Helping or Hindering?
How Browser Extensions Undermine Security

Shubham Agarwal

Princeton University, Princeton NJ 08544, USA
CISPA Helmholtz Center for Information Security
Saarbrücken, Saarland, Germany
shubham.agarwal@cispa.de

Abstract: Browser extensions extend the functionality of web browsers by adding useful features and make tasks easy and efficient. As extensions have access to powerful Web extension APIs provided by the browser, they can access various components of the browser. For example, interact with and manipulate the browser's bookmarking system, get and set cookies, modify various privacy-related browser settings, intercept and control client-server exchanges, etc. Which makes them more potent than web applications that are native to the browser. Thus they can also create security concerns. This paper looks at one such problem where extensions modify security headers which undermines security. And then introduces a systematic procedure to identify such extensions using static and dynamic analysis techniques.

Keywords: Browser Extensions · HTTP Security Headers · Client-side Security

1 Summary

In this paper we focus on security header exchanges between the server and the client. In 2015, Hausknecht et al. [1] demonstrated that extensions may have to modify headers like CSP(Content Security Policy) in order to implement desired functionality. And to enable this they also proposed an endorsement mechanism. However, altering any security headers can have risky repercussions and could result in disabled security mechanisms.

Now we go through the automated pipeline presented by the author and understand how can we detect extensions that modify security headers using a hybrid analysis technique. The Framework is divided into the following four steps.

1.1 Pre-filtering and Static Analysis

If an extension does not declare necessary permissions such as *webRequest* and *webRequestBlocking* in its *manifest.json* file, we can conclude that it does not have the ability to access and modify security headers. Such extensions are not considered for further analysis.

For the other remaining extensions, we then begin by identifying which host permissions are requested by the extension or which hosts activate the extension. There are three ways to achieve this. (i) List all the URLs such as http://www.example.org and wildecards such as http://*.example.org from the manifest. (ii) Most extensions are not specific to any particular domain and thus declare <*all_urls*>, *http://*//** or *https://*/** [2] as target hosts. (iii) Many extensions declare <*all_urls*> in their manifest, but only target specific domains by defining hosts within the *webRequest* API.

Then the URLs/hosts are extracted from the wildcards, storing them for dynamic analysis. For example, *://*.foo/*, and http://*.foo/*/*/* is processed and resolved to http://www.sub.foo.com.

1.2 Extension Instrumentation

In this phase first the background script(s) of each extension is found by analyzing the manifest and any background pages. Then we rewrite the definition of the corresponding webRequest APIs within the extension-space. Which allows a mechanism to capture headers while keeping the intended functionalities intact. By doing this, the framework can track both the original headers received by the API and the modified headers returned by the callback method during a single visit to the target host.

```
function _hooked() {
    let originalHeaders = arguments;
    let newHeaders = callback(arguments);
    recordHeaders(originalHeaders, newHeaders);
    return newHeaders;
};
```

Listing 1: Instrumented API function to collect response headers.

1.3 Dynamic Analysis and Header Collection

In this step we use a tool like Puppeteer[3], which allows automated interaction with web pages, to load instrumented extensions in the browser to visit the target hosts of the extension and record the original and modified headers during runtime, and store them in a logging server.

1.4 Comaprative Analysis and Threat Detection

From the previous step we have the original and modified headers available for analysis. Now two sets of headers are compared to find differences in their values and are classified accordingly. For example, in case of the XFO(X-Frame-Options) header. If an extension modifies this header by replacing DENY with ALLOWALL, the framework flags the extension as *modifiedHeader*. Adding a security header with no ground truth is labeled as *injectedHeader*, and removing or replacing a ground truth header with an empty string is labeled as *stripped-Header*. The framework then reports the modified, injected, and dropped headers as part of the analysis results and detects if the extension shows benign or suspicious behavior.

Using this proposed framework, extensions of Chrome and Firefox were examined, and a large number of extensions which modify security headers were identified. Developers of a small subset of such extensions were notified to address the underlying problem and to understand the design choices behind such actions. The responses from the developers mostly fall in four categories. (i) Acknowledged the report and claimed to either take down the extension or fix the issue. (ii) They were unaware of potential security risks of modifying headers. (iii) Intended benign usage behind altering security headers. (iv) Had an unidentified bug or legacy code.

Many extensions modify headers for their desired functionalities based on the description provided by them. Some of these functionalities include injecting scripts to highlight texts, provide additional tools on social media platforms such as WhatsApp, uMatrix development build & CSP Safe Browsing replaces the server-sent CSP header to prevent all content-based injection attacks, etc. Although many extensions modify headers for benign reasons such modifications can sometimes be extreme or unnecessary and may compromise security, Findings so far suggest that the prevalence of extensions modifying security headers is higher than those that intentionally modify headers or become malicious. In particular, CSP and XFO headers are the most commonly modified headers, affecting millions of users without their knowledge, so it is important to inform users explicitly about these potential risks. This can be achieved by integrating the proposed pipeline into the extension vetting process which will add a warning to the overview page in the extension store if the extension is suspicious.

2 Assumptions

- The paper begins with the assumption that extensions downgrade client-side security by modifying security headers but their are extension that enhance security-related features or by blocking malicious content e.g. NoScript, AdBlocker, and other privacy-preserving extensions.

- It is assumed that extensions that modify security headers are harmful but there are extensions may intend to modify security headers to provide functionality, and such modifications may not necessarily lead to harmful consequences.

- It is assumed that the presented framework can detect all potentially harmful extensions that modify security headers but there can be undiscovered ways in which extensions can tamper with headers or evade detection.

- To find the harmfulness of extension, static analysis is done but it doesn't consider extensions that use dynamically created hosts or manipulate the hosts at runtime.

- Few extensions modify *SameSite* cookies to be less restrictive, but it is not explained how this can enable CSRF(Cross-Site Request Forgery) attacks or what the consequences will be for the users.

3 Technical approach

- When processing wildcards to extract hosts the transformation may yield incorrect results. For example, wildcards like https://facebook.com/*/*/mypage/* and *://*/*/*.pdf would not yield a valid URL. But we can clearly observe that the extension would get activated when browsing a pdf file.

- While extracting URLs, ehe static analyzer extracts literal values from target APIs and fails to resolve pointer to any variables, this reduces the performance of the analyser as it has to rely on hosts found in manifest or in worst case use Tranco top 100 domains.

- If an extension has requested *activeTab* permission, then it can target any host present in the active tab when the user is interacting with the browser. The framework fails to resolve this issue and has to depend on self created test domains.

- Sometimes manipulations depend on some trigger conditions. The Dynamic analysis fails to capture extensions where the target APIs were registered but not triggered.

- If an extension requires event handlers, which can only be installed by user's consent or fulfillment of some criteria. The framework can not examine such extensions because it is automated and based on no user intervention.

4 Analysis and Results

- Presented paper only focuses on security related headers and does not consider the impact of extensions on other aspects which can affect system security.

- Author only finds what extensions are doing to alter the security of the system but doesn't mention how to prevent these modifications in the header other than notifying it.

- Initial analysis in the paper indicate that a lot of extensions are modifying security headers but on further analysis we understand that most of them are for benign reasons. Even though author claims that such modifications may lead to security concerns, no extensions are pinpointed which intend to do such modifications for malicious reasons.

5 Improvements and Extensions

- A more sophisticated procedure can be developed to extract hosts from wildcards and APIs so that the transformation yields valid URLs.

- The analysis is based on a automated pipeline and needs no user interactions, but because of this we have to rule out such extensions which require some kind of user interaction in the runtime. Instead of completely ignoring such extensions a small scale manual analysis can be done on a subset of such extensions.

- In addition to discussing the risks associated with browser extensions, the paper should provide recommendations for users and developers on how to minimize these risks. This could involve discussing best practices for using and developing extensions, as well as highlighting specific security tools or technologies that can help mitigate risks.

- If the goal is to find suspicious extensions, along with analysing only security headers the framework should check other behaviour such as accessing sensitive information, executing malicious code, tracking scripts, stealing user data, etc.

6 References

[1] Daniel Hausknecht, Jonas Magazinius, and Andrei Sabelfeld. 2015. May I?-Content Security Policy endorsement for browser extensions. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.

[2] Chrome Developers. 2020. Match Patterns. https://developer.chrome.com/extensions/match_patterns

[3] Chrome Developers. 2020. Puppeteer. https://developers.google.com/web/tools/puppeteer