RequestPolicy: Increasing Web Browsing Privacy through Control of Cross-Site Requests

Justin Samuel and Beichuan Zhang

University of Arizona
Department of Computer Science
{jsamuel,bzhang}@cs.arizona.edu

Abstract. Many requests that a Web browser makes are not made to the primary site a user is visiting. It is common for websites to instruct browsers to make additional requests to third-party sites for content, advertisements, as well as for purely user-tracking purposes. Current techniques for maintaining user privacy with respect to cross-site requests are limited and inadequate. We propose a client-side whitelist for controlling third-party website requests. We implement this as RequestPolicy, an extension for Mozilla browsers. We look at the usability of RequestPolicy as well its impact on the Web browsing experience. Our extension maintains a high level of usability while safeguarding user privacy against well-known threats in addition to new threats we draw attention to.

1 Introduction

When a browser requests a page from a website, the response sent to the browser frequently includes instructions for the browser to make additional requests for content. These additional requests are often cross-site requests: requests whose hosts are different from the host of original website. Such cross-site requests often result in advertising companies and other websites gaining information about a user's browsing habits, including knowledge about specific pages the user has viewed and when those pages were viewed.

Any website that receives cross-site requests is in a position to collect and use this information. In some cases, this is the intended purpose of the cross-site request, such as with services that provide site traffic analysis. In other cases, the intention is not to provide a third-party site with user data but only to include off-site content in a webpage. In either situation, more information about a user's browsing habits is exposed than many users meant to reveal. This information and the potential for adversaries to link a user's separate website browsing sessions also puts users of anonymizing networks and proxies at risk of de-anonymization.

Other work has also identified specific risks to Tor [1] users due to the lack of a client-side exit policy. In the described attacks, the ability for a malicious party to reveal the identity of a client by means of timing attacks can be greatly increased [2]. This is done by a malicious exit node or destination server responding to the client with a webpage that causes the client to make outgoing requests to nonstandard ports.

More general timing attacks using cross-site requests and differences in completion time for cached and non-cached requests have been known for many years [3]. These methods allow a malicious site to identify URLs of other sites that a client has previously visited. The ability for this information to be discovered by unrelated websites is contrary to the privacy expectations of users.

In this work we identify where existing tools and methods fail to protect users from privacy loss due to cross-site requests. Based on this information, we identify a need for users to have full control over their browser's cross-site request behavior. Other work has recognized the great privacy benefits of cross-site request blocking, but considered the usability of this method to be impractical [4].

We design and implement RequestPolicy, an extension for Mozilla browsers that focuses on usability while providing complete control in the form of a user-maintained cross-site request whitelist [5]. We discuss the user interface challenges in implementing such a tool and the difficulty in maintaining correctness in the face of the need for minimal complexity and high ease-of-use. We also look at the impact our tool has on the functionality of websites and find ways to minimize disruption and avoid user frustration. Within the first few months of its release, RequestPolicy has been downloaded thousands of times.

While implementing RequestPolicy, we encountered new threats to privacy that have not been sufficiently considered elsewhere. DNS prefetching is a new technique used by browsers to decrease page load times by anticipatorily making DNS requests. We draw attention to how DNS prefetching can be abused for user tracking and other purposes. In accordance with our goal of giving users complete control over privacy-damaging requests made by their browser, we add protections against DNS prefetching abuse to RequestPolicy.

A cross-site request whitelist such as that implemented by RequestPolicy offers increased security in addition to privacy benefits. In this work we focus solely on the privacy aspects of RequestPolicy.

The remainder of this paper is organized as follows: In Section 2 we look at related work that provides methods users have available to limit information exposure due to cross-site requests. Looking at where current technologies fail to protect privacy with cross-site requests, in Section 3 we define requirements for a new system. Section 4 looks at the implementation of these requirements as a Mozilla browser extension and Section 5 discusses usability considerations. Future work is discussed in Section 6 and Section 7 concludes.

2 Related Work

Some technologies do exist that allow users to block certain cross-site requests or decrease the amount of information sent in cross-site requests. However, none of these have been developed with the privacy implications of cross-site requests as their primary focus.

Various proxies and browser extensions exist to suppress sending `Referer` headers in cross-site requests [6, 7]. The risk of privacy loss from cross-site requests, however, is not only due to `Referer` headers. Information about the user

is still contained in cookies, the user's IP address, and most fundamentally in the URL being requested. The requested URL can contain information such as the user's session ID from the originating site; this alone may put the user's privacy at risk.

Similarly, most modern browsers have options to block third-party cookies. Browser extensions also exist [8] that allow control over allowed cookies. As with `Referer` headers, blocking cookies does not eliminate the risk of privacy loss from cross-site requests.

A small number of tools do exist that block some cross-site requests. These include browser extensions that use predictive analysis for cross-site request blocking [9] and others that use blacklist-based advertisements blocking [10]. Predictive analysis has the potential to block many undesirable cross-site requests given the correct rules and history, but privacy cannot be guaranteed with such a system due to the occurrence of false negatives. Advertisement blocking, on the other hand, only targets privacy loss due to advertising companies. Further, advertisement blocking systems are generally blacklist-based and thus will have a delay time between false negatives and updates to the blacklist, which is often updated automatically. Other browser extensions exist that focus on general, manual request blacklisting [11], but these extensions have the same inadequacies as subscription-based advertisement blocking systems in addition to having user interfaces not intended for fine-grained cross-site request control.

The use of proxies that hide a user's true IP address from destination websites is common for users with an interest in maintaining privacy [1, 12]. However, as mentioned in Section 1, cross-site requests should be of concern to users of anonymizing proxies due to their potential for de-anonymization.

A solution to cache timing attacks has been proposed through cache partitioning [13]. However, the solution implemented in that work, the Firefox extension SafeCache, has been shown to be easily bypassed [14] and SafeCache is no longer maintained.

Krishnamurthy has added a great amount to the body of knowledge related to privacy and cross-site requests. In [4], various methods and tools for preserving privacy were looked at, concluding that all methods were inferior to blocking cross-site requests. The blocking of cross-site requests, however, was found to have very low usability. Other work by Krishnamurthy has studied metrics for quantifying privacy loss due to cross-site requests, the increase in cross-site requests for user tracking, and the impact of company acquisitions on the centralization of accumulated cross-site request data [15, 16].

3 Requirements

The existing tools that mitigate privacy loss due to cross-site request can be categorized as either reducing information sent with cross-site requests or blocking a portion of cross-site requests. Tools that reduce the amount of sent information fail to preserve privacy due to the fact that certain information they allow, such as the cross-site requests' URL, can cause loss of privacy. Tools that use

defined or predictive blacklists fail to preserve privacy due to only blocking some privacy-impacting cross-site requests. Additionally, these blacklisting tools have a weak ability to block new cross-site requests that do not trigger existing rules within those tools.

To create a tool that provides privacy-concerned users with the control they need over cross-site requests, neither information-decreasing nor blacklist-based approaches will suffice. These users need to have the ability to block cross-site requests unless they specifically choose to allow them. A whitelist solution rather than a blacklist solution is therefore required in order to ensure that all unwanted requests are blocked.

The use of a whitelist for cross-site requests, though, raises several usability concerns.

3.1 User understanding

Blacklist-based tools generally require little or no understanding of how the tool works. Users of such tools may not understand what the privacy risks are but only that they want to guard against privacy loss.

With a whitelist solution, it is even more important to recognize that many users desiring a high level of privacy do not have a full understanding of what cross-site requests are. However, other browser extensions have overcome the hurdle of user-understanding and have provided useful services to many who do not grasp the underlying technical issues. An example of such an extension is NoScript, a popular whitelist-based security extension for Mozilla browsers that has been downloaded more than 40 million times [17]. It is highly likely that a large number of NoScript's users do not fully understand the threats NoScript protects against. Despite this lack of understanding, these users are still able to benefit from a highly-secure whitelist solution.

It is worth recognizing, however, that NoScript's user base is not representative of the average Internet user. Such users are in the minority in that most have specifically chosen a browser other than their operating system's default browser. Additionally, their likelihood of higher-than-average technical understanding is evident in that they are aware of the existence of browser extensions and know how to install them.

3.2 User interface

With respect to user interface, blacklist solutions are generally non-invasive. Many users never have to interact with the interface. This level of automation will not be possible with a whitelist solution. Care will need to be taken to keep the user interface of a cross-site request whitelist tool intuitive.

User intuition is also a key to the usability of NoScript's interface. Users are alerted when the whitelist has restricted components of a website; users then must make whitelisting decisions based on their level of security knowledge, perceived need for a blocked component, and trust in the website the component

is from. Similarly, we will rely upon user judgment for our cross-site request whitelist. As discussed above, basing a standard of user intuition off of the users of a popular addon will likely not translate directly to the intuition of an average Internet user.

3.3 Website functionality

A cross-site request whitelist will undoubtedly impact the appearance as well as the functionality of some websites. Any tool we develop will have very limited ability to compensate for any such breakage. What a whitelist tool should do, however, is enable users to identify as easily as possible what the blocked content is that is causing the breakage. Based on that knowledge, users should be able to quickly whitelist the cross-site requests required for that desired content.

4 Implementation

We implemented RequestPolicy, a Mozilla browser extension that provides a cross-site request whitelist. Browser extensions provide an ideal way to implement changes to the browsing experience. Through a browser extension, software can maintain the user's expectations of appearance and behavior with respect to their existing browser. When a website does not behave as a user expects or requires security decisions to be made by the user, the browser commonly provides users additional information. An example of this is when a user visits a website whose SSL certificate is invalid. Modern browsers will present the user additional information directly in the content pane of the browser. Users do not expect to look to separate windows or applications when there is a problem with their browsing experience.

Browser extensions also provide an ideal way to implement our changes due to the large amount of available information about the user's actions. This same information would not be accessible through, for example, a proxy that can only see and modify requests and responses. This is especially important with cross-site requests. Many cross-site requests are fully intentional by the user. For example, users will often follow links from one site to another. These are not the kinds of cross-site requests we want to block. Rather, we want to be able to selectively allow user-intended cross-site requests while subjecting others to the user's whitelist.

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 [18].

4.1 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.

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

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

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 the extension provides. The Mozilla XPCOM [19] 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. These comparisons will be discussed in more detail in Section 5.2.

Though it may appear that knowing the origin URL and destination URL for any given request would be enough information to make an accurate decision for that request, our decision algorithm needed more information to avoid false positives. For example, when a user clicks a cross-site link or submits a cross-site form, we do not want to block those requests. In order to not subject link clicks and form submissions to the whitelist rules, a combination of methods were used. For simple link clicks and form submissions, event handlers provided by the browser were sufficient to allow the decision algorithm to allow these requests. More difficult but highly important was to also recognize other actions such as choosing to open a link in a new tab or window through the context menu (the menu the displays when a link is clicked with the alternate mouse button). To detect these cases, the browser functions that are called to open links from the context menu were replaced with modified versions of those functions.

Not all cross-site requests could be handled through the nsIContentPolicy interface. Header-based redirects, for example, were not subjected to this interface and neither were prefetched webpages. Header-based redirects could be caught and acted on through a separate observer interface within Mozilla. Prefetched webpages, unfortunately, could not be. Webpage prefetching occurs when a webpage includes a tag that hints to the browser what it believes is a likely next page the user will visit. Webpage prefetching allows some content to be anticipatorily requested and cached. The only way to disable webpage prefetching in our extension was to globally disable this functionality through the browser preferences system. This means that all prefetched webpages are blocked rather than just those that are blocked according to whitelist rules.

4.2 DNS Prefetching

DNS prefetching is a new feature being added in Firefox 3.1. DNS prefetching is where the browser looks at all links on every page and, during browser idle time, performs DNS resolution on domain names in those links. DNS prefetching is an idea first implemented in the Google Chrome browser [20].

Before DNS prefetching was added to Firefox, a security review was performed by the Firefox developers [21]. The review, unfortunately, only considered privacy in terms of how DNS prefetching related to the Private Browsing mode being added to Firefox [22].

Although no cases of privacy abuse through DNS prefetching are currently known to exist, there is clearly the possibility for abuse. User privacy could be violated by having the authoritative DNS servers for a domain record requests for specially-crafted domain names that are used in links. For example, DNS prefetching could be used for user tracking by embedding links in a page that have subdomain parts which include information about the user session and visited page. Such a domain might look like `page-123.session-456.example.com`. The same tactic could be used to obtain email open-rate data for webmail users. Email open-rate information is useful for email marketers, spammers, and phishers. DNS prefetching could even be used by individuals who want to bypass a recipient's potential refusal to provide read receipts from their webmail client.

No granular method for control over DNS prefetching exists in Firefox's current implementation. In order to protect users against privacy loss due to DNS prefetching, we had to use the same approach as with webpage prefetching. Namely, DNS prefetching is disabled by RequestPolicy through Firefox's preferences system.

4.3 Problematic Requests

Some cross-site requests are difficult to manage. When a user clicks a cross-site link, we want to allow that request. However, non-standard links that are actually triggers for JavaScript which redirect the browser to a different site cannot be detected as user-intended actions. As a result, these types of links are blocked

when clicked by a user. Fortunately, while links that trigger JavaScript are common, we have found that the use of such links to trigger cross-site redirection is extremely uncommon. If such a case were to be encountered by a user, the user would be able to follow the link by whitelisting the cross-site request so that it would be allowed.

More difficult are cross-site requests performed by third-party browser plugins (such as Flash and Java) and other browser extensions which can make requests that bypass RequestPolicy's whitelist. In general, there is no way to prevent a separate application installed by a user from bypassing the browser's privacy or security mechanisms. We did find, though, that some plugin-initiated requests do go through the browser's request interface. As a result, these requests are often subjected to the cross-site request whitelist.

4.4 Identifiability of RequestPolicy

For users seeking anonymity on the Web, browser extensions should not be identifiable to websites. Any browser extension that can be identified by analyzing web requests adds to the information an adversary can use to potentially de-anonymize a user. However, it does not seem possible to make RequestPolicy unidentifiable. At best, individual websites may not be able to determine which browser extension is blocking certain requests. Looking at a user's request pattern across multiple websites would likely remove any doubt as to which extension was in use.

5 Usability

The usability of a system is highly dependent on the needs of those using the system. RequestPolicy, fundamentally, does not address a need of all users browsing the Web. Specifically, only users sufficiently concerned about privacy to be willing to make changes to their browsing experience would consider a tool such as RequestPolicy for privacy preservation. Thus, RequestPolicy must meet a level of ease of use that is acceptable to people willing to incur some impact on their browsing. The goal, of course, is to minimize that impact and make RequestPolicy usable to those with the least amount of privacy concern as well as the least patience and technical savvy.

The most important user interface aspects in developing RequestPolicy were how users were to be notified of blocked content and how they would then interact with the extension to control their whitelist. In addition, the aggressiveness of the default cross-site request classification policy needed to be balanced with the privacy concerns of the largest segment of the expected user base.

5.1 User Interface

In order to notify users of blocked content on the current page, some form of notification needed to be added to the browser window. A RequestPolicy icon

was added to the browser's status bar, the bar that runs along the bottom of the browser window. When content is blocked, this icon changes to indicate that there is blocked content. When RequestPolicy was first made available as a beta for public testing, immediate feedback from multiple users was received requesting a similar notification icon be made available in the toolbars at the top of the browser window. An optional toolbar button was soon added.

Some types of blocked content are difficult to indicate directly to the user. For example, blocked CSS stylesheets cannot be easily represented to many users because they don't occupy a specific area of a webpage and, more importantly, many users do not understand what stylesheets are. Images, on the other hand, are understood by users and do occupy a specific region of a 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.

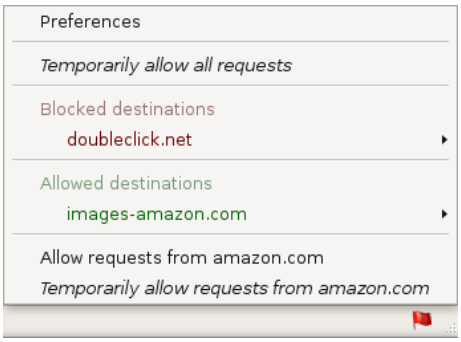

Fig. 1. The RequestPolicy menu while visiting `amazon.com`. The destination `amazon-images.com` has been white whitelisted for requests originating from `amazon.com`.

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 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`.

A primary goal with RequestPolicy was to maintain simplicity in the menu while still providing the information a user needs and quick access to the options they are likely to use. A case where this simplicity was jeopardized was when dealing with situations where there are multiple origins within a single Web page a user views. Multiple origins in a page can happen when that page includes frames or iframes whose origin is different from the main page and those frames make cross-site requests of their own. In RequestPolicy, we refer to such frames as *other origins*. It is not very common for a page to have other origins within it, but situations can occur where a user needs to be able to whitelist requests belonging to one of these other origins. Ultimately, we handled this case in RequestPolicy by adding a single menu item for "other origins within this page" when there are other origins. This provides access to a somewhat complex series of submenus that provides the same level of control over these other origins that a user has over the primary origin of a page. There is likely still room for improvement with this part of the user interface.

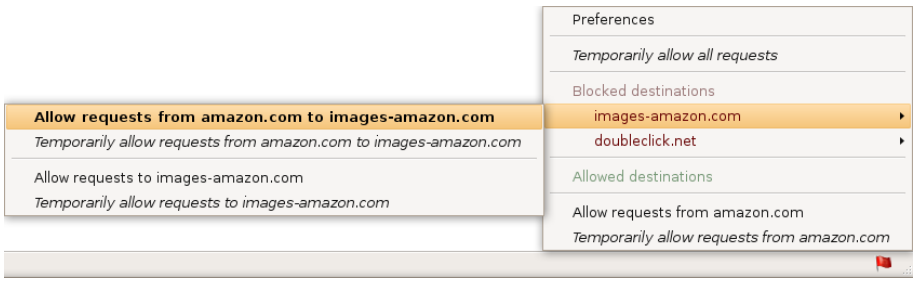

Fig. 2. The RequestPolicy menu while visiting `amazon.com`. No requests have been whitelisted.

5.2 Usability vs. Correctness

Ideally, RequestPolicy should default to the highest level of privacy possible with respect to cross-site request blocking. Link clicks and form submissions are notable exceptions; allowing cross-site requests in these cases is considered to be correct behavior in terms of user intent. However, there is one important case where we decided that the usability impact of the most strict privacy settings was not a good default. This case is the way in which RequestPolicy classifies requests as same-site or cross-site.

It is very common for webpages at a given registered domain to include content from different subdomains. For example, a page may be accessed at `example.com` but that page includes images from `www.example.com`. There are also situations where many different subdomains are used to serve images and other static content for a page. This is especially common for the purpose of speeding up page load time. Browsers generally limit the number of simultaneous requests to a single host. This limit can be worked around by providing content from different hosts. The use of many subdomains within the same page is a common method of doing this. Notable sites that serve images this way include `amazon.com` and `yahoo.com`.

This situation results in a marked increase in the number of distinct blocked destinations when classifying URLs by full domain name as opposed to the registered domain name. Therefore, in order to decrease its impact on many websites, RequestPolicy defaults to using the strictness level of only the registered domain name when determining whether a request is cross-site. If a request's origin and destination have the same registered domain, the request is considered to be a same-site request and is allowed. This is also regardless of protocol and port.

There is a privacy risk with this default setting, though. Allowing all requests within a registered domain allows sites to serve ads, for example, through subdomains pointing to advertising company servers (e.g. a website makes `ads.their-domain.com` a CNAME for an advertising network). Though this technique is not commonly used [4], the potential for its increased use does exist. Having a default setting that ignores the destination port also makes RequestPolicy ineffective by default against the attacks on Tor users described in Section 1.

RequestPolicy therefore allows users to choose a stricter site classification method. Rather than using the default of the registered domain name, requests can optionally be classified as cross-site requests by either the full domain name or the combination of the protocol, domain, and port (that is, the criteria used for the "same origin policy"). This gives increased usability for the majority of users while allowing better privacy for those with greater privacy needs.

5.3 Impact on Websites

The impact that blocking cross-site requests would have on the appearance and functionality of websites was a major concern for the overall usability of RequestPolicy. With RequestPolicy, we found three major ways to categorized sites according to the impact of blocked cross-site requests: those that are not noticeably impacted, those that remain functional but have moderately or drastically altered appearance, and those that do not function as expected when all cross-site requests are blocked.

Not every blocked destination from a given site generally needs to be whitelisted in order to correct the affected appearance or functionality of the site. In fact, when using RequestPolicy's default of classifying sites by registered domain name, it is often the case that at most one destination for a site needs to be whitelisted. Using the stricter classification policies, both the number of

blocked destinations as well as the number of those that need to be whitelisted increases (Table 2).

Additionally, with the stricter classification policies, it can be much more difficult to know which destinations need to be allowed. This is the case even when only one destination may need to be allowed among the many blocked destinations. For example, with `www.yahoo.com`, the following domains are blocked when using the strictness level of the full domain (the specific domains may be dependent on the requesting IP address as well as other factors):

- `l.yimg.com`
- `ads.yimg.com`
- `us.i1.yimg.com`
- `us.bc.yahoo.com`

The only one of those which needs to be allowed in order to have the site look and behave as expected is `l.yimg.com`. This isn't obvious, though. The only item which would intuitively be assumed to not be required is `ads.yimg.com`.

Table 2. The impact of RequestPolicy on popular websites. Impact is classified as None, Appearance (missing images, formatting), or Functionality (does not work as intended). For each site, the number of destinations needing to be whitelisted out of the total number of blocked destinations is given. This varies by classification policy: registered domain, full domain, and "same origin" (protocol, host, and port).

	Impact			Required to Whitelist		
Site	None	Appear.	Func.	Reg. domain	Full domain	Same origin
www.google.com	•			0/0	0/1	0/1
www.yahoo.com		•		1/2	1/4	1/4
www.youtube.com		•		1/2	5/6	5/6
www.youtube.com (videos)			•	2/3	2/7	2/7
www.live.com	•			0/1	0/3	0/3
www.facebook.com			•	1/1	2/2	2/2
www.msn.com		•		1/3	2/9	2/9
en.wikipedia.org		•		1/1	1/2	1/2
www.blogger.com			•	1/2	1/2	1/2
www.myspace.com			•	1/3	2/5	2/5
www.amazon.com		•		1/2	3/4	3/4

After releasing RequestPolicy to the public, general feedback indicated that the period requiring the most interaction with the whitelist menu was the initial one to two weeks of using the extension. After this period of time, most users have whitelisted the majority of cross-site requests required for proper functionality of their frequently-visited websites. In order to ease this transition into using RequestPolicy, we added a dialog window that displays after initial

installation which gives users the opportunity to add common cross-site but same-organization items to their whitelist. Examples of such optional default whitelist items include requests from `wikipedia.org` to `wikimedia.org` and from `yahoo.com` to `yimg.com`. The default whitelist items are available in region-specific groups and, in total, currently offer around 100 origin-to-destination pairs as well as a single destination from any origin, `recaptcha.net`.

5.4 Policy Creep

There are two major ways a user may experience *policy creep*, where the user whitelists broader requests than they may have wanted in order to decrease the number of times they need to whitelist specific origins-to-destinations. The first of these situations is where a user visits a site and, upon finding the site does not function as desired, the user decides to whitelist all requests from that origin rather than the individual origins-to-destinations needed to make the site work properly. The user may do this, for example, because there are many blocked destinations that may be the cause of the breakage and they do not want to determine the subset of the blocked requests that need to be allowed.

The other cause of policy creep is where users may find they often need to allow requests to a certain destination in order to make sites they visit work properly. As a result, a user may decide to whitelist requests from any origin to that destination in order to avoid having to whitelist the destination from many different origins. Notable cases of many websites being dependent on cross-site requests to a small number of destinations are websites using certain Content Delivery Networks (CDNs) and hosted services. For example, this includes sites that make use of JavaScript libraries hosted by Google [23] and Yahoo [24]. In the case of Google's hosted JavaScript libraries, a RequestPolicy user with the default domain strictness settings could potentially allow requests from any site to `google.com`.

6 Future Work

One clear area for usability improvements within RequestPolicy involves the large number of cross-site requests that exist between sites run by the same organization. The privacy concerns of these types of cross-site requests are generally low. Blocking these types of cross-site requests is responsible for a significant amount of site breakage. The addition of optional default whitelist items lessened RequestPolicy's impact on website functionality for many popular websites. However, this form of one-time import provides no way for users to stay updated with a list of same-organization cross-site requests in their whitelist. One possible solution for this is to use a subscription model for additional whitelisting. This type of subscription model is used by extensions such as Adblock Plus [10]. However, this may add unnecessary complexity to the interface, especially in cases where users want to override their subscription's whitelist in specific cases.

We intend to wait for more user feedback before deciding whether to proceed with a subscription model.

We briefly discussed the privacy risk of cross-site requests for users of anonymizing networks. However, more work needs to be done studying the impact of cross-site requests on anonymity in anonymizing networks such as Tor.

RequestPolicy's most notable privacy deficit is the usability decision to default to using only the registered domain name for determining whether a request is cross-site. Further work needs to be done to determine if the simplicity of the user interface can be maintained while defaulting to a stricter classification policy.

7 Conclusion

In this work, we explored the reasons why existing tools fail to provide a high level of privacy with respect to cross-site requests. We then used that knowledge to define the requirements for a system that does provide users full control over data leakage due to cross-site requests. Fundamentally, users need to be able to block cross-site requests by default and whitelist only those they want to allow.

Designing and implementing such a whitelist-based system brought with it serious usability concerns. We found, however, that with proper attention to user interface issues such as making blocked elements of a webpage easy to identify and whitelist, our browser extension was able to remain easy to use. Our extension, RequestPolicy, has been rapidly adopted since its release and has been downloaded thousands of times.

In the process of implementing RequestPolicy, we also discovered a lack of attention to the privacy ramifications of DNS prefetching. We added privacy protections against abuse of DNS prefetching to RequestPolicy and raised awareness of this issue.

Acknowledgments We thank Justin Cappos, John Hartman, and Robert Hansen for their feedback and suggestions for RequestPolicy. We also thank the anonymous reviewers for their suggestions and comments. We are grateful to the Mozilla developers for providing the powerful extension architecture they have and to the many RequestPolicy users who have given us feedback.

References

1. Tor: anonymity online. `http://www.torproject.org/`
2. Abbott, T., Lai, K., Lieberman, M., Price, E.: Browser-Based Attacks on Tor. In: Proceedings of The 7th International Symposium on Privacy Enhancing Technologies (PET). Volume 4776 of LNCS., Ottawa, Canada, Springer-Verlag (June 2007) 184

3. Felten, E., Schneider, M.: Timing attacks on web privacy. In: Proceedings of the 7th ACM conference on Computer and communications security, ACM New York, NY, USA (2000) 25–32
4. Krishnamurthy, B., Malandrino, D., Wills, C.: Measuring privacy loss and the impact of privacy protection in web browsing. In: Proceedings of the 3rd symposium on Usable privacy and security, ACM Press New York, NY, USA (2007) 52–63
5. RequestPolicy - Firefox addon for privacy and security. http://www.requestpolicy.com/
6. Privoxy. http://www.privoxy.org/
7. RefControl. http://www.stardrifter.org/refcontrol/
8. Extended Cookie Manager. http://www.defector.de/blog/category/firefox-extensions/extended-cookie-manager/
9. Karma Blocker. http://trac.arantius.com/wiki/Extensions/KarmaBlocker
10. Adblock Plus. http://adblockplus.org/
11. BlockSite - Firefox Add-ons. https://addons.mozilla.org/en-US/firefox/addon/3145
12. Psiphon. http://psiphon.ca/
13. Jackson, C., Bortz, A., Boneh, D., Mitchell, J.: Protecting browser state from web privacy attacks. In: Proceedings of the 15th international conference on World Wide Web, ACM New York, NY, USA (2006) 737–744
14. Web Security Research - Alex's Corner: Attacking the Safe-Cache Firefox Extension. http://kuza55.blogspot.com/2007/02/attacking-safecache-firefox-extension.html
15. Krishnamurthy, B., Wills, C.: Generating a privacy footprint on the internet. In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, ACM New York, NY, USA (2006) 65–70
16. Krishnamurthy, B., Wills, C.E.: Privacy Diffusion on the Web: A Longitudinal Perspective. In: Proceedings of the World Wide Web Conference. (2009)
17. NoScript - Firefox Add-ons. https://addons.mozilla.org/en-US/firefox/addon/722
18. Firefox Add-ons. https://addons.mozilla.org/
19. XPCOM. http://www.mozilla.org/projects/xpcom/
20. Google Chrome. http://www.google.com/chrome
21. Firefox 3.1 / DNS Prefetching Security Review. https://wiki.mozilla.org/Firefox3.1/DNS_Prefetching_Security_Review
22. Private Browsing. https://wiki.mozilla.org/PrivateBrowsing
23. Google AJAX Libraries API - Google Code. http://code.google.com/apis/ajaxlibs/
24. The Yahoo! User Interface Library (YUI). http://developer.yahoo.com/yui/