Exploiting Leaky Chrome Extension API

6.857 Final Project

Benjamin Chen Tony Ding Xiaolu Guo Adelaide Oh

May 15 2019

Abstract

Given the popular use and reliance on Chrome extensions, it is important to consider their security, especially because these extensions often have access to user data and permissions to modify the browser itself. Extensions can have permissions to sensitive user data, and extension API calls with the browser can be leaky, potentially exposing this data to malicious websites. Consequently, we examined the security threat of malicious websites stealing user data through benign Google Chrome extensions. To demonstrate this, we created a proof-of-concept website and extension that display an attack to read user cookies, as well as expand on how this attack could be manipulated in other ways to do more harm. Based on our results and discussion, we provide recommendations to users, developers, and Chrome to mitigate these security concerns, as well as provide a final assessment on the security of Chrome extensions for our attack vector.

1 Introduction

Over the last few years, Google Chrome has taken off as the most-used internet browsing application, boasting over 60% of the browser market share worldwide [1]. The browser offers a rich variety of user engagement and customizability which is further enhanced by the ability to add free browser extensions. Through the application of extensions, Google Chrome allows developers to build plug-ins that create additional functionality for users. Users are able to download Chrome extensions created by Google and third-party developers alike from the Chrome Web Store for free, allowing users to uniquely customize their browsing experience. As a browser, Google Chrome is incredibly secure, sandboxing the browsing environment and blocking common malware among other things to ensure security. However, the nature of browser extensions creates a measure of vulnerability and a potential threat to unsuspecting users.

Chrome extensions typically possess special privileges to browser data and can often modify the client-side code that a user interacts with. Coupled with the convenience and ubiquity of Chrome extensions, there is a significant risk posed to security. Despite the guidelines Google set for developers to create safe and secure extensions, a study conducted from 2012 to 2015 found that 10% of all unique extensions submitted to the Google Chrome store during this period were found to be malicious [2]. For example, in the past, malicious third-party developers have leveraged Chrome extensions to create botnets [3]. Unwitting developers who do not implement proper privilege separation can create privilege abuse vulnerabilities; another study estimated about 50% of extensions to have some sort of privilege abuse [4].

We explored the security of Google Chrome extensions and ways in which the current security protocol might be improved. Specifically, we focused on how extensions can unwittingly be exploited by malicious websites. In this paper, we will first examine how Chrome extensions work before discussing our various attempts to exploit Chrome extensions through a malicious website. Finally, we will analyze our results and present our recommendations to Chrome extension users as well as to Chrome.

2 Background

2.1 Chrome Web Store

In order to provide some safeguards to users, the Google Chrome store provides a series of guidelines to its developers called the Content Security Policy (CSP). Broadly, the CSP works as a block/allowlist for specific functions, resources, or code executed by the extension in the scope of the web browser. The policy listings for each extension can be found in the `manifest.json` file.

Under the default policy restrictions, eval and related executable API are disabled. This prevents cross-site scripting attacks through strings containing Javascript code. Similarly, inline Javascript is also not executed by default. Finally, only local script and object resources from the extension package itself are loaded.

In addition, Google Chrome recommends that extensions do the following: protect developer accounts, never use HTTP, request minimal permissions, limit manifest fields, include an explicit content security policy, avoid executable APIs, use content scripts carefully, and register and sanitize inputs

Nevertheless, the exact mechanisms to actively enforce these guidelines are hidden. Google Chrome claims that submissions to the Web Store go through basic security scans for plugin behavior and code style to identify malware. All submitted extensions are subject to Enhanced Item Validation, which is described as "a series of automated checks that examine its code and behavior once installed to identify malware. Once the validation is complete, the app is published – usually within an hour" [11]. Through this Enhanced Item Validation, Chrome does reject about 10% of all submitted extensions.

However, Google Chrome does not publicly provide insight about how extensions are tested before being added to the Chrome web store, leaving us to wonder whether these guidelines are recommendations or strongly enforced expectations in the scans.

Google also maintains that it possess the right to remove any items that are deemed to have been "associated with a security vulnerability that could be exploited to compromise another application, service, browser, or system." This seems to be more of a reactive response rather than a preemptive measure on Google's part.

2.2 Extension Architecture

An extension is made up of zipped bundles of HTML, CSS, JavaScript, and images. They have the ability to modify web content users see and interact with or extend and change the behavior of the browser itself.

Each Chrome extension must have a `manifest.json` file that defines the extension's security policy. This `manifest.json` file give the browser information about important files and capabilities of the extension. We elaborate on the permissions of the `manifest.json` file in 2.3.

The background script or page is found in `background.js` and is also registered in the `manifest.json` file in the "background" field [10]. This file runs in the background of the Chrome browser and launches when Chrome itself launches. In general, it contains listeners that wait for events from the browser, such as external messages or a new page being opened.

Content scripts are files that run in the context of web pages. They are injected into the browser and can read the source code of web pages, modify the code, and relay information between the extension and page [10]. Communication between the page and content script is achieved through the shared Document Object Model of the page, with the provided Chrome messaging API. For most extensions, the `content_script.js` file is the main source code file that modifies web pages.

The user interface of the extension includes the visible toolbar and popups that the extension displays to the user. It may include user settings and preferences for the extension.

2.3 Extension Permissions

Extensions are able to access and use a variety of Chrome APIs that can interact with an end-user's computer and browser beyond the limitations of a normal website. However, in order to use such features, the extension must list the permissions it needs within the manifest file. These permissions help limit damage if the extension is compromised by malware [8], and when a user downloads the extension, Chrome can deliver a user-friendly warning prompt stating the potential actions that the extension can do.

This snippet of the `manifest.json` file for Honey, an extension that helps you find coupons for online purchases, contains multiple such permissions. We see that, for example, Honey uses the "webRequest" permission, which allows it to observe and analyze web traffic, and also intercept or modify in-progress requests. By accessing all these permissions, extensions have the ability to leak important information that a website might not normally have access to.

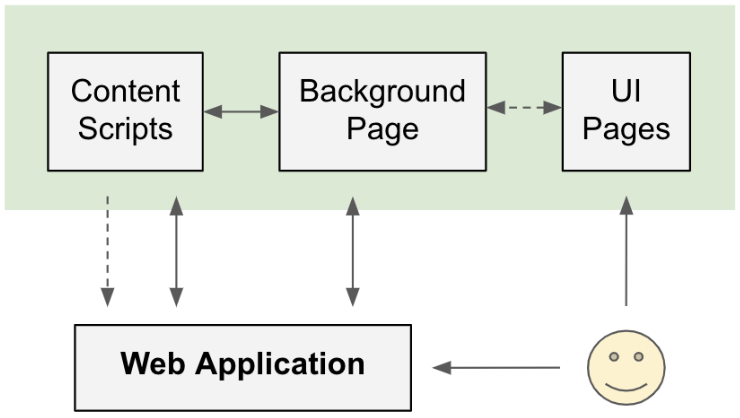
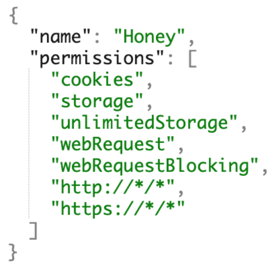


Figure 1: Chrome extension architecture



```
{
  "name": "Honey",
  "permissions": [
    "cookies",
    "storage",
    "unlimitedStorage",
    "webRequest",
    "webRequestBlocking",
    "http://*/*",
    "https://*/*"
  ]
}
```

Figure 2: Honey permissions

2.4 Same Origin Policy

Security policies that apply to websites but not extensions serve as motivations for malicious websites to exploit extensions. In particular, the Same Origin Policy (SOP) is an important security mechanism that blocks content from one origin interacting with content from another. In the context of websites, this would block potentially malicious documents or scripts, thus reducing the vulnerability of the site. As users are likely to visit multiple websites within each browser session, SOP is an easy way to increase security without a user needing to strongly verify the authenticity of each site before.

However, SOP does not apply to extensions. According to Chrome's developer website, "a script executing in an extension's origin can talk to remote servers outside of its origin, as long as the extension requests cross-origin permissions" [9]. Extensions can make cross-origin requests to either specific websites or all websites if the developer chooses to do so. This would then allow a malicious website to take advantage of this loophole and essentially bypass SOP if they were to hijack an extension. A malicious website would then have the power to read and write user data through the permissions granted to the extension.

Google Chrome does not consider this a security vulnerability because a user chooses to manually install and enable permissions on an extension, thus making them more liable for what might follow. In theory, if Chrome established more rigorous security protocols for checking extensions, this might be an adequate model. In practice, lazy developers can be easily exploited and users become at risk.

3 Prior Work

There have been many studies of the security of Google Chrome extensions in the past. These studies have observed the security of both malicious and benign extensions through examining extensions currently available to the public and creating new ones to test for potential security risks.

Previous literature has done extensive work in analyzing the security of Chrome extensions that are publicly available. Carlini et al. examined 100 Chrome extensions - the 50 most popular extensions and 50 random ones - and found "70 vulnerabilities across 40 extensions" [7]. They identified the main three mechanisms employed by Google to mitigate security vulnerabilities: privilege separation, isolated worlds, and permissions. The paper then describes the particular vulnerabilities that each mechanism might have. Carlini et al. did not address the threat of malicious extensions, which is something our group would like to look at in more detail. Despite that, their analysis of the threat from network attackers and web attackers is also something we will consider when building our own Chrome extension.

In the paper, "Chrome Extensions: Threat Analysis and Countermeasures," the authors created their own Chrome extension, Bang!. The authors were then able to use Bang! for a range of malicious activities, including large-scale email spamming, DDoS attacks, and phishing attacks. The authors also discussed cross-site forgery with content scripts and cross-site requests with extension cores. Finally, the authors presented countermeasures to reduce risks from malicious extensions [6]. We hope to use this as a reference and guideline in creating our own extension.

Doliere Francis Some examines topics closer to our goals in "EmPoWeb: Empowering Web Applications with Browser Extensions". He analyzed the communication interfaces exposed to web applications by Chrome, Firefox and Opera browser extensions [5]. Looking specifically at extension APIs of these interfaces, he created web applications that could exploit the privileges of the extension to bypass SOP and access user data.

We distinguish ourselves from previous studies by focusing on malicious websites attacking benign extensions. We created our own Chrome extension that demonstrated the potential areas of vulnerability. In particular, we focus on cross-site scripting injections and leaky message passing and build an app highlighting these issues.

4 Attack Approaches

4.1 Overview

To conduct our attacks, we created a custom extension and website. Our extension is based off the popular Millennial to Snake People Chrome extension and is modified to fit our needs. Our extension is not inherently malicious but has several vulnerabilities due to lazy programming. On the other hand, our website is malicious and aims to exploit the extension to obtain sensitive user information. We will demonstrate that benign and simple extensions, such as the Millennial to Snake People extension, can be exploited by malicious websites with severe consequences.

4.2 Threat Model

The threat model we consider in our extension constructions are primarily malicious websites. In particular, we did not try to make a malicious extension - we know that these already exist and are a current problem. Rather, we focused on identifying how seemingly benign extensions can be exploited by malicious websites, typically by taking advantage of permissions granted to extensions.

Malicious websites can obtain sensitive user data such as cookies and browsing history and download files through leaky extension APIs. These sensitive API calls that potentially handle user data include messaging calls between the extension and website.

Through extensions' APIs, web applications can bypass SOP and access user data on any other web application, access user credentials (cookies), browsing history, bookmarks, list of installed extensions, extensions storage, and download and save arbitrary files in the user's device.

4.3 Exploiting Cookie Permissions

4.3.1 Attack Explanation

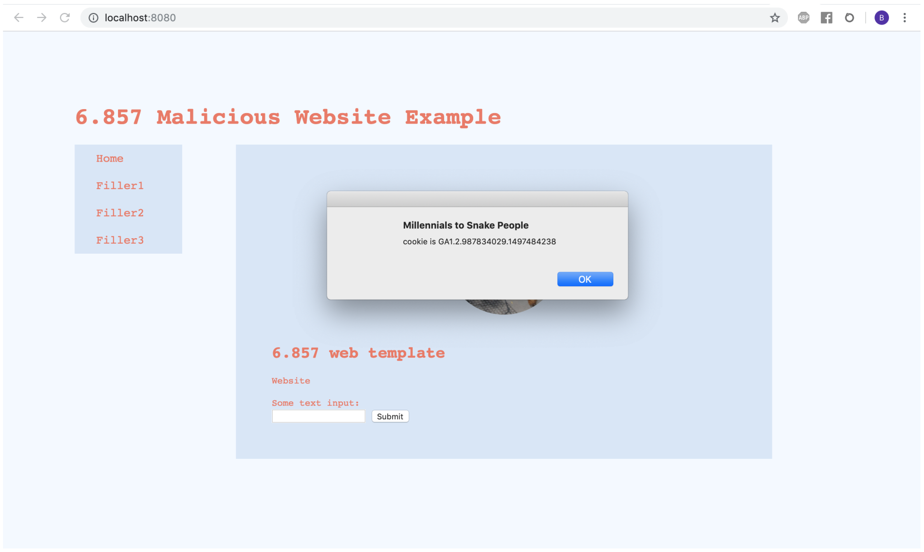
In this attack, we exploit the cookies permission that Chrome extensions possess. Specifically, if the cookies permission in the `manifest.json` file is included, Chrome extensions can view any cookie from a given website through the background.js file using the command `chrome.cookies.get()`. This can enable any malicious Chrome extension to steal user cookies. However, non-malicious Chrome extensions can also unwittingly give up user cookies to a malicious website.

In our attack, our extension allows for any website to send it a message with a list of words or phrases. The extension will not change any of the words or phrases sent in this message. In the background.js file, our extension has an event listener that listens for messages sent by a website through the window. The extension then calls eval to evaluate the received message into a list, which it then iterates over to remove any words or phrases in the list from the words or phrases to be changed by the extension.

However, a malicious website may realize that the extension is using eval by reading the extension's source code. Instead of sending a list of words or phrases, the malicious website can instead send javascript code. Since the extension uses eval on the message that is sent, it will unknowingly run the javascript code in the context of background.js. In our attack, we constructed a malicious website that sends the message:

```
chrome.cookies.get({
    url: ‘https://courses.csail.mit.edu/6.857/2019/studentsOnly/’,
    name: ‘_ga’},
    function (cookie) {
        if (cookie) {alert(’cookie is ’ + cookie.value);}
    }
);
```

Once evaluated, because background.js has cookies permission, it will run the javascript code in the message and alert the value of the user's cookie for the website, 6.857 Top Secret Students Only. Of course, the message sent by the malicious website can be structured such that the cookie's value is instead sent to the attacker directly (for instance, via email) so that the user cannot detect the attack.



Millennials to Snake People

cookie is GA1.2.987834029.1497484238

Figure 3: Malicious website stealing the 6.857 cookie

4.3.2 Attack Analysis

This attack depends on several factors: 1) the enabling of cookies permission, 2) the inclusion of permissions for the required sites, such as the 6.857 Top Secret

Students Only site, and 3) the use of "eval" to evaluate the message passed from the website. We will now analyze each of these factors.

One requirement for our attack is the enabling of cookies permission. Many extensions, such as Honey, enable this permission. In 2011, of the top 30 most popular extensions, 3 extensions had permissions for cookies [6]. As we will discuss in Section 4.4, we can extend this attack to procure other sensitive user information, like browsing history. We can also mess with the user's browser by deleting all of their tabs repeatedly. These extended attacks also depend on popular permissions. For instance, the tabs permission is granted to 27 out of the top 30 most popular extensions in 2011 [6].

Our attack additionally depends on the inclusion of permissions for the required sites, such as the 6.857 Top Secret Students Only site. This requirement is satisfied by many Chrome extensions. By adding permissions for "http://*/*" and "https://*/*", many Chrome extensions enable HTTP request permissions for all websites in their `manifest.json` file. In 2011, of the top 30 most popular extensions, 19 extensions had permissions for "http://*/*" or "https://*/*" [6].

Finally, our attack depends on the use of "eval" to evaluate the message passed from the website. The use of "eval" is generally considered bad programming practice, and is rarely seen in extensions. The primary reason is that if code is passed into eval, it will automatically be run in the context of the eval. As a result, Chrome on default prevents the use of eval in its Content Security Policy. However, it is possible to enable eval by simply including it in the Content Security Policy in the `manifest.json` file. Indeed, because we cannot expect all programmers to follow good programming practice, there are extensions that still use eval. For instance, one researcher discovered a number of extensions, such as erail.in, that are vulnerable due to the use of eval [5]. It should be noted that erail.in has since removed its use of eval.

Our attack would fail on most extensions that follow proper programming guidelines and adhere to Chrome's Content Security Policy. We did not find any notable weaknesses in the overall structure of Chrome extensions. However, our attack does highlight severe vulnerabilities that result from developer negligence. By exploiting these vulnerabilities, malicious websites can obtain sensitive user information or disrupt user experience. In section 5, we will discuss our recommendations to prevent and ameliorate possible negative consequences that these vulnerabilities may cause.

4.4 Other Attacks

Our attack, given a leaky extension, can be extended to procure other sensitive information if the extension allows the use of other Chrome APIs through its permissions. Here, we present a list of some of the possible attack extensions.

- `chrome.history` - our attack would allow the malicious website to access the user's entire browsing history.
- `chrome.downloads` - our attack would allow the malicious website to initiate or manipulate downloads. This could allow the website to put arbitrary

files on the user's device, which could be used for a number of exploits.

- `chrome.storage` - our attack would allow the malicious website to store and retrieve data on the extension's storage. This could be abused for tracking purposes.
- `chrome.tabs` - our attack would allow the malicious website to see and manipulate open tabs by redirecting or closing them. This could allow a website to phish users by silently redirecting their other open tabs when trying to navigate to sensitive pages.

5 Recommendations

5.1 Chrome Extension Users

Chrome extensions can generally protect sensitive user information as long as it adheres to the Content Security Policy. Vulnerabilities generally result from poor programming practice rather than the inherent structure of Chrome extensions. As a result, if Chrome extension users are careful about downloading only reliable Chrome extensions, they will generally be safe from attacks by malicious websites. Our recommendation to Chrome extension users is to carefully select which Chrome extensions to install by looking at the number of installations and the developer that created the extension.

5.2 Developer Recommendations

Given the potential threat models we have exposed in this paper, we would also like to offer the following recommendations in order to mitigate these security concerns. Developers possess the most agency to reduce the risks we have outlined above. In order to prevent malicious websites from hijacking and exploiting leaky extensions, clearly the first line of defense is to create more robust extensions. Restricting the permissions of an extension to only include what is required for the extension to function helps reduce overextending the type of permissions and the way in which malicious websites could take advantage of extensions. Though we observed that some extensions use the "eval" function, this is something that we recommend against - much like the Chrome developer guide - as it creates a lot of room for vulnerabilities such as those enumerated in this paper.

5.3 Chrome Recommendations

It is clear that it is not enough to place the burden of security on developers alone; historically that has proven to be insufficient. Although Chrome does offer a developer guide with specific recommendations, there is not guarantee that a developer will read this guide at all, or choose to follow it even if they do read it. As mentioned in section 2.1, Google Chrome does not publicly release information about the security checks they perform on extensions that are released to the Chrome web store. Despite implementation of the Enhanced Item Validation, we can infer that the checks are not entirely comprehensive, as news of malicious or insecure Chrome extensions are relatively common even today.

Our recommendation to the Chrome store is to enforce more stringent security checks in sweeping submitted extensions. Restricting the use of the "eval" function, or at the very least, performing secondary checks if extensions use certain functions that the Chrome store discourages would be a good way to safeguard against the specific vulnerabilities we have enumerated above.

6 Conclusion

In conclusion, although our paper enumerates several potential security vulnerabilities that may put a user with Chrome extensions at risk, we generally found that Chrome extensions are relatively secure. The attack vectors we looked at typically relied on coding practices that Chrome's developer guide warns against, thus the logic follows that only lazy developers or malicious agents might be susceptible to our attacks. Although a user might be concerned as to how they could gauge the competency of an extension's developer, for the most part, we believe that popular extensions with large user bases avoid these pitfalls and remain relatively secure. Popular extensions are often held to a higher standard given that with more publicity and scrutiny, they must withstand more cyberattacks (i.e. in comparison to perhaps a smaller developer). Furthermore, popular extensions tend to be updated for bugs and patches more frequently, which also helps to reduce vulnerabilities.

Security is always an ongoing battle as there is no such thing as total security. Despite that, given the ubiquity of Chrome extensions and the reliance by users, we must still work to increase the security of Chrome extensions. From previous research and our analysis, we can conclude that although Chrome extensions are widely and frequently used, more can be done on part of both developers and the Chrome store itself in terms of security. While our team exposed a specific security vulnerability that may plague unsuspecting or lazy developers, this represents a small fraction of the vulnerabilities that developers should be cognizant of and work to mitigate.

References

[1] W3Counter. Web Browser Market Share. Retrieved March 15, 2019, from www.w3counter.com/globalstats.php.

[2] Jagpal, N., Dingle, E., Gravel, J., Mavrommatis, P., Provos, N., Abu Rajab, M., & Thomas, K. (2015). Trends and Lessons from Three Years Fighting Malicious Extensions. Retrieved March 18, 2019, from

https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-jagpal.pdf.

[3] Perrotta, R., & Hao, F. (2017). Botnet in the Browser: Understanding Threats Caused by Malicious Browser Extensions. Retrieved March 19, 2019, from https://arxiv.org/pdf/1709.09577.pdf.

[4] Rana, A., & Nagda, R. (2014). A Security Analysis Of Browser Extensions. Retrieved March 19, 2019, from https://arxiv.org/pdf/1403.3235.pdf.

[5] Some, F. Doliere (2019). EmPoWeb: Empowering Web Applications with Browser Extensions. Retrieved April 18, 2019, from https://arxiv.org/pdf/1901.03397.pdf.

[6] Liu, L., Yan, G., Zhang, X., & Chen, S. (2012). Chrome Extensions: Threat Analysis and Countermeasures. Retrieved March 18, 2019, from https://pdfs.semanticscholar.org/0081/6b774f52031ea160c05181af3251a76220e6.pdf.

[7] Carlini, N., Porter Felt, A., & Wagner, D. (2012). An Evaluation of the Google Chrome Extension Security Architecture. Retrieved March 20, 2019, from https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final177_0.pdf.

[8] Chrome. Declare Permissions. Retrieved March 18, 2019 from: developer.chrome.com/apps/declare_permissions.

[9] Chrome. Cross-Origin XMLHttpRequest. Retrieved April 4, 2019 from: developer.chrome.com/apps/xhr.

[10] Chrome. What Are Extensions? Retrieved March 18, 2019 from: developer.chrome.com/extensions.

[11] Ellis, C. Google Removes Malicious Chrome Extensions with Half a Million Downloads. Retrieved April 2, 2019 from: www.techradar.com/news/google-removes-malicious-chrome-extensions-with-half-a-million-downloads.