Wild Extensions: Discovering and Analyzing Unlisted Chrome Extensions

Aidan Beggs and Alexandros Kapravelos

North Carolina State University
{awbeggs,akaprav}@ncsu.edu

Abstract. With browsers being a ubiquitous, if not required, method to access the web, they represent a unique and universal threat vector. Browsers can run third-party extensions virtually invisibly in the background after a quick install. In this paper, we explore the abuse of browser extensions that achieve installations via suspicious methods. We scan the web for links to extension installations by performing a web crawling of the Alexa top 10,000 websites with recursive sub-page depth of 4 and leverage other tools to search for artifacts in the source code of webpages. We discover pages that have links to both listed and unlisted extensions, many times pointing to multiple different extensions that share the same name. Using this data, we were able to find 1,097 unlisted browser extensions ranging from internal directory lookup tools to hidden Google Docs extensions that pose a serious threat to their 127 million users.

Keywords: browser extensions · JavaScript · browser security

1 Introduction

Although they run largely in the background, extensions can be quite useful to the end user. Performing everything from letting users manage their email, to helping people manage their banking and crypto accounts, to blocking invasive ads (one of their most popular uses), extensions can ease the web browsing experience for many users. Upon installation, extensions will request the user for various permissions [9], however, many users have been conditioned to click "accept" without careful analysis of the permissions they are granting [10]. A small subset of the explicit permissions which some extensions request are as follows: "Read and modify your browsing history", "Access your browsing activity", "Read and modify all your data on all websites you visit", "Manage your apps, extensions, and themes", "Manipulate privacy-related settings", and "Access data you copy and paste." Some of these permissions grant considerably more power to extensions than many general users realize. For instance, the "Read and modify all your data on all websites you visit" permission may be marketed as necessary for ad-blocking software. In contrast, a malicious vendor could leverage this to hijack bank account information, but still force Chrome to render a bank account page that appears as normal. Research has been conducted to explore methods for classifying and eliciting malicious behavior from browser extensions [7,13].

Chrome extensions can broadly be broken up into two categories: listed and unlisted. Google has mandated that all extensions, listed or unlisted, be hosted on the Chrome web store, likely in an attempt to curb extension abuse [8]. As a result, we define "listed extensions" as those which may be found by a direct search, and "unlisted extensions" as those which only be accessed via a direct link, from an external advertisement or elsewhere. In this paper, we focus primarily on the security issues which use of unlisted extensions pose to users, in contrast to previous work done analyzing listed extensions and all extensions.

Chrome extension analysis is a category of applications for which for the most part, analysis has been largely overlooked. From the perspective of many, extensions appear to be nothing more than small plugins which help automate many users' day-to-day activities. In reality however, the potential for large-scale data and privacy breaches under the guise of innocent extensions is quite feasible. Although extensions must undergo vetting by Google before making it onto their web store, large-scale in-depth vetting of aforementioned extensions is an infeasible task given the vast scale of extensions submissions to the Chrome web store. In an attempt to increase the ease of filtering potentially malicious extensions, Google has cracked down on obfuscated code in extensions submissions, among other preventative measures [11]. Unfortunately, the potential for widespread abuse of user's privacy via installed extensions is much more nuanced than the typical patterns for malicious applications. The distinction between benign intended and malicious behavior is often blurred. As a result, making this distinction is not a process that can be easily automated, even with the policies which Google has in place regarding extension submission. Extension behaviors, such as exfiltration of user data may appear clearly malicious in certain contexts, however how does one distinguish between malicious tracking and an extension which is meant to verify that a user's pages are safe to visit? Clearly, even with manual analysis, this distinction is difficult to draw. Automating the process, even more so.

In this paper, we found 7,069 extensions by crawling the web, of which 1,097 were unlisted. On these extensions, we performed analysis to determine whether unlisted extensions are a feasible attack vector for such privacy abuses, and attempted to analyze the effectiveness of Google's attempts to reduce the prevalence of privacy abuse and malicious behavior in the extensions which are hosted on their web store. We focused simultaneously on the low-level methods which such extensions may use to circumvent traditional prevents, in addition to the high-level social engineering and advertisement campaign techniques that such extension creators may use to disseminate malicious extensions.

In summary, we frame our key contributions as follows:

- We discover novel suspicious practices that can lead the users to install extensions that are not discoverable on Chrome's webstore.

– We identified 7,069 extensions that are publicly linked from websites and 1,097 extensions that are not searchable via Chrome's webstore. Some of these unlisted extensions have millions of users, for a total of 127 million users among found unlisted extensions alone.
– We provide insights about unlisted extensions and show that they pose a significant threat to users, as we find many of them to use techniques including partial code obfuscation, advertisement injection, and various security vulnerabilities which may be exploited external to the extension.

2 Related Work

Our extension analysis work primarily leveraged Mystique [7], a technology for automating analysis of extension privacy leakages. The reasons for which Mystique was a good fit for our analysis were twofold: to correlate prevalence of privacy leakage with unlisted extensions, and because the Mystique platform captures a large number of web requests upon visiting various websites, which were used for later analysis. Mystique is available through a web interface [2], which we leveraged for this work.

Previous work on browser extension analysis has focused on developing novel techniques for analyzing extensions and detecting particular types of abuse coming from extensions. One line of work has focused on advertisement injections that the extensions might employ to monetize from their userbase [13,19,21]. Another threat that the users face from extensions is the increased fingerprintability that might occur when the adversary can identify the installed extensions of the users [16,18,17,15,12,14]. By having access to the visited pages of the user, extensions also pose a privacy threat, which has been explored in depth in the past [20,6,7]. Our work differs in that we aim to explore *how* users discover and install extensions. None of the previous work, except the internal tools that Google is using, had access to these unlisted extensions that we discovered in this paper. This affects the users security, as what has been studied before is not the complete picture of what the users are exposed when installing extensions. We hope that our work will motivate future researchers to expand their extension analysis to include also unlisted extensions.

3 Design

3.1 Crawling Setup

Before creation of our custom setup as described below, we attempted to find an existing crawling tool which met our requirements. Unfortunately, our specific needs were not met by any existing crawling framework. A large part of our crawling needs included the following:
– A large level of flexibility with regards to re-enqueuing sites
– Ability to run crawling in a highly parallelized setup

– Ability to evaluate Javascript on visited sites before scraping HTML (this was extremely important: if we simply scraped the HTML before evaluating the Javascript, Javascript obfuscation could be effectively used against our crawling setup. Because we evaluate Javascript, our browser will do a large part in de-obfuscating such Javascript by running it

As we found no such pre-existing framework that met all of the above-mentioned needs, we created our own. We used Docker to containerize workers utilizing an RQ [4] work queue to handle job distribution. Each worker made use of Selenium [5] to drive a PhantomJS [3] headless browser, visiting each page, checking for extensions, and re-queueing links found on the page to be queried later. Kubernetes was used to deploy these workers in parallel, with each worker communicating with a stateful MongoDB server hosted elsewhere, to save relevant data for later mining and analysis. The complete architecture of our crawling setup can be found in Figure 1.

In the Fall of 2018, we performed a crawl starting with the Alexa top 10k websites, using the architecture described in Figure 1. We recursively visited hyperlinks found on these pages until a sufficient depth was reached, resulting in approximately one million pages crawled. We weighted pages in other domains higher when evaluating the links to recursively search, to attain a breadth as high as possible. This was done in order to have the highest possible chance of finding unlisted extensions.

Sites with a high prevalence of extension-based advertisements were identified throughout the course of crawling. Later, these sites were visited by hand over the course of multiple sessions, in an attempt to leverage as many extensions as possible through manual analysis. The reason for which these extensions were searched for by hand was the obfuscation types of advertisements and prevalence of these extensions. Each page that held advertisements did so in a cyclical nature: a few advertisements appeared to be hosted for a few days, then after those days had passed, new advertisements were hosted. In addition, the level of user interaction and the variability in the type of user interaction which was required to navigate through these advertisements, and thus eventually chance upon an extension advertisement, was substantially higher than the level for which automation of advertisement interaction via Selenium would have been feasible. As a result, navigating to these pages every few days and finding the new extensions via advertisements that were then available on these pages was the preferred method of extension extraction from advertisements.

3.2 Unlisted Crawling Setup

Compilation of all extensions found via automated crawling and manually navigated advertisement areas, followed by removal of duplicate/dead extensions, yielded 7,069 extensions. Classifications of all found extensions as listed extensions (able to be found via a search in the Chrome Web Store) or unlisted extensions (only reachable via a direct link, and not reachable via a search by name) was achieved by using an automated script, again leveraging PhantomJS with a Selenium driver, in the same Docker environment as before. For every

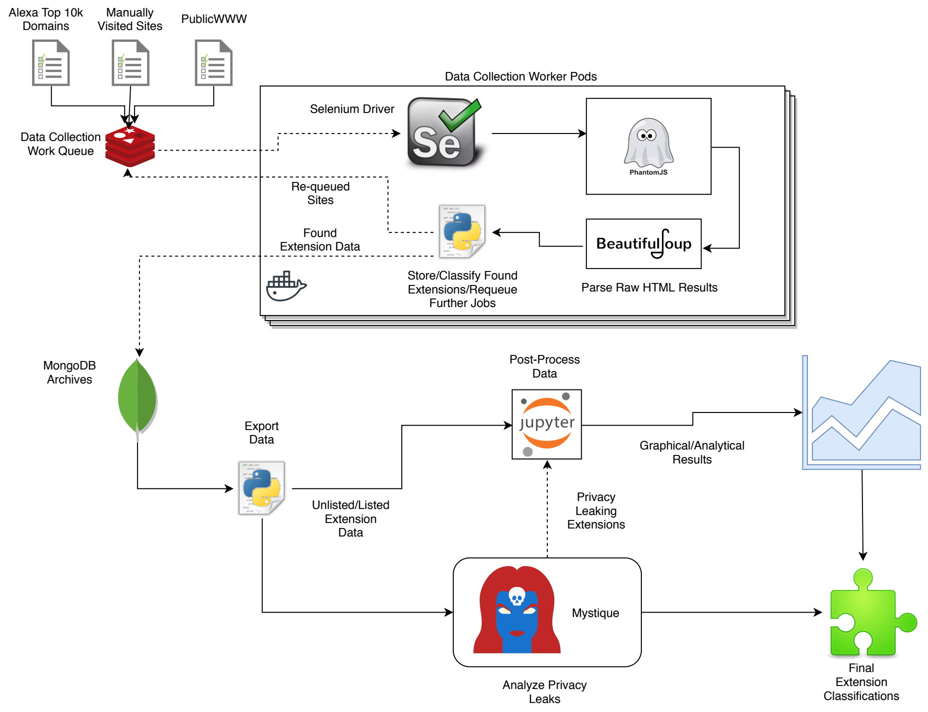

Fig. 1. The extension crawling system. Jobs are distributed automatically using a work queue. Docker containers running headless PhantomJS and controlled with a Selenium script load pages, search for extensions, and then re-queue further pages of interest.

extension that was previously found, the name was extracted from the extension page, and subsequently searched on the Chrome webstore. Search results were filtered to extensions, and precautions were taken to ensure that the results page loaded enough results such that the extension would load, if it were truly listed. Then, the results page was parsed with BeautifulSoup, searching to see whether the extension was returned in the search results, by link to the extension page. If the extension was not found in the results page upon searching for that extension, it was added to the list of unlisted extensions. In total, 1,097 unlisted extensions were found, out of the original 7,069 discovered extensions.

3.3 Duplicate Crawling Setup

Upon completion of crawling for all extensions and filtering for unlisted extensions, we additionally searched for "duplicated" extensions. Duplicated extensions were classified as extensions for which multiple extensions advertising the exact same thing, oftentimes with near-identical images, descriptions, and with inconsistent unlisted/listed statuses, were uploaded to the Chrome web store. One such example of duplicated extensions which we found is `Improve YouTube! (Open-Source for YouTube)`[1,2], with 12,671 users and 205,704 users respectively.

These types of duplicated extensions are considered as a possible attack vector for introducing malicious behaviors to users (perhaps introducing benign extensions listed on the store, and then pushing unlisted versions of the very same extensions, with near-identical descriptions/icons, via advertisements).

The methodology for classifying duplicated extensions is as follows: for every extension found in the overall group of extensions (including both listed and unlisted extensions), the same PhantomJS/Selenium combination navigated to the page, and found the name of the extension, the page being parsed via BeautifulSoup. As with the unlisted algorithm, the name of the extension was searched on the web store, and similar results were collected. The criteria for marking an extension as duplicated were simply finding whether there are one or more extensions with the same name in the returned results.

Consider two cases: extension A is found in the initial crawl, and searched on the web store as part of the duplicated classification. If A is listed, the search will yield extensions A, B, and C, assuming B/C have the same name as A. As a result, extensions A, B, and C will all be added to the duplicate list as part of the same group. If A is an unlisted extension however, the search will simply yield B and C. Despite this, extensions A, B, and C will all be added to the duplicate list. Thus, whether or not A is listed, special care will be taken to ensure that it gets added to the duplicate list, along with found extensions of matching names. In total, out of the original 7,069 extensions, we found 461 instances of these "duplicate extensions".

3.4 Extension Metadata Capture

After the groups of all extensions, groups of unlisted extensions, and groups of duplicate extensions were processed, we ran one last crawl of all extensions to capture relevant metadata on the found extensions, that would be used for later analysis. To perform this scraping, the same Docker container with PhantomJS and Selenium was used, visiting the extension page of all the found extensions. For each extension, the HTML of the store page was processed with BeautifulSoup [1], and the following list of attributes for each extension was extracted:

– Extension Name
– Extension ID/URL
– Extension Category
– Number of Extension Users

3.5 Offline Analysis

Upon completion of grouping extensions into groups by all extensions, unlisted extensions and duplicate extensions, in addition to scraping pertinent metadata

[1] `https://chrome.google.com/webstore/detail/lodjfjlkodalimdjgncejhkadjhacgki`
[2] `https://chrome.google.com/webstore/detail/bnomihfieiccainjcjblhegjgglakjdd`

on each extension, we performed offline analysis of the found extensions. First, all of the extension sources were downloaded locally, for closer evaluation. Primarily, analysis was performed using a Jupyter notebook running Python 3.6, for repeatability and modularity of analysis. The types of offline analysis performed were, in summary, as follows:

- Permissions requested, by our extension types
- User Distribution, by our extension types
- Mystique analysis of all found extensions, and cross-referencing between privacy leaks and our extension types
- Analysis of permissions requested vs. permission used, by our extension types
- Isolation of likely candidates for suspicious extension behavior, and per-extension source analysis

4 Evaluation

4.1 Summary

In the Fall of 2018, we visited approximately one million pages and saved relevant data pertaining to them in an attempt to search for unlisted, and potentially malicious, Chrome extensions. In our evaluation of these pages, we found approximately 7,069 extensions with cumulative installs totaling approximately 600,346,707 users. In addition, we found 1,097 unlisted extensions, with cumulative installs totaling over 127 million users.

For all evaluation of extensions, extensions with less than 100 users were not considered. We found that due to the fact that such extensions with such few users were not widely spread extensions, and that most extensions with such few users would not make a useful attack vector for an agent attempting to carry out malicious actions, they could be safely ignored for the purposes of this analysis, not to mention that they would likely skew analysis of data. In total, 1,097/7,069 of all found extensions were ignored for the purposes of extension analysis. An observant reader may notice that the number of ignored extensions here is exactly equivalent to the number of unlisted extensions we found (1,097). This coincidence exists independently from any sort of correlation between number of ignored extensions and found unlisted extensions: the set of found unlisted extensions and the set of ignored extensions are non-equivalent, in addition to being non-disjoint.

4.2 Extension Permissions

Every extension released on the Chrome web store includes a file called "manifest.json" in its root directory. This file contains key information about the extension, such as the name, description, other metadata, and as will be analyzed here, the permissions which the extension requests. One such example "manifest.json" file may look as follows:

```
{
"update_url": "https://clients2.google.com/service/update2/crx",


  "name": "Loadr - Daily Links",
  "description": "Your favorite bookmarks, only one click away.",
  "version": "1.0.7.1",
  "manifest_version": 2,

  "options_page": "options.html",

  "permissions": [
    "bookmarks",
    "contextMenus",
    "tabs",
    "storage",
    "chrome://favicon/",
    "alarms",
    "notifications"
  ],

  "icons": {
    "16": "img/icon.png",
    "48": "img/icon.png"
  },

  "browser_action": {
    "default_icon": "img/icon.png",
    "default_title": "Loadr - Daily Links"
  },

  "background": {
    "scripts": ["js/background.js"]
  }

}
```

Listing 1.1. Example manifest.json file

As one can see, formatted as a JSON file, the key ”permissions” contains every permission which this Chrome request needs, or at least claims to need, to function. A full list of permissions which may be requested and a brief description for each is available at `https://developer.chrome.com/apps/declare_permissions`. We parsed the ”manifest.json” file for all found extensions, and compiled the resulting permissions requested by extension type. Figure 2 shows the proportion of all extensions, by group (all, unlisted, duplicated), for the most popular permissions requested. This correlation was done in an attempt to see if unlisted/duplicated extensions requested certain permissions at higher rates of incidence than the permissions being requested among all extensions, to see if perhaps unlisted/duplicated extensions were requested more intrusive permissions during the course of potential privacy abuses.

Only permissions that were requested by at least 5% of extensions in at least one group are included. The permission proportions are sorted by the "all" group, but displayed in the same order for the other groups. The permissions displayed in Figure 2, in order by bar from left to right, are as follows:

```
['tabs', 'storage', 'contextMenus', 'webRequest', 'activeTab',
'notifications', 'http://*/*', 'https://*/*', '<all\_urls>',
'webRequestBlocking', 'cookies', 'unlimitedStorage',
'webNavigation', 'alarms', 'identity', 'management',
'background', 'clipboardWrite', 'webview']
```

Listing 1.2. Permissions from Figure 2

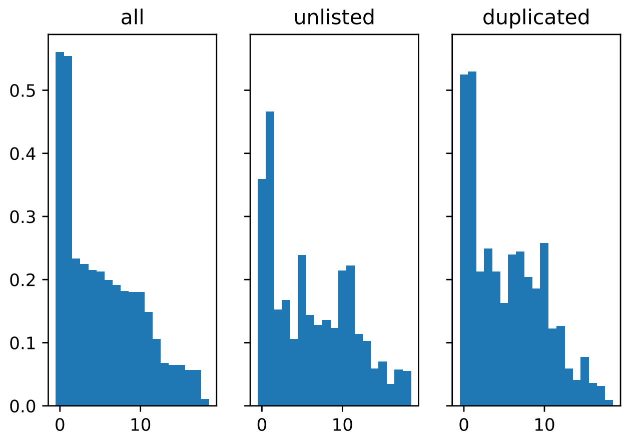

Fig. 2. Proportion of all extensions which request a given permission, by group.

4.3 User Distribution

In this section, we performed post-processing on the metadata found for each extension. This was done in an attempt to correlate number of users/user distribution among all, unlisted, and duplicated extensions, and to see if distribution tendencies vary by extension grouping.

Figure 3 shows the distribution of users across all extensions in each group of extensions for all users. The average number of users in each group was as follows:

– All: 118,505

– Unlisted: 128,961
– Duplicated; 158,114

In Figure 3, each graph is a CDF of the number of users of extensions. In other words, for any given x-value (number of users), the y-value represents the probability that any given extension will have less than or equal to this many users (again, filtered such that only extensions with 100 users or more are considered.) This representation gives us an easy way to visualize the distribution of a large number of users across a large amount of extensions, and to see potential patterns/similarities across data sets.

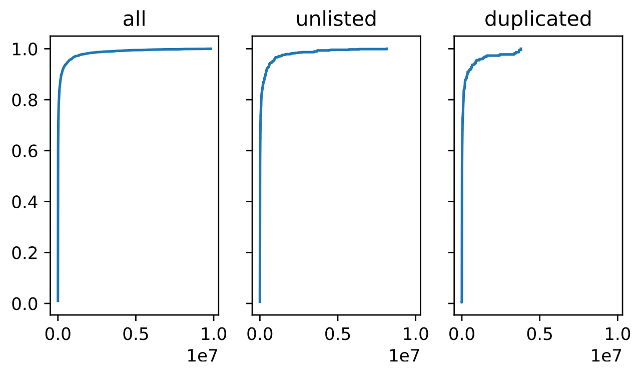

Fig. 3. CDF of number of users, by extension group, for all extensions.

As stated above, Figure 3 shows the CDF of users for all extensions. This figure shows that for extensions with a below-average number of users, there is very little difference between users in the "all", "unlisted", and "duplicated" groups. For extensions with an above-average number of users, this trend holds for the most part, with the exception of a deviation among found "duplicated" extensions. Such above-average user extensions appear to correlate closely for the "all" and the "unlisted" groups, however the "duplicate" group appears to have substantially fewer outliers on the upper end of the spectrum, as evidenced by the dip upwards and lack of trend line past approximately $0.5e7$ users. This may be due in part to the fact that there are certain Google extensions[3], which have a large number of users, are clearly legitimate, and would not show up in the duplicated extensions category. Beyond this however, the lack of outliers among the upper range for duplicated extensions would imply that duplicated extensions that do well perform more consistently. This may be due in part to the advertising campaigns for such extensions that we ran into during the course of our web crawling (the average duplicate extension is likely to do better, as a group willing to spend the time to create duplicate extensions is also likely

[3] An example would be this extension: `https://chrome.google.com/webstore/detail/save-to-google-drive/gmbmikajjgmnabiglmofipeabaddhgne?hl=en`

willing to put the time in to advertise it widely), although such campaigns do not entirely explain this phenomenon.

4.4 Mystique Analysis

In this section, we ran all found extensions through the Mystique web API [2], and ran analysis on the results. We leveraged the Mystique API via a simple script which uploaded all found extensions to the front-end site, interacted with the results via the REST API which we were given access to, and wrote the results to a local file for later offline analysis. In running this analysis, we attempted to correlate extension group with prevalence of privacy leak/violations, according to Mystique. As with before, extensions with under 100 users were not considered, as data may be skewed by such extensions. In Table 1, we detail the results of such analysis:

Table 1. Mystique results, by extension group

Extension Group	Flagged Extensions	Total Extensions	Flagged Percentage
All Extensions	160	3851	4.15%
Unlisted Extensions	25	750	3.33%
Duplicate Extensions	15	223	6.73%

At the $\alpha = 0.01$ level, there is statistically significant evidence to suggest that each group differs meaningfully from the original Mystique result of 2.13% of extensions flagged. In other words, there is a very low chance that this difference happened purely by chance, and thus, there is a high probability that this difference exists as a result of meaningful differences between these groups.

There are multiple factors to which this statistically significant difference may be attributed to. It is likely that the sample populations differ meaningfully between the extensions analyzed in the course of this paper and the extensions analyzed via Mystique. Mystique extensions were crawled via the web store, whereas our extensions were found organically in the wild. Thus, it makes sense that we ran into a higher incidence of privacy leaking extensions, as organic extensions in the wild are a much more effective attack vector than simply public-facing extensions on the web store.

4.5 Permission Usage Analysis

When a user installs an extension, they are notified of all pertinent permissions that an extension requests to have the ability to use, and must accept all permissions before the extension may be installed. At a later date, if the extension updates its permissions and required more/more powerful permissions the extension will be automatically disabled until the user verifies the new permissions as acceptable[4].

A commonly seen method for exposing users to malicious behavior among applications that auto-update is as follows:

1. Create a benign application, generally one that has a legitimate use
2. Distribute your application to a large user base through widespread advertising/other techniques
3. Update your application to include malicious code, thus pushing malicious behavior to all users who had previously downloaded the benign application

This verification poses a problem to anyone trying to carry out the above attack, as it will provide an explicit warning to the user that their extension is requesting more permissions. For many users who don't bother to check such warnings immediately, this will simply delay the timing with which they receive the updated extension. This delay and warning serves to reduce the effectiveness of a potentially malicious extension payload being carried out with this strategy. As a result, some extensions may preemptively request such permissions ahead of time, in preparation for use at a future time. If a user accepts permissions at install time, even if they are not currently used, the user will receive no warning/confirmation upon updating to a version of the extension that uses the aforementioned permissions.

In this section, we analyzed the prevalence of which extensions declared permissions that were never used across extension groups, in an attempt to see if this was a common occurrence, and to see if certain groups of extensions did this at a higher rate.

Table 2. Used/Declared percentage of extension permissions, by group

Permission	All Extensions	Unlisted Extensions	Duplicated Extensions
tabs	992 / 1013 = 97.9%	123 / 127 = 96.9%	57 / 59 = 96.6%
webRequest	386 / 496 = 77.8%	49 / 69 = 71.0%	24 / 35 = 68.6%
webNavigation	153 / 238 = 64.3%	23 / 43 = 53.5%	10 / 14 = 71.4%
downloads	67 / 79 = 84.8%	5 / 8 = 62.5%	3 / 4 = 75.0%
cookies	287 / 391 = 73.4%	63 / 83 = 75.9%	28 / 40 = 70.0%
identity	116 / 125 = 92.8%	20 / 24 = 83.3%	2 / 2 = 100.0%

As can be seen from Table 2, a significant number of extensions do not use all of the permissions for which they request access. As a result, any such extension could begin using these permissions at any time, for any purpose, malicious or otherwise. In addition, this can be done without notifying the user that the extent of permission utilization of the extension has changed. Such automatic update pushing, especially when extensions can request a broad swath of permissions

[4] `https://developers.chrome.com/extensions/permission_warnings#update_permissions`

upon installation, is an attack vector for which a malicious extension could very reasonably, and very suddenly, attack a large number of users.

4.6 Privacy Violation Case Study

Although in Section 4.4 we analyzed privacy leaks in a batch format, we decided to do some further processing on Mystique results to further identify signatures of privacy leaks among extensions. Mystique saves all of the web requests made in the course of evaluating an extension for privacy leaks and through its web-facing API we were able to download all such web requests made, for each of our extensions, in JSON format.

In our analysis, we found two signatures of XMLHttpRequests being made that leaked identifying information about the user. The signatures were as follows, with customized data replaced with placeholders:

```
https://api.amplitude.com/httpapi?api_key=(SHA_256_HASH)&event=%5B%7B%22user_id%22%3
    ↪ A%22(SHA_256_HASH)%22%2C%22event_type%22%3A%22%5BSession%5D%20
    ↪ PageVisit%22%2C%22event_properties%22%3A%7B%22host%22%3A%22(PAGE_URL)
    ↪ %22%2C%22url%22%3A%22https%3A%2F%2F(PAGE_URL)%2F%22%2C%22title
    ↪ %22%3A%22(PAGE_URL)%22%2C%22options_enableInject%22%3Atrue%2C%22
    ↪ options_enableGoogleProducts%22%3Afalse%2C%22options_enabledGoogleDiscounts
    ↪ %22%3Afalse%2C%22plugin_version%22%3A%222.1.3%22%2C%22plugin_source%22%3A
    ↪ %22sp%22%2C%22plugin_browser%22%3A%22ch%22%2C%22plugin_utm_source%22%3
    ↪ Anull%2C%22plugin_utm_campaign%22%3Anull%2C%22plugin_utm_term%22%3Anull%2
    ↪ C%22plugin_utm_content%22%3Anull%7D%7D%5D

https://api.mixpanel.com/track/?data=(B64_ENCODED_IDENTIFYING_INFORMATION)%3D
    ↪ %3D&ip=1&_=1524125554902
```

Listing 1.3. Examples of XMLHttpRequests which leak User Data

As can be seen in Listing 1.3, in the instance of the "api.amplitude.com" link, various identifying information about the user is passed in the URL. In addition, an API key and the last site that the user came from was passed as well. This signature showed up with some elements of the URL added/removed from request to request, however the majority of the request was similar enough to safely group it as likely a part of an extension from the same entity. Again, the instance of the "api.mixpanel.com" link which is shown in Listing 1.3 simply encodes some relevant information about the user (Chrome version, extension version, extension id, etc.) in base64 encoding before sending it as part of the URL. The most interesting observation about the api.mixpanel.com API specifically is that the extension id of the extension from which the request came from is included in the request. Although we found multiple URLs in our list that sent requests to the API, the fact that the extension id is passed alongside some other user data implies that there are multiple extensions from a single entity, all leaking data to this API.

In all, we found 29 instances of user privacy leaking simply between these two signatures, which implies there are a small handful of groups making numerous extensions which operate in very similar ways in the context of privacy-leaking behavior.

4.7 Extensions as a Backdoor

In terms of privacy violations and malicious behavior of extensions, there are two clear categories in which extensions fall:

– Extensions which intentionally leak and/or violate user's privacy
– Extensions which contain vulnerability which may lead to leaks/violations of user's privacy

All of the above analysis has pertained to the first category of extension: extensions which intentionally leak and/or violate user's privacy. This is what we would traditionally classify as a "malicious" extension. In this section, we will investigate a different type of path to perform malicious actions on a user: vulnerable extensions.

For most, the violations which "Extensions which intentionally leak and/or violate user's privacy" commit do not rise to the same level as those committed by viruses and what would be considered by most as "traditionally malicious". That being said, we still found such privacy violations of interest, for multiple reasons. Despite operating in the gray area of "malicious behavior", these extensions nonetheless exhibit a clear violation of not only Google's extension policies, but the user's expectation of privacy when they install such an extension. As a result, these extensions represent an important yet fine line between benign and malicious behavior: a line which may be crossed at any time among such extensions, and thus nonetheless poses a very credible security threat.

In the course of our investigation into duplicated extensions, we singled out a few extensions that met multiple criteria for closer source code analysis. The conditions which we searched for included:

– Relatively large user base
– Duplicated extension, according to the criteria listed in earlier sections
– Mystique privacy leaks in one or more instances of the duplicated extension
– Relatively small number of reviews, compared to its user base
– Preferably, obtained via an online ad campaign

Upon finding multiple such candidates, we zeroed in on a single extension to audit in depth, and to built a potential outline of the attack vector which any such extension may utilize. Through analysis of this extension, we came to several interesting conclusions. Previous versions of this extension appear to feature advertisement injection, and sending user data to external sources. This functionality appears to have been removed in more recent versions of this extension, perhaps signifying that Google's updated screening process for extensions has been an effective means of filtering out such extension behaviors.

The most interesting aspect of this extension, aside from the privacy breaches in previous versions of the extension, appears to be an (allegedly) unintentional security vulnerability which existed in all versions of this extension before March 2018. Found independently[5], this vulnerability would allow any webpage exploiting this vulnerability to implement a Universal Cross-site Scripting (UXSS) attack on any other webpage. This allows the attacker to perform an XSS attack on

[5] `https://bugs.chromium.org/p/project-zero/issues/detail?id=1555`

any website by tricking the extension into loading JavaScript code in a new tab with a URL of the attacker's choosing. Although this issue was fixed in March 2018[6], this bug was present dating all the way back to 2016, affecting over 400k users of this extension alone. Even if unintentional, this vulnerability represents a significant security risk, in addition to being incredibly difficult to detect.

This type of vulnerability opens up another potentially interesting avenue of attack for a malicious user. By either creating an extension with intentional but hidden security vulnerabilities or auditing the source code that is easily available for any extension on the Chrome web store, a malicious user can target massive swaths of users that visit their webpage. Even worse, they can carry out their exploit all while likely avoiding detection via obfuscated exploit code on their website, due to the fact that the extension does not have overtly malicious code itself.

The mitigation techniques for this type of exploit are not entirely straightforward. To mitigate this variety of attack, an auditing system that considers this line of attack is needed for popular extensions. Google does not publicly release any information about the nature of rigor that extensions go under before being accepted to the web store/before updates are accepted, but the time commitment required to audit such a large number of applications, many with largely minified code bases, approaches an insurmountable task.

4.8 Minification or Obfuscation?

In this section, we will consider Google's policy for what level of minification is accepted for web store extensions, to what level these requirements are upheld, and finally, the potential security implications of failure to uphold such requirements. As per Google's own requirements [11], obfuscated code is not permitted, and minified code is only permitted when it exclusively relies on the following techniques:

- Removal of whitespace, newlines, code comments, and block delimiters
- Shortening of variable and function names
- Collapsing the number of JavaScript files

This policy was put in place in an attempt to make manual code review easier. Unfortunately, the line between "minification" and "obfuscation" is blurry. One example of this blurry line can be found in Listing 1.4

```
if (!y) var O = a,
    T = [0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334],
  C = function(e, t) {
    return T[t] + 365 * (e - 1970) + O((e - 1969 + (t = +(1 < t))) / 4) - O((e - 1901 + t) /
       ↪ 100) + O((e - 1601 + t) / 400)
  };
P = function(e) {
    for (var t = "\"", n = 0, o = e.length, i = !S || 10 < o, r = i && (S ? e.split("") : e), a; n < o;
       ↪ n++) switch (a = e.charCodeAt(n), a) {
      case 8:
      case 9:
      case 10:
      case 12:
```

[6]https://bugs.chromium.org/p/chromium/issues/detail?id=827288

```
      case 13:
      case 34:
      case 92:
        t += A[a];
        break;
      default:
        if (32 > a) {
          t += "\\u00" + j(2, a.toString(16));
          break
        }
        t += i ? r[n] : e.charAt(n);
    }
    return t + "\""
}, N = function(e, t, n, o, i, r, a) {
  var d, s, p, c, l, u, _, f, g, y, I, S, T, A, R, M;
  try {
    d = t[e]
  } catch (e) {}
  if ("object" == typeof d && d)
    if (s = m.call(d), s != "[object Date]" || b.call(d, "toJSON")) "function" == typeof d.toJSON && (s != k && s != x && s != E || b.call(d, "toJSON")) && (d = d.toJSON(e));
    else if (d > -1 / 0 && d < 1 / 0) {
    if (C) {
      for (l = O(d / 864e5), p = O(l / 365.2425) + 1970 - 1; C(p + 1, 0) <= l; p++);
      for (c = O((l - C(p, 0)) / 30.42); C(p, c + 1) <= l; c++);
      l = 1 + l - C(p, c), u = (d % 864e5 + 864e5) % 864e5, _ = O(u / 36e5) % 24, f = O(u / 6e4) % 60, g = O(u / 1e3) % 60, y = u % 1e3
    } else p = d.getUTCFullYear(), c = d.getUTCMonth(), l = d.getUTCDate(), _ = d.getUTCHours(), f = d.getUTCMinutes(), g = d.getUTCSeconds(), y = d.getUTCMilliseconds();
    d = (0 >= p || 1e4 <= p ? (0 > p ? "-" : "+") + j(6, 0 > p ? -p : p) : j(4, p)) + "-" + j(2, c + 1) + "-" + j(2, l) + "T" + j(2, _) + ":" + j(2, f) + ":" + j(2, g) + "." + j(3, y) + "Z"
  } else d = null;
  if (n && (d = n.call(t, e, d)), null === d) return "null";
  if (s = m.call(d), s == "[object Boolean]") return "" + d;
  if (s == k) return d > -1 / 0 && d < 1 / 0 ? "" + d : "null";
  if (s == x) return P("" + d);
  if ("object" == typeof d) {
    for (A = a.length; A--;)
      if (a[A] === d) throw h();
    if (a.push(d), I = [], R = r, r += i, s == E) {
      for (T = 0, A = d.length; T < A; T++) S = N(T, d, n, o, i, r, a), I.push(S === w ? "null" : S);
      M = I.length ? i ? "[\n" + r + I.join(",\n" + r) + "\n" + R + "]" : "[" + I.join(",") + "]" : "[]"
    } else v(o || d, function(e) {
      var t = N(e, d, n, o, i, r, a);
      t !== w && I.push(P(e) + ":" + (i ? " " : "") + t)
    }), M = I.length ? i ? "{\n" + r + I.join(",\n" + r) + "\n" + R + "}" : "{" + I.join(",") + "}" : "{}";
    return a.pop(), M
  }
};
```

Listing 1.4. Minified browser extension code that is unreadable.

The snippet in Listing 1.4 appears to fit well into either category, it is not easily understandable. Although not clearly obfuscated, such extraneous functions and long convoluted equations approximates code obfuscation, as opposed to minification. In either case, the purpose of the code is not intuitive, not even after careful observation. Even if not objectively obfuscation, this code appears to go against the spirit of Google's policies, which are intended to make extension source code more understandable. Such extensions pose three options:

- Google does not consider such ”gray areas” of obfuscation as suspicious
- Google has not analyzed this extension well enough to notice this code
- Google genuinely has the resources to examine every instance of such extensions, and has verified them to be without malicious behavior

As discussed in Section 4.7, even intuitive extension code can house non-straightforward vulnerabilities that can lead to wide-scale exploitation of users.

In either scenario, and due to the large prevalence of such code snippets across extensions on the web store, lack of understanding about the extensions that Google themselves purports to verify is a potential source of vulnerability for any user who installs extensions from the web store.

5 Limitations

The largest limitations that we faced during the course of this research were as follows:

- Limitations on reproducibility
- Limitations on crawling speed and depth
- Difficulty automating the scraping of ad campaigns
- Automated analysis of found extensions

Reproducibility. As the internet contains many moving parts, even when controlling for as many initial conditions as possible, one will end up with drastically different results when crawling the web. We controlled for everything possible to maximize the reproducibility of our work, however random chance means that it is difficult to quantify the level to which we succeeded. Our goal was to cast a wide net in our search for extensions to analyze, and as a result (taken in conjunction with the rapidly changing nature of the web and of advertisement campaigns) it is difficult to quantify the degree of reproducibility for our crawler.
Crawling. For our experiment, we leveraged the maximum level of resources which were available. It is likely that a group with access to greater resources (even Google themselves) would be able to find a larger number of extensions in the wild, both listed and unlisted. As the relevance of crawled domains becomes lower and lower, the likelihood of finding prominent extensions decreases. Groups which wish to distribute their extensions to the largest possible user base, have it in their best interest to reach as many extensions for their extensions as possible. As the amount of sites crawled reaches an inflection point, the crawler receives diminishing returns from additional crawls, due to the poorer quality/user counts of extensions found thereafter. Furthermore, we are confident that the degree to which we ran our crawler and the quality/userbase of extensions which were found are a good representation of the extension population of interest.
Advertisements. During our experiment, we found many extensions of interest via advertisement campaigns. Many advertisement sites, although easily navigable by humans (intentionally so, to gain maximum traction), prove immensely difficult to navigate automatically via crawler. Most advertisement sites we found are set up such that for a human (who may easily pick up context clues about where and what to click), navigation through such sites is second nature.

Through techniques, such as heavy code obfuscation, automated scraping of such sites quickly becomes a monumental task. For the purposes of this research, visiting such sites manually every few days and clicking through a few dozen times proved sufficient to deliver interesting results. Nonetheless, this is a clear shortcoming of our crawler, and this topic would yield enough to merit a research paper of its own.

Automated Extension Analysis. One category of analysis which we performed was automated analysis of extensions which we found. This was done largely by either:

- Scraping all found extensions for relevant, well-defined data
- Using existing metadata to narrow down extensions of interest for manual investigation

Although utilizing both of these strategies yielded interesting and novel results, there is room for building on the work done here. Mystique provides a means of investigating potential privacy abuses in Chrome extensions, although does not provide the ability to classify purely malicious behaviours in extensions. As such, the results found here have great potential to be combined with Mystique as a means of classifying purely malicious extension behaviors.

6 Conclusion

In this paper we presented our work on discovering unlisted/suspicious Chrome extensions, and brief work on analyzing the found extensions. In total, we found 1,097 unlisted Chrome extensions, and 461 of the "duplicate but different" extensions mentioned above. Based on these findings, we analyzed these extensions in an attempt to identify suspicious behaviors. We also attempted to identify a correlation between whether or not an extensions was unlisted, and the types/intrusiveness of permissions which were requested by it. Finally, we compared the "duplicate but different" extensions across the population to discover differences in behaviors across these "duplicates" and to correlate these differences to malicious behaviors in the extensions themselves.

As the web becomes increasingly accessible to the general public, the number Chrome extension users becomes larger as well. In this paper, we analyzed the potential routes that a malicious extension could take to exploit a user's privacy. In addition, we discussed the steps which Google has taken to prevent such attacks. We have also assessed the effectiveness of these various countermeasures, and potential responses from a malicious agent. We segregated the extensions identified via crawling into three categories (all, unlisted, duplicated), and performed various analyses comparing each category, including leveraging the Mystique platform to analyze privacy leaks across extension types.

In addition to our statistical comparisons, we have provided two case studies of specific points of interest with extensions. The first case study analyzed the efficacy of Google's new anti-obfuscation policy, and the degree to which it is enforced and is effective. Secondly, we analyzed the potential for the use of extensions as a dormant backdoor, which may be later leveraged by a site taking advantage of the backdoor to cause harm or to exploit the user.

Although we have discussed multiple possible resolutions available to Google to reduce the incidence of such exploitative techniques, educated and informed user action is an equally valuable tool in curbing this malicious behavior. By debunking the idea that extensions are inherently safe because Google hosts them and helping the user realize that the permissions which an extension requests are indeed quite important and making such information more easily readable for users, Google can increase the power of their users to take control of their own privacy. As a result, they can empower users to protect themselves from malicious extension behavior.

It is our hope that our research into the extension ecosystem will motivate others to explore this under-analyzed field, which has potential for large impacts should a malicious agent target it.

Acknowledgements

We would like to thank our shepherd Kapil Singh and the anonymous reviewers for their insightful comments and feedback. This work was supported by the Office of Naval Research (ONR) under grant N00014-17-1-2541, by DARPA under agreement number FA8750-19-C-0003 and by the National Science Foundation (NSF) under grant CNS-1703375.

References

1. Beautiful Soup: We called him Tortoise because he taught us. `https://www.crummy.com/software/BeautifulSoup/`
2. Mystique Extension Analysis Engine. `https://mystique.csc.ncsu.edu/`
3. PhantomJS - Scriptable Headless Browser. `http://phantomjs.org/`
4. RQ: Simple jobs queues for Python. `http://python-rq.org/`
5. Selenium - Web Browser Automation. `https://www.seleniumhq.org/`
6. Aggarwal, A., Viswanath, B., Zhang, L., Kumar, S., Shah, A., Kumaraguru, P.: I spy with my little eye: Analysis and detection of spying browser extensions. In: Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P) (2018)
7. Chen, Q., Kapravelos, A.: Mystique: Uncovering information leakage from browser extensions. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2018)
8. Google: Alternative Extension Distribution Options - Google Chrome. `https://developer.chrome.com/apps/external_extensions`
9. Google: Chrome Permission Warnings. `https://developer.chrome.com/apps/permission_warnings`
10. Google: Declare Permissions and Warn Users - Google Chrome. `https://developers.chrome.com/extensions/permission_warnings#permissions_with_warnings`

11. Google Security Blog: Trustworthy Chrome Extensions, by Default. `https://security.googleblog.com/2018/10/trustworthy-chrome-extensions-by-default.html`
12. Gulyas, G.G., Some, D.F., Bielova, N., Castelluccia, C.: To extend or not to extend: On the uniqueness of browser extensions and web logins. In: Proceedings of the 2018 Workshop on Privacy in the Electronic Society. WPES'18 (2018)
13. Kapravelos, A., Grier, C., Chachra, N., Kruegel, C., Vigna, G., Paxson, V.: Hulk: Eliciting Malicious Behavior in Browser Extensions. In: Proceedings of USENIX Security Symposium (2014)
14. Sanchez-Rola, I., Santos, I., Balzarotti, D.: Extension breakdown: Security analysis of browsers extension resources control policies. In: Proceedings of USENIX Security Symposium (2017)
15. Sjösten, A., Van Acker, S., Sabelfeld, A.: Discovering browser extensions via web accessible resources. In: Proceedings of the ACM on Conference on Data and Application Security and Privacy (CODASPY) (2017)
16. Starov, O., Nikiforakis, N.: Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions. In: Proceedings of the 26th International World Wide Web Conference (WWW) (2017)
17. Starov, O., Nikiforakis, N.: Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In: Proceedings of the International Conference on World Wide Web (WWW) (2017)
18. Starov, O., Nikiforakis, N.: XHOUND: quantifying the fingerprintability of browser extensions. In: Proceedings of the IEEE Symposium on Security and Privacy (2017)
19. Thomas, K., Bursztein, E., Grier, C., Go, G., Jagpal, N., Kapravelos, A., Mccoy, D., Nappa, A., Paxson, V., Pearce, P., Provos, N., Abu Rajab, M.: Ad Injection at Scale: Assessing Deceptive Advertisement Modifications. In: Proceedings of the IEEE Symposium on Security and Privacy (2015)
20. Weissbacher, M., Mariconti, E., Suarez-Tangil, G., Stringhini, G., Robertson, W., Kirda, E.: Ex-ray: Detection of history-leaking browser extensions. In: Proceedings of the ACM Annual Computer Security Applications Conference (ACSAC) (2017)
21. Xing, X., Meng, W., Lee, B., Weinsberg, U., Sheth, A., Perdisci, R., Lee, W.: Understanding malvertising through ad-injecting browser extensions. In: Proceedings of the International Conference on World Wide Web (WWW) (2015)