What is in the Chrome Web Store?

Sheryl Hsu
Stanford University
sherylh@stanford.edu

Manda Tran
Stanford University

Aurore Fass
Stanford University, CISPA Helmholtz Center for Information Security
fass@cispa.de

ABSTRACT

This paper is the first attempt at providing a holistic view of the Chrome Web Store (CWS). We leverage historical data provided by ChromeStats to study global trends in the CWS and security implications. We first highlight the extremely *short life cycles of extensions*: roughly 60% of extensions stay in the CWS for one year. Second, we define and show that *Security-Noteworthy Extensions (SNE)* are a significant issue: they pervade the CWS for years and affect almost 350 million users. Third, we identify *clusters of extensions with a similar code base*. We discuss how code similarity techniques could be used to flag suspicious extensions. By developing an approach to extract URLs from extensions' comments, we show that extensions reuse code snippets from public repositories or forums, leading to the propagation of dated code and vulnerabilities. Finally, we underline a critical *lack of maintenance in the CWS*: 60% of the extensions in the CWS have never been updated; half of the extensions known to be vulnerable are still in the CWS and still vulnerable 2 years after disclosure; a third of extensions use vulnerable library versions. We believe that these issues should be widely known in order to pave the way for a more secure CWS.

CCS CONCEPTS

• **Security and privacy → Web application security**; **Browser security**.

KEYWORDS

Browser Extension, Chrome Web Store, ChromeStats, Security-Noteworthy Extension, SNE, Malware, Policy Violation, Vulnerability, Life Cycle, Maintenance, Vulnerable Library, Code Similarity

ACM Reference Format:
Sheryl Hsu, Manda Tran, and Aurore Fass. 2024. What is in the Chrome Web Store?. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3634737.3637636

1 INTRODUCTION

Browser extensions provide additional functionality and customization for browsers. The most popular desktop browser Chrome (with a market share of 66% [70]) has almost 125k extensions, totaling over 1.6 billion active users [6]. Unfortunately, due to their often specialized or privileged capabilities, browser extensions can either be a tool or a target for attackers. Specifically, attackers are developing *malicious extensions* to, e.g., spread malware via malvertising [1], track users [72], spy on users [14], or steal credentials and other sensitive information [22]. At the same time, other extensions have inherently benign functionalities but *contain vulnerabilities* which, if exploited, lead to, e.g., universal cross-site scripting or leaking of sensitive user data [41, 64, 74]. Finally, merely using extensions (independent of whether they have any known flaws) represents a *privacy risk* for Web users. For example, it is possible to infer the set of extensions a user has installed, by observing side effects some extensions induce (browser extension fingerprinting) [50, 59, 60, 62, 63, 67, 69]. This enables an attacker to track users across websites or infer sensitive information about them [46].

To mitigate these issues, browser vendors review extensions prior to publication, e.g., Google engineers vet extensions before their deployment in the Chrome Web Store (CWS) [32]. Despite this vetting process and a decade of work on securing extensions, malicious, vulnerable, and fingerprintable extensions are still found in the CWS [41, 54]. We argue that a potential solution to this issue requires a comprehensive understanding of the underlying ecosystem of browser extensions. Indeed, and perhaps surprisingly, very little is currently known about what lies in browser extension galleries.

This paper provides a holistic view of the browser extension landscape within the CWS. We focus on the CWS because Chrome is the most popular browser, and Chrome extensions are built using the WebExtensions API–a cross-browser technology compatible with Firefox, Opera, Microsoft Edge, etc [56].

We begin our analyses by investigating overall trends in the CWS, along with potential security problems that could arise from having such a big and diverse code base. We first highlight unexpected volatility in terms of extensions in the CWS, e.g., only 60% of extensions are available for one year. Second, we define and investigate "Security-Noteworthy Extensions" (SNE): we analyze malware-containing, policy-violating, and vulnerable extensions. We find that these SNE are a significant problem: over 346 million users installed a SNE in the last 3 years (280M malware, 63M policy violation, and 3M vulnerable). In addition, these extensions are staying in the CWS *for years*, making thorough vetting of extensions and notification of impacted users all the more critical. Third, we uncover thousands of clusters of similar extensions, and we show that investigating extension codes for similarities may enable us to identify unknown SNE. Subsequently, we develop an approach to extract URLs from extensions' comments and attribute instances of code reuse to these cited URLs. We show concrete evidence that extensions reuse code from public repositories or forums directly, and we exemplify how code reuse leads to the propagation of vulnerabilities. Fourth, we show that extensions are globally not maintained: 60% of the extensions in the CWS have never been updated. This has

ASIA CCS '24, July 1–5, 2024, Singapore, Singapore

ACM ISBN 979-8-4007-0482-6/24/07
https://doi.org/10.1145/3634737.3637636

direct security consequences, e.g., half of the vulnerable extensions discovered in 2021 [41] are still in the CWS and still vulnerable in 2023. Equally worrisome is the fact that developers continue to use deprecated tools, even when (more) secure alternatives are available. For example, a third of extensions use JavaScript libraries with known vulnerabilities, impacting almost 500 million users.

To sum up, our paper makes the following contributions:

- We analyze overall trends in the CWS and highlight the exceptionally *short life cycles of extensions*;
- We define and investigate *"Security-Noteworthy Extensions" (SNE)*. We show that these are a significant issue, affecting hundreds of millions of users and staying in the CWS for years;
- We discover *clusters of extensions with a similar code base* and highlight security implications;
- We characterize a critical *lack of maintenance in the CWS* and discuss the pervasiveness of vulnerable extensions in the CWS.

We are confident that our findings will guide future research and pave the way for a more secure CWS.

2 BACKGROUND: BROWSER EXTENSIONS

Browser extensions are third-party programs that users can install to extend and customize their browser functionality by, e.g., adding ad-blocking capabilities or checking grammar. In this section, we first give an overview of extension architecture and permission system. Then, we describe the main components of extensions.

Overview — Browser extensions are zipped bundles of, e.g., HTML, JavaScript, or CSS files, stored in CRX files. Every extension requires a JSON-formatted file, named `manifest.json` [26], which specifies important information such as the extension's most important components and the extension's permissions. In fact, to use most Chrome APIs, e.g., `activeTab`, `downloads`, or `storage`, an extension must declare the corresponding permissions in its manifest [24]. There are three versions of the manifest [33]. Manifest V1 has been deprecated since 2012. Manifest V2, while deprecated, is still used by almost 62% of extensions as of July 2023[5]. It is currently unclear when Chrome will stop providing support for this version, as the deadline has been extended several times [27]. Manifest V3 was released in November 2020 to improve the security, privacy, and performance of Chrome extensions [25]. For example, manifest V3 prevents extensions from downloading external resources; instead, all resources must be bundled within the extension package.

Main Components — The source code of browser extensions is split into several components, which are specified in the extension manifest file. The core logic of an extension is implemented through *service workers* (*background scripts* for manifest V2 extensions), which run independently of the lifetime of a web page and do not need any user interactions [34]. An extension can inject *content scripts* to run in the context of web pages, similarly to the scripts web pages directly load. While content scripts can use standard DOM APIs to read and modify web pages, they live in an "isolated world" to avoid conflicting with variables defined in web pages [29]. An extension can propose *UI* or *option pages* to enable users to customize their extension's behavior [31]. Finally, an extension can expose *Web Accessible Resources (WARs)*, i.e., files that can be accessed by web pages or other extensions.

Category	# extensions – metadata collected	# extensions – code collected	When collected
SNE	26,014	16,377	
- Malware containing	10,426	6,587	July 5, 2020 – May 1, 2023
- Privacy violating	15,404	9,638	July 5, 2020 – May 1, 2023
- Vulnerable	184	152	March 16, 2021
Benign extensions	226,762	92,482	July 5, 2020 – May 1, 2023

Table 1: SNE and likely-benign extensions collected

3 EXTENSION COLLECTION AND ANALYSIS

To study global trends in the CWS and corresponding security implications, we first need to collect a comprehensive set of extensions. Since Google does not archive browser extensions that used to be in the CWS but are now deleted, we use ChromeStats [6] to collect historical data. We focus on likely-benign (i.e., not currently known to have any security or privacy issues) and security-noteworthy extensions (SNE): malware-containing, privacy-violating, and vulnerable extensions. In this section, we first discuss the collection of our datasets and then analyze overall trends in the CWS. Finally, we dissect the life cycles of extensions in the CWS.

3.1 Extension Collection

ChromeStats — ChromeStats [6] provides historical data since July 5, 2020 for browser extensions that are or were in the CWS. ChromeStats developers designed a crawler to automatically collect extensions and extract their metadata from the CWS once a day. Besides the source code, they also archive, e.g., extension id, name, category, last update, number of users, or permission information. To conduct longitudinal experiments and study global trends in the CWS, we collected all extensions from the CWS between July 5, 2020 and February 14, 2023. In this setting, we do not discriminate between benign and security-noteworthy extensions but consider all available extensions, independently of their intent.

Security-Noteworthy Extensions (SNE) — The CWS contains what we call "security-noteworthy extensions" (SNE). We define SNE as extensions known to either:

- **contain malware**: such extensions aim to, e.g., steal user-sensitive data, track users, spy on them, or propagate malware [14, 22, 72];
- **violate the CWS policies**: all extensions submitted to the CWS have to comply with the developer program policies [23, 28];
- **contain vulnerabilities**: if exploited, such vulnerabilities could lead to, e.g., universal XSS or user data exfiltration [41, 64, 74].

From ChromeStats, we extract a list of extensions that were removed from the CWS for containing malware or violating policies (such extensions were analyzed and flagged accordingly by Google engineers). We collected malware-containing and privacy-violating extensions that were removed from the CWS between July 5, 2020 and May 1, 2023. To retrieve known vulnerable extensions, we contacted Fass et al. [41] who designed an advanced data flow analysis technique to automatically uncover vulnerabilities originating from PostMessages exchanged between web pages and browser extensions. We got access to their dataset of 184 vulnerable extensions (including the corresponding versions), which they collected on March 16, 2021. To have a point of comparison, we also take the set of all 226,762 benign extensions that appeared in the CWS between

July 5, 2020 and May 1, 2023. In this paper, we use the term *benign* extensions to refer to extensions that are not known to be SNE. As summarized in Table 1 (columns 1, 2, and 4), we retrieved metadata for 10,426 malware-containing extensions, 15,404 policy-violating, 184 vulnerable, and 226,762 benign extensions.

Extension Unpacking — We used the ChromeStats API to download the source code of extensions, in the format of CRX files. As highlighted in Table 1 (column 3), we could download the source code of 16,377 out of 26,014 SNE (the rest was not available for download on ChromeStats). Given the large size of the benign extension set, we chose to only download benign extensions from 2023. After downloading the extensions, we unpacked the corresponding CRX files [16] to extract content scripts and service workers / background scripts. We chose to compile all content scripts and all background scripts into a single content and background file, respectively, similarly to prior work [41].

3.2 Overall Trends

As of February 2023, there are 124,094 extensions in the CWS. We first discuss what users install extensions for, as well as the evolution of the number of active users and extensions over time.

What do Users Install Extensions for? — As a proxy to investigate what users install extensions for, we consider the categories of the most popular extensions. In the CWS, extensions are organized into categories which users can search through. The most popular category is "Productivity" with a share of 41%. "Productivity" contains a wide variety of extensions from translation tools to personal dashboards to PDF generators. Of the 10 extensions with the highest number of active users (over 10M; as of July 2023), 9 are from the "Productivity" category; among these, 5 are content blockers (e.g., "AdBlock", "Adblock for Youtube", or "AdGuard"), 1 enables web page layout customization ("Tampermonkey"), 1 performs grammar checks ("Grammarly"), 1 offers translation options ("Google Translate"), and 1 adds PDF editing capabilities ("Adobe Acrobat"). The 10th extension is from the "Shopping" category ("honey"); it looks for and applies digital coupons while shopping online.

Number of Users — We now examine the evolution of the number of extension users over time. The user count was scraped using the CWS API [37]. This API provides exact user counts for all extensions, except for extensions with over 10 million users, which are just listed as having 10,000,000+ users. According to our email exchange with Chrome Web Store Developer Support, the "number of users" displayed on the CWS for a given extension corresponds to "the number of Chromes with the extension installed that are active and checking in to [their] update servers over the previous seven days only, not for all time. It is not equal to the sum of historic installs minus the sum of historic uninstalls". For instance, this means that if a user with an extension installed did not turn on their computer for a month, they would not be counted as an active user for that month. Given this definition, we can therefore expect fluctuations in the number of active users over time. We observe in Figure 1 (green curve) that the number of extension users largely fluctuates over time. The exact reasons for this phenomenon are currently unclear, but we can make some observations. Overall, we see a large drop in the number of users during the holiday seasons

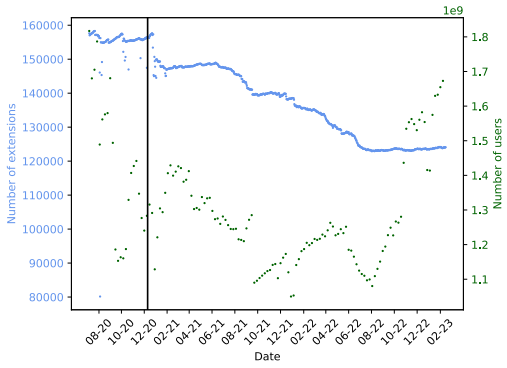

Figure 1: Number of extensions (blue) in the CWS on a given date and weekly average of the number of active users across all extensions (green) between July 2020 and February 2023– The black vertical line denotes the announcement of Manifest V3 on Chromium Blog on December 9, 2020

(the last two weeks of December and the first week of January), which can be explained by people not using Chrome for over seven days as they spend time away from work. We also observe an overall dip in the number of users during the summer months, which could be caused by increased travel and idle school devices.

Takeaway There is unexpected volatility in the number of active users in the CWS. We caution researchers to not overly emphasize precise user counts when measuring the impact of work. We also recommend developers take this into account when reporting or comparing user counts over time.

Regarding the average number of users per extension, we find that the vast majority of extensions have a small user base: 64.22% of extensions have less than 100 users and 17.51% have between 100–1k users. The higher the user count, the fewer extensions there are: 9.70% of extensions have 1k–10k users, 4.17% 10k–100k, 1.13% 100k–1M, and 0.27% over 1M. Overall, extensions with a large user base are the exception rather than the rule. This is problematic from a privacy perspective, as the crowd anonymity of extensions with few users is very small, which can ease user tracking and deanonymization if those extensions can be fingerprinted [46].

Number of Extensions — As shown in Figure 1 (blue curve), the number of extensions is decreasing over time; notably with an initial 157k extensions in July 2020 vs. 124k in February 2023. We observe a large drop in the number of extensions in December 2020–likely related to Chrome's November 9th announcement of Manifest V3 [36] (vertical black line). Note that there were some (now resolved) bugs in the ChromeStats crawler when the developers released it in July 2020–as evidenced by some isolated points in the first 6 months of release and confirmed by the developers.

We observe that, on average, 3,775 extensions are removed from the CWS every month and another 2,687 are added. This information leads us to believe that, depending on the point in time of measurement, researchers may analyze different extensions; thus report on different results.

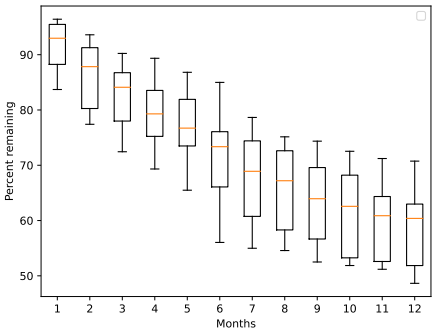

Figure 2: Percentage of extensions still in the CWS on the x^{th} month after having been added–We compute this for extensions first added between January–December 2021 and still in the CWS on the x^{th} following month(s), where $x \in [\![1, 12]\!]$

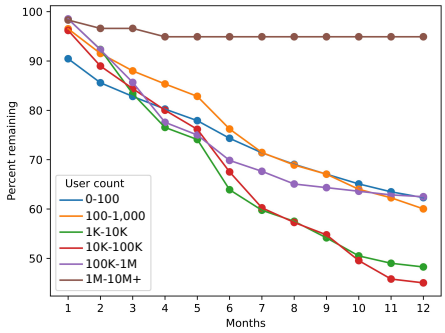

Figure 3: Average percentage of extensions still in the CWS on the x^{th} month after having been added, grouped by an extension last recorded user count

Takeaway Every month thousands of extensions are added or removed from the CWS. Given those fluctuations, we encourage researchers to investigate multiple points in time and evaluate the reproducibility of their findings before generalizing their claims.

3.3 Extension Life Cycle

Very little is currently known about the lifetimes of extensions in the CWS. It is important to investigate, as it sets guidelines for how long study results are valid. Analyses should be run regularly to ensure that research results reflect the current state of the CWS.

We study extension life cycles by computing the percentage of extensions that remain in the CWS after a given number of months. To obtain a full year of data, we perform this analysis on extensions that were added to the CWS between January and December 2021, and we compute the percentage of those extensions that were still in the CWS after $x \in [\![1, 12]\!]$ months. Figure 2 represents the percentage of extensions still in the CWS on the x^{th} month after having been added, with $x \in [\![1, 12]\!]$. For example, we consider extensions first added in January 2021 and still in the CWS in February 2021 ($x = 1$), March ($x = 2$), [...], and January 2022 ($x = 12$). At the same time, we also study extensions first added in February 2021 and still in the CWS in March 2021 ($x = 1$), April ($x = 2$), [...], and February 2022 ($x = 12$). We iterate this process till December 2021 (extensions added to the CWS) and calculate how many were in the CWS after 1–12 months (i.e., from January 2022 to December 2022). This way, we have 12 data points for each 12 x values (x-axis), which we represent with a box plot. As shown in Figure 2, we find that after $x = 12$ months, only 51.86–62.98% of extensions are still available in the CWS (median of 60.39%). Even after $x = 3$ months, we observe a drop, with only 78–86.75% of the extensions still available in the CWS (median of 84.11%).

Intuitively, we expect extensions with a larger user base to remain in the CWS for a longer period, as it takes time to build up a large user base, i.e., developers are more likely to continue maintaining those extensions. In Figure 3, we show the average percentage of extensions still in the CWS on the x^{th} month after having been added, grouped by last recorded extension user count (i.e., so that extension groupings do not change over time). We see that, on average, 94.91% of the most popular extensions (1M–10M+ users) stay in the CWS for at least a year. However, for the rest of the extension groupings, there is no clear trend. For example, the two groups with the highest percentage of extensions remaining after one year are extensions with less than 100 users (62.31%) and 100k–1M users (62.5%). Upon further investigation, we find that there are numerous instances of popular extensions appearing in the CWS, being removed, then reappearing in the CWS. This pattern can be observed for extensions violating the CWS policies: they are removed by Google but can be submitted again after the violation has been fixed [35]. In addition, there may be larger variability in the higher user count categories due to different sample sizes; e.g., 19,528 extensions have 100 users or less[1] vs. 272 have 100k–1M users and 59 have 1M–10M+ users.

Takeaway Surprisingly, we observe that extensions have a very short life cycle in the CWS, e.g., only 51.86–62.98% of extensions are still available after one year. We also observe instances of popular extensions that are added to the CWS, removed, and re-added; likely due to policy violations and subsequent fixes. Thus, analyses on the CWS should be run regularly to ensure that research results reflect the current state of the CWS.

4 SECURITY-NOTEWORTHY EXTENSIONS

Previously, we focused on overall trends in the CWS, independently of whether the extensions were benign or security noteworthy. We now dive into the differences between benign, malware-containing, policy-violating, and vulnerable extensions. Specifically, we compare the number of days in the CWS, user counts, extension ratings, developer patterns, and permissions of these 4 groups of extensions.

4.1 Number of Days in the CWS

We first analyze the average number of days a benign extension vs. a SNE stays in the CWS. For every malware-containing and policy-violating extension, we count the number of days between the date of its last update and the date of its removal. For vulnerable extensions, we count the number of days the reported vulnerable

[1]Note that we conducted this analysis on extensions added between January and December 2021, not on our full extension dataset.

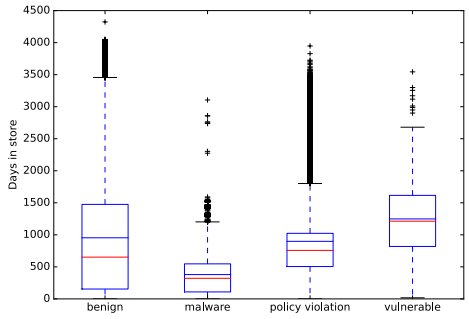

Figure 4: Number of days a benign, malware-containing, policy-violating, or vulnerable extension stays in the CWS– The blue line denotes the means and the red one the median

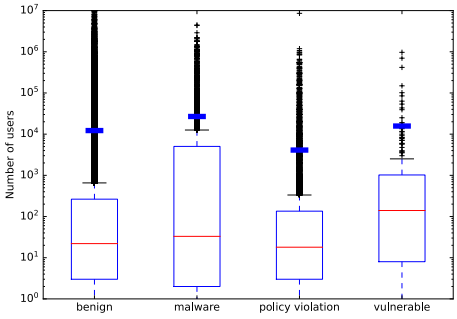

Figure 5: Number of users with a benign, malware-containing, policy-violating, or vulnerable extension installed–The blue tick denotes the means and the red line the median

versions stayed in the CWS (as of May 1, 2023). Of course, it is possible that an extension had security or privacy issues *before* the last update or continues to be vulnerable even after being updated. We acknowledge that our observations are a *lower bound* of the number of days SNE stay in the CWS. As shown in Figure 4, SNE stay in the CWS for an average of 380 days (malware containing) to 1,248 days (vulnerable). This is extremely problematic, as such extensions put the security and privacy of their users at risk *for years*. Interestingly, benign extensions tend to stay in the CWS for less time than vulnerable extensions (1,152 days, with a median of 780 days vs. 1,213 for vulnerable extensions). While the sample sizes are different (226,762 benign vs. 184 vulnerable extensions), there are more fluctuations within benign extensions. Equally worrisome are some outliers. In particular, we found one malware-containing extension that stayed in the CWS for 3,105 days (8.5 years!). This extension, "TeleApp", was last updated on December 13, 2013 and was found to contain malware on June 14, 2022. Similarly, the extension "No More Holidays" was last updated on May 17, 2012 and was found to have a policy violation only on March 9, 2023 (after almost 11 years in the CWS!).

Takeaway Security-noteworthy extensions stay in the CWS *for years*, meaning that their user base can stay at risk *for years*. It is currently unclear why such extensions are not immediately detected by Chrome's vetting system and why SNE stay in the CWS for years even after disclosure (e.g., case of vulnerable extensions).

4.2 Number of Users

Next, we investigate how large the user bases of SNE are. To this end, we consider the number of users they had when removed from the CWS or, if not yet removed, May 1, 2023 (when we conducted this analysis). As a comparison, we also consider the last recorded number of users for benign extensions. We represent the number of users of benign extensions and SNE in Figure 5. As expected, the median number of users is very low: between 18 (policy-violating and benign extensions) and 140 (vulnerable extensions); remember that 65% of extensions have less than 100 users (Section 3.2). However, there are some outliers with extremely large user bases so that, on average, benign extensions have 11k users, policy-violating 4k, vulnerable 16k, and malware-containing extensions 27k. For example, the extension "Casino Volcano" had 8.58M users and was ranked number five in the "Social and Communication" category until it was removed on July 20, 2020 for policy violations. Quite worrisome is the fact that 25% of malware-containing extensions have over 5k users (vs. a 75^{th} percentile of 220 for benign extensions).

In total, we collected the metadata of over 26k SNE (Table 1). Overall, we observe that over 346 million users installed at least one SNE in the last three years: 280M users installed malware-containing extensions, 63.3M policy-violating, and 2.9M vulnerable extensions.[2] We assume that those users are unaware of using SNE. Given both the extremely large number of impacted users and the fact that SNE stay in the CWS *for years*, SNE are a major problem and need to be removed as quickly as possible from the CWS. While Google engineers seem to be looking for malware-containing or policy-violating extensions through their review process [32] (with more or less success); to the best of our knowledge, they currently cannot detect vulnerable extensions [41, 74]. Future work would be beneficial in this area to further secure the CWS.

Takeaway *Over 346 million users* installed a SNE in the last 3 years. Given that SNE tend to stay in the CWS *for years*, it is critical to improve the detection of SNE and notify impacted users.

4.3 Extension Ratings

Given that SNE are only moderately removed from the CWS and that this process can take up *years*, we now investigate if users themselves are able to flag SNE, e.g., by giving such extensions a low rating. On the CWS, extension users can rate extensions from 1 (lowest score) to 5 (highest score); extensions with no ratings are given a score of 0. First, we observe that a large number of extensions have no ratings: 58.63% of policy-violating extensions have no ratings, 52.12% for malware, 47.32% for vulnerable, and 31.52% for benign extensions. Of the extensions that do have a rating, we

[2]Note that we know the number of users for each extension, but we cannot deduplicate the total number of users, in case one user has several SNE installed; so this number does not represent *unique* users.

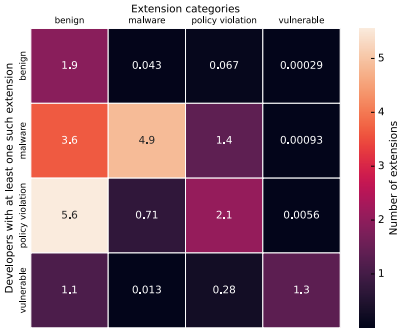

Figure 6: Average number of benign extensions or SNE published by a developer with at least one of these extensions– E.g., a developer with 1 malware-containing extension in the CWS (y-axis) publishes on average 3.6 benign, 4.9 malware-containing, 1.4 policy-violating, and 0.00093 vulnerable extensions (x-axis)

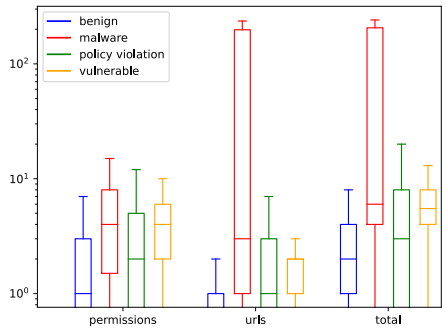

Figure 7: Number of API-based (permissions), host-based (urls), and total (sum of APIs and hosts) permissions of benign, malware, privacy-violating, and vulnerable extensions

do not observe a significant difference in scores between these four groups: the benign and policy-violation sets have a median of 5, 4.997 for malware, and 4.571 for vulnerable. Overall, users do not give SNE lower ratings, suggesting that users may not be aware that such extensions are dangerous. Of course, it is also possible that bots are giving fake reviews and high ratings to those extensions. However, considering that half of SNE have no reviews, it seems that the use of fake reviews is not widespread in this case.

Takeaway Users do not give SNE lower ratings. This makes thorough vetting and review of extensions all the more critical.

4.4 Extension Developers

Next, we focus on the developers designing the SNE that are or were in the CWS. Interestingly, we found numerous cases of developers publishing both SNE and benign extensions. In Figure 6, we represent the number of benign, malware-containing, policy-violating, and vulnerable extensions (x-axis) published by a developer already having one such extension in the CWS (y-axis). For example, a developer having published 1 malicious extension (y-axis) publishes on average 3.6 benign, 4.9 malware-containing, 1.4 policy-violating, and 0.00093 vulnerable extensions. We also found instances of developers having many malicious extensions. For example, the developer "http://newtabexperience.com" has had 1,041 extensions removed for containing malware and 9 for policy violations (as of May 1, 2023, this developer still has 434 extensions in the CWS). Overall, we observe that developers with at least one SNE tend to publish more SNE than developers with one benign extension. In particular, developers publishing malware-containing extensions publish almost 5 of those, on average, whereas other developers publish less than 1. In other words, it is quite unlikely that a developer having published at least one benign extension will publish a SNE. There are some exceptions, though. For example, the developer "New Tab" has 1,041 benign extensions in the CWS but also 2 SNE (1 malicious and 1 policy-violating extension). On the contrary, a developer having a malware-containing or privacy-violating extension will likely publish another one of those. Overall, there are 30 developers with over 100 malware-containing extensions each; and 28 developers each of them with over 100 extensions removed for violating the CWS policies. We assume that such developers multiply their SNE to try to infect or harm as many users as possible, for their own profit. Therefore, extensions by developers known to have created a SNE may require further scrutiny. The trend is different for vulnerable extensions, though. Such developers do not seem to publish more malware-containing or privacy-violating extensions than benign developers, but they tend to publish more vulnerable extensions. So, contrary to the other SNE, it seems that developers publishing vulnerable extensions are *well intentioned* but make mistakes in their implementation, leading to *vulnerabilities*.

Takeaway Developers having published a SNE are more likely to publish SNE than developers having published a benign extension. We endorse flagging and further scrutinizing the extensions submitted to the CWS by developers with known SNE.

4.5 Extension Permissions

Finally, we compare the permissions a SNE vs. benign extension request. In fact, the requested permissions will determine the capabilities of an extension, i.e., the attack surface for SNE. We collected permissions by parsing each extension's `manifest.json` file. For manifest V3, permissions are broken up into *permissions* (APIs such as `storage` or `cookies`) and *host permissions* (URLs or URL patterns that an extension wants to make requests to). For manifest V2, these two categories are combined. For consistency with manifest V3, we separate APIs from host permissions.

First, we look at the number of permissions each extension requires (Figure 7). As expected, SNE require more permissions than benign extensions. For API-based permissions, malware-containing and vulnerable extensions use a median of 4 permissions vs. 2 for policy-violating and 1 for benign extensions. We observe a similar trend for host permissions. If we look at the third quartile, 25% of malware-containing extensions list 198 URLs; 3 for policy-violating, 2 for vulnerable, and 1 for benign extensions. Ultimately, the more permissions an extension has, the larger the attack surface is.

Next, we dive into the specific API and host permissions extensions need. Interestingly, and perhaps surprisingly, both benign

Rank	Benign	Malware	Policy violation	Vulnerable
1	<all_urls>	*://www.google.com.kh/*	*://www.google.com.kh/*	http://*/*
2	https://*/*	*://mail.google.com/*	*://mail.google.com/*	https://*/*
3	http://*/*	*://www.google.com/*	*://www.google.com/*	<all_urls>
4	*://*/*	*://www.google.com.au/*	*://www.google.ps/*	chrome://favicon/
5	http://*/	*://www.google.us/*	*://www.google.co.zw/*	*://*.fliptab.io/*
6	https://*/	*://www.google.nl/*	*://www.google.co.zm/*	*://*/*
7	chrome://favicon/	*://www.google.ca/*	*://www.google.co.za/*	https://*/
8	https://ajax.googleapis.com/	*://www.google.dk/*	*://www.google.ws/*	http://*/
9	https://image.lovelytab.com	*://www.google.co.jp/*	*://www.google.vu/*	*://127.0.0.1/*
10	https://v2.lovelytab.com/	*://www.google.no/*	*://www.google.com.vn/*	*://localhost/*

Table 2: Top 10 host-based permissions for benign and SNE

extensions and SNE seem to more or less use the same APIs (see a list of the top 10 API-based permissions in Table 5 in the Appendix). The only notable exception is `topSites` (permission allowing an extension to access a user's most visited sites), ranking 2 for malicious extensions (4,026 extensions) but not in the top 10 in any of the other groups. This number is likely influenced by the numerous malware-containing extensions that replace a user's homepage on new tabs, which requires this `topSites` permission to operate.

On the contrary, there are differences between the host permissions required by each category of extensions, as illustrated in Table 2. Firstly, all-encompassing URLs, e.g., "<all_urls>" or "http://*/*", are highly ranked in both benign and vulnerable extensions but not in malware-containing nor policy-violating, which instead have a lot of Google sub-domains. One reason for this may be that extensions requesting powerful privileges, such as "<all_urls>", require in-depth review [32], which developers with malicious intent may want to avoid. For example, we discovered a set of 1,130 SNE, 740 of which were removed for malware and 390 for policy violation; all of them had a large list of Google sub-domains in their host permissions. This exemplifies how large clusters of identical extensions can skew results and demonstrates the need to account for duplicated extensions when conducting analyses.

Takeaway SNE request more permissions than benign extensions, likely because the more permissions an extension has, the larger the attack surface is. While host-based permissions differ between benign and malware-containing extensions, API permissions do not enable to discriminate between benign and SNE.

5 CODE SIMILARITY AND SECURITY

Next, we show that analyzing extension codes for similarity could be a useful tool for identifying SNE. Subsequently, we investigate the sources of code similarities and exemplify how code reuse can lead to the propagation of vulnerabilities.

5.1 Extensions with a Similar Code Base

We first present our approach to detect clusters of similar extensions and discuss some case studies.

5.1.1 Methodology. As described in Section 3.1, we downloaded and unpacked our sets of malware-containing, policy-violating, vulnerable, and benign extensions. For each extension, we extracted their content and background scripts, as described previously. We could successfully collect 108,859 extensions (92,482 benign, 6,587 malware-containing, 9,638 policy-violating, and 152 vulnerable). To compare the source code of all these extensions with each other, we compared all pairs of `(content script, content script)`, `(background script, background script)`, and `(background script, content script)`.[3] For all of the described pair types, we performed an ssdeep fuzzy-hash-based clustering [12]. We chose this similarity-hashing algorithm to identify clusters of similar[4] extension scripts. We cluster two extensions together if they have a 100% overlap either with their content script or background script.

5.1.2 Clusters of Similar Extensions. We compared the source code of all 108,859 extensions we successfully collected. We found that 20,822 extensions contain a similar content or background script to other extensions. We found a total of 3,270 clusters. The median cluster size is 2 similar extensions, but the distribution is heavily right-skewed with a mean of 7.32 extensions and a maximum cluster size of 1,397 extensions. By manually analyzing extension source code and CWS pages, we categorize our clusters into three types:

- **Identical extensions:** These extensions look the same and perform the same tasks; their description and images in the CWS are similar. E.g., we observe instances of developers publishing a new extension instead of pushing an update to an existing one.
  Case study: We found 19 "Kobo Book" extensions (e.g., "Free Kobo Books", "Free Kobo Books Deluxe", or "Free Kindle Religious Books"). They all have identical background scripts as well as identical screenshots and descriptions on their respective CWS pages, but some have different names and developers.
- **Extensions with a similar functionality:** These extensions have a similar functionality, but they may look different and have a dissimilar page on the CWS. E.g., there are many home page wallpaper extensions, where a developer publishes hundreds of extensions with a different theme, such as cars or nature. The source code for all of these extensions is the same, with a different URL string in the code to select the specific image category.
  Case study: 9 screenshotting extensions share an identical background script, e.g., "Xsnap", "LTR Screenshot", or "Computub Screenshotter". Although the extensions do similar actions, they have very different UI and branding. The identical background scripts contain code to receive a button click, take a screenshot, and display it to the user.
- **Extensions with generic background scripts:** Such extensions perform very different functions but contain generic code, such as receiving a message or sending a response, which could come as part of an extension builder or sample code.
  Case study: We found a cluster of 1,397 extensions with identical background scripts, e.g., "Read a Newspaper" or "Goleferine". The background script for all of these extensions is a message listener that sends the tab ID. These files are identical, down to the comment "//example of using a message handler from the inject scripts," which makes it likely that all developers copied the code from a Chrome tutorial or StackOverflow post.

[3] We found clusters of similar (content script, background script). E.g., the background script of "PodQueue" is similar to the content script of "Slack Beeegmojis", as they both use the `webextension-polyfill` class in their respective scripts.

[4] We prefer the word *similar* to *identical*: while the majority of the scripts are identical word for word, a few scripts have the same syntactic structure, but a few string constants were changed. We hypothesize they were still clustered with a 100% overlap, because ssdeep returns an integer between 0 (no match) and 100, and rounding up by the algorithm probably gave a 100 score to *nearly* identical extensions.

5.1.3 Suspicious Extension Clusters. Of the 3,270 clusters we identified previously, 2,296 contain only benign extensions, 321 only SNE, and 653 both SNE and benign extensions. The fact that 321 clusters contain only SNE is a good indication that code similarity could enable to quickly remove a full cluster once one of the extensions in the cluster is found to be security critical; or at least to flag the remaining extensions in the cluster for further analyses. In the following, we focus in more detail, first on SNE clusters, and then on clusters containing both SNE and benign extensions.

SNE Clusters — Regarding SNE clusters, 14 of those contain 100 or more SNE, with 2 clusters containing 863 SNE each. One of these 863-SNE clusters mainly contains new tab wallpaper extensions, i.e., when a user opens a new tab, such extensions redirect a user to a custom Google search page with a different wallpaper image. These wallpaper images can come in different themes, e.g., sports or anime, which allows a developer to create hundreds of extensions with the same code but different background images. Such extensions include "Kpop Big Bang Wallpapers New Tab HD" and "Los Angeles Clippers Wallpapers New Tab HD". The majority of these new tab extensions are by the developer "http://newtabexperience.com", although there are extensions by other developers in the cluster. Across the board, new tab extensions are very common, as they can be used to show users advertisements or distribute malware [53]. In fact, almost all of the extensions in this cluster were removed from the CWS for containing malware on October 1–5, 2020.

SNE and Benign Extension Clusters — We now look at the 653 clusters containing both benign extensions and SNE. These clusters total 10,678 extensions: 1,754 malware-containing, 3,370 policy-violating, 2 vulnerable, and 5,552 benign extensions.

One such cluster contains 9 screenshotting extensions that all have the same background script. 3 / 9 extensions were removed for policy violations months apart (January 20, February 11, and March 17, 2023), despite them having *identical code* and not receiving any updates in this time span. As a result, we believe that the CWS developer team is not using code similarity as part of their review process to help detect suspicious extensions. With code similarity-based approaches, once one extension in the cluster had been found to be in violation of the CWS policies, the other 8 identical extensions could have been flagged for additional review, allowing a faster removal of those policy-violating extensions.

Interestingly, we also found examples of truly benign extensions sharing a cluster with SNE. For example, in a cluster of 9 extensions (7 benign and 2 malicious), we see that "Keplr for cosmos: osmosis,akt Wallet" is malicious but not "G-Eye". Despite them having *identical code*, they have *different permissions*: the benign extension is missing the `activeTab` and `scripting` permissions. This makes the "malicious" part in the benign extension dead-code for now (due to the absence of the required permissions). In other words, a simple permission change in the `manifest.json` file could make the extension malicious. Unfortunately, both Fass et al. [41] and Pantelaios et al. [57] reported examples of extensions turning vulnerable or malicious, respectively, after such an update. For these reasons, we believe that the 5,552 "benign" extensions containing *identical code to known SNE* should be flagged for additional review. Maybe some of them are not security critical today, but they could easily become so, and should thus be more closely monitored.

Takeaway We uncovered thousands of clusters of similar extensions. We found that 321 clusters contain only SNE, showing that analyzing extension code for similarities could be a useful tool for identifying SNE: once one extension in a cluster is found to be security-critical, researchers or Google developers could flag the remaining ones for additional screening. Flagged extensions could then be run through more computationally and time-expensive analyses, and should be more closely monitored.

5.2 Sources of Code Similarity

Interestingly, some of the similar extensions we found are written by different developers. We observe that it is common for developers to include URLs to reference the origin of a code snippet or give credit to other authors [43]. As a best-effort strategy to understand sources of code similarity, we scrape the comments from all of the downloaded extension code (JavaScript files) and extract URL-like strings. While sometimes the URLs are simply generic license files, we find that in many cases links cited in commented code specify the origin of the code, i.e., the reason for code similarity.

5.2.1 Methodology. For each extension, we extract the comments from all of the JavaScript files using Esprima [7], a popular JavaScript parser used by prior work [19, 38–42, 52, 55, 65, 66]. To conduct a more fine-grained analysis, contrary to Section 5.1, we consider all JavaScript files from an extension package independently (i.e., we do not merge all content scripts or background scripts). Then, we use the get-urls library to search for and extract URLs from the comments [9]. Since developers also use comments to temporarily remove small sections of code, the get-urls library triggers false positives (i.e., text being incorrectly flagged as URL). To eliminate these false positives for our analysis, we filter for strings that contain one of the top five most popular top-level domains: ".com", ".org", ".ru", ".net", and ".uk" [3]. We also include ".io" and ".edu".

5.2.2 Origins of Code Similarity. Overall, we collected over 8M URLs embedded in comments, over 50k of which are unique (we list the top 10 URLs appearing the most often in Table 6 in the Appendix). These URLs belong to 18,551 unique second-level domains (we list the 10 most popular second-level domains in Table 7 in the Appendix). As a best-effort strategy to understand to which categories of websites these URLs point to, we manually inspected the top 100 most popular URLs. We find URLs primarily from the following categories: licenses, code documentation, code repositories, forum responses, corporate websites, and personal developer websites. Overall, we collected 697 unique URLs that contain licenses, 3,036 from Stack Overflow, and 9,014 from GitHub repositories.

Next, we validate our assumption that identical URLs in code commented blocks are an indicator of code similarity. To this end, we leveraged the ssdeep library [12], as previously, to compute the fuzzy hashes of all the files that contain a given URL in their commented code, and we compared all the hashes. To be able to manually verify our findings, we used this approach on a random sample of 100 extensions, for each of the ten most popular URLs, as listed in Table 3, columns 1 and 2. In column 3, we report the proportion of extensions that have a unique file. For example, 45% of the extensions listing the URL "http://apache.org/licenses/LICENSE-2.0" do not have any files identical to any files from the other 99

extensions listing this URL. This result is not particularly surprising as this URL refers to a license file, which can be applied to a variety of different software and is, therefore, not a strong indication of code similarity. On the contrary, for "http://extensionizr.com", only 15% of the files referencing this URL have unique files not appearing in any of the other 99 extensions. Overall, the results from Table 3 confirm the fact that extensions reuse code from public sources, such as code documentation, code repositories, or forums. This is bad practice, as code reuse multiplies the usage of dated code and propagates vulnerabilities [58]. We give a specific example below.

5.2.3 Security Implications. In this section, we illustrate the security implications of code reuse with one case study. We focus on Extensionizr, an open-source project used for generating boilerplate for Chrome extensions [73]. As shown in Table 3, "http://extensionizr.com" is the fourth most frequent URL found in at least one comment block of an extension. Despite Extensionizr being last updated in 2017 [8], 816 new extensions have been created using this project between 2020 and 2023 (we plot the number of extensions created each year between 2012–2023 using Extensionizr in Figure 12 in the Appendix). Additionally, extensions created with Extensionizr have the option of adding Angular.js or jQuery as add-ons. However, since the project is not maintained, the only options for adding these libraries are Angular.js version 1.0.6 and jQuery version 2.0.0, both of which have known vulnerabilities [4]. To estimate a lower bound of the number of Extensionizr extensions using a vulnerable library, we used RetireJS, an open-source tool to detect known vulnerable libraries [10]. We found that 65.56% of the Extensionizr generated extensions use the default and vulnerable jQuery 2.0.0 and 79.66% Angular.js 1.0.6 (note that those vulnerable libraries are not necessarily exploitable from the context of a web application). This case study rather exemplifies the fact that code reuse is a bad practice that can lead to dated code and vulnerabilities, all the more when the third-party code is not maintained.

Takeaway We find that commented URLs in extension code can indicate the origins of code similarity. Code reuse is a bad practice, as it propagates the usage of dated code and vulnerabilities. For instance, roughly 1,000 extensions use the open-source Extensionizr project, 65–80% of which still use the default and vulnerable library versions initially packaged with the tool, 6 years ago.

6 MAINTENANCE AND SECURITY

Previously, we showed that dated code propagates vulnerabilities. We now investigate extension maintenance in the CWS and discuss security implications. Specifically, we exemplify the prevalence of outdated and vulnerable developer practices.

6.1 Extension Updates

First, we focus on extension updates. On average, 200–600 extensions are updated every day (Figure 8). The number of updates per day was mostly constant over the past two years. Interestingly, there were no update spikes after any key releases about the manifest V3 timeline. However, there are two outliers on April 19-20, 2022 where over 900 extensions were updated on each day. This correlates with the release of the featured and verified badges, which was reported by news outlets on April 19 and officially posted on Google's blog on April 20 [48]. We assume that many developers submitted updates to meet the requirements for the badges.

URL	# unique extensions	% extensions with a unique file
http://apache.org/licenses/LICENSE-2.0	2,622	45%
http://opensource.org/licenses/MIT	2,122	18%
http://opensource.org/licenses/mit-license.php	1,735	74%
http://extensionizr.com	1,651	15%
http://jquery.org/license	1,158	21%
http://underscorejs.org	854	12%
http://code.google.com/p/crypto-js	755	16%
http://code.google.com/p/crypto-js/wiki/License	754	20%
https://github.com/carhartl/jquery-cookie	722	15%
http://mozilla.org/MPL/2.0	712	35%

Table 3: Number of unique extensions a URL appears in and percent of extensions with this URL that have unique files

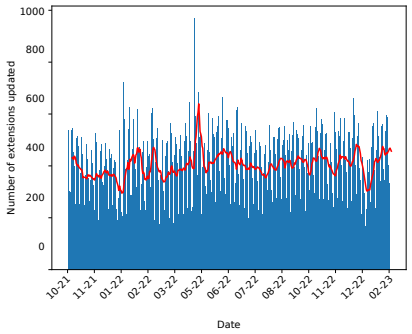

Figure 8: Number of extensions updated on a given day–The red line represents the 7-day rolling average

Surprisingly, the median number of updates an extension receives after a year in the CWS is 0, and the average is 0.16. So, the majority of extensions do not seem particularly maintained (even though some outliers have received over 25 updates). Of the extensions in the CWS on February 14, 2023, 73,865 (59.5%) have never been updated since they were added to the CWS. Interestingly, those extensions tend to have fewer users (average of 1,974) than extensions that have received updates (30,708 users).

To further quantify our claim about the lack of maintenance in the CWS, we represent the number of extensions by time window of their most recent update in Figure 9 (as of February 14, 2023). As expected, the majority of extensions have not been updated within the past year: almost 30k extensions have been last updated 2–4 years ago, and over 40k more than 4 years ago. We even observe over 5k extensions not having been updated within the past decade! Overall, this raises concerns about compatibility issues and potential security or privacy implications of unmaintained extensions. We give a specific example in Section 6.3, where half of known vulnerable extensions have been unmaintained for over 2 years. Nevertheless, we acknowledge that some extensions may already have achieved their full functionality when being first added to the CWS and may not need further updates. Still, Chrome is releasing new APIs and features, such as Manifest V3, whose benefits (also including security and privacy improvements [25]) those unmaintained extensions are missing out on.

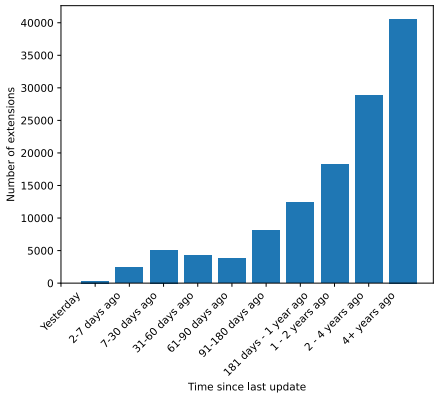

Figure 9: Number of extensions at their last update (as of February 14, 2023)

Takeaway We highlight a critical lack of maintenance in the CWS, e.g., almost 60% of extensions have never received any updates. Given the sensitive nature of extensions, this is problematic, as such unmaintained extensions are missing out on security and privacy improvements such as those offered by Manifest V3.

6.2 Manifest Versions

As discussed in Section 2, Chrome released manifest V3 in November 2020 and stopped accepting new manifest V2 extensions in January 2022 [27]. However, as of February 2023, 74% of extensions are still using manifest V2 (62% as of July 2023). This is especially concerning as Chrome originally planned to force all manifest V2 extensions to set their visibility to unlisted or private in June 2023 and remove all manifest V2 extensions in January 2024, although this is now under review and being postponed [27]. The purpose of manifest V3 is to improve security, privacy, and performance [25]. For example, from a security perspective, extensions with manifest V3 are no longer able to run arbitrary code in their highly-privileged service worker environment. All code must be packaged within the extension, unlike previously where extensions could download and run external code [30]. In particular, it has been shown that extensions running arbitrary code could lead to universal cross-site scripting vulnerabilities [41, 64, 74]. These vulnerabilities (among others) would be fixed if the extensions migrated to manifest V3.

While manifest V3 promises a higher security and privacy posture, transitioning from V2 to V3 is challenging. For example, the transition requires significant changes for many existing extensions: some features are now unavailable, which forces developers to invest considerable time and effort to look for alternatives, which sometimes do not exist. In other cases, developers complain about limited API capabilities: the shift from the `webRequest` API to the `declarativeNetRequest` API limits extensions' ability to modify web requests on the fly, which is problematic, e.g., for adblockers [5].

Takeaway Unfortunately, the transition from manifest V2 to V3 is challenging. As a consequence, 74% of extensions and 1.2 billion users are potentially missing out on some security and privacy benefits offered by manifest V3.

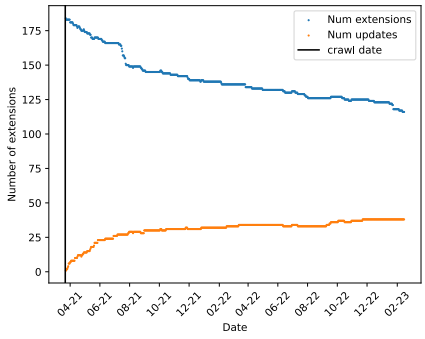

Figure 10: Life cycle of vulnerable extensions–The blue curve represents the vulnerable extensions detected by DoubleX [41] (crawl: March 16, 2021) that are still in the CWS. In orange, we see the number of those extensions that have received at least one update. The blue minus orange line thus represents a lower bound of the number of extensions that are still in the CWS and vulnerable

6.3 Vulnerable Extensions

As shown in Section 6.1, the vast majority of extensions are unmaintained. This is particularly problematic for vulnerable extensions, whose user base can stay at risk *for years* if the vulnerabilities are not fixed. In fact, in 2021, Fass et al. showed that vulnerable extensions persist in the CWS: of the 193 vulnerable Chrome extensions they found in 2020, 160 (83%!) were still in the CWS and still vulnerable one year later [41]. We extended their experiment from 2021 to 2023. To this end, we contacted the authors and received the list of the 184 vulnerable extensions (including the corresponding versions) they found to be vulnerable in 2021. In Figure 10, we represent the life cycle of those vulnerable extensions; the blue curve represents the vulnerable extensions detected by their tool, DoubleX, (crawl: March 16, 2021) that are still in the CWS, over time: 116 / 184 extensions (63%) remain in the CWS as of February 2023. In orange, we see the number of those extensions that have received at least one update. The blue line minus the orange line thus represents a lower bound of the number of extensions that are still in the CWS and vulnerable (in fact, an update does not necessarily equate to fixing a vulnerability, so our results are a *lower bound* of the number of extensions still vulnerable today). Overall, we find that at least 78 extensions (over 42%) are still vulnerable two years after disclosure, impacting over 450k active users.

Takeaway At least 78 / 184 extensions (42%) are still in the CWS and still vulnerable 2 years after disclosure. This shows that, while detecting vulnerable extensions is critical, we also need better incentives to encourage and support developers to *fix vulnerabilities* after disclosure.

6.4 Vulnerable JavaScript Library

After *vulnerable extensions*, we finally investigate the prevalence of extensions using *vulnerable JavaScript libraries*. To detect known vulnerable JavaScript library versions, we leverage RetireJS [10, 11] to scan each Chrome extension. We find that 39,093 extensions

Library	Vulnerability summary	# extensions
jquery	XSS, 3rd party CORS	41,290
jquery-ui	XSS	9,632
bootstrap	XSS	6,746
moment.js	regular expression DoS, path traversal	5,408
angularjs	XSS, CSP bypass, DoS	4,728
jquery-ui-dialog	XSS	2,223
underscore.js	arbitrary code injection	1,693
vue	XSS	1,486
YUI	XSS	1,027
jquery-validation	regular expression DoS	960

Table 4: 10 most commonly used JavaScript libraries containing known vulnerabilities and used by extensions

(31.5%) use at least one JavaScript library with a known vulnerability. Those extensions belong to 32,144 distinct developers. Across all extensions, we identify 87 unique JavaScript libraries with a known vulnerability. We detect over 80k uses of those vulnerable JavaScript libraries, impacting almost 500 million users. We list the 10 most commonly used vulnerable libraries in Table 4, along with the corresponding vulnerabilities, and the number of impacted extensions. The top vulnerable JavaScript library is jQuery, with over 40k extensions loading a vulnerable version (this represents a third of the extensions in the CWS!). As a comparison, the remaining vulnerable libraries we detected are used by less than 10k extensions. We list the top 10 JavaScript library versions with vulnerabilities in Table 8 in the Appendix; the majority of which are jQuery versions. However, using a vulnerable JavaScript library does not equate to a vulnerable extension, since the vulnerability may not be exploitable from the context of a web application. While some vulnerabilities may be exploitable through Web Accessible Resources, we leave a study of the exploitability of vulnerable libraries for future work.

Takeaway Almost a third of extensions (40k) use a JavaScript library with a known vulnerability: we detect over 80k uses of vulnerable libraries, impacting almost 500M extension users.

Next, we investigate how often extensions with a vulnerable library are updated. In Figure 11, we represent the year of the most recent update for each extension in blue and plot the number of extensions that use a library with a known vulnerability, grouped by last update year, in orange. Surprisingly, we observe that extensions which were updated recently are not less likely to have a vulnerable library than extensions that were updated several years ago. Specifically, for extensions with their most recent updates in 2017–2023, between 19.54% and 24.62% are using vulnerable libraries. While extensions with their most recent updates between 2013 and 2016 have a greater fraction of extensions with vulnerable libraries (between 24.49% to 32.88%), extensions with their most recent updates in 2011 and 2012 have a smaller fraction (0.81–14.11%).

Interestingly, and perhaps surprisingly, we find that even when developers are updating their extensions, they are not necessarily maintaining the JavaScript libraries within their extensions. For instance, 21% of the extensions updated in 2022 and later (8,236 extensions) have a known vulnerable library. Overall, this provides further evidence that the use of vulnerable libraries persists even when patched library versions are available and even when developers are maintaining their extensions.

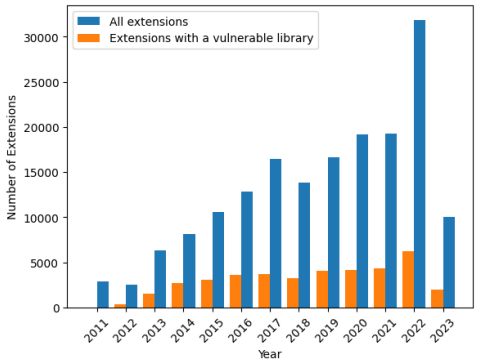

Figure 11: Number of extensions using a vulnerable library (orange), grouped by last update year, compared to the total number of extensions (blue), by last update year

Takeaway Even when developers update their extensions, they often *do not* update vulnerable libraries within their extensions, even after a patched version of a vulnerable library has been released. We encourage developers to not only maintain their extension source code but also libraries within their extension package.

7 DISCUSSION

In this section, we first discuss some limitations of our approach. Then, we summarize our takeaways and recommendations.

7.1 Limitations

Since Google does not archive browser extensions that used to be in the CWS, we are dependent on ChromeStats to access longitudinal data. As mentioned in Section 3.1, for some extensions, we only had access to their metadata and not source code.

Another limitation of our paper relates to our code similarity approach. We chose to compute similarity based on the raw source code (Section 5.1.1). However, this approach is not resistant against variable renaming or if a developer chooses to merge scripts in a different order. In the future, we could consider other methods, such as relying on Abstract Syntax Trees [38]. Regarding our suggestion to use code similarity analysis to identify SNE (Section 5.1.3), we realize that sharing an identical background or content script with a SNE does not necessarily imply that an extension is also security-critical. For example, in the case of policy-violating extensions, we would need to also analyze the reported privacy policies; for vulnerable extensions, we would need to review the extension's permissions. This is why we emphasize this as a tool for flagging extensions for future investigation, not the end-all determination.

Regarding security implications of a lack of maintenance in the CWS, we acknowledge in Section 6.1 that poorly maintained extensions do not necessarily lead to security issues: an extension may have achieved its intended functionality when first added to the CWS and not necessitate any further updates. As for extensions using vulnerable libraries, we emphasize in Section 6.4 that a vulnerability does not equate to exploitability. We leave a study of the exploitability of vulnerable libraries in extensions for future work.

7.2 Recommendations and Future Work

We summarize here our main guidelines to guide future research and pave the way for a more secure CWS.

Volatility in the CWS and Reproducibility — We observe unexpected volatility of extensions in the CWS. It is important to keep in mind that extensions are frequently being removed from the CWS and new extensions are being added. Additionally, extensions have extremely short life cycles, and extension user counts are unstable. Given these constant changes, researchers should try to perform their analyses over multiple points in time (e.g., by using data from ChromeStats [6]) and evaluate the reproducibility of their findings before making claims about broad patterns in the CWS. Specifically, longitudinal studies provide the best assurance of temporal representativeness. To this end, we encourage researchers to open source their extension sets to ease future work and comparisons.

SNE Detection and Extension Vetting — We show that SNE stay in the CWS *for years*. First, additional efforts are needed to detect such extensions *prior* to their acceptance to the CWS. For example, existing tools [41, 74] could help Google engineers flag potential vulnerable extensions. To detect malware-containing and privacy-violating extensions, we strongly recommend thoroughly scrutinizing extensions submitted to the CWS by developers having published SNE in the past. Similarly, we encourage researchers and Google engineers to look for extensions with a similar code base to known SNE. Flagged extensions should be more closely monitored, e.g., run through more computationally and time-expensive analyses to detect and reject verified SNE from the CWS. Second, we suggest to vet extension updates given that "benign" extensions can easily turn malicious after permission changes. Third, we encourage researchers to investigate *why* vulnerable extensions stay in the CWS for years, even after disclosure. While Google engineers may need to be more active and restrictive when vetting extensions, this also raises the question of who should be responsible for having SNE in the CWS. For instance, developers may not have any incentives to fix their vulnerable extensions once they have been accepted to the CWS. In this case, the impacted users should be notified of the security and privacy issues they may be subject to. This is all the more important as users have a limited understanding of security and privacy issues extensions can induce [15].

Extension Maintenance — We obviously encourage developers to maintain their extension source code, including migrating to manifest V3, when applicable. We also recommend updating the libraries within their extension package. We leave an investigation of the exploitability of vulnerable libraries for future work.

8 RELATED WORK

Prior work focused on detecting *malicious*, *vulnerable*, and *fingerprintable* extensions. Our work is in an orthogonal direction. We investigate what is in the CWS and highlight security implications that were overlooked by prior work.

Malicious Extensions — First, malicious extensions are designed by malicious actors to *harm victims*. For example, some extensions may tamper with security headers [13], steal users' credentials, track users [72], spy on them [14], spread malware [1], leak sensitive information [22, 68], or have inconsistencies in their privacy practices [18]. To defend against these issues, several approaches have been proposed to detect malicious extensions, e.g., by monitoring their behavior [45, 71], tracking the reputation of developers [44], or detecting anomalous ratings [57]. In this paper, we highlight the fact that malicious extensions have different patterns (e.g., number of days in the CWS, user counts, or developers) compared to benign extensions. Such patterns could be leveraged to flag suspicious extensions for additional analyses.

Vulnerable Extensions — Second, vulnerable extensions are designed by well-intentioned developers but *contain vulnerabilities*, leading to security or privacy issues. Prior work focused on vulnerabilities in XPCOM Firefox extensions (XPCOM is now deprecated), based on static information flow tracking [17] and JavaScript namespace vulnerabilities [19]. To detect vulnerable Chrome extensions, some approaches primarily rely on manual analysis [21, 64], others focus on a formal approach [20], abstract interpretation [74], or data flow analysis [41]. In this paper, we show that vulnerable extensions have patterns quite similar to benign extensions', which makes them more challenging to detect. Besides detection, it is of utmost importance to encourage developers to fix known vulnerabilities that currently pervade the CWS.

Fingerprintable Extensions — Third, fingerprintable extensions enable an attacker to *track users* across websites, deanonymize them, or infer sensitive information about them (e.g., related to health or religion) [46]. Extensions can be fingerprinted by analyzing their observable side effects. An attacker can detect the extensions a user has installed by leveraging DOM changes [63, 67, 69], style changes [50], Web Accessible Resources [46, 60], user actions [62], or timing-channels [59]. Several defenses to extension-enumeration attacks have been proposed, such as controlling extensions loaded on a website [61], randomizing browser extension fingerprints [2], or creating a parallel DOM [47]. We leave a study of fingerprintable extensions and a comparison with SNE for future work.

Untrusted Libraries and Third Parties — In this paper, we show that extensions are not maintained and use known vulnerable libraries. This is comparable to the npm ecosystem where lack of maintenance causes packages to depend on vulnerable code [75]. In npm, one proposed mitigation was to reduce the attack surface by removing unused code [49]. We could imagine a similar approach for browser extensions. More generally, similar issues are observed on the Web (and still an open challenge), when developers do not maintain libraries, allowing their site to be compromised [51].

9 CONCLUSION

This paper provides a holistic view of the browser extensions' landscape, within the CWS. We leverage historical data provided by ChromeStats to broadly investigate and understand what is in the CWS, along with security implications. Our research fundamentally underlines (a) the extremely *short life cycles of extensions*, (b) the pervasiveness of what we call *"Security-Noteworthy Extensions" (SNE)*, (c) the presence of *clusters of similar extensions*, and (d) a critical *lack of maintenance in the CWS*. We are confident that our results can serve as a foundation for researchers and practitioners looking to study and improve the security of the browser extension ecosystem.

ACKNOWLEDGMENTS

We would like to thank ChromeStats [6], and more specifically Hao Nguyen, for providing us with Chrome extensions, their metadata, and additional information upon request. We also thank the anonymous reviewers for their constructive reviews and helpful suggestions. Finally, we thank Giovanni Apruzzese and the members of the Stanford Empirical Security Research Group for valuable conversations and feedback.

REFERENCES

[1] 2015. Understanding Malvertising Through Ad-Injecting Browser Extensions. In *WWW*.
[2] 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *USENIX Security Symposium*.
[3] Accessed on 2023-04-18. Most popular top-level domains worldwide as of June 2022. https://www.statista.com/statistics/265677/number-of-internet-top-level-domains-worldwide/.
[4] Accessed on 2023-04-21. Retire.js. https://github.com/RetireJS/retire.js/blob/master/repository/jsrepository.json.
[5] Accessed on 2023-07-18. Chrome extensions Manifest V3 migration status. https://chrome-stats.com/manifest-v3-migration.
[6] Accessed on 2023-07-18. Compare and analyze Chrome extensions. https://chrome-stats.com.
[7] Accessed on 2023-07-29. ECMAScript parsing infrastructure for multipurpose analysis. https://esprima.org/.
[8] Accessed on 2023-07-29. Extensionizr.com! https://github.com/altryne/extensionizr.
[9] Accessed on 2023-07-29. get-urls. https://github.com/sindresorhus/get-urls.
[10] Accessed on 2023-07-29. Retire.js. https://github.com/retirejs/retire.js/.
[11] Accessed on 2023-07-29. Retire.js. https://retirejs.github.io/retire.js/.
[12] Accessed on 2023-07-29. ssdeep 3.4. https://pypi.org/project/ssdeep.
[13] Shubham Agarwal. 2022. Helping or Hindering ? How Browser Extensions Undermine Security. In *ACM CCS*.
[14] Anupama Aggarwal, Bimal Viswanath, Liang Zhang, Saravana Kumar, Ayush Shah, and Ponnurangam Kumaraguru. 2018. I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions. In *Euro S&P*.
[15] Ankit Kariryaa, Gian-Luca Savino, Carolin Stellmacher, and Johannes Schöning. 2021. Understanding Users' Knowledge about the Privacy and Security of Browser Extensions. In *SOUPS*.
[16] Aurore54F. Accessed on 2023-07-28. DoubleX/src/unpack_extension.py. https://github.com/Aurore54F/DoubleX/blob/main/src/unpack_extension.py.
[17] Sruthi Bandhakavi, Samuel T Kingand P Madhusudan, and Marianne Winslett. 2010. VEX: Vetting Browser Extensions for Security Vulnerabilities. In *USENIX Security Symposium*.
[18] D. Bui, B. Tang, and K. G. Shin. 2023. Detection of Inconsistencies in Privacy Practices of Browser Extensions. In *S&P*.
[19] Ahmet Salih Buyukkayhan, Kaan Onarlioglu, William Robertson, and Engin Kirda. 2016. CrossFire: An Analysis of Firefox Extension-Reuse Vulnerabilities. In *NDSS*.
[20] Stefano Calzavara, Michele Bugliesi, Silvia Crafa, and Enrico Steffinlongo. 2015. Fine-Grained Detection of Privilege Escalation Attacks on Browser Extensions. In *Programming Languages and Systems*.
[21] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An Evaluation of the Google Chrome Extension Security Architecture. In *USENIX Security Symposium*.
[22] Quan Chen and Alexandros Kapravelos. 2018. Mystique: Uncovering Information Leakage from Browser Extensions. In *ACM CCS*.
[23] chrome. Accessed on 2023-07-18. Chrome Web Store review process – The basics. https://developer.chrome.com/docs/webstore/review-process/#basics.
[24] chrome. Accessed on 2023-07-18. Declare permissions. https://developer.chrome.com/docs/extensions/mv3/declare_permissions/.
[25] chrome. Accessed on 2023-07-18. Extensions platform vision. https://developer.chrome.com/docs/extensions/mv3/intro/platform-vision/.
[26] chrome. Accessed on 2023-07-18. Manifest file format. https://developer.chrome.com/docs/extensions/mv3/manifest/.
[27] chrome. Accessed on 2023-07-18. Manifest V2 support timeline. https://developer.chrome.com/docs/extensions/mv3/mv2-sunset.
[28] chrome. Accessed on 2023-07-18. Program Policies. https://developer.chrome.com/docs/webstore/program-policies/.
[29] chrome. Accessed on 2023-07-29. Content scripts. https://developer.chrome.com/docs/extensions/mv3/content_scripts.
[30] chrome. Accessed on 2023-07-29. Executing arbitrary strings. https://developer.chrome.com/docs/extensions/mv3/mv3-migration/#executing-arbitrary-strings.
[31] chrome. Accessed on 2023-07-29. Give users options. https://developer.chrome.com/docs/extensions/mv3/options.
[32] chrome. Accessed on 2023-07-29. How long will it take to review my item? https://developer.chrome.com/docs/webstore/faq/#faq-listing-108.
[33] chrome. Accessed on 2023-07-29. Manifest Version. https://developer.chrome.com/docs/apps/manifest/manifest_version.
[34] chrome. Accessed on 2023-07-29. Migrating from background pages to service workers. https://developer.chrome.com/docs/extensions/mv3/migrating_to_service_workers.
[35] chrome. Accessed on 2023-07-29. My extension has been removed from the Chrome Web Store. What should I do? https://developer.chrome.com/docs/webstore/faq/#faq-listing-10.
[36] chrome. Accessed on 2023-07-29. Welcome to Manifest V3. https://developer.chrome.com/docs/extensions/mv3/intro/.
[37] Chrome Developers. Accessed on 2023-04-28. Chrome Web Store API Reference. https://developer.chrome.com/docs/webstore/api_index/.
[38] Aurore Fass, Michael Backes, and Ben Stock. 2019. HideNoSeek: Camouflaging Malicious JavaScript in Benign ASTs. In *ACM CCS*.
[39] Aurore Fass, Michael Backes, and Ben Stock. 2019. JStap: A Static Pre-Filter for Malicious JavaScript Detection. In *ACSAC*.
[40] Aurore Fass, Robert P. Krawczyk, Michael Backes, and Ben Stock. 2018. JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript. In *DIMVA*.
[41] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. 2021. DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale. In *ACM CCS*.
[42] HyungSeok Han, DongHyeon Oh, and Sang Kil Cha. 2019. CodeAlchemist: Semantics-Aware Code Generation to Find Vulnerabilities in JavaScript Engines. In *NDSS*.
[43] Hideaki Hata, Christoph Treude, Raula Gaikovina Kula, and Takashi Ishio. 2019. 9.6 million links in source code comments: Purpose, evolution, and decay. In *International Conference on Software Engineering (ICSE)*.
[44] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. 2015. Trends and Lessons from Three Years Fighting Malicious Extensions. In *USENIX Security Symposium*.
[45] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *USENIX Security Symposium*.
[46] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *NDSS*.
[47] Soroush Karami, Faezeh Kalantari, Mehrnoosh Zaeifi, Xavier J Maso, Erik Trickel, Panagiotis Ilia, Yan Shoshitaishvili, Adam Doupé, and Jason Polakis. 2022. Unleash the Simulacrum: Shifting Browser Realities for Robust Extension-Fingerprinting Prevention. In *USENIX Security Symposium*.
[48] Debbie Kim. Accessed on 2023-07-29. Find great extensions with new Chrome Web Store badges.
[49] Igibek Koishybayev and Alexandros Kapravelos. 2020. Mininode: Reducing the Attack Surface of Node.js Applications. In *RAID*.
[50] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. 2021. Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets. In *USENIX Security Symposium*.
[51] Tobias Lauinger, Abdelberi Chaabane, Sajjad Arshad, William Robertson, Christo Wilson, and Engin Kirda. 2017. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *NDSS*.
[52] Suyoung Lee, HyungSeok Han, Sang Kil Cha, and Sooel Son. 2020. Montage: A Neural Network Language Model-Guided JavaScript Engine Fuzzer. In *USENIX Security Symposium*.
[53] Giedrius Majauskas. Accessed on 2023-07-29. Newtab Viruses - How to remove. https://www.2-viruses.com/remove-newtab-virus.
[54] McAfee. Accessed on 2023-07-29. Malicious Cookie Stuffing Chrome Extensions with 1.4 Million Users. https://www.mcafee.com/blogs/other-blogs/mcafee-labs/malicious-cookie-stuffing-chrome-extensions-with-1-4-million-users.
[55] Marvin Moog, Markus Demmel, Michael Backes, and Aurore Fass. 2021. Statically Detecting JavaScript Obfuscation and Minification Techniques in the Wild. In *Dependable Systems and Networks (DSN)*.
[56] Mozilla Developer Network. Accessed on 2023-07-29. Browser Extensions. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions.
[57] Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. 2020. You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas. In *ACM CCS*.
[58] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. 2022. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. In *S&P*.
[59] Iskander Sánchez-Rola, Igor Santos, and Davide Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *USENIX Security Symposium*.
[60] Alexander Sjösten, Steven Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *CODASPY*.

[61] Alexander Sjosten, Steven Van Acker, Pablo Picazo-Sanchez, and Andrei Sabelfeld. 2019. Latex Gloves: Protecting Browser Extensions from Probing and Revelation Attacks. In *NDSS*.
[62] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The Dangers of Human Touch: Fingerprinting Browser Extensions through User Actions. In *USENIX Security Symposium*.
[63] Konstantinos Solomos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. 2022. Escaping the Confines of Time: Continuous Browser Extension Fingerprinting Through Ephemeral Modifications. In *ACM CCS*.
[64] Dolière Francis Somé. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *S&P*.
[65] Pratik Soni, Enrico Budianto, and Prateek Saxena. 2015. The Sicilian Defense: Signature-Based Whitelisting of Web JavaScript. In *CCS*.
[66] Cristian-Alexandru Staicu, Michael Pradel, and Benjamin Livshits. 2018. SYNODE: Understanding and Automatically Preventing Injection Attacks on NODE.JS. In *NDSS*.
[67] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *WWW*.
[68] Oleksii Starov and Nick Nikiforakis. 2017. Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions. In *WWW*.
[69] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *S&P*.
[70] statcounter. Accessed on 2023-04-28. Desktop Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share/desktop/worldwide.
[71] Jiangang Wang, Xiaohong Li, Xuhui Liu, Xinshu Dong, Junjie Wang, Zhenkai Liang, and Zhiyong Feng. 2012. An Empirical Study of Dangerous Behaviors in Firefox Extensions. In *International Conference on Information Security (ISC)*.
[72] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. 2017. Ex-Ray: Detection of History-Leaking Browser Extensions. In *ACSAC*.
[73] Alex Wolkov. Accessed on 2023-07-29. Extensionizr. https://extensionizr.com/.
[74] Jianjia Yu, Song Li, Junmin Zhu, and Yinzhi Cao. 2023. CoCo: Efficient Browser Extension Vulnerability Detection via Coverage-guided, Concurrent Abstract Interpretation. In *ACM CCS*.
[75] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. 2019. Small World with High Risks: A Study of Security Threats in the npm Ecosystem. In *USENIX Security Symposium*.

A APPENDIX

Rank	Benign	Malware	Policy violation	Vulnerable
1	storage	storage	storage	tabs
2	tabs	topSites	tabs	storage
3	activeTab	activeTab	cookies	webRequest
4	contextMenus	webRequest	activeTab	webRequestBlocking
5	notifications	webRequestBlocking	management	contextMenus
6	webRequest	cookies	topSites	cookies
7	scripting	management	webNavigation	activeTab
8	unlimitedStorage	webNavigation	webRequest	notifications
9	cookies	tabs	webRequestBlocking	bookmarks
10	webRequestBlocking	unlimitedStorage	notifications	downloads

Table 5: Top 10 API-based permissions for benign and SNE

Rank	URL	Occurrences
1	http://opensource.org/licenses/mit-license.php	426,224
2	http://apache.org/licenses/LICENSE-2.0	383,040
3	http://jqueryui.com	99,984
4	http://opensource.org/licenses/BSD-3-Clause	87,712
5	http://code.google.com/p/crypto-js	65,376
6	http://code.google.com/p/crypto-js/wiki/License	65,248
7	http://opensource.org/licenses/MIT	65,056
8	http://jquery.org/license	59,552
9	http://codemirror.net/LICENSE	58,240
10	https://github.com/twbs/bootstrap/blob/master/LICENSE	57,312

Table 6: Most popular URLs and number of occurrences of a URL across all JavaScript files of all extensions (total of over 50k URLs)

Rank	Second-level domain	# unique URLs	# unique extensions
1	github.com	9,685	62,285
2	ecma-international.org	516	9,976
3	github.io	1,128	8,461
4	whatwg.org	737	7,491
5	stackoverflow.com	3,036	6,793
6	chrome.com	698	6,422
7	w3.org	870	6,413
8	google.com	1,297	6,319
9	mozilla.org	1,388	6,088
10	opensource.org	48	5,030

Table 7: Most popular second-level domains with the number of unique extensions they appear in (total of 18,551 second-level domains)

Library	Version	# extensions
jquery	3.3.1	4,744
jquery	3.4.1	3,700
jquery	3.2.1	3,326
jquery-ui	1.12.1	2,242
jquery	3.1.1	2,144
jquery-ui	1.11.4	2,015
jquery	2.1.1	1,970
jquery	2.1.4	1,921
jquery	2.0.0	1,871
jquery	1.10.2	1,808

Table 8: Top 10 JavaScript library versions containing known vulnerabilities and used by extensions

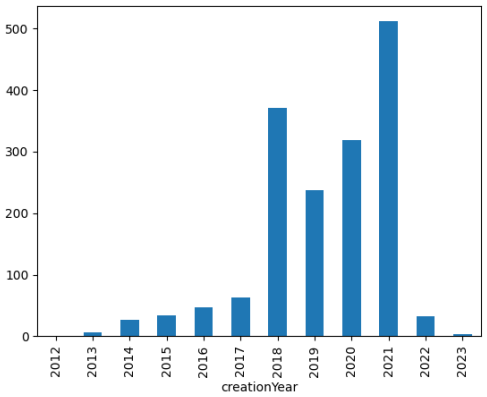

Figure 12: Number of extensions created each year using Extensionizr [73]