Arbitrary Code Injection in Adblock Plus

Eric Li
Information Networking Institute
Carnegie Mellon University
Pittsburgh, U.S.A
xiaoyili@andrew.cmu.edu

Teresa Alberto
Information Networking Institute
Carnegie Mellon University
Pittsburgh, U.S.A
talberto@andrew.cmu.edu

Chengcheng Ding
Information Networking Institute
Carnegie Mellon University
Pittsburgh, U.S.A
chengchd@andrew.cmu.edu

***Abstract*—TODO**

***Index Terms*—browser security, browser extension, ad blocking, XSS**

I. Introduction

Since the very beginning of their introduction, browser extensions expose additional attack surfaces for browsers. Their popularity among users only makes the matter worse: a vulnerability in a popular extension can affect millions of users. One such popular extension is Adblock Plus, which has tens of millions of users on Chrome and Firefox combined [1], [2]. In this article, we will investigate a vulnerability introduced in a previous version of Adblock Plus that enabled the execution of arbitrary code on users' devices.

Adblock Plus has a very well-intended feature that allows users to apply custom filter rules. To do so, users can supply a text file that provides (a list of) rule(s) written in the syntax specified by Adblock Plus. Similarly, users can also apply custom filter rules crafted by others to their Adblock Plus extensions. This introduced another party in the threat model.

A standard custom filter rule mentioned above consists of 2 items:

1) A pattern of URIs. All requests made to a URI that fits such a pattern will be filtered
2) The action to the request that will be filtered. The most obvious and straightforward example would be blocking that request.

In Adblock Plus version 3.2, released on July 17, 2018, a new option for filter action is introduced: rewrite. It allowed users to rewrite requests to a new destination. In that release, it is restricted that the new destination must be of the same origin as the original request. Unfortunately, this introduced new vulnerabilities that allowed arbitrary code execution on user machines that somehow applied malicious filter rules to their extension [3]–[6].

We contribute the following:

1) An explanation of the technology details behind the attack.
2) A detailed guide to re-implementing the attack, that specifies corresponding versions of the products related, and a demo website that showcases the attack.
3) What we can learn from this vulnerability: its impacts, and what vendors can do to mitigate against it.

II. Background and Goals

A. Massive Adaptation of Ad-Blocking Extensions

Extensions play a critical role in the capabilities of modern browsers, enabling customized web experiences. Among them is a popular category of extensions that blocks advertisements on web pages. One of the most popular in that category is Adblock Plus. According to statistics from the Chrome extensions web store and Mozilla Firefox add-on store, tens of millions of users have Adblock Plus installed [1], [2]. According to a study by Pujol et al., 22% of the most active users analyzed browse the Web with Adblock Plus. In fact, Ad-blocking extensions are so popular that advertisement vendors have begun to take active measures against them [7]. Therefore, vulnerabilities from the Adblock Plus extension can impact a considerable amount of internet users. That is why we have chosen to investigate a vulnerability in a previous version of Adblock Plus.

B. Custom Filters in Adblock Plus

The default advertisement-blocking behaviors of Adblock Plus are defined by a filter list included in the extension. In addition to that, Adblock Plus enables custom filters created by users. These filters allow users to block ads and other unwanted content that may not be covered by the default filter lists. Users can create their own filters by specifying URLs or patterns of URLs, and the actions they want to take against them, such as blocking or allowing. Custom filter lists can also be shared with other Adblock Plus users. From a security standpoint, custom filters involve additional third parties in the flow of ad-blocking, potentially creating additional attack surfaces. The filters are not written in a syntax easily understandable for most average users. Hence, attackers can embed malicious filter actions and present them to users as a filter for additional features, if the filter rules' functionalities allow them.

C. The Rewrite Rule: What and Why

In version 3.2 of Adblock Plus, released on July 17, 2018, a new filter option called "rewrite" was introduced. It allows users to rewrite outgoing requests to certain URL(s). In this version, the destination of requests can be rewritten to resources within the same origin. For example, a request to "foo.com/ad.js" can be rewritten to "foo.com/not-ad.js".

There can be multiple correct usages of the rewrite filter. The first (and primary) one is when simply blocking a request to a certain advertisement-related resource will lead to an error. That's when rewriting the request is a reasonable way to block advertisements with minimal impact on user experience.

Another potential way of using the rewrite filter is eliminating certain HTTP request parameters related to privacy. Advertisers have long used HTTP request parameters for cross-site tracking. Parameters in advertisement links help vendors figure out where the click comes from. Therefore, the rewrite filter can be utilized in this scenario to eliminate request parameters related to tracking.

D. Goals of This Project

There are mainly 2 goals for this project, centered around the vulnerability introduced by the first implementation of the rewrite rule, and its implications.

The first goal is to prove that the vulnerability implies a previously understated attack surface. The vulnerability's chain of exploits utilizes vulnerabilities, or even features, that would be considered low-impact or no-impact on their own. For example, a crucial component of this vulnerability, open redirect, would be considered a feature instead of a vulnerability in many websites. As a matter of fact, Google doesn't consider open redirects to be a security vulnerability to be accepted as part of its Vulnerability Disclosure Program, even while acknowledging it can be a tool for phishers. [8]

The second goal of this project is to recreate a proof of concept of the attack scenario. The proof of concept will specify the versions of all of the related software, and provide a skeleton website to demonstrate the capabilities of the attack. The details of the proof of concept can be found in Section V.

III. Threat model

The threat model implies a remote attacker with the ability to abuse the rewrite filter of Adblock Plus installed in the victim's browser. In this scenario, the attacker may be an insider employee of Adblock Plus or an open-source malicious contributor to its codebase. In both cases, the attacker needs to include a malicious rewrite rule in the preinstalled filter lists that are enabled by default when installing the extension. An alternative scenario would be a malicious actor that convinces the victim to install and enable a malicious filter that includes a malicious rewrite rule in it. Eyeo, the company behind Adblock Plus, makes it clear that the filters are maintained by third parties (even the defaults ones that come preinstalled when installing the extension) and that they neither have control over their contents nor are responsible for any damage provoked to the user in relation to them [9].

The victim needs to have installed a vulnerable version of Adblock Plus in the browser, with the malicious filter enabled. For the attack to succeed the victim needs to visit the page that will be rewritten by the malicious rewrite filter.

IV. Attack Outline

TODO

- Victim website: Open redirect combined with fetching and loading JS code provokes worse problems.
- Victim user: install malicious filter list (may be updated automatically), visit victim website.
- Attacker: control the malicious filter list.
- Potential impact.

V. Implementation

In this section, we explain how we reproduce the attack. We provide a fully functioning environment for demonstrating this attack. We write two websites using Flask as the attacker and the victim website. We also add Content Security Policy (CSP) logics to demonstrate the recommended mitigation in [3].

TODO: we make our project open source?

A. Recreating the Environment

As [3] mentions, the rewrite filter vulnerability is only exploitable on Adblock Plus version between 3.2 and 3.5.1. Thus, the first step for us is to find old version of of this extension. Unfortunately, the official Chrome Web Store and Add-ons for Firefox website no longer provide these versions. So we find old versions from unofficial sources.

We are able to find old version of the extension for Chrome on Crx4Chrome [10], a website that provides archived extensions for Chrome. We are unable to find old version of the extension for Firefox easily. However, we argue that since the extension is designed to be browser-independent, the same attack should work on Firefox, Safari, and Edge.

Adblock Plus 3.2 cannot be directly installed on the latest version of Chromium, because the extension is packed in a deprecated format and installing the extension triggers the `CRX_HEADER_INVALID` error. However, it is possible to extract the packed extension as a ZIP archive file and install the extension to Chromium in developer mode [11].

The problem with Adblock Plus 3.2 is that it uses manifest V2. However, manifest V2 is deprecated and planned to be removed, so Chromium browsers in the future will not be able to install Adblock Plus 3.2 [12]. Thus, we demonstrate that it is possible to install Adblock Plus 3.2 on an old version of Chromium.

[13] provides guidance to download old versions of the Chromium browser. We choose Chromium 69.0.3497, a version released close to the release date of Adblock Plus 3.2. Following the guidance, we query the position lookup and find that the branch base position of this version is 576753. Finally, we can download the Chromium browser's snapshot at this version. Adblock Plus 3.2 can be installed directly in Chromium 69, by dragging the CRX package to the browser's extension management page.

In this research, we only test reproducing the attack on Linux. However, since Chromium is a cross-platform browser, we expect the same attack to be reproducible on Windows and macOS.

B. Attacker Website

We implement the attacker website using Flask as the backend. The host name and port of the attacker website are `attacker.local:8080`. We use plain HTML and JavaScript as the frontend. The attacker website has two features.

First, the attacker website hosts an filter rules file that will be loaded to the victim user's browser. To make demonstration easy, we design an interfaces using HTML and JavaScript to modify the filter rules from a browser. The attacker can choose from any subset of the pre-defined 8 rules:

1) Block advertisement image.
2) Block static JavaScript.
3) Rewrite advertisement image with another image.
4) Rewrite XMLHttpRequest (XHR) that sends private information with URL that does not send private information.
5) Rewrite static Javacsript.
6) Rewrite dynamic JavaScript to static file.
7) Rewrite dynamic JavaScript to redirect URL.
8) Rewrite dynamic JavaScript to user uploaded file.

Second, the attacker website needs to host the malicious JavaScript payload to be executed on the victim browser. To simplify our design, we host the JavaScript payload statically. The attacker website sets Access-Control-Allow-Origin to `"*"` to allow the malicious JavaScript from being read by the victim website on the victim browser.

C. Victim Website

Similar to the attacker website, we implement the attacker website using Flask as the backend. The host name and port of the victim website are `victim.local:8080`. We use plain HTML and JavaScript as the frontend. The victim website has multiple sections, as discussed below in detail.

1) Welcome Image: The victim website loads a welcome image using HTML's `<img>` tag. This provides a image for rewriting the advertisement image later.

2) Advertisement Image: The victim website loads an advertisement image using HTML's `<img>` tag. The website also uses JavaScript to detect whether loading the advertisement image is successful. The image is loaded successfully if the `load` event listener is called. The image fails to load if the `error` event listener is called.

3) Targeted Advertisement: The victim website loads an advertisement using XMLHttpRequest (XHR). The request sends information that undermines the user's privacy, such as the user's location. For demonstration purpose we hard code this information. The server returns an advertisement as a string. The frontend displays the string on HTML.

The purpose of this section is to show that rewrite rules can modify HTTP requests to prevent leakage of users' privacy.

Note that we use XMLHttpRequest (XHR) in the entire demonstration. However, other techniques like fetch also work.

4) Static JavaScript: The victim website loads a JavaScript script statically using the `<script>` tag in HTML. The loaded script changes HTML to indicate that the script is loaded correctly. This section demonstrates that Adblock Plus is able to block a static script from loading. However, rewrite rules cannot modify the request to load another script instead.

5) Dynamic JavaScript: The victim website loads a JavaScript script dynamically when the user clicks a button. The browser first requests the content of the script using XHR. The browser then evaluates the downloaded script using `eval()`. This is similar to how Google Maps dynamically load JavaScripts while the user is browsing the map. Similar to Google Maps, the victim website must enable `script-src 'unsafe-eval'` in CSP to make dynamically loading JavaScript working.

6) Open Redirect: One weakness of the victim website is open redirect, where any user can define HTTP redirects in the website [14]. In the victim website, the attacker can control the URL `/redirect` to return an HTTP 302 to any URL. Example URL include a JavaScript on the victim website and a JavaScript on the attacker website.

The I'm Feeling Lucky button provided by Google Search has this weakness in 2019. When the user clicks the button, the website sends a GET request to Google's server. The request contains the user's query and `btnI` parameter to indicate the user clicks the I'm Feeling Lucky button. Google's server returns an HTTP 302 response to the first Google Search result. An attacker can use this feature as an open redirect. The attacker simply needs to construct a Google Search query that places the redirect target as the first Google Search result. This weakness is fixed by Google later, as discussed in Section VI-B.

7) User Uploaded File: Another weakness of the victim website is user uploaded file with JavaScript content [15]. The victim website does not perform any sanitization to the file uploaded, and any user of the victim website can view the uploaded file. Thus, if the attacker can trick the victim browser to load the uploaded file, the attacker can achieve arbitrary code execution.

8) CSP: The victim website allows dynamically configuring the website's CSP. This allows easy understanding the effectiveness of different defense strategies. In an attempt to defend, the victim user can try to remove `script-src *` from CSP and/or to add `connect-src 'self'` to the CSP.

D. Attacker User

The attacker user runs any modern browser that can run JavaScript. The attacker user opens the attacker website and interacts with it. The attacker user also interacts with the victim website by changing the open redirect target and uploading the JavaScript file.

E. Victim User

We setup the victim user in a Linux virtual machine. The virtual machine runs Chromium 69. We install AdBlock Plus 3.2 and add the filter rules file of the attacker website to

AdBlock Plus. The victim user then opens and interacts with victim website. When the attacker updates the filter rules file, the victim needs to click the update button in AdBlock Plus's configuration page.

F. Experiment Results

Using the setup described above, we perform multiple experiments to demonstrate the capabilities of AdBlock Plus and the arbitrary code execution attack.

1) Blocking Image: AdBlock Plus can block the advertisement image in the victim website. However, the victim website can see that the advertisement image fails to be loaded using the JavaScript `error` event listener.

2) Blocking Static JavaScript: AdBlock Plus can block the static JavaScript in the victim website.

3) Rewriting Image: AdBlock Plus can rewrite the advertisement image in the victim website with another image on the victim website. Since the `<img>` tag successfully loads an image, the victim website cannot detect the rewrite using the JavaScript `error` event listener. This demonstrates that the intended use of the rewrite feature [6].

4) Rewriting Request Paramters: AdBlock Plus can rewrite an XHR request to a different URL. In our example, we remove the request URL's query string, which contains private information of the victim user. This is another intended use of the rewrite feature [4].

5) Rewriting Static JavaScript: AdBlock Plus is not able to rewrite static JavaScript in the victim website. During the design of the rewrite feature, rewriting static JavaScript is disallowed due to security considerations [4].

6) Rewriting Dynamic JavaScript: AdBlock Plus is able to rewrite dynamic JavaScript loaded from XHR request. So when the victim website loads a JavaScript from the its own origin dynamically, AdBlock Plus can change to let it load a different JavaScript from the same origin.

7) Attack Using Open Redirect: A victim website that loads dynamic JavaScript and has the open redirect weakness can be attacked. The attacker first adds AdBlock Plus rule to rewrite the dynamic JavaScript's URL to the open redirect URL. The attacker then controls the open redirect to return HTTP 302 to an attacker-controlled website. The attacker controlled website returns the malicious JavaScript code.

Removing `script-src *` from CSP does not defend against this attack. The dynamic JavaScript is downloaded using XHR, which is not controlled by `script-src`.

Adding `connect-src 'self'` to CSP defends against this attack. This rule prevents XHR from being redirected to an origin other than the victim website itself. However, this breaks functionality if the victim website needs to send XHR requests to other websites.

8) Attack Using User Uploaded File: A victim website that loads dynamic JavaScript and has the user uploaded file with JavaScript content weakness can be attacked. The attacker first uploads the malicious JavaScript file to the victim website. The attacker then adds AdBlock Plus rule to rewrite the dynamic JavaScript's URL to the uploaded file.

Neither removing `script-src *` from the CSP nor adding `connect-src 'self'` to CSP to CSP defends against this attack. The original dynamic JavaScript and the attacker uploaded file are both in the victim website's origin. Thus, blocking loading scripts from the victim website's origin will break the victim website's functionality.

G. Reproducibility Analysis

To investigate the software versions this attack can take place, we try our experiment in different versions of Chromium and AdBlock Plus.

We see that this attack is successful regardless of the Chromium version, as long as AdBlock Plus can be installed on Chromium. This attack works on both Chromium 69 (released in 2018) and Chromium 112 (released in 2023).

We verify the reproducible AdBlock Plus version range mentioned in [3]. The attack does not work on AdBlock Plus 3.1 (released in May 2018), which does not introduce the rewrite feature yet. This attack works on AdBlock Plus 3.2 (released in July 2018) and 3.5.1 (released in April 2019). We assume this attack works on the versions in between. This attack does not work on AdBlock Plus 3.5.2 (released in April 2019), when the rewrite feature is redesigned to prevent the attack [10].

We also try reproducing this attack on other vulnerable advertisement blockers mentioned in [3], including AdBlock and uBlock. We are not able to reproduce on uBlock. For unknown reason uBlock does not recognize any custom filter rules we write.

We are able to reproduce this attack on AdBlock, and the reproducible version range agrees with [3]. The attack does not work on AdBlock 3.31.2 (released in June 2018), which does not introduce the rewrite feature yet. The attack works on AdBlock 3.32.0 (released in July 2018) and 3.44.0 (released in April 2019). We assume this attack works on the versions in between. This vulnerability is fixed in AdBlock 3.46.0 (releaesd in April 2019) [16].

VI. Discussion and Limitations

A. Warnings During Design Phase

TODO

- Public warnings previous the deploy of the $rewrite feature.

B. Mitigation by Google

When AdBlock Plus 3.2 is published in 2018, Google is vulnerable to this attack because Google Maps loads dynamic JavaScript, and Google's I'm Feeling Lucky provides an open redirect weakness.

Google does not mitigate this attack by removing the dynamic JavaScript. As of April 2023, Google Maps still loads dynamic JavaScript. The CSP of Google Maps contains `script-src 'unsafe-eval'`, which is required for calling `eval()`.

Google mitigates this attack by removing the open redirect [17]. Google uses the NID cookie to identify different users

[18]. When the user clicks the I'm Feeling Lucky button, Google computes a signature that includes the search query and a secret related to the user. The signature is included in the HTTP request in the `iflsig` query field. If the signature verifies, the Google Search server returns HTTP 302 redirect. Otherwise, the server returns an HTML page asking the user to confirm the redirection.

We think this defense is effective. When an attacker constructs the malicious AdBlock Plus rule, he or she cannot compute the signature required for the `iflsig` query field. It is possible to compute the signature if the attacker is able to inject cookies to the victim user's browser and overwrites the NID cookie. However, we think this attack is out of scope.

C. Defense by Adblock Plus

TODO

- How Adblock Plus defended (limits the ability of filter rules).

D. Proposed Mitigation

We propose another way for websites to mitigate against this attack, even though the users uses a vulnerable extension like AdBlock Plus 3.2. We mitigate through the process of loading the dynamic JavaScript. Instead of executing whatever string returned in the HTTP response, the victim website should cryptographically verify the dynamic JavaScript fetched from the server.

One possible way to implement asymmetric signatures. The victim server holds a public key and private key pair. When the victim browser loads the victim website, the victim server sends the public key to the victim browser through static JavaScript. When the victim browser requests dynamic JavaScript, the victim server sends the JavaScript content and signs hash of the content using the secret key. The victim browser verifies the signature before calling `eval()` on the dynamic JavaScript content.

The disadvantage of this mitigation is that both the server and the browser requires additional computation power and time to compute the cryptographic signature. This will likely increase the delay in the user interaction.

E. Limitations and Future Work

In this research, we only investigate the Chromium browser. The main reason is that Chromium browser is popular and old versions of Chrome extensions can be easily downloaded from third party archives. Though we expect other browsers like Firefox, Safari, and Edge to behave the same, future work is needed to verify this behavior. We expect compiling AdBlock Plus from source when testing on other browsers.

In this research, we also investigate the Linux operating system. Future work is expected to verify whether the attack still works on other operating systems like Windows and macOS.

Since AdBlock Plus 3.2 uses manifest V2, the reproducibility of our demo depends on the browser supporting manifest V2. Chrome is in the process of ending support for manifest V2, but the exact timeline is not determined as of this research [12]. Thus, to reproduce this demo in the future when only manifest V3 is supported, the the Chromium browser must be downgraded.

VII. Related Work

TODO

- Advertisement blocking and detection.

VIII. Conclusion

TODO

Acknowledgment

TODO

References

[1] "Adblock Plus - free ad blocker - chrome web store." [Online]. Available: https://chrome.google.com/webstore/detail/adblock-plus-free-ad-bloc/cfhdojbkjhnklbpkdaibdccddilifddb

[2] "Adblock Plus – Get this Extension for Firefox (en-US)." [Online]. Available: https://addons.mozilla.org/en-US/firefox/addon/adblock-plus/

[3] A. Sebastian, "Adblock plus filter lists may execute arbitrary code in web pages," Accessed: Mar. 30, 2023. [Online]. Available: https://armin.dev/blog/2019/04/adblock-plus-code-injection/

[4] Adblock Plus, "Implement the $rewrite filter option," Accessed: Mar. 30, 2023. [Online]. Available: https://issues.adblockplus.org/ticket/6622/

[5] ——, "Implement the $rewrite filter option," Accessed: Mar. 30, 2023. [Online]. Available: https://codereview.adblockplus.org/29760707/

[6] ——, "How to write filters," Accessed: Mar. 30, 2023. [Online]. Available: https://help.adblockplus.org/hc/en-us/articles/360062733293-How-to-write-filters

[7] M. H. Mughees, Z. Qian, Z. Shafiq, K. Dash, and P. Hui, "A First Look at Ad-block Detection: A New Arms Race on the Web," May 2016, arXiv:1605.05841 [cs]. [Online]. Available: http://arxiv.org/abs/1605.05841

[8] "Open redirectors | Google Bug Hunters." [Online]. Available: https://bughunters.google.com/learn/invalid-reports/web-platform/navigation/6680364896223232/open-redirectors

[9] Adblock Plus, "Known adblock plus subscriptions," Accessed: Apr. 23, 2023. [Online]. Available: https://adblockplus.org/en/subscriptions

[10] Crx4Chrome, "Adblock plus version history for chrome," Accessed: Apr. 20, 2023. [Online]. Available: https://www.crx4chrome.com/history/31928/

[11] Stack Overflow contributors, "Error when installing chrome extension from file: Crx_header_invalid," Accessed: Apr. 20, 2023. [Online]. Available: https://stackoverflow.com/questions/57034873/

[12] Chrome Developers, "About manifest v2," Accessed: Apr. 20, 2023. [Online]. Available: https://developer.chrome.com/docs/extensions/mv2/

[13] The Chromium Projects, "Download chromium," Accessed: Apr. 20, 2023. [Online]. Available: https://www.chromium.org/getting-involved/download-chromium/

[14] The MITRE Corporation, "Cwe-601: Url redirection to untrusted site ('open redirect')," Accessed: Apr. 20, 2023. [Online]. Available: https://cwe.mitre.org/data/definitions/601.html

[15] ——, "Cwe-434: Unrestricted upload of file with dangerous type," Accessed: Apr. 20, 2023. [Online]. Available: https://cwe.mitre.org/data/definitions/434.html

[16] Crx4Chrome, "Adblock version history for chrome," Accessed: Apr. 20, 2023. [Online]. Available: https://www.crx4chrome.com/history/31927/

[17] Super User contributors, "Google 'feeling lucky' url causing redirect notice," Accessed: Apr. 23, 2023. [Online]. Available: https://superuser.com/questions/1496083/

[18] Google, "How google uses cookies," Accessed: Apr. 23, 2023. [Online]. Available: https://policies.google.com/technologies/cookies

APPENDIX

This section records our responds to the peer review.

The first comment is about the replace script changing both static JS and dynamic JS's HTML elements, which creates confusion. As a response, we modify the demo to only change one of static JS and dynamic JS.

The second comment is about Section V mentioning only the start and end version of vulnerable extensions, without mentioning the versions in between. The peer reviewer suggests changing the sentences to "between version X and Y." However, we choose to keep our writing because we want to only report what we tested. We add a sentence to state that the versions in between are assumed to be reproducible, though we did not test them.

The third comment is about the business model in the final presentation. TODO

The fourth comment is about the Brave browser. We consider the Brave browser as out of scope. Through superficial research on its code base, it seems this browser implements advertisement blocking logic through Rust, which is different from AdBlock Plus. The advertisement blocking code does not contain the keyword "rewrite" in the context of the rewrite feature discussed in this paper, so we think Brave browser is not vulnerable.

The fifth comment is about Microsoft Edge. In our older draft of the paper we only mention Chrome (Chromium), Firefox, and Safari. We modify the paper to add Edge, since Edge is also a popular browser.