It's not Easy: Applying Supervised Machine Learning to Detect Malicious Extensions in the Chrome Web Store

BEN ROSENZWEIG, Saarland University, Germany
VALENTINO DALLA VALLE, CISPA Helmholtz Center for Information Security, Germany
GIOVANNI APRUZZESE, University of Liechtenstein, Liechtenstein and Reykjavik University, Iceland
AURORE FASS, CISPA Helmholtz Center for Information Security, Germany

Google Chrome is the most popular Web browser. Users can customize it with *extensions* that enhance their browsing experience. The most well-known marketplace of such extensions is the Chrome Web Store (CWS). Developers can upload their extensions on the CWS, but such extensions are made available to users only after a vetting process carried out by Google itself. Unfortunately, some *malicious* extensions bypass such checks, putting the security and privacy of downstream browser extension users at risk.

In this paper, we carry out a comprehensive real-world security analysis of malicious extensions in the CWS. Specifically, we scrutinize the extent to which automated mechanisms reliant on supervised machine learning (ML) can be used to detect malicious extensions on the CWS. To this end, we first collect 7,140 malicious extensions published in 2017–2023 and which have been flagged as malicious by Google. We combine this dataset with 63,598 benign extensions published or updated on the CWS before 2023, and we develop three supervised-ML-based classifiers—leveraging both original features as well as techniques inspired by prior work. We show that, in a "lab setting", our classifiers work well (e.g., 98% accuracy). Then, we collect a new, and more recent, set of 35,462 extensions from the CWS, published or last updated in 2023, with unknown ground truth. We were eventually able to identify 68 malicious extensions that bypassed the vetting process of the CWS. However, our classifiers also reported over 1k likely malicious extensions which may overestimate their true number. Based on this finding (further supported with other experiments and realistic analyses), we elucidate, for the first time, a strong *concept drift* effect on browser extensions. We also provide factual evidence that commercial detectors (e.g., VirusTotal) work poorly to detect known malicious extensions. Altogether, our results highlight the fact that detecting malicious browser extensions is a fundamentally hard problem which has not (yet) received an adequate degree of attention. This requires additional work both by the research community and by Google itself—potentially by revising their approaches. In the meantime, we informed Google of our discoveries, and we release our artifacts.

CCS Concepts: • **Security and privacy → Web application security**; **Social aspects of security and privacy**; • **Computing methodologies** → *Machine learning*; • **Information systems → World Wide Web**.

Additional Key Words and Phrases: Google Chrome; Browser Extensions; Classification; Concept Drift

ACM Reference Format:
Ben Rosenzweig, Valentino Dalla Valle, Giovanni Apruzzese, and Aurore Fass. 2025. It's not Easy: Applying Supervised Machine Learning to Detect Malicious Extensions in the Chrome Web Store. *ACM Trans. Web* 1, 1, Article 1 (January 2025), 36 pages. https://doi.org/10.1145/3770852

Authors' Contact Information: Ben Rosenzweig, s8berose@stud.uni-saarland.de, Saarland University, Saarbruecken, Germany; Valentino Dalla Valle, valentino.dalla-valle@cispa.de, CISPA Helmholtz Center for Information Security, Saarbruecken, Germany; Giovanni Apruzzese, giovanni.apruzzese@uni.li, University of Liechtenstein, Vaduz, Liechtenstein and Reykjavik University, Reykjavik, Iceland; Aurore Fass, fass@cispa.de, CISPA Helmholtz Center for Information Security, Saarbruecken, Germany.

ACM 1559-114X/2025/1-ART1
https://doi.org/10.1145/3770852

1 Introduction

Web browsers have become indispensable tools: besides allowing users to visit and interact with various websites, browsers now integrate a rich ecosystem of *browser extensions* that enhance their functionalities. Such add-ons can substantially improve users' online experience—e.g., by removing ads, facilitating password management, or automatically spell-checking users' input [55]. In this context, the Chrome Web Store (CWS) stands out as the most popular "marketplace" of extensions supported by chromium-based Web browsers [4]: as of September 2025, several hundreds of extensions in the CWS are installed by millions of devices worldwide [41].

Unfortunately, despite most extensions providing plenty of benefits to their users, some extensions may conceal harmful content that can lead to security (e.g., unauthorized operations [62]) or privacy (e.g., data leakage [39]) violations. Such *malicious extensions* are a cause of concern to Web users, who may inadvertently compromise their devices by installing a malicious extension from a legitimate source—i.e., the CWS. Indeed, the CWS adopts an (apparently) strict policy for the content uploaded by third-party developers: before becoming available to end-users, any new uploaded extension must first pass Google's vetting process [1, 5, 30]. Despite such precautionary measures, a recent work [55] found that the CWS is rich of "security-noteworthy extensions", which bypass the checks enforced by Google, and hence represent a risk for any users installing them onto their Web browser.

Abundant prior work attempted to mitigate the spread of malicious browser extensions [23, 74, 96], either by assessing their real-world prevalence (e.g., [55, 56]) or by studying the risks of certain types of extensions, such as those exfiltrating user data [39, 89, 98] (potentially by also proposing some countermeasures [74, 97]). Yet, we found no recent work that carried out a comprehensive, reproducible, and large-scale security analysis of classifiers reliant on supervised machine learning (ML) to automatically detect malicious browser extensions on the CWS. For instance, some works analyze only hundreds of extensions [38], whereas others do not focus on the CWS [95, 103], are outdated (e.g., 2012–2015 [56, 58, 91, 95, 99]), or propose methods reliant on unsupervised ML techniques—and therefore the detection process requires defining rules/heuristics or manual inspections (e.g., [74]). We systematically discuss related work in Section 9. Intriguingly, however, there is abundant literature that performs holistic analyses focused on the detection of other (and orthogonal) security threats—such as generic Android/Windows malware [36, 49, 75] or Phishing Email/Webpages [43, 64, 71]. This shows that the detection of malicious browser extensions by means of automated methods is still an open problem requiring further scrutiny.

In this paper, we tackle this challenge and carry out a security assessment of the extensions published on the CWS. We begin with a broad research question: can we build a system that automatically learns (via supervised ML) to detect malicious extensions uploaded on the CWS? To this end, we first collect a *large* corpus of over *100k* extensions: 99k are from the CWS (and we are unsure of their true nature), whereas 7k are *verified malicious* extensions taken from ChromeStats [2]; these procedures are outlined in Section 3. Then, we use these extensions to develop three automated detectors—based both on original features and prior work [47], used to train supervised ML-based classifiers. We show that these detectors achieve remarkable performance, with 98% accuracy and less than 2% false positives, by carrying out rigorous experiments and following best practices [26] (including cross validation and verification of the feature importance); this evaluation is discussed in Section 4. Based on these encouraging results, we test our detectors on extensions in the "real world," i.e., on extensions recently (2023) published on the CWS, for which we have no verified ground truth. These experiments are discussed in Section 5 and reveal a stark disconnection with our initial findings. First, our detectors flagged over 1k extensions (out of ~35k tested) as malicious. Then, we manually analyzed a subset (≈20%) of them, and we find that only 68 of these extensions can be claimed to be truly malicious—denoting a large number

```
"manifest_version": 3,
"permissions": ["downloads", "history"],
"host_permissions": ["https://example.com/*"],
"background": {
    "service_worker": "service_worker.js",
},
"content_scripts": [
    {
        "matches": ["<all_urls>"],
        "js": ["script.js"]
    }
],
...
```

Listing 1. Extract of a `manifest.json` file from a browser extension

of false positives. This result suggests that the ecosystem of browser extensions may be affected by the *concept drift* problem. We validate this hypothesis in Section 6, wherein we simulate a realistic (and fair) longitudinal analysis, measuring how the performance of our detectors would have changed over four years. Our results confirm the existence of concept drift, elucidating that automatic detection of malicious extensions is fundamentally hard. We even tested commercial products, such as VirusTotal, and found they work poorly: they cannot detect known malicious extensions taken down by Google.

Contributions. This paper provides technical methods and novel scientific findings. Specifically:

- we collect ≈107k browser extensions and use them to develop, and then evaluate, three detectors (including original ones) of malicious browser extensions;
- we show that—in our "laboratory" setup—our detectors perform well (detection accuracy=98%, ≈1s to analyze an extension end-to-end) which would justify their deployment in the real world;
- by using our detectors in an open-world test on 35k extensions, we provide (for the first time) factual evidence that the CWS ecosystem is affected by concept drift, making automated detection hard in the long term. We analyze the effects of concept drift and explore some mitigations;
- nonetheless, by manually analyzing the extensions flagged by our detectors, we found 68 malicious extensions on the CWS, which affected >13M users. We disclosed our findings to Google.

To pave the way for future research, we discuss the lessons learned from our security analysis in Section 10. We publicly release our artifacts [20] for reproducibility.

2 Background: Browser Extensions

Browser extensions are third-party programs that users can install to improve their browsing experience by, e.g., removing advertisements from Web pages or for password management. To set up the stage for our contributions, we first describe the generic architecture of a browser extension and then outline its main components. Given the fact that we focus on the CWS, the following content pertains to extensions for chromium-based Web browsers.

2.1 Architecture of a Browser Extension

Browser extensions are `crx` files, i.e., compressed archives including mainly HTML, JavaScript, and CSS files. Every extension must have a `manifest.json` file [14]. The `manifest` contains basic configuration and details about an extension, such as its main components and permissions. The current version (as of September 2025) is *Manifest V3* which aims to improve the security of extensions by, e.g., preventing them from downloading external resources [15]. Although the CWS stopped accepting V2 extensions in January 2022, only 76.8% of the extensions currently in the CWS have migrated to V3 [40].

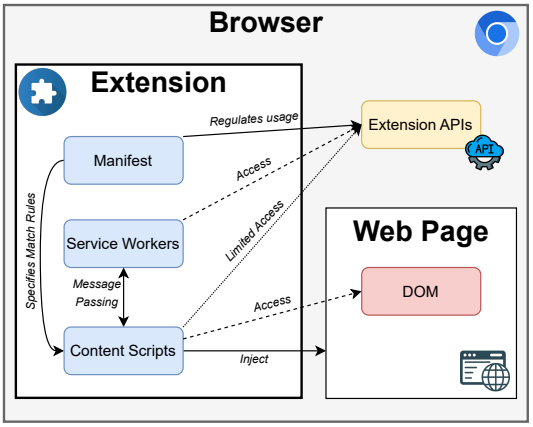


Fig. 1. Typical architecture of a chromium-based browser extension

To use most extension APIs (e.g., `cookies`, `downloads`, `history`), a developer must declare the permissions in the corresponding `manifest` field [9, 17]. In addition, a developer can also specify host permissions (i.e., URLs or URL patterns) to allow extensions to interact only with specific Web pages (useful to, e.g., monitor network requests or fetch data). An example of a manifest file is shown in Listing 1. Here, the extension has the permissions "downloads" and "history" to download arbitrary files and read or modify users' browsing history. Through the host permissions, it is able to send HTTP requests to "https://example.com/*".

2.2 Main Components of a Browser Extension

Figure 1 displays the architecture of browser extensions and their main components relevant to this work. The core logic of an extension is implemented through a *service worker* (or *background scripts* in Manifest V2). This JavaScript file runs in the background of the browser (independently of the lifetime of a window / tab), handles events, and has access to the extension APIs specified in the `manifest` [16]. An extension can inject *content scripts* in a Web page. For example, in Listing 1, a content script "script.js" is defined and injected into any Web page (matching pattern "< all_urls >") visited by the user. Similarly to scripts already loaded by Web pages, content scripts can use the standard DOM APIs to read and modify Web pages. Contrary to the service worker, content scripts have restricted access to the extension APIs (limited to `storage`, `i18n`, and various `runtime` APIs [7]).

2.3 Problem Scope, Motivation, and Threat Model

The starting point of our research is developing detectors (i.e., binary classifiers) for malicious browser extensions that leverage supervised machine learning (ML) techniques. This goal is motivated by the intrinsic benefit of supervised ML methods to "automatically" learn hidden patterns among data for classification tasks—thereby avoiding the burden of manually defining "rules" or "heuristics" that can discriminate whether any given extension is benign or malicious [24]. Such learning is made possible during a training phase necessitating labeled data, used to define the decision boundaries of the ML model. Even though obtaining such labeled data is typically not trivial in cybersecurity [24], the browser-extension context facilitates such a data-collection process because it is possible to use extensions *taken down from the CWS for being "malicious"* as samples with verified ground truth (we will better explain our rationale in Section 3.3).

We observe that our focus on supervised ML implicitly puts works on unsupervised- (e.g., clustering [74]) or heuristic-based methods (e.g., [58]) outside our scope. Indeed, even though

many prior works proposed methods to identify malicious browser extensions, there is no publicly available implementation of a detector that relies on supervised ML techniques specifically designed for malicious browser extensions. We extensively discuss related work in Section 9, wherein we also perform a systematic literature review to validate this claim. The closest work we could find—which we use as a baseline—is JaSt, a detector of malicious JavaScript that relies on supervised ML, but which is not designed to work (and has never been tested) on browser extensions.

To devise our detectors, we assume the following threat model. First, the attacker wants to develop (and massively release) a browser extension that, if installed by (unaware) users in their Web browsers, would enable the attacker to carry out some security violation (e.g., exfiltrating user data). To maximize the reach of such a malicious extension, the attacker wants to upload it on popular (and, ideally, trusted) repositories—such as the CWS. We assume the attacker may leverage simple techniques, such as code obfuscation [68], to conceal the malicious functionality of the browser extension and/or delay the time required to flag the extension as being malicious. Hence, to detect such malicious extensions, we will leverage three complementary sources of information: JavaScript code, which is a crucial component of browser extensions and can embed some malicious functionality; as well as an extension's `manifest`, `service worker`, and `content scripts`, since prior work showed that such elements can conceal traces of malicious behavior [58, 59, 74, 94]. We do not seek to build a detector that is specifically designed to withstand targeted evasion attempts.

Finally, we emphasize that our focus is on "malicious" extensions—which are orthogonal to other classes of security-noteworthy extensions [55], such as vulnerable (e.g., [48]) or fingerprintable (e.g., [22]) extensions. An extended overview of related works is deferred to Section 9.

3 Data Collection and Preparation

As a starting point, we describe how we assembled the extensions that enabled our security analysis. We first explain our data collection procedure. Then, we describe how we preprocessed our data. Finally, we present how we derive the two main datasets used in our investigation.

3.1 Collecting Extensions

We are not aware of any publicly available and (relatively) up-to-date datasets with benign and malicious extensions that are suitable for our purposes. Hence, for a meaningful assessment (and to comply with our threat model), we create our datasets by retrieving chromium-based browser extensions from two trusted sources: the Chrome Web Store [4] and Chrome-Stats [2].

3.1.1 Chrome Web Store. The CWS is the most well-known marketplace for chromium-based extensions and is maintained by Google itself. We use the CWS sitemap [6] to collect the URLs of *all* available extensions. From such URLs, we extract the unique 32-character long IDs of all the extensions, which enables us to directly download these extensions. Specifically, we perform these operations on Nov. 16, 2023. We identified 168,953 extension IDs, but 10,829 extensions were either not available for download or required payment. Hence, we downloaded the remaining 158,124 extensions; their last update was made between 2012 and 2023.

3.1.2 Chrome-Stats. Chrome-Stats [2] is a tool that provides metadata (e.g., last update, user count, description, or user reviews) as well as the source code of extensions that are—or were—in the CWS. Chrome-Stats engineers developed a crawler that has been automatically collecting extensions from the CWS and extracting their metadata from the CWS, once a day, since July 5, 2020. We use Chrome-Stats, because it also stores data of extensions that have been removed from the CWS. By using the "Advanced Search Functionality" [3], we can look for extensions that have been removed from the CWS due to being malware (with the filter option "obsoleteReason" and the value "malware"). Note that those extensions have been flagged as malicious by Google engineers, and

Dataset	Label	Source	Last update	#Extensions
`Dataset L`	benign	CWS	before 2023-01-01	63,598
`Dataset L`	malicious	Chrome-Stats	any	7,140
`Dataset U`	unlabeled	CWS	after 2023-01-01	35,462

Table 1. Summary of `Dataset L` and `Dataset U`

taken down from the CWS. Our search (done on Nov. 16, 2023) yielded 10,814 malicious extensions, but 3,131 of these were not available for download.[1] Hence, we downloaded all the remaining 7,683 *malicious* extensions. Finally, for every downloaded extensions (both from the CWS and from Chrome-Stats), we then collected metadata information from Chrome-Stats.

3.2 Preprocessing: Extension Unpacking

Having acquired 165,067 extensions (7,683 from Chrome-Stats and 158,124 from the CWS), we must now unpack them (recall that extensions are `crx` files; see Section 2.1) to analyze them. Hence, we rely on a well-known tool, DoubleX (CCS'21 [10]), to extract the `manifest`, content scripts, and service worker of each extension we collected. In doing so, we take extra care to: *(i)* remove all extensions that do not have a valid `manifest`, and *(ii)* all extensions that do not have at least one content script or service worker—this is to align our analysis with the scope of our paper (see Section 2.2). After this filtering, we obtain a total of 106,200 extensions: 99,060 *likely benign* (from the CWS) and 7,140 *malicious* (from Chrome-Stats, and flagged as malicious by Google).[2]

3.3 Data Partitioning and Extension Datasets

We partition our extensions in two distinct datasets (summarized in Table 1), described below.

Rationale — Among our goals is analysing the effectiveness of detectors of malicious extensions based on supervised ML—i.e, binary classifiers. To develop such classifiers, we need data provided with *ground truth*—i.e., we need a labeled set of benign and malicious extensions. Unfortunately, there is no existing publicly available dataset of *labeled* browser extensions. As a best-effort strategy, we consider all extensions from the CWS published or last updated *before Jan. 1, 2023* to be "benign".[3] In contrast, we know that the extensions we collected from Chrome-Stats are malicious, since Google engineers analyzed and flagged those extensions as such through their own (proprietary and closed-source) checks. Finally, we consider all extensions published or last updated on the CWS *after Jan. 1, 2023* to be "unlabeled," and we will use these for our open-world assessment. Based on this rationale, we define our two datasets:

3.3.1 `Dataset L`. We create our "labeled" `Dataset L` by considering all extensions from the CWS whose last update occurred *before January 1, 2023*. Such a cutoff-date leads to `Dataset L` having 63,598 (out of 99,060) benign extensions from the CWS. While there may be some malicious extensions in those, it is impossible to manually vet over 63k extensions; thus, our approach is a *best-effort strategy* to reduce the number of mislabeled extensions while enabling a comprehensive analysis. Then, we add all 7,140 malicious extensions from Chrome-Stats. For our primary investigation,

[1] Chrome-Stats told us some developers asked for their extensions' source code to be removed from Chrome-Stats' database.
[2] This filtering procedure validates our choice of focusing on `manifest`, `service worker`, and `content scripts`: of the "known malicious" extensions collected from Chrome-Stats, only 543 (out of 7,683, i.e., 7%) have been excluded due to our filtering.
[3] We acknowledge that there may be a few malicious extensions that evaded Google's vetting process and that are still in the CWS. However, since the extensions we deem as "benign" have been in the CWS for almost a year (we collected our dataset in Nov. 2023, see Section 3.1), and malicious extensions take on average one year to be detected [55], we assume that the fraction of extensions that we (incorrectly) label as benign to be negligible (we further discuss this limitation in Section 8.2).

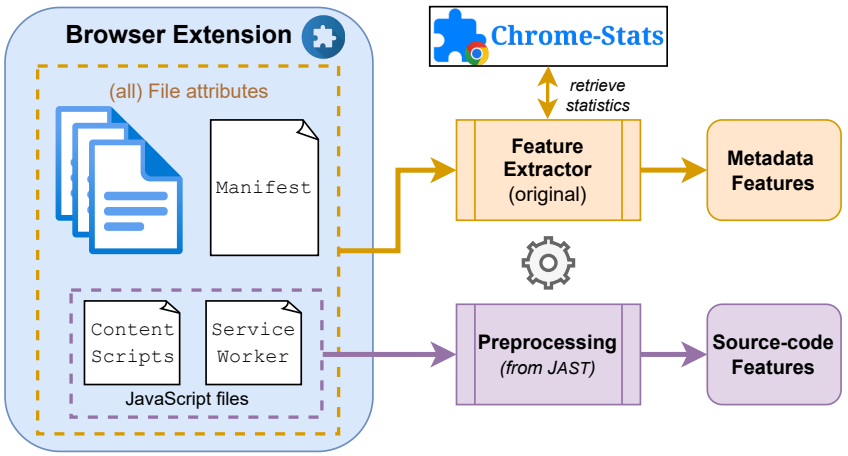


Fig. 2. Feature extraction and preprocessing phase of our approach.

focused on testing our detectors in a "lab setting", we further split `Dataset L` into disjoint training and test sets. We randomly sample 80% of the benign and 80% of the malicious extensions to create our training set. The remaining 20% of `Dataset L` are used to test our classifiers.

3.3.2 `Dataset U`. For an open-world evaluation to test our classifiers on "unknown" extensions, we create `Dataset U`, an *unlabeled* dataset (with no ground truth). `Dataset U` serves to investigate to what extent there are malicious extensions in the CWS that have not yet been detected. `Dataset U` contains the remaining 35,462 extensions last updated (or published) *after Jan. 1, 2023*.

4 Detectors of Malicious Extensions

We describe our ML-based approach to detect malicious extensions—our technical contribution. First, for each extension, we extract a feature vector from its source code and metadata. Then, we define three classifiers (`Source code classifier`, `Metadata classifier`, and `Combined classifier`) using our extracted features. Finally, we evaluate our classifiers on our labeled `Dataset L` of benign and malicious extensions, and we measure the runtime performance of our approach.

4.1 Feature Extraction

Our feature-extraction process is fully automated. A schematic representation of our pipeline is shown in Figure 2. In what follows, we describe and justify the procedure to extract the feature representation of each browser extension.

4.1.1 Source Code Features. Our "baseline" classifier is based entirely on prior work, and analyzes features extracted from extensions' source code. Specifically, we leveraged JaSt, a well-known detector of malicious JavaScript code [47] which has never been applied in the browser-extension context. JaSt uses the parser Esprima [54] to build the Abstract Syntax Tree (AST) of an input JavaScript file. Then, it traverses the tree to extract syntactic units and build 4-gram features. JaSt leverages the relative frequency of each 4-gram to produce a feature vector of length 2,457. In the JaSt paper [47], these features were extracted from JavaScript samples taken from emails or Web pages and used to train a supervised-ML classifier, yielding a detector of malicious JavaScript with 99.5% accuracy, 0.54% false negatives, 0.52% false positives [47]. Despite these promising results, however, the JavaScript code of emails or Web pages (i.e., which was the focus of the JaSt paper [47]) does not fully align with that of browser extensions (which is the focus of our paper).

Table 2. Top 10 hosts permissions (train set)

Rank	Benign extensions	Malicious extensions
1.	<all_urls>	*://www.google.com.kh/*
2.	https://*/*	*://mail.google.com/*
3.	http://*/*	*://www.google.com.au/*
4.	*://*/*	*://www.google.us/*
5.	http://*/	*://www.google.ca/*
6.	https://*/	*://www.google.de/*
7.	https://ajax.googleapis.com/	*://www.google.dk/*
8.	chrome://favicon/	*://www.google.fr/*
9.	https://*.1688.com/*	*://www.google.co.jp/*
10.	https://www.youtube.com/*	*://www.google.nl/*

Table 3. Top 10 content script matches (train set)

Rank	Benign extensions	Malicious extensions
1.	<all_urls>	<all_urls>
2.	https://*/*	*://duckduckgo.com/*
3.	http://*/*	*://gl-search.com/*
4.	*://*/*	*://redirect.lovelytab.com/*
5.	https://www.youtube.com/*	*://str-search.com/*
6.	https://mail.google.com/*	*://search.yahoo.com/search*
7.	https://twitter.com/*	*://www.google.com/search*
8.	https://github.com/*	*://www.bing.com/search*
9.	*://*.youtube.com/*	https://happyhey.com/*
10.	file://*/*	https://www.happyhey.com/*

However, we hypothesize that *(i)* the AST of benign and malicious extensions may still differ; and *(ii)* extensions may be using code transformation techniques (which leave traces in the AST [68]), even though this would be a violation of the CWS policies [51]. Therefore, we use JaSt to extract AST-based features from the source code of extensions. Since JaSt takes a single JavaScript file as input, we concatenated the service worker and content scripts into a single file (see Section 3.2).

4.1.2 Metadata Features. To develop our "original" classifiers, we extract features purely based on an extension's metadata, i.e., from its `manifest`, file attributes, and Chrome-Stats' statistics. In principle, some of these features are inspired by previous work (e.g. permissions [23, 55, 96] or ratings [74]). However, the low-level implementation of *76% of our metadata features is novel*: specifically, we develop a custom feature extractor which we publicly release in our repository [20] (we are not aware of any prior work that released a similar tool). Given the novelty of our features, we will validate them while describing them (at a high-level) below.

- *Permissions*: based on Chrome documentation, we collected all 70 permissions an extension can specify in its `manifest` [17]. Then, for each extension in our dataset, we analyze its `manifest` to extract the permissions it specifies. (We do not consider optional permissions here, because they require user approval during the runtime of an extension.) This way, each extension has a permission feature vector of length 70 containing booleans: a value of "true" indicates that the extension lists the corresponding permission in its `manifest` and "false" otherwise.
- *Host Permissions*: since host permissions can be arbitrary string-matching patterns, we cannot extract an exhaustive list as before. Instead, we used our training set (i.e., 80% of `Dataset L`) to extract a subset of the 400 most popular host permissions. Hence, each extension has a boolean feature vector of length 400, where each value indicates whether the extension has the corresponding host permission ("true") or not ("false"). We list the top 10 host permissions for benign and malicious extensions in Table 2. Benign extensions tend to use host permissions that can match arbitrary URLs, while malicious extensions list many Google subdomains. These findings align with those in [55], validating our choice.
- *Content Script (CS) Matches*: we use the same approach as for the "Host Permissions" to extract the top 400 CS matches from the `manifest` of the extensions in our training set. As before, we get a boolean feature vector of length 400. We list the top 10 CS matches in Table 3. The most popular CS match is "<all_urls>", i.e., CS are injected into any Web pages. Malicious extensions rather inject their CS in search engine Web pages like "*://duckduckgo.com/*" or "*://gl-search.com/*". This may be due to large clusters of malicious extensions redirecting users' search queries [77]. Conversely, benign extensions inject their CS into popular pages like YouTube or Twitter.
- *Number of CS and Service Worker (SW)*: the last features we obtain from the `manifest` are the number of CS and SW (or background scripts for Manifest V2) an extension defines, which are integer values.

Table 4. Most common keywords among the extensions in the *training set*

Rank	Benign Ext.	Malicious Ext.
1.	EXTENSION	GOOD
2.	WORK	LOVE
3.	GREAT	EXTENSION
4.	GOOD	LIKE
5.	THANK	GREAT
6.	NOT WORK	WORK
7.	USE	THANK
8.	LIKE	USE
9.	LOVE	NICE
10.	TIME	TIME

(a) Top 10 review keywords

Rank	Benign Ext.	Malicious Ext.
1.	EXTENSION	EXTENSION
2.	CLICK	NEW
3.	PAGE	TAB
4.	CHROME	TIME
5.	USE	WALLPAPER
6.	ADD	FAVORITE
7.	NEW	THEME
8.	TIME	HIGH
9.	WEBSITE	BACKGROUND
10.	BROWSER	FEATURE

(b) Top 10 summary keywords

Rank	Benign Ext.	Malicious Ext.
1.	EXTENSION	NEW
2.	CHROME	TAB
3.	PAGE	WALLPAPER
4.	TAB	HD
5.	ADD	CHROME
6.	WEB	THEME
7.	BROWSER	WALLPAPERS
8.	WEBSITE	USEFUL
9.	ALLOW	UTILITY
10.	CLICK	LOT

(c) Top 10 description keywords

- *Number of Users*: inspired by [56], we use Chrome-Stats to collect the number of users of each extension. According to the Chrome Web Store Developer Support, the number of active users of an extension is "the number of Chromes with the extension installed that are active and checking in to [their] update servers over the previous seven days only, not for all time. It is not equal to the sum of historic installs minus the sum of historic uninstalls" [55].
- *Number of Ratings and Average Rating Score*: users can rate extensions with 1–5 stars on the CWS. We use the total number of ratings received by an extension, as well as an extension's average rating, as features (for the latter, the value is 0 if an extension has no rating).
- *Description, Summary, and Review Keywords*: we extract features from the description, summary, and reviews of an extension. The description is a short text describing the functionality of an extension. The summary is typically a longer article that contains more information about the behavior of an extension. The reviews are written by users on the CWS to evaluate an extension. In all three cases, we need to process these free texts before being able to extract features. To this end, we use spaCy [18], a natural language processing tool to *(i)* remove stop words and punctuation; and *(ii)* obtain the basic form of words, e.g., this turns an "is" into a "be". Finally, we split each text into a set of single words and remove duplicates per text. This way, we get three sets of words for each extension (for description, summary, and reviews). As before, we compute the top 400 words for description, summary, and reviews; and we check for their presence for each extension we analyze. Tables 4 show the top 10 description, summary, and review keywords. Interestingly, the keywords "NEW", "TAB", and "WALLPAPER" are frequently used in the description and summary of malicious extensions. This is probably due to the cluster of "new tab wallpaper" extensions, which change the appearance and functionality of a newly opened tab [74] and sometimes also show advertisements or contain malware [55].
- *Same Developer Count*: we consider the number of extensions by the same developer (i.e., same username and email address). We obtain this number by using Chrome-Stats' advanced search functionality for each extension, as also done in [55].
- *File Count,* `CRX` *Size, JavaScript File Count, JavaScript Size*: additionally, we use file attributes as features. We consider the size of a `CRX` file in bytes, the number of files and JavaScript files inside of an extension, and the size of those JavaScript files in bytes.
- *Related Permissions*: for each extension, we extract the permissions of the first four similar extensions recommended by Google. (We manually verified that the recommendation list is stable over time—as also done by [63].) We then compare the permissions specified in an extension's `manifest` with those of the four recommended extensions. For each extension, we have a feature vector of length 70 (the maximum number of declarable permissions). Initially, all the values

are set to 0. If the analyzed extension declares a permission that is not listed by n ($n \in [\![1, 4]\!]$) recommended extension(s), then we decrease the value by n.

Remark. In devising our metadata features, we use the extensions in the *training set*, i.e., we follow the recommendations by Arp et al. [26] and do not commit the "data snooping" crime—which would unfairly increase the performance of our classifiers on the test set.

4.1.3 Features Summary. Table 5 summarizes all the extracted features (metadata and source code), including the length of each feature vector and how we extracted each feature. The low-level technical details that enable complete reproduction of our feature-extraction procedure can be found in our repository [20].

Feature name	Feature length	Feature origin
Permissions	70	`manifest`
Host Permissions	400	`manifest`
Content Script Matches	400	`manifest`
Number of Content Scripts	1	`manifest`
Number of Service Workers	1	`manifest`
Number of Users	1	Chrome-Stats
Average Rating Score	1	Chrome-Stats
Number of Ratings	1	Chrome-Stats
Description Keywords	400	Chrome-Stats
Summary Keywords	400	Chrome-Stats
Review Keywords	400	Chrome-Stats
Same Developer Count	1	Chrome-Stats
CRX Size	1	File attributes
File Count	1	File attributes
JavaScript File Count	1	File attributes
JavaScript Size	1	File attribute
Related Permissions	70	`manifest` + Chrome-Stats
Source Code (AST)	2,457	JaSt [47]

Table 5. Summary of the extracted features for each extension

We mention that, during the extraction process of the source-code features, the generation of the AST by JaSt led to syntax errors (raised by Esprima) for 7,448 out of 106,200 extensions. In contrast, we encountered no errors in the generation of the metadata-related features. Despite this small discrepancy (explicitly quantified, for both `Dataset L` and `Dataset U`, in Table 6), we will not exclude extensions for which we could obtain only their metadata-related features. We will discuss the consequences of such a design choice in Section 8.2.

	Metadata features	Source code features
#Extensions	106,200	98,752
- `Dataset L`	70,738	68,748
- `Dataset U`	35,462	30,004

Table 6. Extensions with metadata and source code features extracted

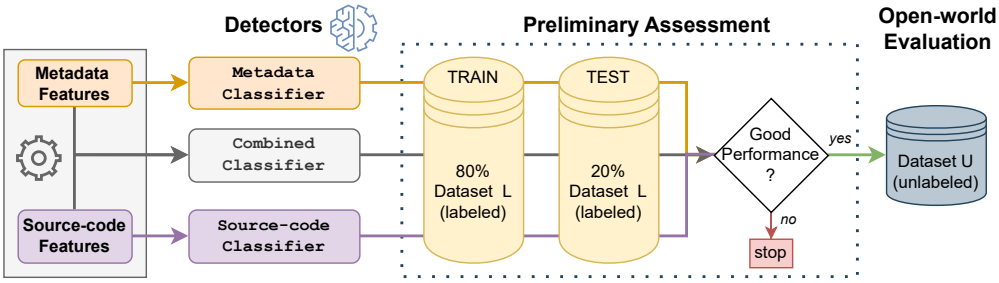


Fig. 3. Workflow: iif our detectors exhibit good performance on `Dataset L`, we use them on `Dataset U`

4.2 Classifier Development

For our large-scale assessments of supervised-ML methods, we develop three (binary) classifiers:

- **`Source code classifier`**: this "baseline" classifier (inspired by JaSt [47]) leverages only features from the extensions' source code (see Section 4.1.1);
- **`Metadata classifier`**: this "original" classifier uses only the extensions' metadata features (see Section 4.1.2);
- **`Combined classifier`**: this classifier uses both metadata and source code features—thereby enhancing JaSt [47] with our custom-defined metadata features.

As shown in Figure 3, we use `Dataset L` to train each classifier (i.e., a detector) and preliminary assess its performance in a "lab setting". Then, if the performance is good, we will use `Dataset U` to evaluate the classifier in an open-world setting (covered in Section 5). In the remainder, we use the terms "classifier" and "detector" interchangeably; a "positive" denotes a "malicious" sample.

To develop our classifiers, we empirically evaluated several classification algorithms available in scikit-learn (e.g., gradient boosting, naive bayes, and even deep neural networks) and eventually chose random forest [33], which provides the best detection performance (after cross-validation on the training set). This result is in line with prior work [46, 47].[4]

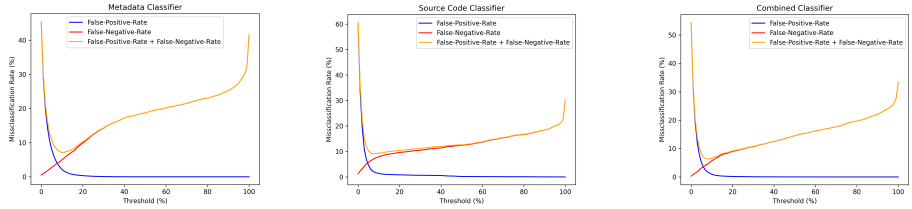
Given that the benign extensions in the training set vastly outnumber malicious extensions (i.e., the benign:malicious ratio is almost 9:1) we set the class weight to "balanced", which adjusts the weights inversely proportional to the number of samples in each class in the training set. To select the optimal set of hyperparameters, we performed 5-fold cross validation on the training set. We empirically inferred that 300 decision trees provide the best trade-off between detection accuracy and runtime performance for all three classifiers. As an original design choice, our classifiers seek to minimize both the false-positive and false-negative rates (instead of, e.g., focusing on minimizing just one of these two metrics[5]). We do this by leveraging the Youden's J statistic [79, 101]:

$$J = \text{sensitivity} + \text{specificity} - 1 = TPR - FPR$$

We empirically assessed which threshold value achieves the lowest cumulative sum of the false-negative and false-positive rates; and we averaged the value across the 5 cross-validation runs (always performed on the training set) for each of the three classifiers. We display the results in Figure 4. The optimal threshold value for the `Metadata classifier` is 11.6%; i.e., if 11.6% of the decision trees predict that an extension is malicious, it will be flagged as malicious. For the `Source code classifier`, the threshold is 8.8% and 9.2% for the `Combined classifier`.

[4]Indeed, tree-based algorithms are known [53] to be superior to deep learning ones for analyzing (security-related) tabular data—both in terms of detection performance and operational runtime [24]. In our case, a deep neural network (using a multi-layer perceptron) took 2.5x the training time and classified everything as benign.

[5]We will, however, also consider detectors that minimize the false-positive rate (see Section 7.3).



(a) `Metadata classifier` (b) `Source code classifier` (c) `Combined classifier`

Fig. 4. Evolution of the false-positive and false-negative rates depending on the threshold value

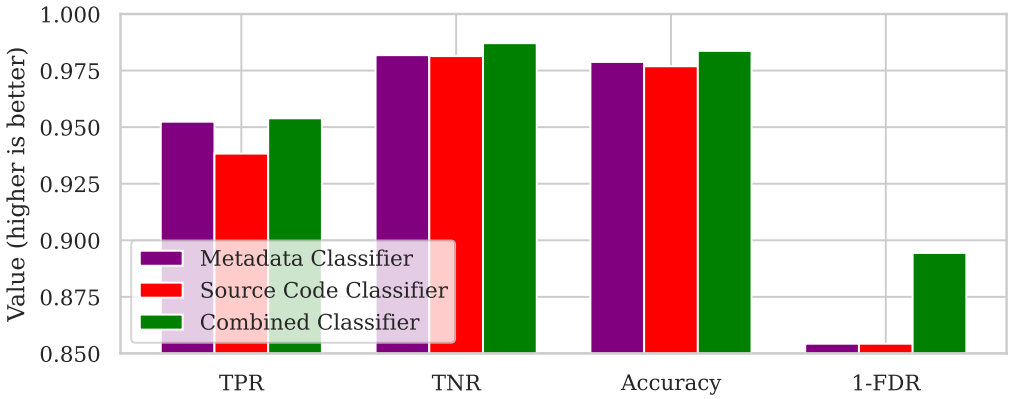


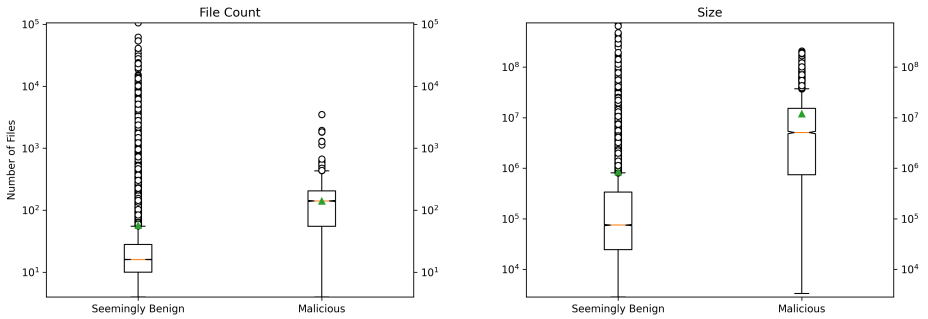
Fig. 5. Accuracy, TPR, TNR, and precision (1-FDR) of our classifiers

4.3 Evaluation (before real-world deployment)

We present in Figure 5 the results of our three classifiers (`Metadata`, `Source code`, and `Combined classifier`) on the test set of our labeled `Dataset L`. Let us discuss these—very encouraging—results.

4.3.1 `Source code classifier`. First, we consider our "baseline" classifier solely based on source-code features from JaSt [47]. This detector achieves a high accuracy, with 97.69% of the extensions being correctly classified (Figure 5). In addition, we have a high-true negative rate of 98.14% (12,107 / 12,337) and true-positive rate of 93.83% (1,324 / 1,411). These results reveal that our `Source code classifier` (which analyzes the JavaScript code of browser extensions) has a slightly inferior performance to JaSt (which was designed for detecting malicious JavaScript in Web pages or emails). Indeed, the results in [47] show an accuracy of 99.5%, true-positive rate of 99.46%, and true-negative rate of 99.48%. This means that, while the AST of benign and malicious extensions differ, the differences are less pronounced than on benign vs. malicious Web pages or emails. Nevertheless, our results suggest that using AST-related features can be an effective way to detect malicious browser extensions—representing a solid baseline, and validating our hypotheses.

4.3.2 `Metadata classifier`. Next, we focus on our "original" classifier. We find that malicious extensions can be detected with 97.88% accuracy by using only metadata information (Figure 5). In particular, 12,488 (out of 12,720) benign extensions are correctly classified as benign; this represents a high true-negative rate (TNR) of 98.18%. We also have a high true-positive rate (TPR) of 95.24%, with 1,360 (out of 1,428) malicious extensions correctly identified. However, due to the class imbalance,



(a) Number of files (b) Size of an extension in bytes

Fig. 6. Distribution of two features for benign & malicious extensions

we have a false-discovery rate (FDR) of 14.57%, i.e., of the 1,592 extensions flagged as malicious, 232 are actually benign—suggesting that there is room for improvement. Nonetheless, altogether, these results suggest that the `Metadata classifier` has similar performance to the `Source code classifier`. This is interesting given that the malicious functionality of extensions is reflected in their *source code*, and yet it is still possible to detect such malicious extensions by only using metadata-related information (a finding that contrasts what was suggested in some prior work [28]).

To better understand the logic behind the predictions of our classifier, we *studied the importance of each feature*, by leveraging the mean decrease in impurity [67]. The most important feature is the number of files included in an extension ("File Count"). As shown in Figure 6a, benign extensions include a median (orange line) of 16 files vs. 141 for malicious extensions. Similarly, benign extensions contain 57.23 files on average (green triangle) and malicious extensions 140.18. We assume that malicious extensions may split their source code into multiple files to make their analysis more difficult, which we empirically confirmed in Section 5.3, during our manual analyses. Besides summary and description keywords related to New Tab Extensions (e.g., "TAB", "WALLPAPER", or "NEW"), the fourth most important feature is related to the size of an extension. Figure 6b shows the distribution of the "Size" among benign and malicious extensions. Malicious extensions are larger (mean of 11.94 MB) than benign extensions (0.85 MB). In particular, we observed that malicious extensions contain obfuscated code and dead code to make their analysis more difficult. The fifth most important feature is the "Same Developer Count", which indicates how many extensions were published on the CWS by the same developer. While a developer with only benign extensions publishes on average 2.7 (median of 1) extensions, a malicious developer publishes 337.58 extensions on the CWS on average (median of 18). This finding shows that developers who upload malicious extensions tend to be "repeated offenders", and hence *should be closely monitored by Google* for future uploads.[6]

4.3.3 `Combined classifier`. Finally, we investigate to what extent the combination of metadata and source-code features improves the detection of malicious browser extensions. As shown in Figure 5, this detector outperforms the other two: it accurately classifies 98.37% of the extensions (13,524 out of 13,748). It also has a higher true-negative rate (98.71%) and higher true-positive rate

[6]Moreover, "repeated abuse" should result in the closure of the account [52], but such accounts were still able to publish hundreds of extensions. Nonetheless, once Google detects (and removes) malicious browser extensions by a given developer, other (malicious) extensions from the same developer can stay in the CWS for months before being removed by Google [55].

	Benign	Malicious	Total
`Metadata classifier`	31,669	3,793	35,462
`Source code classifier`	27,366	2,638	30,004
`Combined classifier`	27,039	2,965	30,004

Table 7. Open-world classification results on `Dataset U`, which require further analyses. N.b.: we cannot extract source-code features for some extensions due to parsing errors in their source files (Section 4.1.1)

(95.39%) than the other two classifiers. In addition, we have a false-discovery rate of 10.56% (159 false positives out of 1,505 extensions classified as malicious), which is lower than before.

4.3.4 Runtime Performance. We evaluated the operational runtime of our approach end-to-end, i.e., feature extraction and classification, plus training. We carry out our experiments on a server mounting four AMD EPYC 7H12 CPUs (each having 64 cores and 128 threads) and 2 TB RAM; we do not use GPUs. We parallelize our experiments to use all cores available. Overall, given an extension, 0.112s are required to extract the metadata features, and 0.673s for the source code features. To train our classifiers (on 80% of `Dataset L`), we need: 25s for `Metadata classifier`, 27s for `Source code classifier`, and 44s for `Combined classifier`. The classification time (per extension) is: 1.88ms for `Metadata classifier`, 2.81ms for `Source code classifier`, 4.69ms for `Combined classifier`. So, on our testbed, an extension can be analyzed end-to-end in less than 1 second with our classifiers.[7]

Takeaway Our three classifiers can detect malicious extensions with high accuracy (>97.69%) as well as low false-positive (<1.86%) and false-negative rates (<6.17%). The `Combined classifier` achieves the best results given the combination of both metadata and source code information. Moreover, our detectors can analyze an extension end-to-end in less than 1 second. Altogether, these results show that our detectors can *in theory* be used to automatically flag malicious extensions whenever a new extension is uploaded to the CWS.

5 Analyzing Extensions in the Open World

Our experiments in Section 4 showed that our classifiers perform well to detect malicious extensions in a "lab setting". Hence, we now assess our detectors' performance in an open-world setting—i.e., to scrutinize the presence of malicious extensions in our unlabeled `Dataset U`.

5.1 Classification of Unlabeled Extensions

Classifier Predictions — As described in Section 3.3.2, `Dataset U` has extensions that were published or received an update on the CWS *after Jan. 1, 2023*. The findings of prior work suggest [55] that this dataset may contain some malicious extensions that have not yet been detected by Google nor security researchers. Therefore, we analyze the 35,462 extensions from `Dataset U` with our three detectors. Table 7 summarizes the predictions of our classifiers. Surprisingly, between 2,638 (`Source code classifier`) and 3,793 (`Metadata classifier`) extensions are flagged as malicious. *This represents almost 10% of our dataset.* Even if we take the false-discovery rate of our classifiers into account (between 10.56–14.80%, see Section 4.3), we would still flag between 2,248 and 3,240 extensions as malicious with each classifier.

[7]We also repeated the ML-runtime experiments (training and testing) on a different setup: an AMD Ryzen 5800X3D @ 4.5GHz (with 8 cores and 16 threads) having 32GB of RAM. To simulate a worst-case scenario, we forced the usage of only one thread (which should significantly impact the training time). Over five independent runs, it takes an average of 140s to train the `Combined classifier`, and the test phase requires 2.2s, i.e., around 0.16ms per sample.

Intersection of Flagged Extensions — We have reason to believe that our results overestimate the number of malicious extensions—but confirming this hypothesis requires manual review of each extension flagged as malicious. However, it is not feasible to manually check *thousands* of extensions. Hence, as a best-effort strategy, and to increase the likelihood that a reported extension is truly malicious, we focus on extensions that are *classified as malicious by all three detectors*.[8] In this way, we are using our classifiers as an ensemble architecture. On our unlabeled `Dataset U`, there are 1,131 extensions that are classified as malicious by the ensemble. We will further scrutinize these extensions to identify which ones are truly malicious. To this end, we first consider using operational products, and then resort to manual analyses.

5.2 Effectiveness of Existing Security Tools

We first investigate if well-known security solutions can be used to provide meaningful analyses for the sake of ascertaining the "maliciousness" of browser extensions.

VirusTotal — As a starting point, we consider VirusTotal, which is well known in security research [104]. As of Oct. 2024, VirusTotal uses over 70 different security vendors (including Kaspersky, McAfee, and Avira) to analyze files, domains, IPs, and URLs to detect malware. However, the results of VirusTotal may be inconsistent across engines [104]. Moreover, a study carried out in 2017 [44] found that VirusTotal was able to detect only 5 out of 9k malicious extensions. Therefore, to estimate the reliability of VirusTotal to detect malicious browser extensions "today," we verified *if VirusTotal was able to detect known malicious extensions*. [Results] We sent our set of 7,140 *known* malicious extensions (recall that those extensions were flagged as malicious by Google itself) from `Dataset L` to the VirusTotal API. Of those, only 293 were flagged as malicious by at least one engine. More specifically: 120 were flagged by one vendor, 104 by two, 26 by three, 8 by four, 3 by five, 8 by 6–10 engines, 15 by 11–20, and 9 by 21–30. According to prior work [104], it is recommended to consider at least more than one engine to derive sound conclusions: this would lead to VirusTotal being able to detect only 173 malicious extensions out of the 7,140 we submitted—i.e., a *false-negative rate of 97.6%*. It is unclear why VirusTotal is not able to detect the huge majority of known malicious extensions identified by Google. So, this "negative result" (on recent data) led us to look for alternative solutions to analyze the 1,131 extensions from `Dataset U` flagged as malicious by our classifiers. Nevertheless, the fact that commercial security engines cannot detect *known* malicious extensions is a sign that more work is needed in this area.

CRXcavator — This tool, developed by Duo's Corporate Security Engineering team [8], provides a *Risk Score* that can be used to estimate the "maliciousness" of a given extension. It leverages factors, such as an extension permissions, inclusion of vulnerable third-party JavaScript libraries, or missing details from the associated description on the CWS to assign "points of risk" to each extension. The resulting Risk Score is the sum of all the points of risk: lower (higher) values denote smaller (greater) chances that an extension presents security risks. We hence retrieve the corresponding scores for the 1,131 extensions flagged as malicious by our ensemble on `Dataset U`. Moreover, to provide a more comprehensive overview, we also retrieve the risk scores for the remaining 34,331 "unlabeled" extensions of `Dataset U`, i.e., those that have not been flagged as malicious by the ensemble (and which we consider as less likely to be malicious). [Results] Table 8 shows the percentage of extensions with a given Risk Score. We see that 73.42% of flagged extensions have a "moderate" risk score, ranging from 400–599; in contrast, only 51.02% of the remaining extensions fall within this range. However, by focusing on extensions with the highest risk score (above 600),

[8]To justify this choice, we carry out this "intersectional" experiment on our labeled `Dataset L`. Overall, 1,321 extensions were classified as malicious by all three classifiers, and only 25 were false positives. This corresponds to a false-discovery rate of 1.89%, which is significantly lower than the 10.56–14.80% we found in Section 4.3. Moreover, the false-negative rate is only slightly impacted (8.15% vs 4.61–6.17%).

	<300	300-399	400-499	500-599	600-699	>699	#Ext
Flagged by all 3 classifiers	0.46	23.12	41.71	31.71	1.90	1.10	1,131
`Dataset U` minus flagged	0.34	44.47	44.52	6.50	2.17	2.00	34,331

Table 8. Risk Score (CRXcavator), in percent of extensions flagged as malicious by the 3 classifiers and remaining `Dataset U`

	0	1	2	3	4	#Ext
Flagged by all 3 classifiers	11.09	34.61	34.25	11.98	8.07	1,131
`Dataset U` minus flagged	12.32	6.18	53.96	26.13	1.41	34,331

Table 9. Risk Likelihood (Chrome-Stats), in percent of extensions flagged as malicious by the 3 classifiers and remaining `Dataset U`

we see that the remainder of `Dataset U` (which is less likely to be malicious) has more candidates than the extensions in the flagged set (which are more likely to be malicious). This is true both in absolute (1,431 vs 34) and relative (4.17% vs 3%) terms. Hence, the Risk Score is not a metric we can rely on to assess the maliciousness of the extensions flagged as malicious by our ensemble.

Chrome-Stats — As a final attempt, we consider another risk-based metric. Indeed, Chrome-Stats provides the *Risk Likelihood*, which measures the risk that an extension is malicious based on, e.g., the reputation of its developer and of the extension on the CWS, or how long it has been available. The risk is an integer between 0 (low) and 4 (high). To align this assessment with the one for CRXcavator, we retrieve the Risk Likelihood for all extensions in `Dataset U`. [Results] Table 9 shows the percentage of extensions with a given Risk Likelihood score. By focusing on the lower scores, we observe that almost half (45.7%) of the 1,131 extensions flagged as malicious by our ensemble have a low risk score of 1 or 0; and this percentage raises to 80% if we also include a "medium" risk score of 2. This may suggest that over 80% of our 1,131 extensions are not truly malicious. However, by focusing on the higher scores (denoting a higher risk of maliciousness), we see an intriguing result: while 20% (231) of the extensions flagged as malicious by our ensemble have a score of 3 or 4, this holds true for 27.54% (9,454) of the remaining extensions in `Dataset U`, which are less likely to be malicious. Moreover, by focusing on the *absolute number* of extensions with a score of 4, a total of 575 in `Dataset U` achieve the highest risk likelihood: if we deem these as malicious, it would mean that the CWS would have hosted 575 malicious extensions (on Nov. 16th, 2023) that were reported as malicious by Chrome-Stats—which we consider as an unrealistically high number. Hence, we believe that the Risk Likelihood metric cannot be used either to verify if our flagged extensions are truly malicious or not.

Takeaway Commercial products like VirusTotal perform poorly and cannot detect most extensions *known* to be malicious. Risk scores provided by Chrome-Stats and CRXcavator also provide contrasting results. Hence, these tools do not enable us to assess if the extensions flagged by our three detectors are indeed malicious.

5.3 Manual Analysis

Since we cannot rely on existing tools to identify malicious extensions, we manually reviewed a subset (200, i.e., 18%) of the 1,131 extensions flagged as malicious by our ensemble of three classifiers.

Methodology — To carry out a feasible manual analysis, we *(i)* group our extensions into clusters, similarly to [74], and then *(ii)* analyze a few extensions for each cluster. To this end, and inspired by prior work [55], we used ssdeep [19] to compute the fuzzy hash of the concatenated (content

	#Extensions	#Clusters
Proxying network requests through endpoints	2	1
Navigating the user to sites VirusTotal reports as malicious	3	3
Retrieving the list of friends and account details and publishing a Facebook post that includes the CPU model, CPU, and RAM usage automatically and without permission	2	1
Requests & exfiltrate credentials for popular websites	6	1
Exfiltrating the IP address, operating system, and a generated UUID to various endpoints	22	19
Collecting system information and exfiltrating it to endpoints reported as malicious on VirusTotal	2	1
Ad Blockers that send every visited URLs to an endpoint reported as malicious on VirusTotal	3	3
Total	40	29

Table 10. Summary of malicious behaviors manually identified

scripts, service worker) files, and we grouped extensions with a similarity score >90 in the same cluster. We could group 732 (out of 1,131) extensions into 93 clusters, having a minimum cluster size of 2 extensions, maximum of 157, and average of 8. We ignored the remaining extensions for our manual analysis, since they could not be grouped with others. Then, for each cluster, we inspected a subset of the extensions by reviewing their code. When necessary, we used js-deobfuscator [13] to deobfuscate JavaScript code. We examined the permissions requested in the `manifest` and verified what (sensitive) information an extension accesses and how it is processed. If any endpoint is contacted to send or retrieve content during execution (e.g., script inclusion, fetch, or XHR), we inspected the payloads of the requests and responses, and we verified with VirusTotal if the domain was malicious. For complex or large extensions, code analysis was challenging. Therefore, we also examined the runtime behavior of those extensions by installing them in an isolated environment and interacting with them, also using extensions' developer tools for breakpoint debugging and network request monitoring.

Manual Effort (qualitative analysis) — We observed a considerable disparity in the degree of (apparent) effort attackers invested in crafting their malicious extensions. As a result, the manual analysis time varied significantly: some extensions required only a few minutes to make an educated guess, as merely inspecting the code was sufficient to identify the malicious operations, while others necessitated several hours of detailed examination and demanded the use of multiple tools to uncover the malicious activities, such as debugger or network inspectors.

Results — Between our data collection (Nov. 2023) and manual analyses (May 2024), 28 of the 1,131 extensions flagged as malicious by our ensemble had already been removed from the CWS by Google for being malicious (overall, these 28 extensions affected >2M users before being taken down). Since, these extensions had been confirmed malicious by Google, we did not manually analyze them. Of the 200 extensions we manually reviewed, we identified 40 malicious extensions that *(i)* had not been detected previously and *(ii)* were still in the CWS before our disclosure to Google in May 2024; these malicious extensions belong to 29 clusters, as shown in Table 10. We observe different classes of malicious behavior: some malicious extensions, e.g., redirect users to malicious websites, steal user sensitive data, spy on users, or track them across sites. As a rule, we consider an extension to be malicious when it puts the security or privacy of its users at risk, in line with [55]. In total, our classifiers could therefore identify (at least) 68 malicious extensions

that were unknown to be malicious in Nov. 2023 when we collected our snapshot from the CWS; overall, the 40 malicious extensions still on the CWS in May 2024 affected 11M users (one had >5M users). Besides these, we found an extra 12 extensions with a "suspicious behavior". Similarly to [58], suspicious means that we could identify potentially harmful actions and risks to users but without certainty that these represent malicious actions. We have also inspected the Risk Likelihood score provided by Chrome-Stats for our 40 manually-verified malicious extensions: only 11 have a maximum Risk Likelihood of 4, revealing that harmful extensions do "bypass" such a metric—and justifying our decision not to rely on it for our analyses.[9]

Ethical Disclosure to Google (and post-acceptance update) — In May 2024, we disclosed to Google the 40 malicious extensions we identified in the CWS. Google responded in August 2024, acknowledging some of our findings. We did not receive any follow up from Google since the last message on August 2024. Moreover, as of September 2025 (i.e., *after our disclosure*) 17 of our 40 reported extensions have been taken down from the CWS (altogether, these 17 extensions had 1.76M users at the time of their removal) and another 17 have been "updated" (altogether, these 17 extensions now have 7.1M users). The remaining 6 extensions that we reported but have not been updated or taken down count 93k users. Hence, and under the assumption that the 17 "updated" extensions are not malicious anymore, it can be said that 99% of the users that had at least one of our reported malicious extensions installed are not affected by such a problem any longer.

Takeaway. Our manual analysis revealed that our ensemble of detectors detected *at least* 68 malicious extensions from the CWS that were unknown to be dangerous when we collected our data in Nov. 2023. Of those, Google took down 28 extensions (for being malicious) prior to our analyses; another 17 extensions were removed after we disclosed our findings to Google, and 17 have been updated. As of September 2025 only 6 are, unchanged, still on the CWS.

6 Concept Drift & Browser Extensions

Our open-world evaluation indicated that our detectors (even if organized in an ensemble) struggle to detect malicious extensions. Even though our manual investigation revealed that our classifiers identified some malicious extensions, the open-world results (in Section 5.1) denoted a stark contrast with the "near-perfect" performance achieved during the development phase of our detectors (in Section 4.3). We assert that such difference has its root on the intrinsic evolution of the browser extension ecosystem: every day, both benign and malicious extensions "change", thereby making it difficult for supervised ML-based classifiers to maintain their performance over long periods of time [50]—a phenomenon typically known as *concept drift* [24, 29]. To the best of our knowledge, no prior work attempted to "quantify" the presence of concept drift in the malicious browser extension context ("concept drift" was only hinted by [56, 91] in 2015, but never explicitly investigated).

In what follows, we provide factual evidence that concept drift does indeed affect this domain. To this end, we carry out new experiments by proceeding backwards.

6.1 Preliminary Assessment on `Dataset U`

As a first step, we examine our last experiments in Section 5. Recall (see Section 3.1) that the extensions in `Dataset U` have been published (or last updated) between Jan.–Nov., 2023. To get a rough idea of the impact of concept drift (which ultimately leads to misclassifications), we need a more recent dataset of confirmed malicious extensions. As a best-effort approach, we queried Chrome-Stats in Jan. 2024[10] looking for extensions that have been removed from the CWS for being

[9]We also manually reviewed 40 extensions with Risk Likelihood=4: at least 5 of these extensions were benign (i.e., FPR>12.5%).
[10]We did this experiment *before* our manual analysis (Section 5.3).

Classifier	Update date...	Benign	Malicious	Total	FNR
`Metadata classifier`	before 2023	51	1,318	1,369	3.73%
	in 2023	170	144	314	54.14%
`Source code classifier`	before 2023	83	1,273	1,356	6.12%
	in 2023	216	59	275	78.54%
`Combined classifier`	before 2023	51	1,305	1,356	3.76%
	in 2023	182	93	275	66.18%

Table 11. Classification results of *malicious* extensions published or last updated before vs. in 2023; the classifiers have been retrained without any extensions from 2023 in their training set

malicious *after November 2023*. We found that 60 of the extensions returned by our query were included in `Dataset U`.

Our `Metadata classifier` detects 41 (out of 60) extensions as malicious. This represents a false-negative rate of *at least* 31.66%. This result is in stark contrast with the false-negative rate of only 4.76% achieved during the development phase of this classifier (Section 4.3.2). We observe a similar trend for the `source code` and the `combined classifiers` (which, due to parsing errors of Esprima, could only analyze a subset of 41 of these 60 extensions): both of these classifiers can only detect 34 (out of 41) malicious extensions, i.e., a false-negative rate of *at least* 17.07% (up from 4.61–6.17%, see Sections 4.3.1 and 4.3.3).

Altogether, these results (based on malicious extensions flagged by Google) suggest that our initial hypothesis was likely correct: malicious extensions frequently mutate, thereby leading to evasion even of well-trained classifiers. However, these findings rely on a small sample and cannot strongly support our hypothesis by themselves.

6.2 Time-aware Evaluation on `Dataset L`

To provide further evidence of the existence and effects of concept drift in the browser extension ecosystem, we did another experiment by changing the training and test sets used for developing (and preliminary testing) our detectors. Recall that we created our train and test sets through an 80:20 split by randomly sampling from the entire `Dataset L` (see Section 3.3.1), which led to malicious extensions from 2023 being included both in the train and test sets. Hence, to investigate the temporal shifts in malicious extensions, we remove the extensions published in 2023 from the training set, and put them in the test set. This leads to a new training set with *no malicious extensions* published in 2023 (and 80% "benign" extensions from before 2023); and a new test set with all malicious extensions (taken down from the CWS) from 2023, as well as all malicious and benign extensions of the original test set.

We retrain our classifiers on the new training set, and then test their performance again. Table 11 shows the predictions of our classifiers on *malicious* extensions from the new test set, differentiating between the publication date of the extensions and the classification output (benign or malicious). We make two observations. First, the detection results of malicious extensions last updated *before* 2023 are very similar to those of Section 4.3: we have a false-negative rate (FNR) of 3.73–6.12% compared to 4.61–6.17% in Section 4.3. Second, the majority of malicious extensions that have been updated *in 2023* are not flagged as malicious by the classifiers: we observe false-negative rates ranging from 54.14% (`Metadata classifier`) to 78.54% (`Source code classifier`). Another intriguing finding is that the `Metadata classifier` has a remarkably lower FNR w.r.t. the `source code` and `Combined classifier`. This may be a sign that attackers are obfuscating their code or at

Rank	2019		2020		2021		2022		2023	
(top-5)	**Feature Name**	**Value**	**Feature Name**	**Value**	**Feature Name**	**Value**	**Feature Name**	**Value**	**Feature Name**	**Value**
#1	fileCount	0.0272	fileCount	0.0271	fileCount	0.0241	fileCount	0.0213	fileCount	0.0206
#2	Full-Summary LOVE	0.0212	Permissions topSites	0.0245	Full-Summary WALLPAPER	0.0223	Full-Summary WALLPAPER	0.0196	Full-Summary FAVORITE	0.0154
#3	Full-Summary HIGH	0.0211	Full-Summary THEME	0.0196	size	0.0178	size	0.0170	Description NEW	0.0151
#4	(source code #157)	0.0209	Full-Summary WALLPAPER	0.0185	Full-Summary FAVORITE	0.0170	Description NEW	0.0146	Full-Summary TIME	0.0131
#5	(source code #105)	0.0171	Full-Summary FAVORITE	0.0175	Description TAB	0.0164	Full-Summary THEME	0.0126	size	0.0117

Table 12. Evolution of the *feature importance* (of `Combined classifier`) after retraining it yearly on `Dataset L`

least hiding the harmful functionality of their malicious extensions, while some malicious patterns are still apparent in the metadata.

This additional experiment reinforces our hypothesis of the existence of concept drift in browser extensions. However, insofar, we have only considered *malicious* extensions (and only published in 2023). The following section will provide the last piece of the puzzle, revealing that the entire extension ecosystem is (and has been for years) affected by concept drift (from an ML perspective).

6.3 Further Validation: Longitudinal Analysis

"*Is the browser extension ecosystem affected by concept drift?*" To provide a convincing answer to this question, we use `Dataset L` to carry out a longitudinal analysis wherein we train and test the performance of our detectors over time. Inspired by [75], we proceed as follows: given any year Y between 2019–2022, we train our detectors on all extensions published or updated before Y, and test them on the extensions published or updated in Y. In this way, we can approximate a realistic scenario wherein we progressively assess the performance of our detectors if deployed in the real world (to analyze "new" extensions) over our considered 4-year timespan—thereby elucidating the degradation (if any) caused by concept drift. We provide in Figure 9 (in the Appendix) the temporal distribution of our extensions in `Dataset L`. We note that such an experiment is *fair* from an ML perspective: any given extension will only be included either in the train or in the test set—never in both (otherwise, this would naturally inflate the results due to data leakage). Our assumption is that, if there is no concept drift, then our results should be "good".

Results — First, we mention that we follow the exact hyperparameter-optimization procedure discussed in Section 4.2 to develop our classifiers, and that our cross-validation results (on each training set, and for each classifier) always yielded near-perfect detection performance. However, the outcome of our longitudinal analyses reveals a different story. We visualize these results in Figure 7 for the `Combined classifier`, and in Figure 10 (in the Appendix) for the `Metadata classifier`. These figures show the TPR, TNR, accuracy (Acc) and precision (1-FDR) across the years. For the `Combined classifier`, we see that the performance is somehow stable in 2019 and 2020 (despite it being retrained at the end of 2019 with the extensions which appeared in 2019). However, the performance of `Combined classifier` degrades substantially in the last two years. The TPR drops substantially (~0.5) denoting that malicious extensions present many differences from those "seen" during the training phase of our detectors. Moreover, we also note a remarkable drop in the precision, with drops to ~0.4 in 2021 and to ~0.3 in 2022. This denotes that more than half of the extensions flagged as malicious by our detectors were actually benign. Such results (which are reflected also by the `Metadata classifier`, as shown in Figure 10) are in stark contrast with those achieved during the cross-validation phase, and significantly different from those we presented in Section 4.3 (see Figure 5). In summary, these "negative results" are a strong indicator of concept drift.

Feature Importance — Previously, we discussed *how* browser extensions are affected by concept drift (by looking at the misclassifications of the detectors); now we focus on *why* extensions are affected by concept drift by analyzing the feature importance learned by our classifiers over the years. This is instructive to highlight "changes" in the data distribution, which may explain the

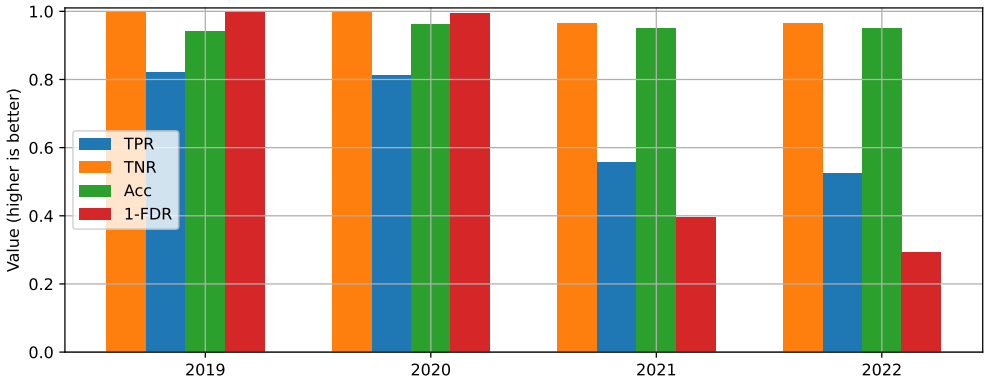


Fig. 7. Longitudinal test. We test the `Combined classifier` on the extensions updated every year from 2019–2022 after training it on the extensions published or updated in the previous years

source of concept drift. Hence, we retrieve the ranking of the top-5 most important features of the `Combined classifier` (which our initial assessment in Section 4.3 showed to perform best) after every year; we also report it for 2023 (i.e., by training the `Combined classifier` on extensions that appeared before Jan. 1, 2023). The results are in Table 12. We see that the "fileCount" feature is always ranked 1st. However, we also observe many changes over the years: the "Full-Summary WALLPAPER" feature is ranked 4th in 2020 and 2nd in 2021 and 2022, but is outside the top-5 in 2019 and 2023; "Permission topSites" is ranked 2nd in 2020, and outside the top-5 in any other years; source-code features (two) are in the top-5 only in 2019. This analysis indicates that, over the years, the overall feature distribution of malicious and benign extensions changes. Such dynamics explain *why* our detectors have an underwhelming performance when tested "outside the lab" to analyze extensions in the real world—on the CWS.

Takeaway We provided factual evidence that the browser extension ecosystem is also impacted by concept drift and discussed reasons why this detrimental phenomenon occurs. To reduce its impact, detectors need to be retrained on a regular basis—requiring a constant stream of new (and correctly labeled) benign and malicious extensions (see suggestions in Section 7.2).

7 Additional Experiments (and Potential Countermeasures)

Our datasets and classifiers can be used to shed further light on the evolution of the browser-extension landscape from a security viewpoint. We take this opportunity to carry out three complementary experiments, summarised below and discussed in the following subsections.

- **Susceptibility of Extensions to Concept Drift.** We investigate if certain *categories* (or *manifest versions*) of extensions lead to more prominent misclassifications, indicating a more impactful drift. For benign (resp. malicious) extensions, the "search-tools" (resp. "photos") category tends to have a very unstable behavior; moreover, the release of Manifest V3 (MV3) had a substantial impact on the detection of malicious extensions—which is an expected result.
- **Active Learning to Mitigate Concept Drift.** Active learning is a well-known technique to optimize the labeling efforts that can mitigate the impact of concept drift [24]. However, active learning has never been explored in the browser-extension ecosystem. We are the first to do so. We found that the best returns are achieved by re-training the classifiers monthly on 15 labeled extensions ("actively-suggested" via *uncertainty sampling*).
- **Alternative Design Choices for our Classifiers.** We explored *(i)* if the performance of our classifiers could be improved by better balancing the training sets—but none of our experiments

were successful; and also *(ii)* if it was possible to minimize the FPR (at the expense of the TPR) by tinkering with the detection threshold of our random-forest classifiers. We found that (in our "lab setup") we can reduce the FPR from 0.0107 to 0.00047 by sacrificing only 5% of the TPR.
The code of each experiment is provided in our repository [20].

7.1 Which Browser Extensions are more Impacted by Concept Drift?

Given that we have complete information on each extension, provided by Chrome-Stats, it is instructive to carry out a low-level analysis to measure the impact of concept drift on specific clusters of extensions. Specifically, we focus the attention on two aspects: *(i)* the *category* of extensions, and *(ii)* the Manifest version.

Categories of extensions — The goal of this analysis is to investigate the performance over time of our `Combined classifier`, but on specific categories of extensions: categories for which the performance is underwhelming are those that are most likely affected by concept drift. [*Method*] For clarity, we consider the nine most-prevalent categories of extensions included in `Dataset L`: "productivity", "fun", "web-development", "communication", "accessibility", "shopping", "photos", "news", and "search-tools"; we aggregate all remaining extensions (21 in total, i.e., less than 0.1% of `Dataset L`) in a single "other" group. We report in Table 13 the number of extensions for each group (and for each year), as well as the accuracy achieved by the `Combined classifier` on the respective category of extensions.[11] [*Analysis*] We see that, in the case of benign extensions, the three most problematic categories are "photos", "fun", and "communications": apparently, benign extensions of these categories become more similar to previously seen *malicious* extensions. In contrast, for malicious extensions, the most drifting categories are "search-tools", "productivity", and "communications": the classification accuracy is ≈50% every year. Nonetheless, and also as evidenced by the recent extensive analysis by Hsu et al. [55], certain categories of extensions do not have many malicious extensions (e.g., for "news", only 6 extensions are malicious).

Manifest version — We now focus on the current Manifest version (MV3), whose introduction in 2021 led to a substantial change in the browser extension ecosystem. Our goal is quantifying the extent to which the release of MV3 impacted the performance of our `Combined classifier`. We report the results of this study in Table 14. We find it intriguing that the performance of our `Combined classifier` on *benign* extensions has been minimally impacted by the release of MV3 given that the accuracy is always above 97%. However, for malicious extensions, the situation is substantially different: while the `Combined classifier` could detect over 80% of malicious extensions in both 2019 and 2020 (all being MV2 extensions), the detection accuracy on malicious MV3 extensions dropped to 15% in 2021 and only got slightly better (to 44%) in 2022. Such a fluctuation is because the `Combined classifier` was not trained on any MV3 extensions when they came out in 2021, which explains the underwhelming performance; after training the `Combined classifier` on the (few) malicious MV3 extensions that appeared in 2021, its performance improved—but over 55% of malicious MV3 extensions released in 2022 still bypass its detection. Put simply, this finding confirms the existence of concept drift in the browser extension ecosystem.

7.2 What are some Ways to Mitigate the Effects of Concept Drift?

It is evident that the browser-extension ecosystem is affected by a concept-drift problem: the constant changes in new browser extensions make it hard for classifiers based on supervised ML to retain practical performance. However, a promising strategy to mitigate the disruptive effects of such changes (which affect both benign and malicious extensions) is that of *active learning* [32].

[11] For these experiments, we use the same setup as in Section 6.3 (we simply breakdown the results according to the specific category). However, we use a fixed threshold=8.8% because the one of 9.2% was derived by analysing data until 2023, whereas here the training data changes incrementally. For simplicity and consistency, we hence use a slightly different threshold.

Category	Class	2019		2020		2021		2022	
		Accuracy	Total	Accuracy	Total	Accuracy	Total	Accuracy	Total
productivity	Benign	100.00%	2941	99.86%	4417	97.51%	5784	97.33%	7308
	Malicious	0.00%	34	27.38%	168	39.25%	186	47.86%	117
fun	Benign	100.00%	899	99.60%	1011	90.53%	1151	89.35%	1605
	Malicious	1.08%	9	30.65%	124	77.17%	127	83.33%	90
web-development	Benign	100.00%	886	99.90%	1048	98.36%	1160	98.79%	1488
	Malicious	0.00%	30	12.50%	16	26.67%	15	46.67%	15
communication	Benign	100.00%	727	99.79%	969	96.20%	1027	95.87%	1138
	Malicious	0.00%	18	13.11%	61	47.83%	23	29.63%	81
accessibility	Benign	99.85%	651	100.00%	870	96.74%	1195	97.65%	1447
	Malicious	0.00%	11	68.97%	261	50.00%	44	34.04%	47
shopping	Benign	100.00%	335	100.00%	541	98.44%	771	98.17%	763
	Malicious	-	0	-	0	93.10%	13	14.29%	7
photos	Benign	98.04%	51	100.00%	70	72.48%	109	77.86%	98
	Malicious	82.95%	2105	99.38%	70	88.89%	9	100.00%	8
news	Benign	100.00%	102	100.00%	164	96.89%	193	98.16%	163
	Malicious	100.00%	2	0.00%	2	100.00%	2	100.00%	2
search-tools	Benign	100.00%	1	100.00%	3	100.00%	1	100.00%	2
	Malicious	99.87%	763	37.50%	40	50.00%	28	63.89%	36
(others)	Benign	100.00%	11	100.00%	1	100.00%	1	100.00%	2
	Malicious	-	0	-	0	-	0	100.00%	1
TOTAL	Benign	99.97%	6514	99.85%	9093	96.51%	11391	96.35%	14056
	Malicious	82.04%	3061	81.25%	2139	55.70%	1465	52.49%	402

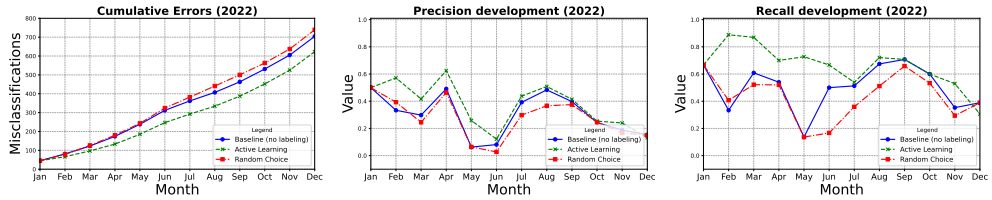
Table 13. Accuracy (by `Combined classifier`) and number of extensions for every year (cells in green highlight an accuracy over 70%)

Manifest	Class	2019		2020		2021		2022	
		Benign		Malicious		Benign		Malicious	
		Accuracy	Total	Accuracy	Total	Accuracy	Total	Accuracy	Total
MV2	Benign	99.97%	6,514	99.86%	9,093	95.93%	9,377	94.27%	3,841
	Malicious	82.03%	3,061	81.25%	2,139	61.43%	407	83.13%	83
MV3	Benign	-	-	-	-	99.21%	2,014	97.13%	10,215
	Malicious	-	-	-	-	15.52%	58	44.51%	319

Table 14. Accuracy (as by the `Combined classifier`) and total number of extensions per manifest version (MV2 vs. MV3) from 2019 to 2022

Hence, we will now assess how practical such techniques are in this domain via original experiments. We stress that, to the best of our knowledge, we are the first to empirically evaluate active-learning methods in the context of malicious browser extension detection—hence, our approach can be considered as an original technical contribution.

Design Choices — First, we consider the application of active learning based on *uncertainty sampling* [24, 75]. At a high level, active learning seeks to optimize the development process of ML-based classifiers by "actively suggesting" a small number of datapoints (in our case, extensions) that should be labeled—and then update the classifier by re-training it on these (new) labeled samples. The idea is that some extensions are more informative than others for detection purposes, and



(a) Cum. Errors (lower is better) (b) Precision (higher is better) (c) Recall (higher is better)

Fig. 8. Experiments with active learning. The blue line corresponds to the `Combined classifier` trained on all extensions before Jan 1st 2022 (and never re-trained). The green line corresponds to the `Combined classifier` that is re-trained monthly on an extra 15 extensions for which the model was *"the most uncertain"* the previous month. The red line corresponds to the `Combined classifier` that is monthly re-trained by including an extra 15 *randomly chosen* extensions of the previous month.

hence having the analyst label such "very informative extensions" can result in a better learning phase of the corresponding classifier. In the case of uncertainty sampling, the suggestions are provided by considering extensions for which the classifier is "uncertain": indeed, extensions for which the detector is certain that they are malicious (or benign) are likely very similar to those in the training set; in contrast, extensions for which the detector is uncertain are likely to present characteristics that would help the classifier to refine its decision boundaries.

Implementation — We carry out our experiments as follows. First, we consider a time-period of 12 months and, specifically, the entire 2022 year. This is because it is the most recent year for which we have complete availability of extensions with ground truth (see Section 3). Then, we consider only the `Combined classifier` (with the fixed threshold used in §7.1), since it is the classifier with the best performance (according to Section 4.3.3). In this setup, we then iteratively test and update (by means of active learning) the detector after every month. Specifically:

- we begin by training the `Combined classifier` by considering all extensions published (or last updated) before January 1st, 2022 (54,056 extensions);
- then, we test the detector on the extensions published (or last updated) in January 2022. Afterwards, we take the "probabilities" (provided by the *predict_proba* method of scikit-learn) and rank all these extensions according to those that are closer to the decision boundary—i.e., representing those extensions for which the classifier is the most uncertain;
- we select the K top-ranked extensions, and put them in the training set (by assigning the correct label). Finally, we retrain the "active-learning updated" `Combined classifier` (having 54,056+K extensions) and test it on extensions published (or last updated) in February 2022.

We repeat the procedure above for every month of 2022, i.e., in December 2022 our detector will be trained on a dataset of 56,056+(11*K) extensions—so that, every month, only the best K extensions are used to update the `Combined classifier`.

Evaluation and Results — For comparison, we consider the performance of the "baseline", representing the `Combined classifier` trained only with the extensions before January 1st, 2022; and also consider the performance of the `Combined classifier` trained by randomly choosing K extensions (instead of using active learning) every month, which can be considered an ablation study to ascertain the benefit of our active-learning implementation [24]. The performance metrics of choice are the precision and recall (measured on a monthly basis) as well as the cumulative errors (i.e., the overall number of misclassifications across the entire test window). We tested our implementation by considering various values of K: 10, 15, 25, 50; however, we obtained the best results (in terms of labeling effort and performance gains) with K=15 (for instance, we found no substantial improvement for K=50 despite requiring more than 3x the labeling effort). We report the results for K=15 in Figure 8. We can see that overall, *active learning provides a substantial benefit*

compared to the baseline (and also compared to the random choice), which is evident by observing the cumulative errors (Figure 8a). However, in some cases (e.g., for June 2022, see the drop in Figure 8b) the "new" extensions are so different from those which appeared before that our detector struggles even after the update. Another intriguing finding is that *updating the classifier by randomly drawing extensions can be detrimental*: in most plots, the "random choice" (red line) is worse than the "baseline" (blue line), despite the fact that the "random choice" is trained on a superset of the training set of the "baseline". This is why we recommend developers to adopt active-learning practices if they seek to deploy (and update) automatic classifiers based on supervised ML.

7.3 Alternative Design Choices for our Classifiers

To develop the classifiers used in our analysis, we adopted various design choices. Here, we describe some experiments wherein we explore alternative ways to develop our classifiers—which can be instructive for future work that seeks to improve our detectors.

Sub-sampling — The number of benign extensions in `Dataset L` is over 9 times larger than that of malicious extensions. To account for such class-imbalance, we set a "balanced" class weight when we defined our classifiers in scikit-learn. However, we have also experimented by considering subsets of (randomly drawn) benign extensions. Specifically, from our training set (i.e., 80% of `Dataset L`), we isolate 10%, 20%, 40%, 80% of the available benign extensions, which are used alongside all the available malicious extensions to train the `Combined classifier`. We then measure the performance on the test set (i.e., 20% holdout of `Dataset L`) and repeat the experiment five times to account for potential bias introduced by the random draw. We achieved the following results:

- For 10% (i.e., an equal composition of benign / malicious): *Acc*=0.60±0.01; *FPR*=0.43±0.01, *1-FDR*=0.20±0.01, *TPR*=0.99±0.005;
- For 20% (i.e., benign are twice the malicious): *Acc*=0.81±0.005; *FPR*=0.20±0.005, *1-FDR*=0.34±0.005, *TPR*=0.98±0.001;
- For 40% (i.e., benign are four times the malicious): *Acc*=0.93±0.01; *FPR*=0.07±0.01, *1-FDR*=0.62±0.01, *TPR*=0.97±0.01;
- For 80% (i.e., benign are eight times the malicious): *Acc*=0.98±0.01; *FPR*=0.01±0.01, *1-FDR*=0.86±0.01, *TPR*=0.96±0.01.

In light of these results, we can hence conclude that our choice of using all benign samples, and accounting for imbalance via the "balanced" class weight parameter, was a valid choice. We recommend not to adopt subsample strategies: doing so may lead to classifiers which are not provided with vital information to detect malicious extensions.

Minimizing the FPR — Our classifiers minimize the cumulative sum of false positives and false negatives. However, it is legitimate to opt for alternative strategies focused on, e.g., minimizing the rate of false positives. We explored such a possibility by choosing a different threshold and assessing its effects. We did this for the `Combined classifier`, given that it was the best detector. By looking at the plots in Figure 4, we chose a higher threshold=35% (instead of 9.2%, used in the experiments done in this paper), and we tested the resulting performance on the same test set (i.e., 20% of `Dataset L`); we also repeated this experiment five times to account for any potential fluctuation. We found that such a threshold leads to a detector with TPR=0.9054 and FPR=0.00047 (compared to a TPR=0.9551 and FPR=0.0096 achieved with a threshold of 9.2%). In other words, it is possible to substantially reduce the FPR at the cost of only 5% in the TPR. Therefore, future work that seeks to develop detectors that minimize the FPR can use such a strategy to better comply with certain requirements.

8 Discussion and Recommendations

We first summarize our major findings and then discuss the limitations of our research. Finally, we make some considerations on the robustness of our detectors to targeted evasion attempts.

8.1 Major Findings

We elucidated three novel and complementary discoveries.

First, in Section 4, we developed three classifiers (using a mix of novel feature engineering and prior work) to detect malicious browser extensions. We showed that these classifiers performed well on a dataset of real browser extensions collected from the official gallery (see Section 3), and our extracted features also provide insights on underlying properties of benign/malicious extensions. Moreover, we also showed that our detectors can be leveraged to identify malicious extensions "overlooked" by Google on the CWS (we identified 68 malicious extensions, impacting over 13M users). We publicly release the implementation of our classifiers [20]. This contribution is an important stepping stone for future work: we are not aware of open-source detectors of malicious browser extensions that have been validated on large datasets of browser extensions (we discuss related work in Section 9).

Second, in Section 5, we found that automated solutions for analyzing browser extensions—with the aim of detecting *malicious* ones—present limitations. This spans not only our custom-developed classifiers, but also supervised-ML-based detectors proposed by prior work (e.g., JaSt [47] ultimately seeks to detect malicious JavaScript, which is a crucial component in browser extensions) as well as commercial tools such as VirusTotal and CRXcavator. We also observe that even though VirusTotal had been shown to be not very reliable by DeKoven et al. [44], their analysis was carried out in 2017, i.e., over 8 years ago; in contrast, our analysis is more recent. Altogether, these findings underscore that *automatically* detecting malicious browser extensions is a fundamentally hard problem and that manual analysis is still needed for accurate verification of ground truth.

Third, the collective findings in Section 6 (on both `Dataset L` and `Dataset U`), reveal that concept drift significantly affects the browser extension ecosystem—explaining why applying supervised-ML for automated detection of malicious browser extensions is challenging.[12] Over time, malicious extensions become "more similar" to benign extensions that appeared previously; and, in turn, some benign extensions also become "more similar" to malicious ones—leading to misclassifications that degrade the overall performance of detectors based on supervised ML.[13] This discovery, which echoes the results achieved by prior work in other cyber security applications of supervised ML classifiers (e.g., Android or Windows malware [57], intrusion detection [100]), emphasizes the need for more work in the browser-extension context to address this problem.

Remark. Our last finding is *fair*. When we begun our research, we did not expect to find such a strong concept-drift effect. The detectors used in our analysis (both the novel ones and those based on prior work) achieved good performance on `Dataset L`. We thought our initial goal (i.e., developing automated and practical detectors of malicious browser extensions by relying on supervised ML) was achieved—but we were wrong: despite being automated, the performance of our detectors is underwhelming in practice.

[12] Note that the decreased performance is not due to "overfitting" [45]. Our experiments in our lab setting showed good performance on a test set disjoint from the training set—which is the correct evaluation protocol by assuming that the i.i.d. assumption holds. Our time-aware analysis revealed that the issue stems from the fact that the generative process of `Dataset L` does not follow an i.i.d. assumption, but rather follows the chaotic evolution of the browser-extension ecosystem. Hence the samples in the test set should be drawn from those that chronologically follow the samples in the training set.

[13] Note: the purpose of our research was to examine the feasibility of applying supervised ML to detect malicious browser extensions. Our findings indicate that doing so is not easy. We acknowledge that there exist other ML-based approaches that could potentially be used for the same purpose (e.g., some works proposed using LLMs in the browser-extension context, but for ancillary tasks [70]). Yet, our results suggest that any sort of "machine-based" detector requires constant updating due to the dynamic nature of the browser-extension ecosystem.

8.2 Limitations

For our research, we relied on various "best-effort" strategies, and we encountered some errors in the implementation of some methods. However, we have reason to believe that none of such occurrences threaten the validity of our conclusions.

From an ML perspective, it is desirable if a classifier is trained on a "sufficiently large" [24] number of datapoints—and, ideally, if the various classes are relatively balanced. Unfortunately, collecting a meaningful dataset of *verified* benign and malicious extensions *for realistic* experiments is challenging. The analysis carried out in our paper relies on a set of 106,200 extensions, of which 7,140 are known to be malicious, while the remainder are ultimately unverified:[14] we assumed (and justified—see Section 3.3) that those (63,598) whose last update occurred before Jan. 1, 2023 are "benign", but such an hypothesis may not hold universally, and we cannot exclude that our choice may have impacted our results. However, our choice is based on the findings of a very recent work [55] revealing that only dozens of malicious extensions remain in the CWS *for years*. Given that our `Dataset L` contains 63k extensions (from the CWS) whose last update spans across 2012–2022, the likelihood that the number of mislabeled benign extensions in `Dataset L` could have impacted our results to the point of threatening our conclusions is negligible. For instance, our temporal analysis does show that malicious extensions (for which we do have verified ground truth) are subject to frequent changes over the years. Nonetheless, it is well known that providing accurate ground truth is hard [24, 32], and even practitioners adopt coarse labeling strategies [32, 93]. Nevertheless, we could have "artificially" increased the number of extensions (thereby extending our datasets) by considering individual versions of an extension as a stand-alone extension: we did not do so because such an approach would skew the results, since it would strongly violate the "independent and identically distributed random variables" assumption of ML [24].

As justified in Section 2.2, we rely on `manifest`, content scripts, and service worker (and Chrome-Stats) to extract the features to detect malicious extensions. Our choice is confirmed by our empirical analysis (only 7% of the malicious extensions in Chrome-Stats are malicious despite not having these files). Our analysis thus provides a lower bound of malicious extensions. In other words, there may be other ways to develop detectors of malicious browser extensions via supervised ML. However, our results (in Section 4.3) suggested a good performance of our classifiers, thereby justifying our subsequent analyses. Nonetheless, we leave the investigation of malicious behaviors included in elements not considered in our analysis to future work. Our publicly available tools [20] should facilitate development of such "enhanced" detectors.

As mentioned in Section 4.1.1, parsing errors of Esprima prevented us from analyzing some extensions with JaSt [47]. We reduced the syntax errors by: *(i)* updating the Esprima version used by JaSt in 2018, which enabled us to analyze new constructs such as Optional Chaining; and *(ii)* concatenating content scripts with service worker and putting them in two separate `BlockStatement`, which avoided errors for duplicate constant values. Out of 106,200 extensions, Esprima could not parse 7,448. Given our goals, we did not omit these, which could still be analyzed by `Metadata classifier`. Moreover, as co-authors of JaSt [47], we confirm that using JaSt to detect malicious source code is sensible in the browser-extension context, also because browser extensions do not even use as many code-transformation techniques as client-side JavaScript.

8.3 Robustness Considerations

In our threat model (see Section 2.3), we envision an attacker that does not deliberately attempt to evade our detectors. This is motivated by the fact that, to the best of our knowledge, we are the first

[14] Such a lack of ground truth prevents one from using existing tools/methods to *reliably* find out if an extension is malicious or not (and our assessment on "commercial" detectors revealed that existing tools are not very accurate).

to develop (and openly share) detectors powered by supervised ML to flag malicious extensions. However, we acknowledge that real-world attackers may resort to adversarial tactics to bypass our detectors—especially in a white-box setting (which is made possible given our public release of our implementation). Here, we discuss some ways attackers may use to have their malicious extensions be classified as benign by our detectors. We argue that doing so is is possible, but not trivial.

First, attackers may try to use code-transformation techniques [68], such as obfuscation (which is a clear violation of the CWS policies [51]), to conceal malicious functionalities. In these cases, our source-code features may be impacted; however, given that such features are analyzed by JaSt, and given that JaSt was designed to handle obfuscated code [47], it is safe to assume that attempting to evade our `source-code` and `combined classifiers` in this way is unlikely to succeed. Alternatively, attackers may try to embed malicious functionalities in segments of an extension that are not included in JavaScript (e.g., by manipulating the CSS [78]). These attempts would be unnoticed by the `source-code classifier`; yet, such malicious extensions may still be detected by the `metadata` or `combined classifiers` because signals of maliciousness may be reflected in the features extracted from the extension's metadata.

Attackers with knowledge of the features (and, particularly, the metadata-related) used by our classifiers may try to avoid detection by inducing manipulations that alter features important for a correct classification. For instance, being aware of our findings in Table 4 or in Table 12 may induce attackers to avoid mentioning terms such as "WALLPAPER" in an extension's summary, since this is a common occurrence across the malicious extensions in our datasets (as also found in [77]). Such attempts may make detection harder for the `metadata classifier`, but would not impact the source-code features used by the `source-code` and `combined classifier`.

In other words, evading the `combined classifier` is not simple because, by analyzing both metadata and source-code features, it can cover the blind spots of the `metadata` and `source-code classifiers`. Moreover, it is possible to use fuzzying or feature-fusion methods to make such evasion attempts more difficult [25, 27]; or even preventing them altogether via feature-removal [83]. Nevertheless, the worst threat against our detectors are "adversarial ML attacks" [25], which can be staged with full-knowledge of (and/or complete access to) the targeted classifier. In these cases, attackers may guess the manipulation that guarantees evasion—but in the feature space. However, due to the "inverse-mapping" problem, it is not trivial to understand how to manipulate the extension *in the problem space* so as to induce a feature-representation that leads to evasion [42].

Assessing the effectiveness and feasibility of all these different adversarial use-cases is a potential avenue for future work—which are facilitated by the full release of our resources [20].

9 Related Work

We discuss prior work on the security of browser extensions, and then highlight the major differences with competing papers, emphasizing our novelty.

9.1 Browser Extensions and Security (& Privacy)

Early works in this domain are the 2012 papers by Carlini et al. and Liu et al., who assessed the effectiveness of Chrome security mechanisms by respectively semi-manually reviewing only 100 extensions [38] and designing practical attacks through extensions [66]. After these seminal works, abundant prior literature showed that *malicious extensions* can, e.g., track users [97], inject ads [91], hijack Facebook accounts [56], facilitate malvertising [99], conceal botnets [76], or exfiltrate sensitive user information [23, 39, 74]. These efforts confirm that malicious extensions pose a plethora of risks to (millions of) Web users.

Among the most recent related studies, Hsu et al. [55] investigate the prevalence of "security-noteworthy extensions" in the CWS. However, Hsu et al. [55] do not focus on detecting such

security threats: aside from showing that malicious extensions have some similarities (which is an assumption we use to base our research), Hsu et al. [55] do not propose any detector (plus there is no "open world" assessment as we did in Section 5), and their findings are entirely based on ground truth provided by Google (via their proprietary and closed-source mechanisms).

We focus on *malicious* extensions, i.e., extensions designed by malicious actors to harm victims. However, there are other classes of extensions that are source of security or privacy concerns. In particular, *fingerprintable* extensions (covered by, e.g., [22, 60, 61, 65, 80–82, 84, 85, 88, 90, 92]), can be abused by attackers to (uniquely) identify users having a specific set of extensions installed on their browser, leading to, e.g., user tracking. Also, *vulnerable* extensions (covered by, e.g., [35, 37, 48, 63, 87, 102]), are designed by well-intentioned developers, but attackers can exploit their elevated privileges to put the security and privacy of the extension users at risk. These types of extensions (fingerprintable/vulnerable) are outside our scope (as also remarked in §2.3).

9.2 Detecting Malicious Browser Extensions

Many works proposed ways to "detect" malicious extensions. After providing a summary, we carry out a systematic literature review to underscore our unique traits.

Closely related to us, Jagpal et al. developed WebEval, a semi-automatic system that is (or was) used by Google to analyze ≈99k extensions between 2012–2015 [56]. However, WebEval is *not publicly available*, and also *requires regular inputs from human experts* to limit false negatives; furthermore, the findings of [56] are *almost 10 years old*, and the landscape of browser extensions has substantially changed since 2015 [55]. Equally close to us, Wang et al. [96] leveraged 51 static and dynamic features to train an SVM classifier to detect malicious Chrome extensions, and achieve 96% accuracy on a dataset of nearly 5k extensions collected in 2015–2016. However, they handpicked their features also on the test set, hence overfitting; furthermore, they do not carry out a temporal evaluation—which we proved is necessary to ensure that the results reflect the real world. Also related, in 2018, Aggarwal et al. [23] developed a classifier to detect "spying" extensions, and achieve 90% precision on a dataset of 43k extensions (with manually-verified ground truth). However, their experimental pipeline also does not account for the temporal axis, leading to overestimation of their detector's performance in the long term.

Some works also focused on detecting malicious extensions via heuristics or dynamic analyses supported by *unsupervised* ML—which is a different category of ML-based techniques that, despite not requiring labeled data, necessitate custom-defined rules/heuristics to detect malicious extensions. Such works typically find only a small number of malicious extensions via manual verification. For instance, Kapravelos et al. [58] analyzed 48k Chrome extensions and found 130 malicious ones; Wang et al. [95] analyzed 2.5k Firefox extensions, finding no malicious behavior. Olsson et al. [72] used fake reviews on the CWS to identify 86 malicious extensions among 115k. Notably, Pantelaios et al. [74] leveraged the evolution of extensions' ratings to identify potentially malicious clusters, finding 143 malicious extensions out of over 206k.[15] Nonetheless, some of our metadata features are inspired by [72, 74], albeit we use these in a supervised-ML context.

9.3 Systematic Literature Analysis

We carry out a systematic literature review to highlight some relevant differences w.r.t. prior work, but also to corroborate our previous claims (see Sections 1 and 2.3). Specifically, we show the *lack*

[15]In [74], it is stated 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"—thereby admitting that their tool has limitations which we sought to overcome through the application of supervised-ML methods.

Paper (1st author)	Year	Considered Browser	Code Public?	Detect Malicious?	Supervised ML used?	Number of Extensions
Kapravelos [58]	2014	C		✓	✗	48k
Jagpal [56]	2015	C		✓	✓	100k
Kurt [91]	2015	C,F,E		✓	✗	50k
Xing [99]	2015	C		✓	✗	18k
Buyukkayhan [35]	2016	F		✓	✗	323
Sanchez-Rola [80]	2017	C,F,S				21k
Starov [90]	2017	C				10k
Starov [89]	2017	C		✓	✗	10k
Weissbacher [97]	2017	C,F		✓	✓	43k
Aggarwal [23]	2018	C		✓	✓	178k
Chen [39]	2018	C,O	✓	✓	✗	19k
Trickel [92]	2019	C	✓			11k
Sjosten [82]	2019	F,C				11k
Some [87]	2019	F,C,O				79k
Starov [88]	2019	C	✓			58k
Karami [60]	2020	C				29k
Pantelaious [74]	2020	C	✓	✓	✗	209k
Borgolte [31]	2020	C,F	✓			8
Laperdrix [65]	2021	C	✓			116k
Fass [48]	2021	C,F	✓			154k
Solomos [84]	2022	C	✓			40k
Karami [61]	2022	C	✓			5.7k
Solomos [85]	2022	C				38k
Picazo-Sanchez [77]	2022	C		✓	✓	159k
Agarwal [21]	2022	C,F	✓			186k
Kim [63]	2023	C,F,S	✗			<100
Bui [34]	2023	C				47k
Moreno [69]	2023	C	✓			–
Yu [102]	2023	C	✓			145k
Xie [98]	2024	C	✓			113k
Nayak [70]	2024	C		✓	✗	160k
Olsson [72]	2024	C				115k
Hsu [55]	2024	C				226k
Solomos [86]	2024	C				60k
Agarwal [22]	2024	C,F	✓			88k
This work	2024	C	[20]	✓	✓	106k

Table 15. **State of the Art.** Papers having "extension" in the title published in 2014–2024 in top-tier conferences. For each paper, we report the considered browser (F=Firefox, C=Chrome, O=Opera, S=Safari, E=Explorer), if the code is public (if so, the ✓ links to the repository; the link for [63] is broken), if it focuses on the "detection" of malicious extensions (and, if so, if supervised ML is used), and the number of extensions analyzed.

of public source code, preventing direct comparison; and the *non-reliance on supervised-ML methods*, preventing assessments of "concept drift" (which is a problem pertaining to supervised ML [50]).

Method — We consider *all papers published in top-tier conferences having "extensions" in the title.* We considered the 2014–2024 editions of: USENIX Sec, TheWebConf, IEEE S&P / EuroS&P, ACM CCS / AsiaCCS, and NDSS. We found 35 papers that matched our criteria, which were then inspected independently by two researchers. Specifically, we first analyzed the abstract to determine if the paper was about "detection of malicious browser extensions" (we resolved cases of disagreement via discussions). Then, we did whole-text keyword searches for three common ML-related terms ("machine learning", "training", "deep learning"), and if we found more than 3 matches we inspected the paper to see if some form of supervised ML / classifier was used in the detection procedure. For the code availability, we looked for links to repositories, and checked if they were still active.

Results — The results are in Table 15, in which we also report the considered browser and size of the dataset (in terms of unique extensions) used in the assessment (if any). At a high level, we see that, with the exception of one paper [35], all other works consider Chrome, confirming our choice to focus on this Web browser; we also see that, in terms of dataset size, only eight works (i.e., [21, 23, 48, 55, 70, 74, 77, 102]) carry out a substantially larger (i.e., >30%) assessment than ours. Let us focus on the most crucial factors of our systematic literature review:

- *(Not) Open Source:* only 15 (out of 35) papers publicly release their code; however, the repository of one of these (i.e., [63]) is not active (as of July 2025). This result echoes that of a recent work revealing that less than half of papers published at top-tier conferences release their artifacts [73].

- *Different Focus:* only 12 (out of 35) papers focus on the "detection of malicious extensions"; other works focus on different classes of security-related problems of the browser-extension ecosystem (e.g., [98] focuses on detecting vulnerable extensions, whereas [69] shows a vulnerability of Chromium-based browsers, and do not carry out any analysis of browser extensions).
- *Limited Consideration of Supervised ML:* of the 12 papers that focus on the detection of malicious browser extensions, only 4 (i.e., [23, 56, 77, 97]) attempt to do so by relying on some form of supervised-ML technique (e.g., [74] uses unsupervised methods)–and their code is not public.

We also looked for occurrences of "concept drift": no paper among these 35 mentioned this term.

Consequences — Let us explain how the aforementioned results affect our research. First, *there is no work that uses supervised ML to detect malicious browser extensions and which publicly shares its implementation* (e.g., [77] uses supervised ML, but there is no code available). This result implicitly prevents any sort of fair comparison with a previously proposed method; and it also reinforces our choice to consider JaSt [47] as the most valid baseline (despite not being designed to work on browser extensions). Second, we observe that there are two works (i.e., [39, 74]) for which an artifact is available and that focus on the detection of malicious browser extensions, but not by means of supervised-ML. However, we cannot compare against [74] due to a substantially different setup: our datasets include a single version for each extension, whereas the method proposed in [74] require *(i)* multiple versions of each extensions, and—crucially—that *(ii)* only the last version of a malicious extension is malicious, whereas all the previous ones are benign. Due to lack of ground truth on benign extensions, we cannot be sure if the previous version of the malicious extensions contained in our datasets are truly benign (indeed, Hsu et al. [55] found that malicious extensions can stay in the CWS for years). Nevertheless, we tried using the open-source implementation of the method proposed in [39], but we did not succeed. This negative result echoes a remark made in [98], stating that *some prior tools are "incapable of operating in a modern context" due to advances in "modern browsers, the expressiveness of extensions APIs, and the web itself".* We also contacted the authors of [39, 58], who confirmed that these tools are not possible to compare against anymore due the significant evolution of the Chrome browser and its extension ecosystem, which led to deprecated APIs not supported by current versions of Chrome. We believe that this is yet another finding supporting our claim that the browser-extension ecosystem is affected by concept drift.

10 Conclusions and Lessons Learned

We performed a security assessment, of the extensions published on the Chrome Web Store (CWS). We collected 7,140 known malicious extensions (removed from the CWS by Google) and provided by Chrome-Stats, along with 63,598 benign extensions available on the CWS and last updated before 2023. We devised and implemented three supervised-ML-based classifiers (including one based entirely on prior work [47]) using features we extracted from extensions' metadata and source code. These classifiers *perform well* in a "lab setting", with an accuracy of 98% and can analyze an extension in less than 1s end-to-end (even on commodity hardware). Subsequently, we show that these detectors are *underwhelming in an open-world setting* by using them to analyze an extra 35k extensions from the CWS with unverified ground truth. Despite enabling us to identify 68 malicious extensions (which were still present in the CWS at the time of our analysis, and which cumulatively affected over 13M users), our detectors flag over 1k extensions as malicious—which is an unrealistically high number that may conceal many false positives. We also provide evidence that *commercial detectors perform poorly* to detect known malicious extensions, i.e., *existing methods may give a false sense of security*—confirmed by our tests showing that, e.g., the detectors of VirusTotal have a false-negative rate over 97%. Finally, we show that the browser extension ecosystem is *affected by concept drift.* Altogether, our results serve as a foundation for researchers

and practitioners to improve the detection of malicious extensions in the CWS, which *affect millions of end-users*.

We can hence derive **lessons learned** addressed to the entire browser-extension community:

- First, *ML evaluations can be misleading*: a detector performing well in a research setting is no guarantee of practical applicability in an open-world scenario. In fact, to the best of our knowledge, we are the first to demonstrate that the browser extension ecosystem is affected by concept drift. Hence, from the perspective of a researcher, it is crucial to train and test models via time-aware evaluations for real-world assessments. From a practical perspective, we underscore that detectors' performance deteriorate quickly and need constant updates with new labeled data, i.e., both benign and malicious. This, however, leads to another problem: labeling extensions is hard. As we showed, existing tools, e.g., VirusTotal, are unreliable.
- Second, to improve the security of the browser extension ecosystem, *multiple stakeholders must cooperate*. End users should be willing to report suspicious extensions. Security researchers need to design more reliable tools and carry out more realistic security assessments. Maintainers of extension galleries should be more active in fighting these threats: e.g., the CWS contains numerous malicious extensions made by developers known to publish dangerous extensions.
- Third, to spearhead development of appropriate solutions, we have (for the first time) carried out experiments on *active learning* (described in Section 7.2): our findings (in Figure 8) suggest that it is possible to optimally improve the performance of our classifiers by retraining them on a monthly basis on a small number (we found 15 is a sweet spot) of correctly labeled extensions.

Finally, our findings mostly pertain to Chrome. However, our approach (and, likely, our conclusions) is also applicable to other browsers, such as Firefox and Microsoft Edge, with minor adjustments, e.g., by leveraging data from Firefox-Stats [12] and Edge-Stats [11].

Acknowledgments

We would like to thank ChromeStats [2], and more specifically Hao Nguyen, for providing us with Chrome extensions, their metadata, and additional information or support upon request. We also thank the anonymous TWEB reviewers for their constructive reviews and helpful feedback. Parts of this research was funded by the Hilti foundation.

References

[1] [n. d.]. *Check on your review status.* https://developer.chrome.com/docs/webstore/check-review?hl=en#the_lifecycle_of_a_chrome_web_store_item Accessed on 2024-10-24.
[2] [n. d.]. Chrome-Stats. https://chrome-stats.com. Accessed on 2024-05-21.
[3] [n. d.]. *Chrome-Stats Advanced Search.* https://chrome-stats.com/advanced-search Accessed on 2024-04-03.
[4] [n. d.]. *Chrome Web Store.* https://chrome.google.com/webstore Accessed on 2023-11-15.
[5] [n. d.]. *Chrome Web Store review process.* https://developer.chrome.com/docs/webstore/review-process Accessed on 2024-10-24.
[6] [n. d.]. *Chrome Web Store Sitemap.* https://chrome.google.com/webstore/sitemap Accessed on 2024-04-03.
[7] [n. d.]. *Content scripts.* https://developer.chrome.com/docs/extensions/mv3/content_scripts/ Accessed on 2024-10-23.
[8] [n. d.]. CRXcavator. https://crxcavator.io. Accessed on 2024-07-10.
[9] [n. d.]. *Declare permissions.* https://developer.chrome.com/docs/extensions/develop/concepts/declare-permissions Accessed on 2024-10-23.
[10] [n. d.]. *DoubleX Unpacker.* https://github.com/Aurore54F/DoubleX/blob/main/src/unpack_extension.py Accessed on 2024-04-15.
[11] [n. d.]. *Edge-Stats.* https://edge-stats.com/ Accessed on 2024-10-24.
[12] [n. d.]. *Firefox-Stats.* https://firefox-stats.com/ Accessed on 2024-10-24.
[13] [n. d.]. *JavaScript deobfuscator.* https://github.com/ben-sb/javascript-deobfuscator Accessed on 2024-05-28.
[14] [n. d.]. *Manifest file format.* https://developer.chrome.com/docs/extensions/reference/manifest Accessed on 2024-05-21.
[15] [n. d.]. Manifest V3 migration checklist. https://developer.chrome.com/docs/extensions/develop/migrate/checklist. Accessed on 2024-10-24.

[16] [n. d.]. *Migrate to a service worker.* https://developer.chrome.com/docs/extensions/develop/migrate/to-service-workers Accessed on 2024-10-24.
[17] [n. d.]. *Permissions.* https://developer.chrome.com/docs/extensions/reference/permissions-list Accessed on 2024-10-23.
[18] [n. d.]. *spaCy.* https://spacy.io/ Accessed on 2024-04-09.
[19] [n. d.]. *ssdeep 3.4.* https://pypi.org/project/ssdeep Accessed on 2024-05-28.
[20] 2025. Our Repository. https://github.com/its-not-easy/tweb25.
[21] Shubham Agarwal. 2022. Helping or Hindering? How Browser Extensions Undermine Security. In *ACM CCS*.
[22] Shubham Agarwal, Aurore Fass, and Ben Stock. 2024. Peeking through the window: Fingerprinting Browser Extensions through Page-Visible Execution Traces and Interactions. In *ACM CCS*.
[23] Anupama Aggarwal, Bimal Viswanath, Liang Zhang, Saravana Kumar, Ayush Shah, and Ponnurangam Kumaraguru. 2018. I Spy with My Little Eye: Analysis and Detection of Spying Browser Extensions. In *IEEE EuroS&P*.
[24] Giovanni Apruzzese, Pavel Laskov, and Aliya Tastemirova. 2022. SoK: The impact of unlabelled data in cyberthreat detection. In *IEEE EuroS&P*.
[25] Giovanni Apruzzese and VS Subrahmanian. 2022. Mitigating adversarial gray-box attacks against phishing detectors. *IEEE Transactions on Dependable and Secure Computing* 20, 5 (2022), 3753–3769.
[26] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and don'ts of machine learning in computer security. In *USENIX Security*.
[27] Abiodun Ayantayo, Amrit Kaur, Anit Kour, Xavier Schmoor, Fayyaz Shah, Ian Vickers, Paul Kearney, and Mohammed M Abdelsamea. 2023. Network intrusion detection using feature fusion with deep learning. *Journal of Big Data* (2023).
[28] Dénes Bán and Benjamin Livshits. 2019. Extension Vetting: Haven't We Solved This Problem Yet?. In *NDSS Workshop MADWeb*.
[29] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending transcend: Revisiting malware classification in the presence of concept drift. In *IEEE S&P*.
[30] Google Security Blog. Accessed on 2024-07-09. Staying Safe with Chrome Extensions. https://security.googleblog.com/2024/06/staying-safe-with-chrome-extensions.html.
[31] Kevin Borgolte and Nick Feamster. 2020. Understanding the performance costs and benefits of privacy-focused browser extensions. In *TheWebConf*.
[32] Tobias Braun, Irdin Pekaric, and Giovanni Apruzzese. 2024. Understanding the Process of Data Labeling in Cybersecurity. In *ACM SAC*.
[33] Leo Breiman. 2001. Random Forests. In *Machine Learning*.
[34] Duc Bui, Brian Tang, and Kang G. Shin. 2023. Detection of Inconsistencies in Privacy Practices of Browser Extensions. In *IEEE S&P*.
[35] Ahmet Salih Buyukkayhan, Kaan Onarlioglu, William Robertson, and Engin Kirda. 2016. CrossFire: An Analysis of Firefox Extension-Reuse Vulnerabilities. In *NDSS*.
[36] Haipeng Cai. 2020. Assessing and improving malware detection sustainability through app evolution studies. *ACM TOSEM* (2020).
[37] Stefano Calzavara, Michele Bugliesi, Silvia Crafa, and Enrico Steffinlongo. 2015. Fine-Grained Detection of Privilege Escalation Attacks on Browser Extensions. In *Programming Languages and Systems*.
[38] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An Evaluation of the Google Chrome Extension Security Architecture. In *USENIX Security*.
[39] Quan Chen and Alexandros Kapravelos. 2018. Mystique: Uncovering Information Leakage from Browser Extensions. In *ACM CCS*.
[40] Chrome-Stats. [n. d.]. *Chrome extensions Manifest V3 migration status.* https://chrome-stats.com/manifest-v3-migration Accessed on 2025-09-22.
[41] Chrome-Stats. Accessed on 2025-09-22. Chrome extension statistics. https://chrome-stats.com/stats.
[42] Jacopo Cortellazzi, Erwin Quiring, Daniel Arp, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2025. Intriguing Properties of Adversarial ML Attacks in the Problem Space [Extended Version]. *ACM Transactions on Privacy and Security* (2025).
[43] Qian Cui, Guy-Vincent Jourdan, Gregor V Bochmann, Russell Couturier, and Iosif-Viorel Onut. 2017. Tracking phishing attacks over time. In *TheWebConf*.
[44] 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 *USENIX Workshop CSET*.
[45] Tom Dietterich. 1995. Overfitting and undercomputing in machine learning. *ACM Computing Surveys (CSUR)* (1995).
[46] Aurore Fass, Michael Backes, and Ben Stock. 2019. JStap: a static pre-filter for malicious javascript detection. In *ASAC*.

[47] Aurore Fass, Robert P. Krawczyk, Michael Backes, and Ben Stock. 2018. JaSt: Fully Syntactic Detection of Malicious (Obfuscated) JavaScript. In *DIMVA*.
[48] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. 2021. DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale. In *ACM CCS*.
[49] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A Systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security* (2022).
[50] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM CSUR* (2014).
[51] Google. [n. d.]. Code Readability Requirements. https://developer.chrome.com/docs/webstore/program-policies/code-readability. Accessed on 2024-10-24.
[52] Google. Accessed on 2024-07-09. Repeat Abuse. https://developer.chrome.com/docs/webstore/program-policies/repeat-abuse.
[53] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data?. In *NeurIPS*.
[54] Ariya Hidayat. [n. d.]. ECMAScript Parsing Infrastructure for Multipurpose Analysis. http://esprima.org. Accessed on 2024-05-22.
[55] Sheryl Hsu, Manda Tran, and Aurore Fass. 2024. What is in the Chrome Web Store?. In *ACM AsiaCCS*.
[56] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. 2015. Trends and Lessons from Three Years Fighting Malicious Extensions. In *USENIX Security*.
[57] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *USENIX Security*.
[58] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *USENIX Security*.
[59] Soroush Karami, Panagiotis Ilia, and Jason Polakis. 2021. Awakening the web's sleeper agents: Misusing service workers for privacy leakage. In *NDSS*.
[60] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. 2020. Carnus: Exploring the Privacy Threats of Browser Extension Fingerprinting. In *NDSS*.
[61] Soroush Karami, Faezeh Kalantari, Mehrnoosh Zaeifi, Xavier J Maso, Erik Trickel, Panagiotis Ilia, Yan Shoshitaishvili, Adam Doupé, and Jason Polakis. 2022. Unleash the Simulacrum: Shifting Browser Realities for Robust Extension-Fingerprinting Prevention. In *USENIX Security*.
[62] Amin Kharraz, Zane Ma, Paul Murley, Charles Lever, Joshua Mason, Andrew Miller, Nikita Borisov, Manos Antonakakis, and Michael Bailey. 2019. Outguard: Detecting in-browser covert cryptocurrency mining in the wild. In *TheWebConf*.
[63] Young Min Kim and Byoungyoung Lee. 2023. Extending a Hand to Attackers: Browser Privilege Escalation Attacks via Extensions. In *USENIX Security*.
[64] Daniele Lain, Kari Kostiainen, and Srdjan Čapkun. 2022. Phishing in organizations: Findings from a large-scale and long-term study. In *IEEE S&P*.
[65] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. 2021. Fingerprinting in Style: Detecting Browser Extensions via Injected Style Sheets. In *USENIX Security*.
[66] Lei Liu, Xinwen Zhang, Guanhua Yan, and Songqing Chen. 2012. Chrome Extensions: Threat Analysis and Countermeasures. In *NDSS*.
[67] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. 2013. Understanding variable importances in forests of randomized trees. In *NeurIPS*.
[68] Marvin Moog, Markus Demmel, Michael Backes, and Aurore Fass. 2021. Statically Detecting JavaScript Obfuscation and Minification Techniques in the Wild. In *Dependable Systems and Networks (DSN)*.
[69] José Miguel Moreno, Narseo Vallina-Rodriguez, and Juan Tapiador. 2023. Chrowned by an Extension: Abusing the Chrome DevTools Protocol through the Debugger API. In *IEEE EuroS&P*.
[70] Asmit Nayak, Rishabh Khandelwal, Earlence Fernandes, and Kassem Fawaz. 2024. Experimental Security Analysis of Sensitive Data Access by Browser Extensions. In *TheWebConf*.
[71] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. 2020. PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *USENIX Security*.
[72] Eric Olsson, Benjamin Eriksson, Pablo Picazo-Sanchez, Lukas Andersson, and Andrei Sabelfeld. 2024. FakeX: A Framework for Detecting Fake Reviews of Browser Extensions. In *ACM AsiaCCS*.
[73] Daniel Olszewski, Allison Lu, Carson Stillman, Kevin Warren, Cole Kitroser, Alejandro Pascual, Divyajyoti Ukirde, Kevin Butler, and Patrick Traynor. 2023. "Get in Researchers; We're Measuring Reproducibility": A Reproducibility Study of Machine Learning Papers in Tier 1 Security Conferences. In *ACM CCS*.

[74] Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. 2020. You've Changed: Detecting Malicious Browser Extensions through their Update Deltas. In *ACM CCS*.
[75] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *USENIX Security*.
[76] Raffaello Perrotta and Feng Hao. 2018. Botnet in the browser: Understanding threats caused by malicious browser extensions. In *IEEE S&P*.
[77] Pablo Picazo-Sanchez, Benjamin Eriksson, and Andrei Sabelfeld. 2022. No Signal Left to Chance: Driving Browser Extension Analysis by Download Patterns. In *ACSAC*.
[78] Pablo Picazo-Sanchez, Juan Tapiador, and Gerardo Schneider. 2020. After you, please: browser extensions order attacks and countermeasures. *International Journal of Information Security* (2020).
[79] David M. W. Powers. 2011. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies* (2011).
[80] Iskander Sánchez-Rola, Igor Santos, and Davide Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *USENIX Security*.
[81] Alexander Sjösten, Steven Acker, and Andrei Sabelfeld. 2017. Discovering Browser Extensions via Web Accessible Resources. In *Conference on Data and Application Security and Privacy (CODASPY)*.
[82] Alexander Sjosten, Steven Van Acker, Pablo Picazo-Sanchez, and Andrei Sabelfeld. 2019. Latex Gloves: Protecting Browser Extensions from Probing and Revelation Attacks. In *NDSS*.
[83] Charles Smutz and Angelos Stavrou. 2012. Malicious PDF detection using metadata and structural features. In *Proceedings of the 28th annual computer security applications conference*. 239–248.
[84] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. 2022. The Dangers of Human Touch: Fingerprinting Browser Extensions through User Actions. In *USENIX Security*.
[85] Konstantinos Solomos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. 2022. Escaping the Confines of Time: Continuous Browser Extension Fingerprinting Through Ephemeral Modifications. In *ACM CCS*.
[86] Konstantinos Solomos, Nick Nikiforakis, and Jason Polakis. 2024. Harnessing Multiplicity: Granular Browser Extension Fingerprinting through User Configurations. In *ACSAC*.
[87] Dolière Francis Somé. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *IEEE S&P*.
[88] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. 2019. Unnecessarily Identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *TheWebConf*.
[89] Oleksii Starov and Nick Nikiforakis. 2017. Extended Tracking Powers: Measuring the Privacy Diffusion Enabled by Browser Extensions. In *TheWebConf*.
[90] Oleksii Starov and Nick Nikiforakis. 2017. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *IEEE S&P*.
[91] 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 *IEEE S&P*.
[92] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. 2019. Everyone is Different: Client-side Diversification for Defending Against Extension Fingerprinting. In *USENIX Security*.
[93] Thijs Van Ede, Hojjat Aghakhani, Noah Spahn, Riccardo Bortolameotti, Marco Cova, Andrea Continella, Maarten van Steen, Andreas Peter, Christopher Kruegel, and Giovanni Vigna. 2022. Deepcase: Semi-supervised contextual analysis of security events. In *IEEE S&P*.
[94] Giorgos Vasiliadis, Apostolos Karampelas, Alexandros Shevtsov, Panagiotis Papadopoulos, Sotiris Ioannidis, and Alexandros Kapravelos. 2023. WRIT: Web Request Integrity and Attestation against Malicious Browser Extensions. *IEEE Transactions on Dependable and Secure Computing* (2023).
[95] Jiangang Wang, Xiaohong Li, Xuhui Liu, Xinshu Dong, Junjie Wang, Zhenkai Liang, and Zhiyong Feng. 2012. An Empirical Study of Dangerous Behaviors in Firefox Extensions. In *International Conference on Information Security*.
[96] Yao Wang, Wandong Cai, Pin Lyu, and Wei Shao. 2018. A Combined Static and Dynamic Analysis Approach to Detect Malicious Browser Extensions. In *Security and Communication Networks*.
[97] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. 2017. Ex-Ray: Detection of History-Leaking Browser Extensions. In *ACSAC*.
[98] Qinge Xie, Manoj Vignesh K M, Paul Pearce, and Frank Li. 2024. Arcanum: Detecting and Evaluating the Privacy Risks of Browser Extensions on Web Pages and Web Content. In *USENIX Security*.
[99] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. 2015. Understanding Malvertising Through Ad-Injecting Browser Extensions. In *TheWebConf*.
[100] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and explaining concept drift samples for security applications. In *USENIX Security*.
[101] W. J. Youden. 1950. Index for Rating Diagnostic Tests. In *Cancer*.

[102] Jianjia Yu, Song Li, Junmin Zhu, and Yinzhi Cao. 2023. CoCo: Efficient Browser Extension Vulnerability Detection via Coverage-guided, Concurrent Abstract Interpretation. In *ACM CCS*.
[103] Yufei Zhao, Liqun Yang, Zhoujun Li, Longtao He, and Yipeng Zhang. 2021. Privacy model: detect privacy leakage for chinese browser extensions. *IEEE Access* (2021).
[104] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online {Anti-Malware} engines. In *USENIX Security*.

A Extra Figures and Tables

We provide additional figures and tables to support our claims.

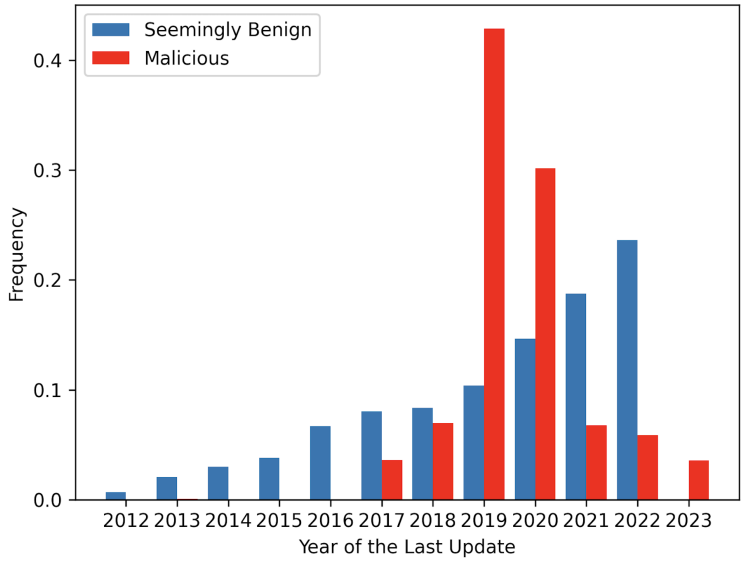


Fig. 9. Temporal distribution of the extensions in `Dataset L`. (N.b.: all extensions in `Dataset U` have had their last update in 2023)

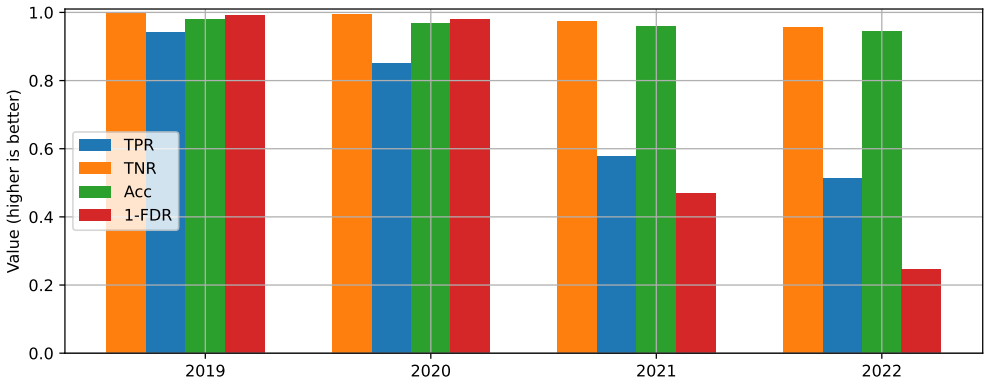


Fig. 10. Performance over time. We test the `Metadata classifier` on the extensions updated every year from 2019–2022 after training it on the extensions published or updated in the previous years.