No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns

Pablo Picazo-Sanchez
Chalmers University of Technology
Gothenburg, Sweden
pablop@chalmers.se

Benjamin Eriksson
Chalmers University of Technology
Gothenburg, Sweden
beneri@chalmers.se

Andrei Sabelfeld
Chalmers University of Technology
Gothenburg, Sweden
andrei@chalmers.se

ABSTRACT

Browser extensions are popular small applications that allow users to enrich their browsing experience. Yet browser extensions pose security concerns because they can leak user data and maliciously act on behalf of the user. Because malicious behavior can manifest dynamically, detecting malicious extensions remains a challenge for the research community, browser vendors, and web application developers. This paper identifies download patterns as a useful signal for analyzing browser extensions. We leverage machine learning for clustering extensions based on their download patterns, confirming at a large scale that many extensions follow strikingly similar download patterns. Our key insight is that the download pattern signal can be used for identifying malicious extensions. To this end, we present a novel technique to detect malicious extensions based on the public number of downloads in the Chrome Web Store. This technique fruitfully combines machine learning with security analysis, showing that the download patterns signal can be used to both directly spot malicious extensions and as input to subsequent analysis of suspicious extensions. We demonstrate the benefits of our approach on a dataset from a daily crawl of the Web Store over 6 months to track the number of downloads. We find 135 clusters and identify 61 of them to have at least 80% malicious extensions. We train our classifier and run it on a test set of 1,212 currently active extensions in the Web Store successfully detecting 326 extensions as malicious solely based on downloads. Further, we show that by combining this signal with code similarity analysis, using the 326 as a seed, we find an additional 6,579 malicious extensions.

CCS CONCEPTS

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

KEYWORDS

Web Security; Browser Extensions

ACM Reference Format:
Pablo Picazo-Sanchez, Benjamin Eriksson, and Andrei Sabelfeld. 2022. No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns. In *Annual Computer Security Applications Conference (ACSAC '22), December 5–9, 2022, Austin, TX, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3564625.3567988

ACSAC '22, December 5–9, 2022, Austin, TX, USA
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9759-9/22/12.
https://doi.org/10.1145/3564625.3567988

1 INTRODUCTION

Browser extensions are popular small web applications that users install in modern browsers to enrich the user experience on the web. Google's official extension repository, Chrome Web Store, currently has more than 180,000 extensions between browser extensions, apps, and themes, with many extensions having millions of users. Driven by the popularity of Chrome extensions, browser extension ecosystems have been adopted not only by Chromium-based browsers like Opera, Brave, and Microsoft Edge but also by browsers like Firefox and Safari. The latter browsers draw on the same architecture, allowing developers to export their Chrome extensions easily. When an extension is installed, the browser typically sends a message showing the permissions this new extension requests. The extension is installed and integrated within the browser upon user approval.

The benefits of using browser extensions come at the high price of granting access to a vast amount of sensitive information. Extensions get and interact with all the content of users' web pages. Also, suppose the extension defines the corresponding permissions. In that case, it can run some of the restricted APIs the browser exposes to extensions to retrieve sensitive information such as cookies, history and even modify the network traffic without the user's knowledge. This raises serious security and privacy concerns [55, 56, 71].

Chrome Web Store. Extensions are usually stored in private repositories managed by vendors, where extensions developers upload them to be freely distributed afterward. The most popular browser extensions repository is the Web Store governed by Google, which banned the possibility of manually installing browser extensions from other sites different than the Web Store years ago [12].

The Web Store implements a Collaborative Filtering Recommendation System (CFRS) [25] in such a way that extensions are ranked or featured to make it easier for users to find high-quality content. This ranking is performed by a heuristic that considers user ratings and usage statistics, such as the number of downloads and uninstalls over time.

Inherent to CFRS, attackers have always been trying to promote or demote apps by automatically modifying ratings and raters [10, 45], or faking the downloads [8, 18]. Also, the proliferation of crowdsourcing sites like Zeerk, Peopleperhour, Freelancer, Upwork, and Facebook groups, have helped on this matter [44]. Among other things, by boosting some apps, developers may get funding from venture capitalists when their apps are popular among users [24].

The Web Store implements a set of fraud detection and defense mechanisms so that attackers cannot alter the ranking that easily [45]. Similar to Android Google Play, users can only review and rate an extension only if they 1) are logged in to the Web Store, and;

2) install it first, being easier for Google to detect fake users trying to exploit the CFRS. However, this is not the case with downloads. To download and install extensions, users need a Chromium-based browser, e.g., Chromium, Chrome, and Brave. Therefore, the number of downloads can be easily altered by automatic processes, being difficult to differentiate between real users and automatic downloads. In this paper, we are particularly interested in how the downloads of the extensions can be used for grouping browser extensions based on the download patterns and identifying malicious ones based on such patterns.

Extensions' Downloads. We monitored the number of downloads of browser extensions over 6 months and observed that the function defining the number of downloads is monotonically increasing over time for most extensions. However, there is a remarkable number of extensions whose downloads: i) do not follow an organic download pattern, i.e., they are synced with others, following the same pattern; ii) experience many fluctuations thus not following a monotonic function, and; iii) deviate from the usual growing pattern, i.e., they grow and/or decrease various orders of magnitude within two or three days. This leads to the insight that the number of downloads of the extensions can be leveraged as a useful signal for analyzing browser extensions. Hence, we pose three research questions:

RQ1: Are there extensions that follow similar download patterns?
RQ2: Is there any relationship between download patterns and malicious code?
RQ3: Can we find malicious extensions based on their download patterns?

To answer these questions, we crawled the Web Store daily for 171 days and analyzed the download patterns of over 160,000 extensions. We clustered the extensions concerning such patterns and found 135 clusters. Later, we analyzed the security of the extensions that compose these clusters and identified 61 of them to have at least 80% malicious extensions. Using a supervised learning algorithm, we trained two classifiers and evaluated them against 1,212 currently active extensions in the Web Store. The first classifier predicts which cluster the test set extensions are in in the training set. Afterward, a threshold is used to mark all extensions in a cluster as either malicious or benign based on the fraction of malicious extensions in the cluster. The second classifier directly predicts if an extension is malicious or benign. The first classifier successfully detects 326, and the second detects 289 extensions as malicious solely based on downloads. The classifiers consider both the pattern and the security labels, meaning any pattern is not an indication of maliciousness, it must closely match malicious patterns. Extensions that receive fake downloads, also known as astroturfing, are only marked as malicious if they match a malicious pattern. While the classifier can find malicious extensions alone, it can also be beneficially combined with other methods. To this end we combine the download pattern signal and code similarity analysis to discover an additional 6,579 malicious extensions. In this case, the code similarity approach needs malicious seed extensions and would not find any malicious extensions without the extensions from the download signal.

Contributions. In detail, our contributions are:

- We describe the methodology we use to retrieve and analyze the data from the Web Store (see Section 3);
- We show our results, supporting that *RQ1)* Extensions follow similar patterns. *RQ2)* These patterns can be correlated to maliciousness *RQ3)* We can find new active malicious extensions based on the download patterns. (see Section 4);
- We present a set of 29 malicious extensions whose downloads are all synced and hijack the search queries of the users, in combination with code analysis we discover 6,579 extra hijacker extensions that remain hidden in the Store (see Section 5).

We introduce some basic definitions for an easy understanding of the paper in Section 2, discuss the threats to validity in Section 6, offer a summary of the most relevant related work in Section 7 and conclude the paper in Section 8.

Coordinated disclosure. We reported to Google 6,579 the malicious extensions detected in our empirical study as well as our methodology. The Chrome Web Store team removed 4,858, while 1,721 are still under investigation.

Artifacts. We open-source our code and data needed to reproduce the results presented in this paper [40].

2 PRELIMINARIES

In this section, we summarize the security and privacy threats that extensions pose, some basic concepts of time-series, some definitions we use in the paper and introduce the threat model.

2.1 Browser Extensions' Security & Privacy

Browser extensions are small applications that can help developers and users develop new web applications or surf the Internet. However, due to the amount of sensitive information the extensions have access to when they run in the users' browsers, the security and privacy of such data have cast doubt on adopting the extensions.

Extensions are composed of two main parts, content scripts and background pages. The former are scripts automatically injected into the web pages the extension defines in the manifest file under the `content_scripts` key. On the other hand, background pages are scripts with no direct access to the web content but with access to a set of restricted and privileged APIs the browser exposes, e.g., network traffic, cookies, and history. To access these APIs, the extension has to define the permissions associated with every API it attempts to use in the manifest file.

2.2 Time-Series Analysis

A time series is a set of data points ordered by time. There are different metrics to compare two time series, i.e., how similar they are, usually based on the data points' distance of the series. The most common examples are the Euclidean distance, the Longest Common Subsequence (LCS), and the Dynamic Time Warping (DTW), being this last one the most common distance used to compare time-series [17]. In addition, Canonical Time Warping (CTW) is a method based on Dynamic Time Warping (DTW) that aligns time series under rigid registration of the feature space, not being needed that time series share the same size nor the same dimension.

Learning in Machine Learning (ML) can be classified into two main families: supervised and unsupervised. Supervised learning needs labeled data to learn the mapping function from an input to an output, whereas in unsupervised learning algorithms, there is no labeled data; therefore, they learn patterns from the input data. Research in supervised and unsupervised learning algorithms applied to time series is quite active nowadays.

Clustering is an unsupervised learning technique by which similar data are grouped with little or no knowledge in advance about the data. Time-series clustering is a particular case where the series, a large number of points measured chronologically, are handled as single objects to extract patterns among them [3]. Examples of algorithms used for time-series clustering are Self-Organizing Map (SOM) [3], k-means [27], and k-shapes [38].

Classification is a supervised learning technique by which an algorithm analyzes a training dataset and outputs function used for determining the labels of new examples. Some examples of algorithms for time-series classification are KNeighbors [58], Support Vector Machines (SVM) [29], Rocket [15], and Minirocket [16].

2.3 Definitions

In the following, we explain in detail each one of the key concepts used throughout this paper.

Downloads. The number of downloads the Web Store publicly exposes for every extension (e) at time t.

Increment of downloads (Δ_{d_e}). Is the difference, in absolute value, of two consecutive downloads of an extension e, i.e., $\Delta_{\mathrm{d}_e} = |t_i - t_{i+1}|$.

Average of the increment ($\overline{\Delta_{\mathrm{d}_e}}$). Is the arithmetic mean of the increment of the downloads, i.e., $\overline{\Delta_{\mathrm{d}_e}} = \frac{\sum_{i=1}^{n} \Delta_{\mathrm{d}_e}}{n}$, where $i < n$ and n is the number of measurements per extension.

Average of the average of the increment ($\overline{\overline{\Delta}}$). Is the arithmetic mean of all the average of the increment of the downloads of the extensions, i.e., $\overline{\overline{\Delta}} = \frac{\sum_{i=1}^{e} \overline{\Delta_{\mathrm{d}_i}}}{n}$, where $i <= n$, n is the number of extensions and $\overline{\Delta_{d_i}}$ is the average of the increment of an extension i.

2.4 Threat Model

Extensions can pose many threats to users' privacy and security. Previous works have analyzed extensions that inject adware, track and fingerprint users, takeover search engines, modify security headers, execute remote code, persuade and steal user's search queries [4, 11, 20, 33, 37, 50–52, 54–56].

In this work, we focus on a subset of these attacks: the search query stealing attack. This is prominently used by "Wallpaper" extensions user's search queries [20]. These extensions override the new tab functionality of the browser such that when the user opens a new tab, this is replaced by the one the extension provides. They usually provide a search bar with some arbitrary wallpaper backgrounds. However, these extensions can redirect search requests containing sensitive queries and redirect them to different URLs (see Table 5). Generating the security ground truth for extensions is a huge challenge and requires manual effort. We can implement efficient and accurate methods for this analysis by focusing on one class of attacks (see Section 3.2. Simply reusing previous analysis methods would be futile as Google now knows these and can remove the corresponding malicious extensions.

3 SCRUTINIZING THE WEB STORE

A Collaborative Filtering Recommendation System (CFRS) is a system that keeps track of the users' preferences to use it afterward to offer new suggestions to other users [35]. Youtube, Amazon, and Netflix are examples of applications that implement CFRSs [47, 66]. This is also the case with the Google Web Store, the online marketplace where browser extensions are freely distributed. The Web Store implements a CFRS where extensions are ranked based on parameters like user ratings, number of downloads, and uninstalls over time.

Even though the algorithms used by the CFRS are usually unknown, researchers found attacks against the recommendation system, being pollution attacks the most common ones [23, 47, 66]. Such attacks consist of generating fake data, typically in the form of new users who interact with the system by watching videos, reading books, and rating or downloading items. By doing so, attackers may promote or demote items as desired.

Some of the information the Web Store offers for each browser extension is the category it belongs to, the name of the developer, the company, a general description, some privacy practices, users' reviews, the number of downloads, the rates that users give or metadata like the version of the extension, and when it was updated.

This section presents the methodology we follow to identify download patterns as a useful signal for analyzing browser extensions. We leverage machine learning for clustering extensions based on these patterns, confirming at a large scale that many extensions follow strikingly similar download patterns (see Section 4).

We split our methodology into three main tasks (see Figure 1):

Data Gathering Daily monitoring of the Web Store to extract downloads of all the browser extensions;

Security Analysis We combine static, manual, and dynamic analysis to mark malicious extensions, and;

Time-Series Analysis Firstly, group extensions according to the downloads function they describe (based on Δ_{d_e}) and look for patterns (clustering phase) and label them based on the security analysis. Secondly, we implement a learning algorithm based on the downloads.

3.1 Data Gathering

Between March 2021 and Aug 2021, we crawled the Web Store daily, monitoring the extensions' downloads (see Appendix A) and their version. At the time of writing, there are 10,941 extensions whose downloads are not shown on the page. Unfortunately, we do not know why some are hidden. From all the download patterns, we extract all the extensions whose $\overline{\Delta_{\mathrm{d}_e}}$ is larger than $\overline{\overline{\Delta}}$. These are the extensions whose downloads fluctuate more than the global average in the store.

Dataset Filtering. After the extraction of all the public data of the extensions, we filtered the dataset in terms of size (number of extensions) and data information (number of measurements). This allowed us to perform a more accurate security analysis in Section 3.2 and better clustering in Section 3.3. This filtering process

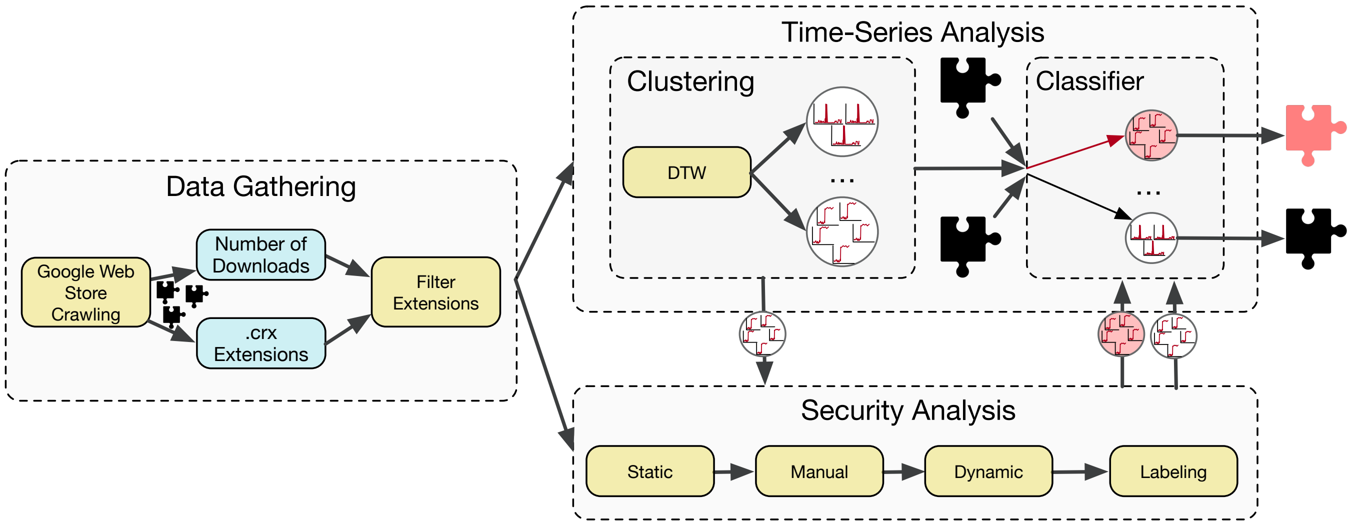


Figure 1: Systematic methodology to cluster and extensions.

neither affects the results nor the methodology presented in this paper. With more manual effort, in terms of time spent on security analysis and clustering, and increasing the frequency of the data gathering, we believe this method can be extended to any group of extensions and attacks.

First, we focused on a subset of extensions called wallpapers, i.e., browser extensions that override the starting page the user previously had and replace it with a random background image that changes every time the user refreshes the webpage and a search box in the middle of the screen. We did so because researchers recently showed that many extensions attempt to steal users' search queries [20]. Wallpapers can be found in the 11 categories of the Web Store. To get all the wallpapers, we filtered out extensions that do not define `chrome_url_overrides` in their manifests. Second, we analyzed those extensions with more than 90 measurements, which, given our crawling frequency, corresponds to at least 90 days.

3.2 Security Analysis

In accordance with our second research question *Is there any relationship between download patterns and malicious code?* We need to perform a security analysis of the extensions to label them as malicious or benign. Using these security labels we can search for relationships between clusters and malicious code.

While extensions can perform many possible attacks, we focus on malicious extensions that change the user's search engine without notice or steal their queries. This can be accomplished either by redirecting the search or using analytics. To detect this, we develop a fully automatic dynamic analysis method and verify it using a combination of static and manual analysis (see Appendix D).

3.3 Time-Series Analysis

Ideally, after 171 days of daily monitoring, we should have gathered 171 measurements per extension. However, extensions can be deleted or added, thus affecting the number of downloads collected

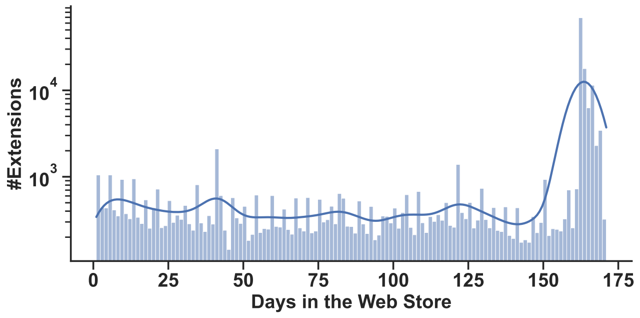


Figure 2: Number of days (x-axis) monitoring the extensions (y-axis in log scale).

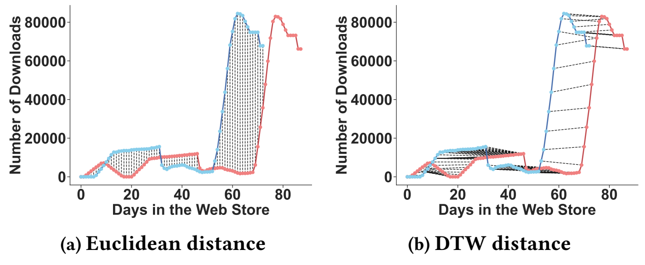


Figure 3: DTW vs Euclidean distance applied to extensions downloads time-series.

per extension. We show in Figure 2 the distribution of how long the extensions in our dataset were online in the Web Store.

Clustering. To answer our first research question (**RQ1**): *Are there extensions that follow similar download patterns?* We apply state-of-the-art clustering methods. The results from this clustering will illuminate if there are any clusterable download patterns.

Given the heterogeneous distribution of the data, i.e., extensions are alive in the Web Store for different periods, we could not implement classical clustering methodologies based on euclidean distances like DBSCAN or other statistical values such as mean like K-means. The reason is that to compare two time-series using the euclidean distance, both series need exactly the same amount of data as well as being synchronized (see Figure 3a). Instead, we adopted a well-known technique in time-series clustering named Dynamic Time Warping (DTW) that solves the aforementioned constraints by computing a discrete matching between the elements of both series rather than using their time sequence [3] (see Figure 3b).

In this paper, we follow a so-called Human-in-the-Loop (HitL) methodology [65] combined with DTW to cluster time-series downloads of browser extensions. To do so, we deploy an instance of dtadistance library [34] combined with COBRAS-TS [62], an interactive version of COBRA [13] that allows semi-supervised clustering of time series. However, this process can be fully automatized without including humans in the clustering algorithm.

Classification. To answer our third research question: *Can we find malicious extensions based on their download patterns?* We create two classifiers that aim to classify extensions as malicious solely based on download patterns. The first one classifies directly based on download patterns, while the second cluster similar patterns before classifying each cluster as malicious or benign. The second predicts which cluster from the training set the extensions in the test set are closest to. Finally, we compare a threshold t to the fraction of malicious extensions in the cluster. We mark the extension as malicious if t is greater than this fraction.

We implement and evaluate an instance of MiniRocket [16]. We split our extensions dataset into training and test sets. To simulate a realistic scenario of our approach, we use all extensions that had been deleted at the end of the data-gathering phase as the training set and the still-active ones as our test set.

We evaluate our model according to three main metrics: precision, recall, and F1-Score. Precision measures how many positive predictions are true, i.e., $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$. Recall measures how many positive classes the model can predict, i.e., $\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})$. Finally, F1-Score is the harmonic mean of both recall and precision, i.e., $2(\mathrm{recall}\cdot\mathrm{precision})/(\mathrm{recall}+\mathrm{precision})$.

4 RESULTS

This section presents the results from our data gathering, security, and time-series analysis. We use these results to answer our research questions.

4.1 Data Gathering

After 171 days of monitoring, we collected download patterns for 159,572 extensions. Figure 4 shows the distribution of the average of the increments of the downloads ($\overline{\Delta_{d_e}}$) of all the extensions of the Web Store. Interestingly, we can see that there are many outliers, i.e., extensions whose $\overline{\Delta_{d_e}}$ is higher than 10, 1,000, or even 10,000 downloads. We marked with a green triangle in Figure 4 the average of $\overline{\overline{\Delta}}$ of the extensions (around 97.6). Even though we could have analyzed all the extensions of the Web Store, we restricted ourselves to the 8,165 outliers extensions whose average of $\overline{\Delta_{d_e}} \geq 97.6$.

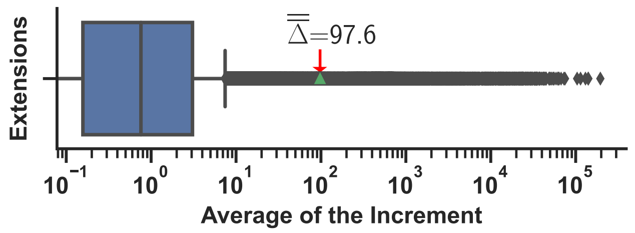


Figure 4: Distribution of the average of the increment of downloads ($\overline{\Delta_d}$). The $\overline{\Delta_d}$ of 8,165 extensions is higher than the average ($\overline{\overline{\Delta}}$).

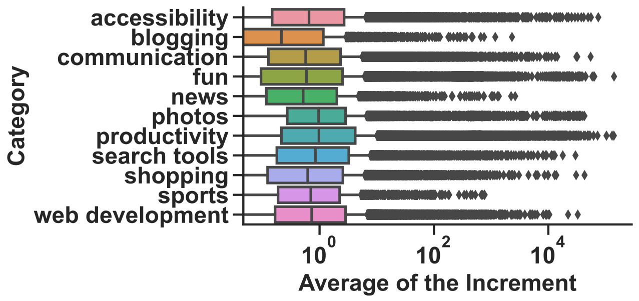


Figure 5: Distribution of the average of the increment of downloads group by the 11 categories of the Web Store.

In Figure 5 we split the dataset into the categories the extensions belong to. We also extracted the last public downloads the Web Store offered per extension and include in Figure 10 (Appendix B) the download distribution of extensions split into categories. Although there are some extensions with millions of downloads, in general, we can observe that most of the browser extensions have been downloaded less than a hundred times, with even fewer downloads in some categories, including "blogging".

In summary (see Figure 6), we collected 159k download patterns from the Web Store. From these, 35k are wallpaper extensions, where 22k are still active, and 13k are deleted. From them, we first filtered extensions with interesting download patterns, (i.e., $\overline{\Delta_{d_e}} \geq 97.6$), getting 1,629 and 2,673 still-active and deleted wallpapers respectively. Finally, because of our crawling frequency (once a day), to increase the useful information of the downloads and thus reduce the false positives, we analyze the downloads of the extensions that remained in the Web Store longer than 90 days, resulting in a total of 1,212 and 2,059 alive and deleted extensions. We use these 3,271 wallpaper extensions for security analysis and clustering.

4.2 Security Analysis

This section presents the results of our automatic security analysis, where we find 1,292 malicious extensions. These labels are used in Section 4.3 to answer our second research question: *Is there any relationship between download patterns and malicious code?*

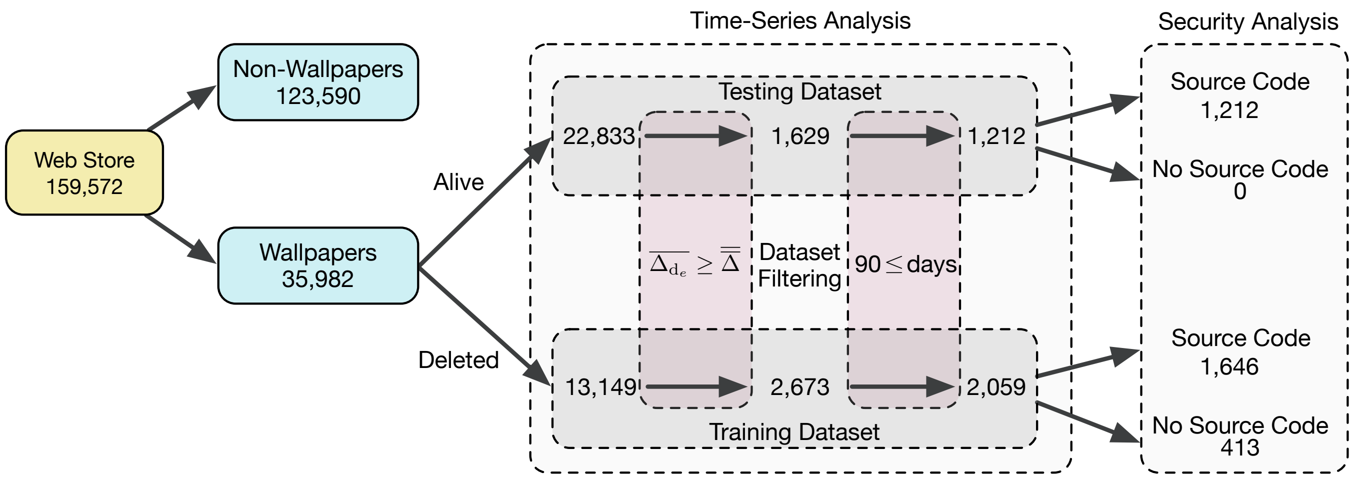


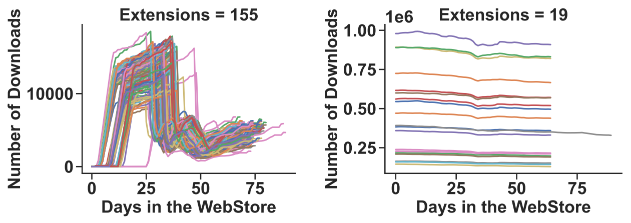
Figure 6: Filtering Process. Extensions on every step.

Table 1: Popular domains used by query stealing extensions.

Domain	#Extensions
cse.google.com	146
mc.yandex.ru	134
gundil.com	116
cors-anywhere.herokuapp.com	100
www.google-analytics.com	92
completion.amazon.com	60
s.bingparachute.com	42
addiyos.com	16
the-theme-factory.com	14
chromethemesonline.net	11

Table 2: Domains scanned in Phase 2.

Domain	Malicious?	#Extensions
www.tabhd.com	Yes	667
www.ultitab.com	Yes	184
themes.wallpaperaddons.com	No	1



(a) **TabHD extensions** (b) **MyWay extensions**

Figure 7: Download patterns for two malicious clusters.

We dynamically executed and analyzed 2,858 extensions, which is the number of extensions we had the source code for (see Figure 6). For the remaining 413 extensions, we marked them as benign since we can not prove they are malicious.

Scanning extensions. We first analyzed extensions that immediately stole queries instead of waiting before stealing them. Here we found 441 malicious extensions stealing search queries. These use a combined total of 182 different domains for their query stealing. However, one extension can use multiple domains, e.g., one extension[1] uses search.myway.com for searching while simultaneously using Google Analytics to log the query.

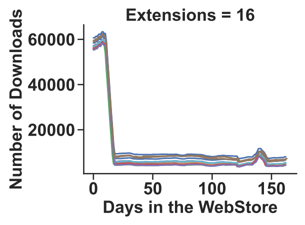
We present the ten most used domains in Table 1. Note that these domains are not necessarily malicious but are used by malicious extensions. For example, cse.google.com is not malicious but is commonly used by spyware [14].

Scanning websites. To detect delayed attacks, we also analyzed the websites used by extensions marked as benign. We found three domains used by 852 extensions (see Table 2). We analyzed each for an hour and detected both www.tabhd.com and www.ultitab.com switch from benignly using Google to maliciously using gundil.com. themes.wallpaperaddons.com was consistently using Google and was therefore marked as benign. Combining the two scans, we found 1,292 malicious extensions. Finally, we verify these labels by manually checking a sample of 100 extensions labeled as benign. Three out of these were incorrectly marked as benign. We discuss the general limitations more in Section 6, and more details about the verification can be found in Appendix E.

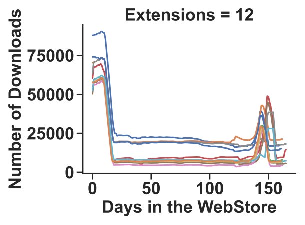
4.3 Time-Series Analysis

Clustering. This section presents the results of our clustering and how it relates to our first research question (**RQ1**): *Are there extensions that follow similar download patterns?* After clustering

[1]bcdhacjdengeibbbhmdjodiecaiciehc



(a) FreeAddon extensions



(b) FreeAddon extensions

Figure 8: Download patterns for two benign clusters.

Table 3: Distribution of malicious clusters versus the extensions they are composed of.

	Maliciousness					#Total
	0%	0%-50%	50%-80%	80%-100%	100%	
#Clusters	52	15	7	6	55	135
#Extensions	902	1130	299	401	539	3,271

the extensions based on their download patterns, as explained in Section 3.3, we find a total of 135 clusters composed of 3,271 extensions, with an average of 24 extensions per cluster and 39 clusters with more than 10 extensions (see Table 3). We show four examples of these clusters in Figures 7 and 8 whereas we include more examples in Appendix F.

Note that the cluster in Figure 7b might seem inactive, but around day 30, a slight drop corresponds to hundreds of thousands of downloads. Another example is the cluster in Figure 8b, where the extensions are inactive for almost 100 days before all gaining many downloads simultaneously. These clear patterns confirm that there exist extensions that follow similar download patterns.

Malicious Clusters. To answer (**RQ2**): *Is there any relationship between download patterns and malicious code?* We further analyzed the 135 clusters and found that similar clusters have similar attack patterns. For example, in one cluster with 155 extensions, 124 used tabhd.com to steal queries (see Figure 7a). We did not have the source code of 30 out of the remaining 31 extensions in that cluster and therefore marked those as safe (we give more details in Section 6). Using an unofficial repository [21] we could confirm that those 30 extensions also used tabhd.com. While, in another cluster, 13 out of 19 extensions used search.myway.com instead (see Figure 7b). In this case, we miss the code of 5 extensions. Similar to malicious clusters, Figure 8a shows two clusters of 16 and 12 extensions are all benign, all from FreeAddon.

Based on these results, we can confirm that there is a relationship between download patterns and malicious code in extensions. Furthermore, in addition to a correlation between download patterns and maliciousness, we also find some cases where the download pattern correlates directly with the specific details of the attack, for example, the use of tabhd.com as a website.

Classifier. This section presents the results relating to our final research question (**RQ3**): *Can we find malicious extensions based on their download patterns?* First, we present the result of using

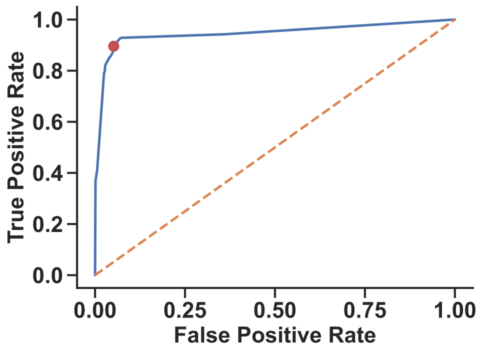


Figure 9: ROC curve of our classifier. The red circle marks the best F1-Score (0.89), correctly classifying 326 out of 382 malicious extensions.

the classifier to predict which cluster an extension matches then we present the fully automatic solution where security labels, i.e., malicious or benign, are directly predicted.

We train our classifier on download patterns for 2,059 extensions and use 1,212 as test. Our classifier will first match each extension in the test set with a cluster from the training set then, based on a threshold t, determine if the extension is malicious.

Figure 9 depicts the ROC curve of our classifier. In it, we see how the threshold t affects the recall and precision of our classifier, achieving the best F1-Score of 0.89 at $t = 0.26$, with a precision and recall of 0.88 and 0.90. At this threshold, the classifier finds 326 out of the 364 malicious extensions in the test set demonstrating that we successfully find malicious extensions solely based on their download patterns.

Directly predicting labels results in an F-score of 0.89 with a precision of 0.92 and a recall of 0.80. This is slightly worse than predicting clusters with the optimal threshold. We believe this is because the semi-automatic clustering is better than the fully automatic solution, which translates into better security label classification. Solely predicting labels also fails to capitalize on the fact that there are cases where download patterns correlate with code or attack similarities.

5 USE CASE: SEARCH HIJACKING WALLPAPERS

In this section, we perform an in-depth analysis of a group of active malicious extensions we found using our classifier. We further analyze the code and structure of these extensions. Initially, the extensions seem harmless as they do not ask for any permission nor have they any known harmful files. In Listing 2, we include the manifest file of these 29 extensions, having all of them the same manifest but with different icon (`<iconFileName>`), background page (`"js/<name>.js"`) and version (`<version>`).

However, when we automatically installed them all, we realized that all of them overwrite the `"newtab"` property, automatically loading the output.html web page whose content is `<head ><script src="js/main.js"></script><head>`, i.e., this files just loads the main.js file. Interestingly, such a JavaScript file just has

Table 4: Servers where the 29 extensions redirect users to.

URL	#Extensions
https://www.ultitab.com	19
https://www.tabhd.com	10

one line: `document.location='<url>/<name>';` where `<url>` is the server URL and `<name>` is usually the name of the extension.

When accessing any of these URLs (https://www.tabhd.com or https://www.ultitab.com), the servers automatically generate a random name and redirect users to https://<url>/<name> where `<name>` is the name of a random wallpaper extension. The content of all the URLs is the same, i.e., there is a search bar in the center and some links to the privacy policy, settings, or user agreement.

The key part is the search bar that automatically redirects the user's search queries to Google during the first 30 minutes. After that, the web server provides almost the same content, but the queries the users introduce in the search bar are redirected to https://gundil.com/index.html?q=<query> instead. Such a website looks identical to Google, but the first 4 entries are always advertisements. We also certify that this website tracks users by retrieving information such as the IP address, browser type, browser version, the pages that users visit, the time and date, and the time spent on those pages, unique device identifiers, and other diagnostic data.

5.1 Wallpapers Discovering

To discover more malicious extensions utilizing the same attack, we statically analyzed all the wallpapers (i.e., 35,982) stored in the Web Store looking for extensions that, similar to the analyzed ones: 1) implement `chrome.browserAction.onClicked.addListener()` and `chrome.runtime.onInstalled.addListener()` event listeners and the `chrome.tabs.create()` function in them; 2) add the `newtab` property of the `chrome_url_overrides`, and; 3) include any URL in their scripts.

We classified extensions into 38 groups according to the URL they include and visited each one of the websites as stated in Section 3.2. The look & feel of all the 38 sites were similar: apart from having some icons for social networks and some other shortcuts, all of them provide a search input similar to Google and Bing pages.

The most used URL by the extensions is https://mytab.me/<alias>, where <alias> is a string that can be found in the app.js file. As an example, the app.js of the "Burst Gyro Anime Anime New Tags Hot HD Themes"[2] extension is `window.app={domain: 'https://mytab.me', name: 'anime1-6'};` and the redirection is carried out in the index.js (`(() => {location.href = `${app.domain}/${app.name}/`;})();`), script that is loaded by the HTML file that overwrites the `"newtab"` property of the browser. Hence, redirecting the user to an external server whenever she opens a new tab. In Listings 3 and 4 we present the background and content script that 2,554 and 3,637 extensions using https://mytab.me have in common respectively. The background script of the remaining 1,083 extensions is the same but without the last `chrome.runtime.setUninstallURL()` statement.

[2]kidpaijejhfcmlceagcmkmcpbfmclhfi

Table 5: Top 10 of the most used URL by extensions to hijack search from users.

URL	#Extensions
https://mytab.me	3,637
https://www.tabhd.com	857
https://www.ultitab.com	216
https://chromethemesonline.net	153
https://www.searchcapitol.com/	137
https://pimp-up-your-browser.com	127
https://the-theme-factory.com	124
https://epic-chrome-themes.com	121
https://chrome-themes.online	89
https://tab-e-licious.com	86

We realized that some web pages initially use a legitimate search engine like Google or Bing. However, after some time (between 1 to 30 minutes) they redirect the search queries to other search engines like https://gundil.com/index.html?q=<query> and https://str-search.com/results.php?q=<query>. Also, similar to the extensions we detected by analyzing the downloads, they extract personal metadata to fingerprint users as well as adding custom ads.

In addition to that, a few of them perform a double redirection, i.e., they first redirect the query to an external server and redirect the user later to a legitimate search engine, being almost unnoticeable to the user. Let us give two examples, one using Google Chrome and another one using Bing.

Cute HD Wallpaper New Tab[3] is an extension that replaces the `"newtab"` by opening a new website (https://qtab.io/cute81/). When users introduce a search query in the input bar, the server first redirects the user to another server (inifinitynewtab) that hijacks the user's search query and other information before finally redirecting to Google Chrome. Similarly, Breaking News Tab[4] extension replaces the `"newtab"` web page with another one provided by the extension itself. The difference is that this time the search queries go to another server (https://www.searchcapitol.com). That server redirects the user to Bing instead once the query is received.

In Table 5, we included the Top-10 of most used URLs by extensions to hijack the search queries after performing the discovering strategies previously explained. Finally, we automatically installed all the extensions we found and checked whether they indeed opened the web pages we found when the user opened a new tab.

In summary, we analyzed the downloads pattern of browser extensions and detected 1,292 malicious ones that hijack users' searches. We combined this method with Dedup.js to detect similar extensions that were not analyzed because their increment of downloads did not pass the filter (what we call "*sleeping*" extensions), and we found 5,288 extra malicious extensions, totaling 6,579.

6 DISCUSSION & LIMITATIONS

Web Store Downloads Granularity. In September 2021, Google changed the way the Web Store shows the information affecting the

[3]ieclinianmfccihifhicbaofnkhndamd
[4]jgginkfhlcakpkjfkkbbcnjpeoladhih

granularity of the number of downloads of the extensions. Numbers are no longer as precise, e.g., the same number of downloads of the extension used in Listing 1 is now represented as "5,000+ users". Although the analysis presented in this paper is based on the precise number of downloads that Google used to offer, we believe that if our methodology was deployed by vendors, then precise numbers of the downloads would be used. Since they do not need to crawl the store, they could increase the sampling rate to potentially detect malicious extensions faster.

Adaptive attackers. Adaptive attackers can try to match benign patterns. Similar to previous work in this field [37], in our case, attackers would need the downloads of all the extensions to learn these patterns. This is currently hard as Google removed exact download details. Although there are promising advances in adversarial ML attacks in malware detection [41], this is a research topic itself and not the core of this paper. Adaptive attacks become more difficult if there as an asymmetry between the granularity of the defender's and attacker's data. This is arguably the case here where only Google has exact download statistics.

Reasoning about patterns. Unfortunately, knowing the reason why the download patterns of extensions are similar is not trivial. Google explicitly states that it implements a heuristic algorithm using parameters like ratings, usage statistics like the number of downloads and uninstalls over time, and fuzzier factors like design [25] to facilitate users to find high-quality content. As mentioned in the introduction, crowdsourcing sites like Zeerk, and Facebook groups are dedicated to increasing the popularity of applications in repositories like the Web Store. Hence, the modification of the number of downloads directly affects such an algorithm, thereby achieving more potential victims (users) who install them. Therefore these patterns become useful signals when malicious developers are artificially increasing the downloads on their extensions. However, this is hard, if not impossible, to prove for us given that we do not have access to such information (only Google might know that).

Dynamic analysis. While it is unlikely to get false positives using dynamic analysis, we do find cases of false negatives. We identify three main limitations in our analysis that result in false negatives. These are 1) installation wizards; 2) HTML elements blocking the search bar, and; 3) state destruction.

Some extensions use installation wizards where the user can fill in preferences and import data before being able to use the extensions. Our dynamic analysis does not attempt to solve these wizards in general. Therefore it can fail to reach the search bar and test it for query stealing. Using our manual and static verification analysis, we identified a group of 27 extensions that used tabturbo.com for stealing queries. However, due to a multi-step installation wizard, we were not able to reach the search bar.

Another general limitation is the invisibility or obstruction of the search bar. For example, pricehelpers.com requires the user to click a drop-down menu before being able to search. However, our dynamic approach does not find the drop-down menu needed for the search. The opposite problem is when extensions use modal windows to obstruct the search bar until the user closes the modal window. Our approach could handle this in many cases since many of the modal windows did not cover the entire search bar. In the case of total obstruction, our method would fail. An example of search bar obstruction is Naruto New Tab Page Top Wallpapers Themes[5]. Interestingly, we input DeDup.js with the files of this extension, and it detected other 3,637 malicious extensions using mytab.me as an external web server to steal users' queries.

Finally, we use a best-effort approach to find the search bar on the new tab page. If multiple text inputs have different functionalities, our approach will only test the first one. We did find one such case where an extension had one search bar for Gmail where it did not steal the query and a second search bar for Google where it did steal the query. In that particular case, we only checked the first search bar. To improve this, the extension, or the new tab page, should be analyzed once per input element.

As with any "potentially unwanted programs", the *potentially* part could result in false positives. If users truly want a search engine other than the major ones or if they want their queries to be sent to multiple servers in addition to their regular search engine. To allow for flexibility, we make it easy to change our code or relabel the data later, to add or remove servers considered safe.

Missing Code. There were 413 extensions for which we could not collect the source code before Google removed them from the Web Store. In these cases, we followed the presumption of innocence and considered the extensions benign. If we could have analyzed their code, our false positive rate would likely have decreased.

Missing Attacks. In this paper, we focus on the attack of stealing queries. This means we might mark an extension as safe even though it performs other malicious actions, like injecting advertisements on pages. As we already find malicious clusters that share code-specific parts, such as the URLs used, we believe our analysis is strong enough to find malicious clusters. By including more attacks in the security analysis, we might have found more clusters for other attacks that are now marked as safe instead.

Code Similarity. While code similarity is useful in many cases, it also has limitations. Indeed, there are cases where our pattern analysis is more effective. One reason is that benign and malicious extensions can have very similar code. For example, the safe extension *jabbaohcijedbmkbdjldjicnohlcdkdp* and the malicious *dbkbnddmcjkjkclnlpagncoebgfaoile* both have a `main.js` file where only 4 out of the 391 lines differ. The main difference here is the URL used by the extension. We can also use techniques based on hash files like DeDup.js [39]. Here, main.js does not match, but other scripts, CSS, and images are still the same in both. There have been multiple earlier cases where malicious authors copy benign code and change a small part to make them malicious[6]. By focusing on download patterns, we can distinguish these as two different clusters.

Another reason is that malicious extensions can have a small footprint by relying on external websites, i.e., they do not need dangerous permissions or powerful API calls to steal queries (e.g., [2, 28, 37]). In fact, the previously mentioned malicious extension (dbkbnddmcjkjkclnlpagncoebgfaoile) does not define a single permission. This makes them hard to detect by traditional methods.

[5]bemmphgeeoaljepcaficneogmlndijbi
[6]https://freeaddon.com/warning-adware-virus-distributors-are-making-fake-extensions-based-on-freeaddon-sportifytab/

7 RELATED WORK

We can differentiate between white-box [68, 69] and black-box attacks [22, 53, 64] against CFRSs. In white-box attacks, attackers have some knowledge about the recommendation algorithms or how the data (users/items) are related. This knowledge can be gathered by, for instance, interacting with the system, similar to what users can do. On the contrary, in black-box attacks, attackers know nothing about the recommendation system or the implemented algorithms. Examples of real attacks on CFRSs, and more concretely on Android Google Play have been proven to be successful [10, 44–46] altering the rating and the number of raters.

Recently, Dou et al. [18] deployed a honeypot in App Market—an alternative non-official Android store—to track the number of downloads of the apps, categorizing the download fraud problem in mobile App markets. In comparison to our proposal, we proved that the download fraud problem in browser extensions can be directly associated with security attacks and showed how the download pattern can be used to detect malicious extensions.

Browser Extensions. In recent years, browser extensions have attracted the research community's interest. In 2015, Google engineers [28] claimed they catch 70% of the malicious ones within 5 days, whereas some extensions can remain months or even years without being detected [5].

A few examples of how private researchers in the *industry* help in detecting malicious extensions are: maladvertising and cryptocurrency [1, 19]; spyware [48]; phishing [42]; proxy scripts [31]; remote code execution [5], and; ransomware [9, 43, 60].

In *academia*, among other attacks, researchers demonstrated that extensions suffer from maladvertising [4, 59, 63, 67]; fingerprinting [32, 33, 50, 51, 54–57, 61]; JavaScript injection attacks [7, 20, 52], and showed how over-privileged the extensions are [2, 28]. When detecting malicious extensions, researchers usually rely on static [6, 26, 70] and dynamic analysis [11, 30, 36, 57, 71] whereas only a few authors included machine learning techniques [2, 28, 37].

ML in Browser Extensions. Jagpal et al. [28] were one of the first who use machine learning to detect malicious extensions. In concrete, they used Logistic Regression (LR) to train a model after extracting API, permissions, DOM operations, and behavioral signal of over 90k extensions monitored during 3 years, obtaining an overall recall of 96.5% and precision of 81% for one year.

Aggarwal et al. [2] detect malicious extensions by extracting their API sequence and using these sequences to train a Recurrent Neural Network (RNN), achieving high precision (90.02%) and recall (93.31%) in detecting spying extensions.

Regarding clustering, recently, Pantelaios et al. [37] used DBSCAN to cluster extensions to detect malicious ones based on their reviews, ratings, and descriptions. Despite being a common technique for detecting anomalies in the time series, i.e., peaks that should not be there, it is not feasible to cluster them.

Contrarily to previous work, we present a novel methodology to automatically detect malicious extensions based on the number of downloads without analyzing the extensions' source code. Similar to previous work, our classifier achieves 0.88 and 0.90 values for precision and recall, respectively.

Comparison. Compared with Eriksson et al. [20], none matches ours in the over 4,000 extensions they found. This is expected as their method focuses on detecting extensions stealing queries by intercepting search engine traffic instead of extensions presenting their own search forms.

Compared with the dataset from Pantelaios et al. [37], they found 143 extensions. Of these, one appeared in our test set, and we both agreed that it was malicious. Since our focus is on downloads, we are not limited by the need for 50 comments. In a sample of 15 extensions we mark as malicious, none have over 50 comments. We believe both methods can detect different sets of extensions based on different meta-data. Since Google removed many extensions, it is hard to recreate the datasets.

Finally, we compared it with DeDup.js, a code similarity method [39]. Here we included JS, HTML, and CSS files. Used the malicious extensions in our training set and checked the code similarity with each extension in the test set. Since DeDup.js only calculates the number of files in common, we need to pick a threshold for the number of files needed to be considered "similar". To be as fair as possible, we tried all thresholds between 1 and 50 files, and 1 resulted in the best F-score. The F-score is 0.47 (compared to our 0.89). In more detail, DeDup.js achieves: 349 (TP) 15 (FN) 781 (FP) 67 (TN), whereas our method achieves: 326 (TP) 38 (FN) 44 (FP) 804 (TN). As this shows, DeDup.js finds more malicious extensions than we do but at the cost of many false positives.

8 CONCLUSIONS

This paper has spotlighted the patterns of browser extension downloads and suggested an approach of leveraging these patterns as a signal to drive the analysis of extensions. Using a semi-supervised clustering algorithm, we derive 135 clusters from the 2,858 extensions to discover that the download patterns are often strikingly similar, answering our first research question positively: *Are there extensions that follow similar download patterns?*

We combined static, manual, and dynamic analysis to mark all the 2,858 extensions as malicious or benign concerning the attacks in our threat model. Using these security labels with our clusters, we showed that 61 of the 135 contain more than 80% malicious extensions. This affirms our second research question: *Is there any relationship between download patterns and malicious code?*

We showed that by creating a classifier trained on download patterns of already deleted extensions, we can find the malicious extensions that are still active in the Web Store, affirming our last research question: *Can we find malicious extensions based on their download patterns?*

Finally, driven by the download pattern signal, we leveraged a code-similarity analysis to find a total of 6,579 malicious extensions, uncovering "sleeping" extensions whose download patterns contain too little information.

Funding. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council (VR), and Meta.

REFERENCES

[1] AdGuard 2021. Over 20,000,000 of Chrome Users are Victims of Fake Ad Blockers. https://adguard.com/en/blog/over-20-000-000-of-chrome-users-are-victims-of-fake-ad-blockers.html.

[2] A. Aggarwal, B. Viswanath, L. Zhang, S. Kumar, A. Shah, and P. Kumaraguru. 2018. I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions. In *Euro S&P.*

[3] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah. 2015. Time-series clustering – A decade review. *Information Systems* 53 (2015), 16–38.

[4] S. Arshad, A. Kharraz, and W. Robertson. 2016. Identifying Extension-Based Ad Injection via Fine-Grained Web Content Provenance. In *RAID.*

[5] Avast 2021. Backdoored Browser Extensions Hid Malicious Traffic in Analytics Requests. https://decoded.avast.io/janvojtesek/backdoored-browser-extensions-hid-malicious-traffic-in-analytics-requests/.

[6] S. Bandhakavi, N. Tiku, W. Pittman, S. T. King, P. Madhusudan, and M. Winslett. 2011. Vetting Browser Extensions for Security Vulnerabilities with VEX. *Commun. ACM* 54, 9 (2011).

[7] A. Barua, M. Zulkernine, and K. Weldemariam. 2013. Protecting Web Browser Extensions from JavaScript Injection Attacks. In *ICECCS.*

[8] Bots 2022. iOS Developers Use "Well-Known" Download Bots To Manipulate App Store Rankings. https://www.cultofmac.com/146438/ios-developers-use-well-known-download-bots-to-manipulate-app-store-rankings-report/.

[9] Catch-All 2021. "Catch-All" Chrome Extension Silently Steals Your Data. https://blog.barkly.com/catch-all-malicious-google-chrome-extension.

[10] H. Chen, D. He, S. Zhu, and J. Yang. 2017. Toward Detecting Collusive Ranking Manipulation Attackers in Mobile App Markets. In *Asia CCS.* 58–70.

[11] Q. Chen and A. Kapravelos. 2018. Mystique: Uncovering Information Leakage from Browser Extensions. In *CCS.*

[12] Chromium 2021. No more silent extension installs. http://blog.chromium.org.

[13] T. Van Craenendonck, S. Dumančić, and H. Blockeel. 2017. COBRA: A Fast and Simple Method for Active Clustering with Pairwise Constraints. In *IJCAI.* 2871–2877.

[14] cseGoogleSpyware 2021. Cse.google.com - Jan 2021 update. https://www.2-spyware.com/remove-cse-google-com.html.

[15] A. Dempster, F. Petitjean, and G. I. Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[16] A. Dempster, D. F. Schmidt, and G. I. Webb. 2021. MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification. In *KDD.* 248–257.

[17] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. 2008. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. In *VLDB*, Vol. 1. 1542–1552.

[18] Y. Dou, W. Li, Z. Liu, Z. Dong, J. Luo, and S. Y. Philip. 2019. Uncovering download fraud activities in mobile app markets. In *ASONAM.* 671–678.

[19] Droidclub 2022. Malicious Chrome Extensions Found in Chrome Web Store, Form Droidclub Botnet. https://blog.trendmicro.com/trendlabs-security-intelligence/malicious-chrome-extensions-found-chrome-web-store-form-droidclub-botnet/.

[20] B. Eriksson, P. Picazo-Sanchez, and A. Sabelfeld. 2022. Hardening the Security Analysis of Browser Extensions. In *SAC.*

[21] extpose 2022. ExtPose - Track your browser extension app store performance and get competitive advantage. https://extpose.com/.

[22] W. Fan, T. Derr, X. Zhao, Y. Ma, H. Liu, J. Wang, J. Tang, and Q. Li. 2021. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. In *ICDE.* 1583–1594.

[23] M. Fang, G. Yang, N. Z. Gong, and J. Liu. 2018. Poisoning Attacks to Graph-Based Recommender Systems. In *ACSAC.* 381–392. https://doi.org/10.1145/3274694.3274706

[24] S. Farooqi, A. Feal, T. Lauinger, D. McCoy, Z. Shafiq, and N. Vallina-Rodriguez. 2020. Understanding Incentivized Mobile App Installs on Google Play Store. In *IMC.* 696–709.

[25] Google 2022. How are items ranked in the store? https://developer.chrome.com/docs/webstore/faq/#faq-gen-24.

[26] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. 2011. Verified Security for Browser Extensions. In *S&P.*

[27] Xiaohui H., Yunming Y., Liyan X., Raymond L., Nan J., and Shaokai W. 2016. Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences* 367-368 (2016), 1–13.

[28] N. Jagpal, E. Dingle, J.P. Gravel, P. Mavrommatis, N. Provos, M.A. Rajab, and K. Thomas. 2015. Trends and Lessons from Three Years Fighting Malicious Extensions. In *USENIX.*

[29] A. Kampouraki, G. Manis, and C. Nikou. 2009. Heartbeat Time Series Classification With Support Vector Machines. *IEEE Transactions on Information Technology in Biomedicine* 13, 4 (2009), 512–518.

[30] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *USENIX.*

[31] Kreb On Security 2021. Is your Browser Extension a Botnet Backdoor. https://krebsonsecurity.com/2021/03/is-your-browser-extension-a-botnet-backdoor/.

[32] P. Laperdrix, N. Bielova, B.t Baudry, and G. Avoine. 2020. Browser Fingerprinting: A Survey. *ACM Trans. Web* 14, 2, Article 8 (April 2020).

[33] P. Laperdrix, O. Starov, Q. Chen, A. Kapravelos, and N. Nikiforakis. 2021. Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets. In *USENIX.*

[34] W. Meert, K. Hendrickx, and T. Van Craenendonck. 2020. *wannesm/dtaidistance v2.0.0.* https://doi.org/10.5281/zenodo.3981067

[35] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. 2004. Collaborative Recommendation: A Robustness Analysis. *ACM Trans. Internet Technol.* 4, 4 (Nov. 2004), 344–377.

[36] K. Onarlioglu, M. Battal, W. Robertson, and E. Kirda. 2013. Securing Legacy Firefox Extensions with SENTINEL. In *DIMVA.*

[37] N. Pantelaios, N. Nikiforakis, and A. Kapravelos. 2020. You've Changed: Detecting Malicious Browser Extensions through Their Update Deltas. In *CCS.* 477–491.

[38] J. Paparrizos and L. Gravano. 2015. K-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD.* 1855–1870.

[39] P. Picazo-Sanchez, M. Algehed, and A. Sabelfeld. 2022. DeDup.js: Discovering Malicious and Vulnerable Extensions by Detecting Duplication. In *ICISSP.*

[40] Pablo Picazo-Sanchez, Benjamin Eriksson, and Andrei Sabelfeld. 2022. *No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns.* https://doi.org/10.5281/zenodo.7056322

[41] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing Properties of Adversarial ML Attacks in the Problem Space. In *S&P.* 1332–1349. https://doi.org/10.1109/SP40000.2020.00073

[42] Proof Point 2022. TA413 Leverages New FriarFox Browser Extension to Target the Gmail Accounts of Global Tibetan Organizations. https://www.proofpoint.com/us/blog/threat-insight/ta413-leverages-new-friarfox-brow ser-extension-target-gmail-accounts-global.

[43] Rabbit 2021. How dangerous is Bad Rabbit Ransomware and how to avoid it. https://safebytes.com/dangerous-bad-rabbit-ransomware-avoid/.

[44] M. Rahman, N. Hernandez, B. Carbunar, and D. H. Chau. 2018. Search Rank Fraud De-Anonymization in Online Systems. In *HT.* 174–182.

[45] M. Rahman, N. Hernandez, R. Recabarren, S. I. Ahmed, and B. Carbunar. 2019. The Art and Craft of Fraudulent App Promotion in Google Play. In *CCS.* 2437–2454.

[46] M. Rahman, M. Rahman, B. Carbunar, and D. H. Chau. 2016. Fairplay: Fraud and malware detection in google play. In *SDM.* 99–107.

[47] S. Rani, M. Kaur, M. Kumar, V. Ravi, U. Ghosh, and J. R. Mohanty. 2021. Detection of shilling attack in recommender system for YouTube video statistics using machine learning techniques. *Soft Computing* (2021), 1–13.

[48] Reuters 2021. Exclusive: Massive spying on users of Google's Chrome shows new security weakness. https://www.reuters.com/article/us-alphabet-google-chrome-exclusive/exclusive-massive-spying-on-users-of-googles-chrome-shows-new-security-weakness-idUSKBN23P0JO.

[49] I. Sanchez-Rola, M. Dell'Amico, D. Balzarotti, P. Vervier, and L. Bilge. 2021. Journey to the center of the cookie ecosystem: Unraveling actors' roles and relationships. In *S&P.*

[50] I. Sánchez-Rola, I. Santos, and D. Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *USENIX.*

[51] A. Sjösten, S. Van Acker, P. Picazo-Sanchez, and A. Sabelfeld. 2019. LATEX GLOVES: Protecting Browser Extensions from Probing and Revelation Attacks. In *NDSS.*

[52] D. F. Somé. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *S&P.*

[53] J. Song, Z. Li, Z. Hu, Y. Wu, Z. Li, J. Li, and J. Gao. 2020. PoisonRec: An Adaptive Data Poisoning Framework for Attacking Black-box Recommender Systems. In *ICDE.* 157–168.

[54] K. Soroush, I. Panagiotis, S. Konstantinos, and P. Jason. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *NDSS.*

[55] O. Starov, P. Laperdrix, A. Kapravelos, and N. Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the Fingerprintability of Browser Extensions Due to Bloat. In *WWW.*

[56] O. Starov and N. Nikiforakis. 2017. Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions. In *WWW.*

[57] O. Starov and N. Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *S&P.*

[58] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. RuÃŸwurm, K. Kolar, and E. Woods. 2020. Tslearn, A Machine Learning Toolkit for Time Series Data. *Journal of Machine Learning Research* 21, 118 (2020), 1–6.

[59] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. Mccoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab. 2015. Ad Injection at Scale: Assessing Deceptive Advertisement Modifications. In *S&P.*

[60] Threatpost 2021. Malicious Chrome Extension Steals Data Posted to Any Website. https://threatpost.com/malicious-chrome-extension-steals-data-posted-to-any-website/128680/.
[61] E. Trickel, O. Starov, A. Kapravelos, N. Nikiforakis, and A. Doupé. 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *USENIX*.
[62] T. Van Craenendonck, W. Meert, S. Dumančić, and H. Blockeel. 2018. COBRAS-TS: A New Approach to Semi-supervised Clustering of Time Series. In *Discovery Science*. Springer International Publishing, 179–193.
[63] G. Varshney, S. Bagade, and S. Sinha. 2018. Malicious browser extensions: A growing threat: A case study on Google Chrome: Ongoing work in progress. In *ICOIN*.
[64] C. Wu, D. Lian, Y. Ge, Z. Zhu, E. Chen, and S. Yuan. 2021. Fight Fire with Fire: Towards Robust Recommender Systems via Adversarial Poisoning Training. In *SIGIR*. 1074–1083. https://doi.org/10.1145/3404835.3462914
[65] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He. 2021. A Survey of Human-in-the-loop for Machine Learning. arXiv:2108.00941 [cs.LG]
[66] X. Xing, W. Meng, D. Doozan, A. C. Snoeren, N. Feamster, and W. Lee. 2013. Take This Personally: Pollution Attacks on Personalized Services. In *USENIX*. 671–686.
[67] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee. 2015. Understanding Malvertising Through Ad-Injecting Browser Extensions. In *WWW*.
[68] G. Yang, N. Z. Gong, and Y. Cai. 2017. Fake Co-visitation Injection Attacks to Recommender Systems.. In *NDSS*.
[69] Y. Zhang, J. Xiao, S. Hao, H. Wang, S. Zhu, and S. Jajodia. 2020. Understanding the Manipulation on Recommender Systems through Web Injection. *IEEE Transactions on Information Forensics and Security* 15 (2020), 3807–3818.
[70] B. Zhao and P. Liu. 2013. Behavior Decomposition: Aspect-Level Browser Extension Clustering and Its Security Implications. In *RAID*.
[71] R. Zhao, C. Yue, and Q. Yi. 2015. Automatic Detection of Information Leakage Vulnerabilities in Browser Extensions. In *WWW*.

A WEB STORE DOWNLOADS

Even though Google hides the downloads for some extensions, we found out that such a number is still in the source code of each browser extension but commented under the <`PageMap`> HTML label, and more concretely it comes as a text field under the <`Attribute`> label whose attribute name is "user_count" (see Listing 1).

```
<!--<PageMap>
  <DataObject type="document">
    ...
    <Attribute name="user_count">
    6281
    </Attribute>
    ...
  </DataObject>
</PageMap>-->
```

Listing 1: Number of downloads hidden in the Web Store's HTML source.

B DATASET DISTRIBUTION

In Figure 10, we extracted the last public download the Web Store offered per extension and represented the distribution of the downloads according to the category they belong to in the Web Store.

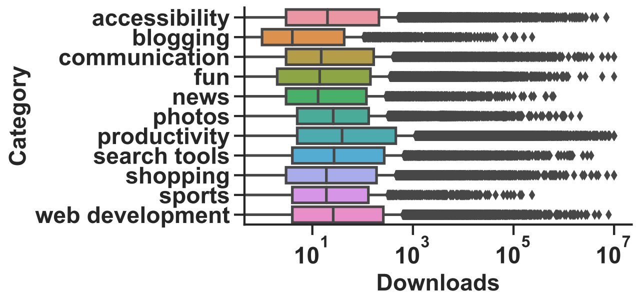


Figure 10: Distribution of the downloads of the categories the Web Store is composed of.

C SOURCE CODE

In Listing 2, we include the manifest file of 29 malicious extensions, having all of them the same manifest but with different icon (<`iconFileName`>), background page ("`js/<name>.js`") and version (<`version`>).

```
{
    "background": {
        "persistent": true,
        "scripts": [
            "js/<name>.js"
        ]
    },
    "browser_action": {},
    "chrome_url_overrides": {
        "newtab": "output.html"
    },
    "default_locale": "en",
    "description": "__MSG_description__",
    "icons": {
        "128": "<iconFileName>"
    },
    "manifest_version": 2,
    "name": "__MSG_name__",
    "offline_enabled": false,
    "update_url": "https://clients2.google.
        com/service/update2/crx",
    "version": "<version>"
}
```

Listing 2: Manifest file of 1,315 extensions.

In Listings 3 and 4 we present the background and content script that 2,554 and 3,637 extensions using https://mytab.me have in common respectively. The background script of the remaining 1,083 extensions is the same but without the last `chrome.runtime.setUninstallURL()` statement.

```
(() => {
  chrome.browserAction.onClicked.addListener
      (function() {
      chrome.tabs.create({
        url: `${app.domain}/${app.name}/`
      });
    });
  chrome.runtime.onInstalled.addListener(
    function(details) {
      if (details.reason == "install") {
          chrome.tabs.create({});
      }}
  );
  chrome.runtime.setUninstallURL('https://
      mytab.me/?from=uninstall')
})();
```

Listing 3: Background script of 2,554 sleeping malicious extensions.

```
(() => {
  location.href = `${app.domain}/${app.name
      }/`;
})();
```

Listing 4: Content script of 3,637 sleeping malicious extensions.

D GROUND TRUTH GENERATION

D.1 Dynamic Analysis

To detect if an extension steals search queries, we use dynamic analysis to interact with the extension. The goal of this interaction is to trigger a search if the extension has a search function.

Challenge. The challenge is that this search function can be implemented in many different ways. For example, simple HTML forms with a predefined action URL for the search engine could be used. In this case, a simple static analysis would likely be enough. However, the search bar could instead be a dynamically generated text-field with JavaScript events connected to complex and obfuscated frameworks, making static analysis more difficult. Additionally, extension can use iframes or redirect users to other new tab pages where the search stealing takes place.

The query stealing can also be implemented in different ways. Normally, server-side redirects are used to send users and their queries to other servers before arriving at the intended search engine. For example, when searching, the query is sent to newtab.com?query=secret which then redirects it to google.com?query=secret. Another method frequently used is JavaScript analytic scripts that use XHR to send the queries to an analytic platform before sending the user to the intended search engine. Analyzing analytic frameworks statically is also challenging, which is why we believe the dynamic analysis is better suited.

Orthogonal to the complexity of statically scanning code, extensions can also implement delays before they start misbehaving. If these delays are implemented in the extension's source code it could be detected statically. However, if instead extensions rely on external websites for the new tab features, these servers could implement the time delay, making it impossible to detect statically.

To tackle these challenges we divide our dynamic analysis into two phases. The first one scans the extensions while keeping track of any search queries being stolen. Also, it records if the extensions rely on external websites for the functionality. If the first phase does not detect the extension as malicious and it is using a website, then, in the second phase, we scan the website for a prolonged period. Using a long enough time span we increase the chances to catch the switch from benign to malicious behavior.

Phase 1 - Scanning extensions. To detect search query stealing, we use Puppeteer[7] to control a Chrome instance running with a potentially malicious extension installed. We build our solution on the dynamic analysis infrastructure developed by Eriksson et al. [20]. However, our approach to interact with the extensions are markedly different. Compared to previous approaches that tried to interact with extensions using honey pages or visiting different websites, our approach is more akin to web crawling and web vulnerability scanning. Honey pages are good at detecting how extensions interact with web pages; however, we want to interact with a web page generated by the extension, i.e., the new tab page.

By using Puppeteer we can mimic how a user would interact with the extension by clicking on search fields and typing queries with their keyboards. This is an important difference from using JavaScript to change the values of search fields as that might not trigger events and analytic scripts.

In Algorithm 1, we give an overview of the algorithm we use to interact with the extension. As some extensions load their new tab functionality in an iframe we need to ensure that we check those first using the same method. We perform two core interactions: 1) First clicking on each text input of either type `text` or `search`. Then we type a predefined query, i.e., "Secret00133700Query". This will trigger any event listeners waiting for a user to type their query; 2) We once again click on each element but this time simulating pressing enter on the keyboard. This submits the search query.

After the interaction method we check all the network traffic to detect any requests containing our query. Similar to previous work [49], we also check for transformations on the query, e.g., lowercase, uppercase, BASE64 etc. If we detect the query in a request to or from a server that is not one of the four major search engines (Google, Yahoo, Bing, and DuckDuckGo), we mark it as malicious. It takes approximately 30 seconds to analyze one extension but for stronger or multiple computers it can easily be parallelized across both more threads.

Phase 2 - Scanning websites. As some extensions rely on websites for their new tab functionality, we also scan these websites in an attempt to detect delayed malicious behavior. For example, a new tab website can start with using Google as the search engine but after some time start redirecting the search queries.

To find websites worth scanning we analyze the results from *Phase 1*. We extract all pairs of *source* and *target* URLs where the target contains our search query. For example, when analyzing an extension we found tabhd.com as the source and google.com/?q=QUERY as the target. In this case, we scanned tabhd.com to test if it switches from google.com to something else.

[7] https://github.com/puppeteer/puppeteer

Algorithm 1 Dynamically scan extensions for query stealing

```
1: procedure SCAN($extension, query$)
2:     $page \leftarrow loadNewtab(extension)$
3:     for $f \in getIframes(page)$ do
4:         for $i \in getInputs(f)$ do
5:             Click on $i$
6:             Type $query$
7:         end for
8:         for $i \in getInputs(f)$ do
9:             Click on $i$
10:            Press Enter
11:        end for
12:    end for
13:    for $i \in getInputs(page)$ do
14:        Click on $i$
15:        Type $query$
16:    end for
17:    for $i \in getInputs(page)$ do
18:        Click on $i$
19:        Press Enter
20:    end for
21: end procedure
```

For each website we find in *Phase 1* we pick an extension relying on this website and scan it repeatedly using the setup in *Phase 1*. We reload the new tab page every 30 seconds for an hour to detect any changes where search queries are sent. If we find that a website starts redirecting queries after some time, we mark the website as malicious and all extensions which use the website.

D.2 Verification Method

While it is unlikely that the dynamic analysis results in false positives we still manually test some of the extensions to detect any false negatives. Note that false negatives found in this verification step do not affect the final results we present, i.e., the security labels we use in later stages for clustering and classification are based on the output from the dynamic analysis. The reason for false positives being unlikely is that our dynamic analysis only marks an extension as malicious if it detects a request from an unknown server with our query. To better understand the general limitations of our dynamic analysis and approximate the false negatives we perform a manual analysis of extensions marked as benign and use static analysis to find similar ones.

Manual. From the set of extensions marked as benign by the dynamic analysis, we pick a subset to inspect manually. To do so, we open it in Google Chrome and follow any installation guide the extension needs. We also accept any pop-ups, either modal windows or window pop-ups that might show up during this process. From there we locate the search bar if one exists. Using the network tab in developers tools we note all requests being sent both when typing in the search bar and when submitting the query. If any unknown server receives our query we mark the extension as malicious.

Static. To increase our coverage, we include static analysis to automatically find similar extensions to the ones we find in the manual analysis. We achieve this by manually creating simple signatures for each new malicious extension we find. For example, Listing 5 shows a code snippet that could be a signature for a group of extensions:

```
var url_analytics = 'https://s.example.com'
```

Listing 5: Example of code used as a signature to find more similar extensions.

Using these signatures we search all manifests, HTML, and JavaScript files for the signature. If we find a match we mark these as malicious.

E VERIFICATION RESULTS

During our verification, we picked a sample of 100 benign extensions to manually review. As discussed in Section 6 we are mainly interested in finding false negatives, therefore we only consider benign extensions for manual review. Most of these were correctly marked as benign, being developed by either *FreeAddon* or *Choosetab*. We found three extensions wrongly marked as benign.

To better understand our limitations we look outside the sample for similar extensions to the ones we missed. Here we find a total of 32 extensions out the 2,858 extensions. The biggest miss was due to a group of 27 extensions using s.tabturbo.com to steal the queries.

There was another group of three extensions we failed to correctly mark as malicious due to a limitation of Puppeteer, and possibly a bug in the extension. Upon loading the extension, it tries to redirect the user to extension://index.html, on Ubuntu this causes Chrome to prompt the user to execute an external program. This prompt can not be closed by Puppeteer and at the same time makes it impossible to interact with the web page.

Finally, there were two unique extensions with different problems. The first one had two search input forms, but only one malicious, and we only tested the benign one. The second one had a hidden search input which our dynamic approach failed to click on. We discuss these cases more in Section 6.

F CLUSTERS

In Figure 11 and Figure 12 we present more of the benign and malicious clusters we find.

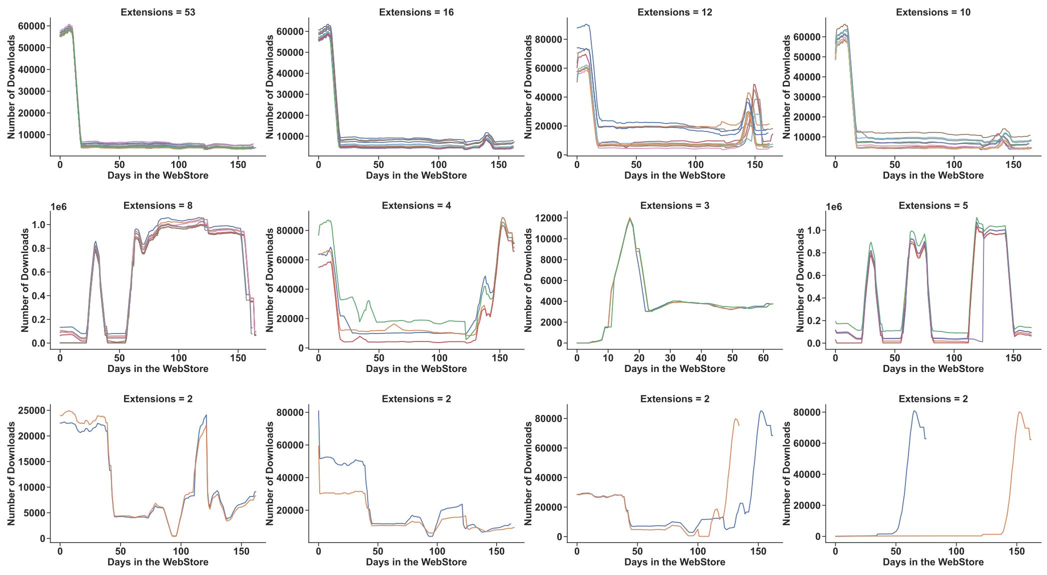


Figure 11: Benign Clusters.

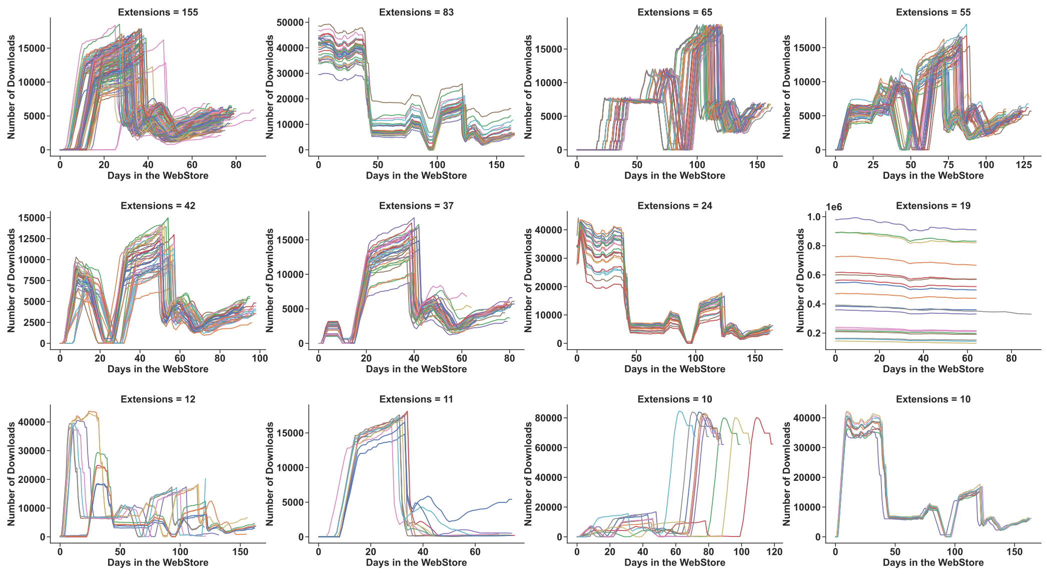


Figure 12: Malicious Clusters.