*

[20] J. Winter and M. Johns. LocalRodeo: Client-side protection against JavaScript Malware, 2007.

STATEMENT BY AUTHOR

SIGNED: _____