

Security Analysis of Chrome Extensions

Ryan Chipman, Tatsiana Ivonchyk, Danielle Man, Morgan Voss

rchipman, ivonchyk, daniman, vossm

May 12, 2016

Abstract

Permissions models, when properly implemented, allow software users to evaluate and limit an application’s ability to act maliciously. We have evaluated the permissions model used by Google Chrome browser extensions, and have identified areas where the permissions model fails to effectively and accurately inform a user of an extension’s capabilities. We have developed a proof-of-concept extension that demonstrates some of these failures, and have included a set of recommended modifications to the Chrome extension permissions model to prevent future developers from exploiting these failures.

1 Introduction

Chrome is largely regarded as the web’s most secure browser. It provides a sandboxed browsing environment and it safeguards its users from common malware and phishing attacks, among many other things. Chrome is also the web’s most-used browser [1]. In addition to standard web-browsing functionality, Chrome provides developers with extension APIs that can extend the capabilities of the browser. By downloading Chrome Extensions, users can customize their browsing experience by changing the appearance and behavior of web pages¹ and of Chrome itself. Extensions have become increasingly popular since their introduction, and the Chrome Web Store is full of extensions developed both by Google and by independent companies or developers.

In order to provide these web browsing enhancements, Chrome extensions are allowed more flexibility and control than traditional web applications [2]. However, it is not desirable (nor is it prudent) to allow every extension to make use of all these advanced capabilities. Every browser approaches this problem differently; notably, Firefox and Safari both have approaches that differ from the one Chrome uses. Chrome employs a permissions model where the user must explicitly allow the extension to use certain subsets of the extensions API.

Our goal with this project was to investigate Chrome’s permissions model, and to see if we could expose any shortcomings. A perfect permissions model would ensure that a user is aware of exactly what a particular extension can and cannot do. However, when the permissions model falls short on that front, discrepancies between what an extension can do and what the user *thinks* it can do leave room for exploitation. Finding such discrepancies was the focus of this research project. Our proof-of-concept extension² demonstrates compelling attacks that can still be perpetrated within the constraints of the existing Chrome permissions policy.

¹The Drumhinator chrome extension translates the text “Donald Trump” to “Donald Drumph” on any website you visit. AddBlock, one of the most popular Chrome extensions, removes ads from pages you visit and even blocks them from playing on media such as YouTube videos.

²Our extension is available on the Chrome Web Store as [Chroak](#). The details of Chroak will be discussed in the sections to come.

In the following section, we will give an overview of the security policies that three major Internet browsers have for extensions (Safari, Firefox, and Chrome). Next, we will discuss the implementation details of our proof-of-concept extension, Chroak, and discuss the conclusions that we drew about Chrome’s permissions model. Finally, we will recommend some changes to the permissions model to help mitigate the vulnerabilities we discovered.

2 Extension Security Policies

There are interesting differences in how Safari, Firefox, and Chrome each handle ensuring the security of the third-party extensions (also known as ‘extensions’ in Safari and as ‘add-ons’ in Firefox) that are listed in their official extension stores. In this section, let the term “security of an extension” not only cover good coding practices by the developer and protecting an extension from outside attacks, but also the security of the end-user from the extension itself.

2.1 Safari

Safari’s parent company Apple reviews and signs all extensions before hosting them in the Safari Extensions Gallery, thus taking on the responsibility for ensuring extension security itself. Apple does list some security recommendations in the Safari Extensions Development Guide, such as not to use imported text and not to use the ‘eval’ method. If followed, these recommendations can prevent attacks such as cross-site scripting. Safari also recommends only using HTTPS to import items, in order to prevent man-in-the-middle attacks, in which an adversary could inject malicious content into the packets being transferred [3]. It is stated that extensions containing malicious or misleading content or code will be rejected, but the developer is not given more specific guidelines as to what is considered malicious [4].

After a developer submits an extension for review, Apple runs through tests to ensure that the extension is not vulnerable to common attacks such as XSS and is not malicious. Notably, Apple has the final word about what is deemed ‘malicious’. When an end-user visits the Safari Extensions Gallery and downloads an extension, there is no obvious information or notification about what a specific extension can access or do when installed on a user’s computer. In order to reveal whether an extension can read which URLs a user is visiting, for example, a user must dig through the extension’s or developer’s personal website. There is no existing notification system on the Safari Extensions Gallery. Thus, the security of the user is ultimately left to Apple’s extension review process.

2.2 Firefox

Add-ons on Mozilla’s Firefox browser, which are listed on addons.mozilla.org (AMO), the equivalent of the Safari Extensions Gallery, are also required to be signed by Mozilla, starting with Firefox version 40. Similar to Apple, Mozilla provides add-on developers with a ‘Security best practices in extensions’. This document outlines implementations and behaviors that Mozilla either requires or recommends. The list includes many of the same suggestions as Apple’s developer guide, focusing on attacks carried out on a developed add-on by malicious outsiders [5]. After an add-on is developed, if the developer wishes the add-on to be listed on the AMO, it must be submitted to the AMO for either an automated or manual code review, done by employees or volunteers. Listed add-ons can undergo either Preliminary or Full Review, which extends Preliminary Review with feature testing and performance checks and results in more prominence on AMO. [6].

Firefox’s Review Policies document [5] provides more detailed information about which behaviors are disallowed in add-ons than the equivalent document for Safari Extensions. In addition to

elaborating on disallowed coding practices, which would render an add-on vulnerable to external attacks, Mozilla gives examples of add-on behaviors that are inappropriate. This list includes “making unexpected changes to the browser or web content”, “preventing users from reverting changes made by the add-on”, and “preventing the user from disabling or uninstalling the add-on”, among other unsuitable behavior. Mozilla list that developers should “clearly disclose all user data handling in a Privacy Policy” as a requirement as well [7]. This privacy policy is provided next to the download button on the add-ons page, when the add-on has access to data that could identify a user. However, if an end-user does not visit this page and downloads the add-on directly from the AMO homepage, they will not be provided any of the information that is listed in the Privacy Policy before installation is started. Thus, like Safari, most of the user’s security is left to Firefox’s extension review process.

2.3 Chrome

Google Chrome has a “Content Security Policy” document that describes policies regarding resource loading and execution by an extension. The goal of the document is to provide developers with a guideline for developing extensions that will not be vulnerable to malicious attacks from outside parties.

A Chrome extension is required to have a “manifest.json” file, which defines the extension’s security policy. Within this document, an extension must declare a ‘manifest_version’. In order to be listed on the Chrome Web Store, an extension must have a manifest_version of at least 2, which by default limits the source of scripts and objects used by the extension to “self”, meaning that “eval” and other related functions are disabled, inline javascript will not be executed, and only local script and object resources can be loaded³ [9]. In order to relax these limitations, an extension must specify the changes to the policy in the manifest file. Changes to the default content security policy will warn users downloading the extension that modifications have been made and that the extension may be vulnerable to attacks.

In addition to notifying users about changes to the default Content Security Policy, Chrome extensions also notify users about the different permissions that an extension is using. The full list of available permissions is shown in Table 1 [10]. These permissions give developers of extensions access to various Chrome APIs, which can interact with an end-user’s computer and browser beyond the limitations of a normal website. In order to use a specific permission, the permission must be declared in the extension’s “manifest.json” file. This declaration prompts a short, user-friendly warning that Chrome has deemed informative and appropriate, to be displayed to the user when an extension is downloaded on the Google Chrome Web Store. An example warning is: “[My Extension] has requested additional permissions. It could: Access your tabs and browsing activity.” A pop-up containing a list of all warnings generated from an extension’s permissions will unavoidably be generated when an end-user downloads a Chrome extension [11].

Unlike Safari and Firefox, Chrome does not provide details on any testing that is performed on extensions prior to their listing in the Chrome Web Store. The responsibility to determine whether an extension trustworthily uses the permissions it asks for is left solely to the end-user.

Interestingly, there exist a subset of permissions that are not considered potentially malicious or noteworthy by Chrome. When these permissions are included in an extension’s manifest file, they generate a warning reading, “Requires no special permissions”. The permissions that Google Chrome has deemed “non-special” are listed in Table 2. Thus, there is no indication to the end-user that an extension uses these permissions, even though the permissions give extensions more

³Having a manifest version of at least 2 has been enforced since January 2014 [8].

activeTab	declarativeContent	notifications	system.storage
alarms	desktopCapture	pageCapture	tabCapture
background	downloads	power	tabs
bookmarks	fontSettings	printerProvider	topSites
browsingData	gcm	privacy	tts
clipboardRead	geolocation	proxy	ttsEngine
clipboardWrite	history	sessions	unlimitedStorage
contentSettings	identity	storage	webNavigation
contextMenus	idle	system.cpu	webRequest
cookies	management	system.display	webRequestBlocking
debugger	nativeMessaging	system.memory	

Table 1: Available extension permissions.

activeTab	declarativeContent	notifications	system.storage
alarms	desktopCapture	pageCapture	tabCapture
background	downloads	power	tabs
bookmarks	fontSettings	printerProvider	topSites
browsingData	gcm	privacy	tts
clipboardRead	geolocation	proxy	ttsEngine
clipboardWrite	history	sessions	unlimitedStorage
contentSettings	identity	storage	webNavigation
contextMenus	idle	system.cpu	webRequest
cookies	management	system.display	webRequestBlocking
debugger	nativeMessaging	system.memory	

Table 2: Extension permissions that are “not special”.

privilege than a website. While Google Chrome seems to imply that no malicious activity can be conducted through requesting these “non-special” permissions, we suspected the policy was not foolproof. Our project explored potential attacks and exploits completed by an extension utilizing only these “non-special” extension permissions.

3 Extension Development

When we started the development of our extension, we were not sure what exactly we would find. Thus we took an exploratory approach and decided to look at each non-special permission individually, to see if it gave us access to potentially identifying information, or potentially malicious actions. We found a number of permissions which fit this criteria, and they are highlighted in yellow in Table 3.

Our goal was not the creation of a malicious extension. If enough permissions are requested, it is trivial to make an extension that can to *something* bad. Furthermore, in order to be effective, a malicious extension needs someone to actually install it. Ultimately, this would not be a very interesting pursuit, as any results obtained or attacks executed would be predicated on a user providing the extension with the very permissions necessary to carry out the attacks! Instead, we focused on an educational extension to prove our concept. Ultimately, we simply wanted to show users just how much information and control they are giving to each extension they install, even

activeTab	declarativeContent	notifications	system.storage
alarms	desktopCapture	pageCapture	tabCapture
background	downloads	power	tabs
bookmarks	fontSettings	printerProvider	topSites
browsingData	gcm	privacy	tts
clipboardRead	geolocation	proxy	ttsEngine
clipboardWrite	history	sessions	unlimitedStorage
contentSettings	identity	storage	webNavigation
contextMenus	idle	system.cpu	webRequest
cookies	management	system.display	webRequestBlocking
debugger	nativeMessaging	system.memory	

Table 3: Extension permissions that we used.

if the extension is not *special*.

Our extension’s homepage shows up every time a user opens a new tab. It has action buttons which a user can click to self-inflict (and undo) malicious actions, and it has a list of information that the extension knows about the user’s computer at any given time. A screenshot of the extension can be seen in Figure 1. Exactly what malicious actions and identifying information we can access is discussed in the [results](#) section of this paper.

4 Results

At the end of the exploration and development phases of our project, we were able to find a surprising amount of information and could perform a variety of attacks. *Disclaimer – We make no claim that we have exhaustively found all of the potential loop holes in the system or potential attack vectors with non-special permissions. Further research and study would be required. Yet, what we have found is compelling and worthy of an update in Chrome’s permissions model.*

4.1 Information Gathered

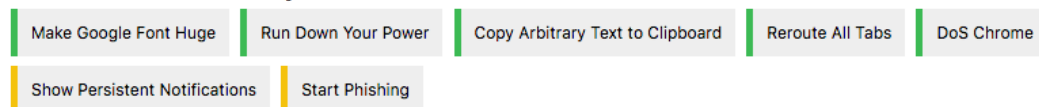
We were able to gather information from the following permissions:

- **chrome.tabs**: see the number of tabs and windows open (without explicitly requesting the tabs permission).
- **chrome.system.cpu**: see information about the user’s CPU including number of processors, processor activity time, processor architecture, and the CPU model.
- **chrome.system.display**: see information about the user’s displays, including the number of displays, the primary display, and other display properties.
- **chrome.sessions**: see the user’s chrome sessions and number of connected devices.
- **chrome.system.memory**: see the user’s total memory and current memory usage.
- **window**: see the user’s IP address, network connection status, operating system, system language, and chrome version.
- **chrome.system.storage**: see the number, types, and names of mass storage devices attached to the user’s computer.

Well, hello there.

This is **Chroak**, a Chrome Extension that teaches users to be more wary of permission lending and points out loopholes in Chrome's security policy for extensions.

Here are some actions you can take:



A horizontal list of seven buttons with colored vertical bars on their left sides. The buttons are: 'Make Google Font Huge' (green bar), 'Run Down Your Power' (green bar), 'Copy Arbitrary Text to Clipboard' (green bar), 'Reroute All Tabs' (green bar), 'DoS Chrome' (green bar), 'Show Persistent Notifications' (yellow bar), and 'Start Phishing' (yellow bar).

Here are some things your extension knows:

- You are currently **connected** to the internet and your IP address is **18.111.104.106**.
- You are running on a **MacIntel** platform in the language **en-US**.
- You are running **Chrome 50**. Other general system info: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94 Safari/537.36.
- You have **20** tabs open in **1** different windows.
- You have a **Intel(R) Core(TM) i5 CPU M 520 @ 2.40GHz** and are running **4** processors with **x86_64** architecture.
- You have **1** display(s) running.
- You have **0** other device(s) connected to Chrome.
- You have **8589934592** total bytes of physical memory capacity and **110768128** bytes of available physical memory capacity remaining.
- You are using **1** storage devices. Their names are **Macintosh HD** and they are of the types **fixed**.

Enabled Permissions:

background, clipboardWrite, fontSettings, power, sessions, system.display, system.storage, system.cpu, system.memory

Figure 1: Screenshot of Chroak, the Chrome extension we developed.

When combined, this is enough information to identify users with reasonable confidence. Additionally, information such as a user's CPU data is extremely relevant to anyone hoping to cultivate a botnet, among other things.

4.2 Attack Vectors

This section includes a brief discussion of the attacks we could carry out. Each of the following attacks is demonstrated in our extension, and we encourage readers to try the attacks out for themselves through our extension, in addition to reading about them in this paper! Our extension executes these attacks in a user-controlled manner; however, users should know that we can execute these attacks at our discretion, including upon initial installation. In the case of our Denial of Service attack, we can render a user's browser useless the moment they accept the installation.

4.2.1 Power Settings

`chrome.power` allows the extension to prevent your computer from suspending on its own. This does not prevent manual suspends, but could cause a laptop user's battery to be drained faster than expected. As with notifications, this would amount to no more than an annoyance, but would be difficult to trace back to a Chrome extension given that this permission falls under the umbrella of "no special permissions" (i.e. the user would not have been warned on installation that this extension might interfere with suspend).

4.2.2 Notifications

`chrome.notifications` allows the extension to show notifications to the user. The worst we were able to do with this permission was to show persistent notifications that kept reappearing when closed by the user. This is certainly the most benign of our findings here; at worst, this would be a minor annoyance. It is also worth noting that, as of version 50, Chrome upgraded notifications to be a "special" permission.

4.2.3 Clipboard

With `chrome.clipboardWrite`, the extension can write arbitrary text to the system clipboard. In most cases, it is unlikely that this could be used to cause more than a major annoyance to the user. However, one could imagine using it to trick the user into navigating to a malicious URL or accidentally entering some malicious text into another application when pasting from the clipboard.

4.2.4 Font Size Modification

By using `chrome.fontSettings`, our extension can change the default font size, which will affect the size of the font on any web page that does not set a font size on the root html element (wikipedia.org, for example). By setting the font size to something very large (10,000pt, say), such web pages will be rendered unreadably large. In many cases, almost nothing will be visible on the screen. On an arbitrary web page, this is not much of an attack, but is rather just an annoyance; many web pages will not even be affected by this change.

However, this will also change the font size on special chrome pages, such as <chrome://extensions>, rendering them unusable. Without going into the developer console and modifying the stylesheet, there is no easy way to circumvent these effects (and certainly no way that is accessible to non-power-users). As such, if a malicious extension developer were to couple this with a way of removing the extension's icon from Chrome's task bar, it would become impossible for an average user to uninstall the malicious extension!

4.2.5 DoS

The most compelling attack we were able to execute without special permissions was a denial-of-service attack, where our extension closes all open Chrome tabs, and *continues* to close any newly-opened Chrome tabs or windows. This makes it impossible to open chrome windows or use chrome at all without restarting Chrome in dev mode or otherwise removing the extension (in the case of Chroak, a restart in dev mode will allow you to untoggle the button and stop the DoS). Furthermore, since Chrome exits cleanly when DoSed in this way, there will be absolutely nothing to suggest to the user that the extension is responsible! While the “attacks” mentioned up to this point are at best minimally exploitable, this one is a plausible attack that any extension can carry out with no special permissions!

4.2.6 Phishing

Phishing is the only attack discussed here that uses a “special” permission (this attack uses the special permission `chrome.tabs`). In this attack, the extension waits until the user opens a new tab with a specific URL, and then silently redirects that tab to an arbitrary URL (in Chroak, this is the 6.857 homepage; a real attack would redirect it to a phishing page that looked like a clone of the actual URL the user tried to navigate to).

This attack is particularly compelling because it can redirect legitimate links from websites that are not under the extension’s control. Thus, it can target even power-users who responsibly inspect link’s URLs before clicking on them. Additionally, we can redirect any URL we desire. Thus, if a user receives an email from their bank and clicks a link on it, we can redirect that click from their actual bank to our own phishing site.

It is worth noting that redirecting URLs is possible without any “special permissions”. The `chrome.tabs` permission is needed solely to access to the URL in question, so we can selectively redirect. Otherwise, we could redirect the URL of every link the user clicks, but this would be too conspicuous to effectively phish.

5 Recommendations

The biggest issue we found with the Chrome extension security policy was that it intends to put control in the hands of the user, but it inhibits the user’s ability to actually make informed decisions. In light of this, we have made recommendations for both the users of Chrome extensions, and for Chrome’s policy makers to improve their security model. *We want to make it clear that we’re not reporting bugs or vulnerabilities in Google code. Rather, we’re reporting loopholes in existing policy that enable developers to perpetrate malicious activity under the guise of normalcy, something they should not be able to do.*

5.1 Chrome User Recommendations

The responsibility for vetting extensions before installation falls completely on the user. With the right permissions, extensions can do many malicious things, so it is not sufficient to assume an extension is safe just because it is on the Chrome Web Store (there is ostensibly some review/scanning of extensions, but it seems to be somewhat cursory compared to Firefox and Safari). Users should therefore actively investigate the extensions that they choose to download and decide whether the permissions requested represent an acceptable level of risk. Furthermore, we advise to never download a random extension from outside the Chrome Web Store unless they know and trust the developer - and maybe even only if they have access to the source code.

Users should be aware that there is still malicious activity available to developers even if the extension says it has no special permissions. Users should be especially aware of extensions that are overly-privileged and request the majority, if not all, of the permissions. It's important to notice when an extension requests a privilege that doesn't appear to be necessary for its purported function.

Users must also be aware that accepting one dialogue box may actually be allowing multiple privileges. For example, "Access your browsing activity" can be the warning for either the `chrome.tabs` or the `chrome.webNavigation` permission. Lastly, it's advised for users to always have the latest version of Chrome running, as out-of-date versions may have known vulnerabilities.

5.2 Chrome Policy Recommendations

We found that despite claiming to leave security decisions up to users, Chrome's security policy makes inherent judgements about what should and shouldn't raise flags for users. This leaves users inadequately informed due to the obscurity. Our recommendations for Chrome fall under two main ideas: 1) increased visibility and opportunities for users to be more informed, and 2) refined use policy for certain non-special permissions that actually are special.

5.2.1 Increased Visibility

We see no reason why extension manifests, which declare the permissions extensions use, cannot be public to users who are downloading the extension. Exposing the manifest will not guarantee that all users will look at it, but it gives especially conscientious users the opportunity to be informed.

Furthermore, warning messages should clearly indicate what each permission does. Right now, the messages are designed to be understandable by low-tech users; however, this actually inhibits users to make informed decisions, because there isn't enough given information. We recommend implementing an expanded warning view, where interested users can read more about exactly what they are accepting. This is the best way to ensure that users are adequately informed.

Lastly, "special" is a vague term that doesn't adequately define what permissions can actually do. It also doesn't allow for circumstances in which users might view some permissions as more or less special. There should be available documentation where users can read about what this actually means.

5.2.2 Refining Existing Policy

Throughout our explorations, we found that certain permissions should have refined policies. Specifically:

1. The `chrome.tabs` API should not be available without permission, as it can completely alter a user's browsing experience by closing windows and redirecting tabs. This power significantly extends the ability of normal web pages, and users should be informed. If anything, the ability to close and reroute tabs *should be its own permission*.
2. The `chrome.fontSettings` API can be used maliciously to great affect very easily. This permission should warn users that it can change fonts and thus the views of pages, as well as have a capped maximum font size.

Enforcing these constraints would protect against our three most compelling attacks: phishing, denial-of-service, and unreasonable font sizes.

6 Contributions

1. We discovered a number of loopholes in Chromes security policy that enable developers to create Phishing and Denial of Service attacks, among others, with “no special permissions” on the extensions.
2. We found that Chrome extensions give access to a large amount of unexpected system information, such as tab state, display setup, and external device connectivity, which users aren’t aware of.
3. We built a Chrome extension that showcases the policy loopholes we found, and the information we can gather.
4. We analyzed Chrome’s security policy and compared it to the policies of other web browsers, namely Firefox and Safari.
5. We recommended policy improvements for Chrome to implement that would create a more secure and transparent extension using experience for users.

We will also be submitting this paper to Chrome through their [vulnerability reporting system](#) with the hope that it will help them understand flaws in their security model and that they will implement our recommendations.

7 Appendix

Our code can be found on [Github](#), and our extension is live on the Chrome Web Store as [Chroak](#).

References

- [1] "Browser Information," Browser Information. [Online]. Available at: <http://www.w3schools.com/browsers/default.asp>. [Accessed: 17-Mar-2016].
- [2] L. Liu, X. Zhang, G. Yan and S. Chen, "Chrome Extensions: Threat Analysis and Countermeasures", 2012.
- [3] "Extensions Overview", Developer.apple.com, 2016. [Online]. Available: https://developer.apple.com/library/safari/documentation/Tools/Conceptual/SafariExtensionGuide/ExtensionsOverview/ExtensionsOverview.html#//apple_ref/doc/uid/TP40009977-CH15-SW2. [Accessed: 11- May- 2016].
- [4] "Mac App Store Review Guidelines - Apple Developer", Developer.apple.com, 2016. [Online]. Available: <https://developer.apple.com/app-store/review/guidelines/mac/#safari-extensions>. [Accessed: 11- May- 2016].
- [5] "Security best practices in extensions", Mozilla Developer Network, 2016. [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/Security_best_practices_in_extensions. [Accessed: 11- May- 2016].
- [6] "Signing and distributing your add-on", Mozilla Developer Network, 2016. [Online]. Available: <https://developer.mozilla.org/en-US/Add-ons/Distribution>. [Accessed: 11- May- 2016].
- [7] "Review Policies", Mozilla Developer Network, 2016. [Online]. Available: <https://developer.mozilla.org/en-US/Add-ons/AMO/Policy/Reviews>. [Accessed: 11- May- 2016].
- [8] "Manifest Version - Google Chrome", Developer.chrome.com, 2016. [Online]. Available: <https://developer.chrome.com/extensions/manifestVersion>. [Accessed: 11- May- 2016].
- [9] "Content Security Policy (CSP) - Google Chrome", Developer.chrome.com, 2016. [Online]. Available: <https://developer.chrome.com/extensions/contentSecurityPolicy>. [Accessed: 11- May- 2016].
- [10] "Declare Permissions - Google Chrome", Developer.chrome.com, 2016. [Online]. Available: https://developer.chrome.com/extensions/declare_permissions. [Accessed: 11- May- 2016].
- [11] "chrome.permissions - Google Chrome", Developer.chrome.com, 2016. [Online]. Available: <https://developer.chrome.com/extensions/permissions>. [Accessed: 11- May- 2016].