Two Heads are Better Than One: Analysing Browser Extensions Across Stores

Abdullah Hassan Chaudhry
CISPA Helmholtz Center for Information Security
abdullah.chaudry@cispa.de

Valentino Dalla Valle
CISPA Helmholtz Center for Information Security
valentino.dalla-valle@cispa.de

Aurore Fass
Inria Centre at Université Côte d'Azur
aurore.fass@inria.fr

***Abstract*—Browser extension stores operate independently of each other and each have their own governance structure, creating a situation where threats identified on one platform can persist on others. We present the first cross-store analysis of security inconsistencies between the Chrome Web Store (CWS) and Edge Add-ons Store (EAS). We study extensions published on both stores, and discover 11 malicious extensions (affecting almost 134k users) that were present on the EAS, despite having already been removed from the CWS for containing malware. These extensions persisted on Edge for an average of 551 days (1.5 years) *after* their Chrome counterparts were removed for malware, with some even receiving updates during this period.**

We additionally find that malicious extensions change their names and developer names more often than other extensions and that these changes are larger. We also examine extensions that have been reinstated after having been removed (e.g., for containing malware), revealing inconsistencies in extension store governance. These findings show that malicious actors can exploit the lack of coordination in an interconnected extension ecosystem.

I. Introduction

The Web has become the dominant global technology, with the Web Browser serving as its primary access point. Browser extensions are software components that users can install to augment browser functionality with features such as password managers, shopping assistants or ad blockers. These functionalities are enabled through powerful browser APIs that are not available to ordinary websites [23], [41]. Such APIs allow extensions to read browsing history, intercept and modify network requests, manipulate web page content, store data persistently across sessions, and download files directly to users' devices. While these capabilities are essential for many legitimate use cases, they also create opportunities for abuse.

Previous research has documented numerous instances in which extensions engage in harmful activities [23], [30], [34], [36], [37], including stealing login credentials, session cookies or browsing history, injecting ads, and affiliate fraud. Beyond intentional malicious behaviour, extensions may contain vulnerabilities that enable exploitation by third parties [22], [31], [44], [46]. Collectively, vulnerable and malicious extensions have been referred by prior work as *security noteworthy*, and their prevalence has been studied extensively [34].

The number of extensions available in browser extension stores further exacerbates these security challenges. As of December 2025, the Chrome Web Store (CWS) hosts nearly 200,000 active extensions [6], with thousands being added to the store each month [34]. Security-noteworthy extensions have been observed to persist in extension stores for extended periods, sometimes months or even years before being removed [34], [41]. Moreover, extensions that were initially benign may later introduce malicious functionality through updates [25], [29], [41], highlighting the need for continuous scrutiny. These factors collectively demand automated detection systems, though the expertise of human analysts remains invaluable [36], [41].

Chrome is the most popular web browser [5], and the CWS is the largest extension marketplace. By comparison, as of December 2025, the Edge Add-on Store (EAS) has under 28,000 active extensions [7], while the Firefox Add-on Store contains fewer than 57,000 [8]. As a result, academic research on browser extension security has predominantly focused on the CWS, leaving other major extension stores comparatively under explored.

In this paper, we address this gap by examining extensions across the CWS and EAS, with a particular focus on how the two stores classify and govern the same extensions. We use the cross-store extension mapping process from ChromeStats [20] that is based on metadata (e.g., developer names and extension descriptions) to identify a dataset of equivalent extensions (*counterparts*) on the CWS and EAS. We study extension counterparts where the Chrome extension was removed from the CWS for malware. We then analyse both historical and current versions of these extensions and apply code similarity analysis to identify malicious extensions that persist on EAS.

Our analysis uncovers 20 malicious extension versions on the EAS that were never removed by the store administrators. These versions correspond to 13 unique extensions, 11 of which remained active at the end of our study period (March 2025). We demonstrate that extensions removed from one store can remain active and continue to receive updates on another store for years, despite containing identical malicious payloads. Finally, we examine a potential evasion technique that hinders cross-store mapping and discuss its applicability

Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2026
27 February 2026, San Diego, CA, USA
ISBN 978-1-970672-06-0
https://dx.doi.org/10.14722/madweb.2026.23027
www.ndss-symposium.org

as a signal for malware detection.

To summarise, this paper makes the following contributions:

- We present the first cross-store study of browser extensions aimed at uncovering malware. We show that mapping extensions across stores is an effective method for identifying malicious behaviour.
- We apply code similarity techniques to compare extension versions across stores. This led to the identification of 20 malicious extension versions, corresponding to 13 unique malicious extensions on the EAS. Of those, we reported the 7 extensions that were still malicious and available on the EAS on December 16, 2025; within a day, Edge confirmed the malicious behaviour of 5 extensions and promptly removed them from the EAS.
- We investigate a potential evasion strategy to limit cross-store extension mapping, and we show that this approach is already used for malicious extensions in the wild.
- We release our analysis code publicly[1] to support follow-up work and reproducibility.

II. Background

This section provides background on the architecture of browser extensions and the security risks they introduce. Then, we describe how extensions are published through extension marketplaces.

A. Browser Extensions and their Architecture

Browser extensions are installable software components that modify, extend, or personalize browser behaviour. They are packaged as `.crx` archives containing HTML, JavaScript, CSS, and a mandatory `manifest.json` file. The manifest defines an extension's metadata, declared capabilities, and the set of functional components used by the extension.

A typical extension architecture consists of three main components. The primary component is a *background script* or *service worker*, which implements the extension's main logic, listens for browser events, and performs privileged operations through the APIs declared in the manifest [28]. *Content scripts* are injected into matching web pages, enabling direct interaction with the DOM and page resources; these scripts operate in a sandbox with limited API access but can communicate with the background logic for privileged actions [27]. Extensions may also include optional *user interface components*, such as pop-ups, side panels, or settings pages, used for user interaction or configuration.

B. Malicious Browser Extensions

Browser extensions operate with elevated privileges relative to regular web content, and many request broad access to user data trough powerful browser extension APIs. As a result, they represent a significant attack surface.

A *malicious browser extension* is an extension that intentionally violates user expectations or platform policies by engaging in harmful behaviour. Such extensions may disguise themselves as benign utilities and may activate malicious behaviour only after a delay or remote trigger, complicating detection and enforcement.

C. Browser Extension Marketplaces

Browser extensions are primarily distributed through centralized marketplaces maintained by browser vendors, such as the Chrome Web Store (CWS), Firefox gallery, and Microsoft Edge Add-ons Store (EAS). These platforms serve as the main channel through which users discover, install, update, and review extensions, and are therefore positioned as the first line of defence against harmful software.

Different stores exist because each browser vendor maintains its own security policies, technical requirements, and developer ecosystems. Some Chromium-based browsers, such as Brave, choose not to operate their own marketplace and instead rely on the Chrome Web Store for extension distribution to reduce maintenance cost and benefit from existing developer and user bases. In contrast, browsers like Microsoft Edge maintain a proprietary store to enforce additional security requirements, branding, or platform-specific business strategies [17], [18].

To limit abuse, extension stores implement a vetting process that may include automated analysis, manual review, and policy compliance checks. Extensions may be rejected or removed for violating security requirements, engaging in deceptive or harmful behaviour, or breaching terms such as excessive data collection or prohibited monetization strategies [32], [33]. While the Chrome Web Store publicly exposes removal reasons that can help researchers categorize malicious behaviour, other marketplaces provide limited transparency, making cross-platform comparison challenging.

III. Dataset Trends

A. Extension Dataset

In line with prior works [24], [34], [43], we rely on ChromeStats [20] to assemble our extension dataset. ChromeStats maintains a historical archive for the Chrome Web Store (CWS) [10], the Edge Add-ons Store (EAS) [12], and the Firefox Add-ons Store [2], with information collected from these sources every 24 hours. This data includes all versions of extensions, extension metadata, and extension details. By *extension metadata*, we refer to information about an extension that is not directly provided by the developer, for example number of users, time active, reviews etc. By *extension details*, we refer to developer-provided information about an extension, for example extension names, author names, description etc.

Chrome and Edge are both Chromium-based browsers, and Edge supports almost all Chrome extension APIs, excluding those specific to ChromeOS [19]. Firefox, in contrast, has more incompatibilities with the Chrome extension APIs, including APIs with design differences (e.g., Proxy and Sidebar APIs), as well as variations in the content script lifecycle [9].

Because our goal is to compare extensions across stores with minimal confounding factors, we restrict our analysis to

[1]https://github.com/abdchau/two-heads-are-better

Chrome and Edge. We assemble our dataset from ChromeStats, downloading historical data of all active extensions in the CWS and EAS between June 2022 and March 2025.

B. Counterpart Pairs

Due to the high degree of API compatibility between Chrome and Edge, developers can publish virtually identical extensions to both stores. We define a *counterpart pair* as a pair of extensions (one on the CWS and one on the EAS) that have very similar extension details. We use *counterpart* to refer to an individual extension within this pair. An example of a counterpart pair is the Adobe Acrobat extension, available both on Chrome [3] and Edge [4]. We rely on the counterpart detection provided by ChromeStats [20], which uses a proprietary method to link extensions with identical or highly similar extension details. Under this method, counterparts are very likely to represent the same extension published by the same author across different stores. ChromeStats identifies 6,000 such counterpart pairs between the CWS and EAS.

C. Reasons for Removal

Browser extension stores are dynamic ecosystems, with extensions being added and removed constantly. In this subsection, we examine the documented reasons for extension removals and discuss the phenomenon of extensions reappearing after removal. Extensions may be removed either by their authors or by store administrators. For Chrome extensions removed by store administrators, ChromeStats can provide the following removal reasons:

- *Malware*: extensions that exhibit malicious behaviour, e.g. stealing browsing history;
- *Policy violation*: extensions that violate the CWS policies;
- *Potentially unwanted software*: extensions that subvert users' expectations with functionality that is not advertised.

These labels are sourced by ChromeStats directly from the CWS and have been used in prior research [24], [34], [43] as a ground truth for, e.g., assembling datasets of malware extensions.

Despite these categories of reasons for removals, the majority of extension removals lack an explicit reason: 72% of removed Chrome extensions have no published removal label (see Table I). In such cases, it is not possible to determine whether the extension was removed by the store or voluntarily taken down by its author. The situation is even more opaque for Edge and Firefox, as neither store publishes removal reasons. This lack of transparency limits the ability of researchers to analyse malicious or policy-violating extensions on these platforms in the same way as on the CWS.

D. Comeback Extensions

We define *comeback extensions* as extensions that reappear on the same store after having been removed. We observe this phenomenon on both the CWS and the EAS. However, because the EAS does not publish removal reasons, we are only able to analyse comeback behaviour on the CWS.

TABLE I
NUMBER OF EXTENSIONS REMOVED ON CHROME BY REMOVAL REASON.

Removal Reason	**Count**	**%**
No Reason	154,848	71.6
Potentially Unwanted Software	34,253	15.8
Policy Violation	20,816	9.6
Malware	6,382	3.0
Total	216,299	100

On the CWS, comebacks occur across all removal categories, including malware. In total, we identify 104 comeback extensions that were at some point removed for malware. This finding contrasts with earlier reports suggesting that malware removals are permanent [36], and indicates that Chrome's enforcement or reinstatement policies have evolved over time.

A particularly concerning pattern is that removal reasons may change over time without a corresponding version change. For example, the extension `kbehlacdlbldkoajppkpkgnjfbjjkkhb` was removed and reinstated multiple times between November 2022 and July 2024 without having undergone any version updates. After the last reinstatement, which coincided with a version update, it was removed again in July 2024 without a reason being published. The removal reason was finally changed to malware in March 2025, without the extension being reinstated or updated.

Figure 1 displays this timeline. Among the 104 comeback extensions that have had malware recorded as a reason for removal, 21 saw their removal reason change, even though they were removed only once. This suggests potential inconsistencies in how removal reasons are assigned or updated.

E. Patterns in Comeback Timing

The CWS exhibits a distinctive temporal pattern in which large batches of extensions are removed on a single day, followed shortly thereafter by a comparable number of reinstatements. We observe hundreds of such near-immediate comebacks occurring within days of mass removals (see Figure 2). The phenomenon could be related to developers quickly updating their extensions to comply with new policies, or to policy rollbacks. However, the scale and regularity of these events suggest systematic changes in removal criteria, potentially driven by automated detection mechanisms or policy updates.

In contrast, we do not observe a comparable pattern on the EAS (Figure 3). While Edge also experiences large numbers of single-day removals, suggesting automated enforcement, these are not followed by similarly large-scale reinstatements. This divergence indicates fundamental differences in how the two stores handle extension reinstatement: automated or semi-automated comeback mechanisms may be used by the CWS and not on the EAS.

IV. Security Inconsistencies in Extension Counterpart Pairs

In this section, we study Edge counterparts of the Chrome extensions that were removed for malware. Because counter-

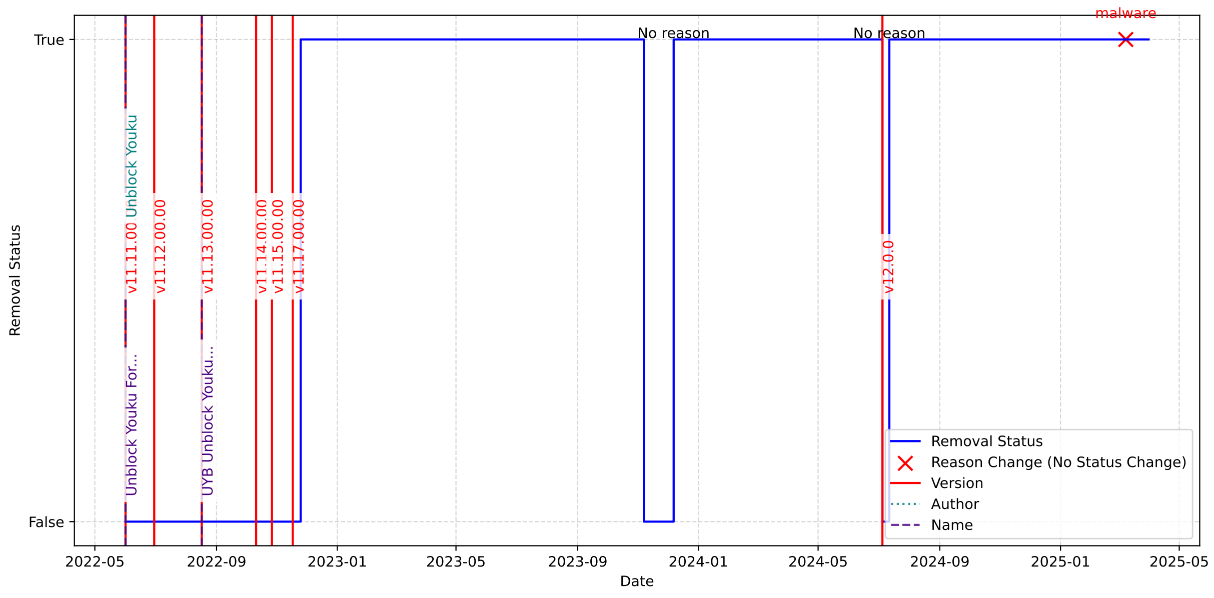


Fig. 1. Visual representation of removals, comebacks, and removal reasons for Chrome extension `kbehlacdlbldkoajppkpkgnjfbjjkkhb` over time.

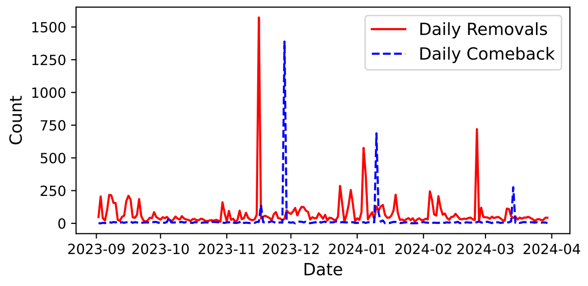


Fig. 2. Daily removals and comebacks for Chrome. Hundreds of extensions removed on the same day can make simultaneous comebacks only a short while later.

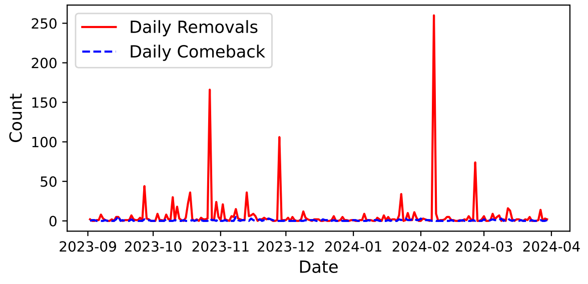


Fig. 3. Daily removals and comebacks for Edge, showing the absence of the comeback pattern observed in the same time frame in Chrome.

parts are published by the same author and are intended to provide equivalent functionality across stores, we hypothesise that the conditional probability of an extension being malicious given that its counterpart was found to be malware is substantially higher than the baseline probability of any extension being malicious. Consequently, when a platform maintainer identifies a malicious payload in one store, the corresponding counterpart on other platforms should receive additional scrutiny to ensure that the same malware payload is absent and never introduced in subsequent updates.

Under this assumption, if an Edge counterpart is also removed after its Chrome counterpart was flagged as malware, it was likely removed for similar reasons. Conversely, if the Edge counterpart remains available, this may indicate a detection gap or delayed enforcement by the EAS.

In this work, we analyse counterpart pairs in which the Chrome extension was at some point removed for malware. This includes extensions that were removed in the past and potentially later reinstated (cf. Section III-D).

A. Defining "Interesting" Version Pairs

We begin by enumerating all Chrome–Edge counterpart pairs for which at least one Chrome version was labelled as malware by the CWS. This set includes extensions that were removed for malware and later reinstated.

Importantly, identical version identifiers across stores do not guarantee identical functionality: version 1.1 of a Chrome extension may differ from version 1.1 of its Edge counterpart. We therefore operate at the level of extension versions rather than extension identifiers.

For each counterpart pair, we construct a set of *interesting pairs* $\{e_i, c_j\}$, where e_i represents an Edge version and c_j represents a Chrome version from the same counterpart pair, and i, j are version identifiers. We initially enumerate the Cartesian product of all Edge and Chrome versions for that counterpart, and retain a version pair as *interesting* if at least one of the following conditions holds:

TABLE II
HOW MANY VERSION PAIRS AND UNIQUE EXTENSION PAIRS ARE REMOVED ON THE RESPECTIVE STORES?

Pairs	Total	Removed on Edge	Malware on Chrome	Malware CWS + removed EAS
Interesting	586	333	221	44
Counterpart	79	43	79	43

1) The Chrome version c_j was removed for being malware (but since then, it may have come back);
2) The two versions share the same version identifier $i = j$;
3) The Edge version e_i was removed (but since then, it may have come back).

As illustrated in Figure 4, a single counterpart pair may therefore give rise to multiple interesting version pairs. Version pairs that do not satisfy any of these criteria are excluded from further analysis. Given these criteria, we find 586 interesting version pairs. Table II provides a summary of store removals for these pairs.

B. Comparing Counterpart Code

To assess code similarity between counterparts, we first extract all JavaScript files (`.js`) from each extension version and concatenate them into a single file per version. Before concatenation, we use Retire.js [14] to identify and exclude third-party libraries, preventing shared dependencies from artificially inflating similarity scores. We then compute context triggered piecewise (fuzzy) hashes [38] using the Python `ssdeep` library [16]. This technique was introduced by Hsu et al. to study security-noteworthy extensions in the CWS [34]. Using this approach, we obtain non-zero similarity scores for only 196 of the 586 interesting pairs.

Manual inspection reveals that this low coverage is largely due to separate minification processes applied independently to Chrome and Edge versions. Even when two extensions are semantically identical, differences in identifier names, formatting, or function ordering can substantially reduce string-based similarity scores. To address this limitation, we adopt more structured comparison techniques based on Abstract Syntax Trees (ASTs).

C. Obtaining Abstract Syntax Trees

For each extension version in the interesting pairs, we generate an Abstract Syntax Tree (AST) using Acorn [1]. Prior to parsing, all JavaScript files are formatted with Prettier [13] to normalise token layout. We then parse each non-library JavaScript file and combine the resulting ASTs into a single tree per extension version.

Separate minification processes introduce noise into ASTs in the form of renamed identifiers and differing source locations. This can be seen in the example of Listing 1 where syntactic noise is introduced by independent minification.

To mitigate this effect, we produce two representations for each AST:

1) a *Simple AST*, retaining all identifiers and location information, and

```javascript
// afbegdjlledbjnldkkealalienoigmcp (Edge, v1.0.3)
// popup.js
t = {};
function o(r) {
  var c = t[r];
  if (void 0 !== c) return c.exports;
  var n = (t[r] = { exports: {} });
  return e[r](n, n.exports, o), n.exports;
}

// epeigjgefhajkiiallmfblgglmdbhfab (Chrome, v1.0.7)
// popup.js
var o = {};
function n(t) {
  var r = o[t];
  if (void 0 !== r) return r.exports;
  var c = (o[t] = { exports: {} });
  return e[t](c, c.exports, n), c.exports;
}
```

Listing 1: Two minified versions of the same code results in different identifiers.

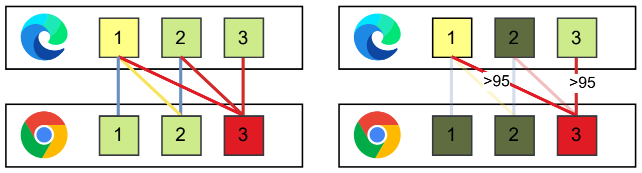


Fig. 4. Interesting pairs (left) and suspicious pairs (right) for a counterpart extension pair. Each edge between versions indicates a pair. We use blue to indicate the same version identifier, yellow to indicate a version that was removed, and red to indicate a version that was removed for malware.

2) a *Stripped AST*, from which identifier names and location metadata are removed.

Both representations are used in subsequent analyses.

D. Computing API Deltas

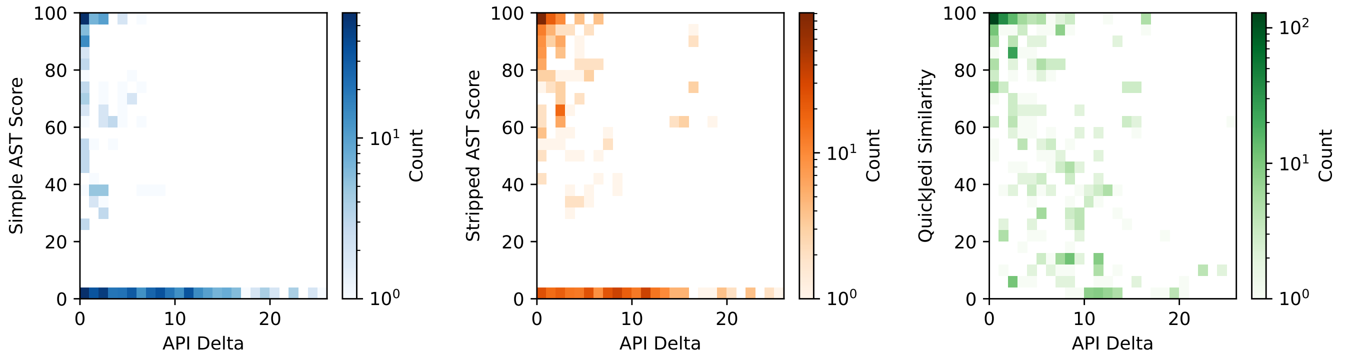
As a coarse-grained similarity measure, we compute the *API delta* between versions in an interesting pair. We augment the set of APIs identified in prior work by Pantelaios et al. [41] with 51 additional browser runtime APIs that we empirically observed to be frequently abused by malware extensions. For the complete list of APIs used, see Tables XI and XII in the Appendix.

Using the Simple ASTs, we detect the presence or absence of calls to each malicious API in a given version. Multiple occurrences of the same API are counted only once. The API delta between two versions is defined as the number of malicious APIs that appear in exactly one of the two versions. For example, if one version uses A, B and the other uses A, C, the API delta is 2.

E. Computing AST Similarity

We complement API deltas with finer grained structural similarity measures based directly on ASTs. We expect similarity scores to be inversely correlated with API deltas: as code diverges, the number of APIs present in only one version should increase.

1) Using ssdeep to Compare ASTs: We first apply ssdeep to ASTs by serialising them as raw strings. For each interesting pair, we compute similarity scores for both Simple ASTs and Stripped ASTs, yielding two metrics: the *Simple AST*



(a) Simple AST Score over API Delta (b) Stripped AST Score over API Delta (c) QuickJedi Similarity over API Delta

Fig. 5. Comparison of similarity scores against API deltas for different AST comparison methods. Note that the count scale is logarithmic.

Score and the *Stripped AST Score*. This approach increases coverage to 294 non-zero similarity scores. However, as shown in Figures 5a and 5b, most scores remain clustered near zero, indicating limited discriminative power. While Stripped ASTs improve over Simple ASTs, ssdeep remains sensitive to syntactic rearrangements, such as function reordering, that preserve semantics but significantly alter string representations.

2) Using Tree Edit Distance: To overcome these limitations, we compute the Tree Edit Distance (TED), defined as the minimum number of operations required to transform one tree into another [47]. Due to computational constraints and sensitivity to minification artifacts, we compute TED only over Stripped ASTs. Because Acorn produces ASTs in JSON format, we employ the QuickJedi algorithm by Hütter et al. [35] which computes edit distance between JSON objects. QuickJedi runs in $O(n^2 d \log d)$ time and $O(n^2)$ space for JSON trees of size n and maximum degree d.

Using QuickJedi, we successfully compute exact distances for 190 of the 586 interesting pairs over approximately ten days. Including the ssdeep scores and QuickJedi distance, we now have non-zero similarity scores or distance for 397 out of 586 pairs. For the remaining pairs, we compute a conservative **lower bound** on the QuickJedi **distance**—namely, the absolute difference in AST sizes—which corresponds to an **upper bound** on **similarity**.

We convert distances into a Jaccard-like similarity score:

$$\text{similarity} = \left\lfloor \frac{\text{size}_1 + \text{size}_2 - \text{distance}}{\text{size}_1 + \text{size}_2 + \text{distance}} \times 100 \right\rfloor$$

where size_1 and size_2 denote the number of nodes in the two ASTs and distance is the QuickJedi distance between them. Where the distance could not be computed, we use the absolute difference between the sizes of the two ASTs. Figure 5c shows the inverse relationship between QuickJedi similarity and API delta. We note that the QuickJedi and AST similarity scores are not directly comparable, as they rely on fundamentally different principles.

F. Looking for Malware

We now use these similarity measures to identify potentially malicious Edge extensions. From the 586 interesting pairs, we extract *suspicious pairs* $\{e_i, c_j\}$ where the Chrome version was removed for malware, the Edge version was never removed, and **any** of the following conditions holds:

1) The Simple AST Score is greater than 95;
2) The Stripped AST Score is greater than 95;
3) The QuickJedi similarity is greater than 95;
4) If the QuickJedi similarity could not be computed, the upper bound for it is greater than 95.

We empirically identified the threshold of 95 as a good trade-off between the number of extensions to manually analyse and the risk of missing true malware Edge extensions (false negatives). As with interesting pairs, a counterpart pair can give rise to multiple suspicious pairs (see Figure 4). Figure 6 summarises our detection pipeline.

This procedure yields 48 **suspicious** version **pairs**, corresponding to 28 unique **counterpart pairs**. Ten of these counterpart pairs contain *multiple* Edge versions that are highly similar to malware-removed Chrome versions. Five of the 28 counterpart pairs saw the Edge counterpart removed from the EAS during the study period, although removal reasons are not disclosed by the EAS. We manually analysed all 48 suspicious pairs for malicious behaviour in the Edge versions. The analysis included inspecting the code, installing the extensions, and attempting to trigger malicious behaviour locally. We confirmed malicious behaviour in 20 **suspicious** pairs, representing 13 unique Edge extensions. Of these, 11 extensions remained active on the EAS at the end of the study period (March 2025). 8 of these remained undetected by the EAS until our report. Table III lists a summary of the counterpart and corresponding suspicious pairs.

For 7 counterpart pairs (corresponding to 15 suspicious pairs), we could not conclusively identify malicious behaviour in the Chrome versions, despite their removal for malware by the CWS. These extensions exhibited invasive but ambiguous

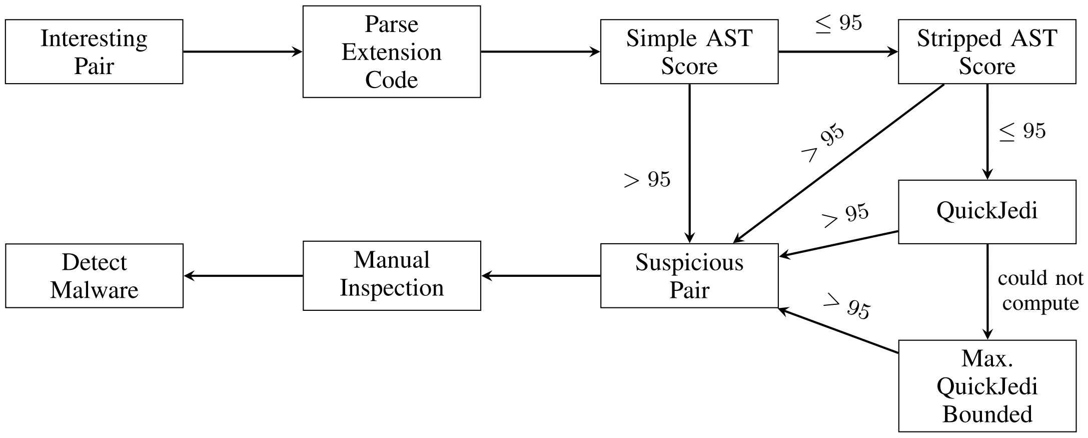


Fig. 6. Pipeline for detecting malware in Edge extension counterparts. Extension code is parsed to produce Simple ASTs, which are then stripped of identifiers to create Stripped ASTs. Similarity is computed using ssdeep on both AST types and QuickJedi on stripped ASTs (with bounded approximation when needed). Results from all similarity measures feed into manual inspection, which leads to malware detection.

TABLE III
DISAMBIGUATING COUNTERPART AND SUSPICIOUS PAIRS THAT WE FOUND TO CONTAIN MALWARE

	Counterpart Pairs (#Extensions)	Suspicious Pairs (#Versions)
Total Malware on Edge	13	20
Active at End of the Study Period	11	16
Reported to Edge	7	13

TABLE IV
TIME IN DAYS (MIN, MAX, AND AVERAGE) SPENT ACTIVE ON THE EAS DURING STUDY PERIOD BY SUSPICIOUS EDGE VERSIONS (COUNT). MALWARE ON EDGE WAS CONFIRMED MANUALLY.

Edge Malware	Latest Version	Count	Min	Max	Avg
Yes	Yes	9	136	1034	466.3
Yes	No	11	8	279	92.5
No	Yes	9	124	1034	371.7
No	No	19	5	691	124.3

behaviours, such as browser usage tracking.[2] We therefore conservatively assume that either the Chrome versions contained malware we could not detect or behaviour considered malicious under CWS policies.

According to the EAS, the 13 unique confirmed malicious extensions we find account for a total of 150,500 users. On average they remained active on the EAS for 480 days (1.32 years) *after* the Chrome counterpart was removed for malicious behaviour. In particular, for the 8 extensions that were not detected by the EAS administration, an average of 638 days had passed since the Chrome counterpart was removed. These extensions alone accounted for 96,577 users at the time of writing (December 2025). Table IV presents statistics for the time spent active on the EAS for the Edge *versions* in suspicious pairs.

From the 8 undetected extensions, one was updated after the end of our study period and the malware logic was removed. We reported the remaining 7 extensions to the EAS administrators on December 16, 2025 (see Table XIII in the Appendix). Just one day after our report, Edge had already validated the malicious behaviours of 5 extensions we reported (affecting 56,055 users) and removed them from the EAS.

The identified malware extensions exhibit a variety of malicious behaviours, the most common being the exfiltration of browsing data, observed in 8 unique extensions. For example, 3 undetected extensions use Google Analytics [11] as a remote endpoint for user browsing history exfiltration, disguising their malicious activity behind a legitimate vendor.

Case study: The extension `fkfilbkmmgekgaj-hflamglahakhalhch` is advertised as a "free screen recording and annotation extension designed for privacy-conscious users". Its CWS counterpart was removed on August 29, 2024 with the reason *malware*. Despite this, the extension remained available on the EAS and was updated twice shortly after the Chrome removal. Manual inspection of the EAS version revealed a clear malware payload. The extension invokes the `chrome.cookies.getAll` API to collect all browser cookies and selectively filters those associated with the Facebook origin. The extracted cookies are then Base64-encoded and exfiltrated to an external server. This payload executes once every two days, enabling repeated exfiltration whenever cookie values change. The same malicious logic was present in the Chrome counterpart at the time of its removal.

Subsequent updates on the EAS introduced only minor modifications to the malware code, without altering its underlying behaviour, alongside unrelated changes to the extension's main functionality. Overall, this malicious extension remained active on the EAS for more than a year after its Chrome counterpart had been removed. It was ultimately detected by our analysis pipeline and reported to the EAS administrators, who then promptly removed it from the store.

[2]For example, the counterpart pair {chrome: `diapmighkmmnpmdkfnmlbpkjkealjojg`, edge: `okfohlpcgpbeeimalodgaafjfheidfbg`}

TABLE V
DISTRIBUTION OF EDGE-CHROME COUNTERPART PAIRS BY REMOVAL REASON ON CHROME.

Chrome Reason	Edge Removed	Edge Not Removed	Total
Malware	26	37	63
Policy Violation	29	72	101
Potentially UWS	5	28	33

V. NAME AND AUTHOR CHANGES AS EVASION STRATEGIES

In the previous section, we used ChromeStat's counterpart connections based on *extension detail* similarity to find malicious extensions on another store. An evasion strategy would be to make counterpart details different enough to not be connected. If successful, this would prevent a newly uploaded extension from being connected to an established malware instance on another store, thereby avoiding additional scrutiny. This motivates a closer investigation of how much extension details differ across counterparts.

More broadly, extensions in browser stores are identified by descriptive details such as their names and the author or publisher fields. However, this identification can be subverted: malicious developers may intentionally alter extension names or author information to evade not only cross-store counterpart mapping but also within-store recognition. Consequently, we also investigate the changes to extension details for extensions within stores. We analyse historical extension details from both the CWS and the EAS to quantify the extent to which benign and malicious extensions change their names and author identifiers.

A. Changes Among Counterpart Pairs

We examine counterparts between the EAS and the CWS, focusing on extensions that were removed from the CWS for various reasons. Table V summarizes the distribution of Edge-Chrome counterparts across different removal categories. The categories shown correspond to the labels which the CWS assigns to removed extensions. Extensions removed without any reason for removal being assigned are not reflected in the table. Also excluded are extensions that were removed for a particular reason, but have since made a comeback.

Of particular interest are the 63 Edge extensions whose Chrome counterparts have been removed for being malware. Of these, 26 are also removed on the EAS, whereas 37 are still active. This discrepancy reinforces the notion of potential gaps in detection and removal policies across the stores.

To measure the extent of detail changes between counterparts, we compute Levenshtein distances between extension names and author names for Edge-Chrome counterparts. The Levenshtein distance [39] measures the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. We use this as a metric for similarity between names.

We analyse all 6,000 Edge–Chrome counterpart pairs provided by ChromeStats. Within this set, 450 pairs include an Edge extension that was removed, and 881 pairs include a Chrome extension that was removed; 258 pairs were removed from both stores. Table VI reports summary statistics for author-name distances across these subsets.

TABLE VI
AUTHOR NAME LEVENSHTEIN DISTANCE STATISTICS FOR EDGE-CHROME COUNTERPARTS.

	Max	Min	Mean	Med	SD	N
All	104	0	13.03	14	9.04	6,000
Removed on Edge	42	0	13.58	14	8.34	450
Removed on Chrome (all)	104	0	12.62	13	8.90	881
Removed on Both	42	0	13.40	14	8.59	258
Malware on Chrome	104	0	16.76	14	13.21	63
Malware, Edge Removed	32	1	17.04	17	7.35	26
Malware, Edge Active	104	0	16.57	13	16.22	37

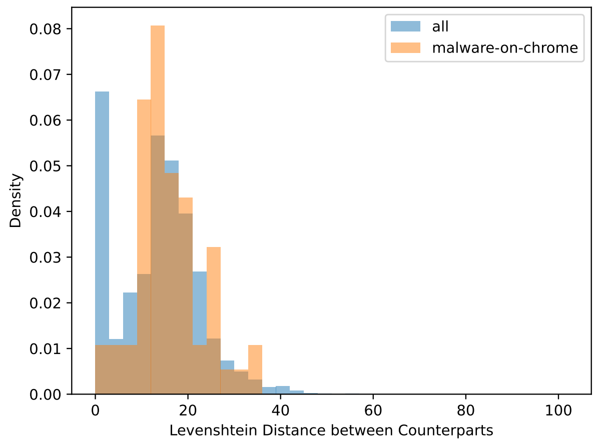


Fig. 7. Distribution of author name Levenshtein distances Edge-Chrome counterpart pairs, for all and malware counterparts on Chrome. The malware distribution contains significantly fewer counterpart pairs with low distance.

Chrome-malware counterparts exhibit larger author-name divergence than the overall population (mean 16.76 vs. 13.03). This means that malicious extension developers are more likely to use different author names across stores for the same extension. The subset of Chrome malware extensions that remain active on Edge shows particularly high variance (standard deviation of 16.22) indicating that some malware-linked counterparts adopt substantially different author names across stores. Figure 7 shows a visual comparison of the distributions for malware extensions and the overall population for counterparts.

Table VII presents corresponding statistics for extension name distances. Across all counterpart pairs, extension

TABLE VII
EXTENSION NAME LEVENSHTEIN DISTANCE STATISTICS FOR EDGE-CHROME COUNTERPARTS.

	Max	Min	Mean	Med	SD	N
All	63	0	1.29	0	5.18	6,000
Removed on Edge	46	0	2.26	0	7.13	450
Removed on Chrome	47	0	1.20	0	4.83	881
Removed on Both	32	0	1.25	0	4.91	258
Malware on Chrome	32	0	2.40	0	6.88	63
Malware, Edge Removed	27	0	1.96	0	6.81	26
Malware, Edge Active	32	0	2.70	0	7.01	37

TABLE VIII
DISTRIBUTION OF DETAIL CHANGES IN CHROME EXTENSIONS BY CATEGORY. PERCENTAGES ARE RELATIVE TO THE TOTAL IN EACH CATEGORY. ONLY EXTENSIONS THAT WERE AT SOME POINT ACTIVE DURING THE STUDY PERIOD ARE INCLUDED.

Category	**Author Changed**	**Name Changed**	**Version Changed**	**Name/Author Never Changed**	**Total**
All Extensions	23,033 (11.0%)	19,051 (9.1%)	75,108 (35.9%)	172,451 (82.4%)	209,167
Malware	235 (20.0%)	180 (15.3%)	641 (54.6%)	825 (70.2%)	1,175
Comeback	2,877 (17.0%)	3,296 (19.4%)	9,600 (56.6%)	11,873 (70.0%)	16,965
Benign	22,798 (11.0%)	18,871 (9.1%)	74,467 (35.8%)	171,626 (82.5%)	207,992
Malware Comeback	21 (32.3%)	18 (27.7%)	59 (90.8%)	34 (52.3%)	65

names are largely consistent (median distance 0; mean 1.29), indicating that most counterparts share identical names. Nevertheless, malware-related counterparts show greater divergence: the mean name distance increases to 2.4 for Chrome-malware counterparts, and is highest among the subset that remains active on Edge (mean 2.7; SD 7.01).

Overall, benign extensions tend to maintain consistent names and author identifiers across stores, whereas malware-associated extensions exhibit greater variability in these fields. The elevated variance for Chrome-malware extensions that remain active on Edge is consistent with deliberate attempts to avoid cross-store linkage. This further suggests that our analysis of counterparts in Section IV likely underestimates the true number of malicious cross-store counterparts, as some may evade matching by sufficiently diverging their details.

B. Changes Within Chrome Extensions

We now examine detail changes within Chrome extensions over time, focusing on extensions that were active at some point during our study period. Extensions that were removed before this period and never reappeared are excluded.

Table VIII presents the distribution of extensions that had changes in their author names, extension names, or received version updates across different categories. Malware extensions are more likely to have detail changes than benign extensions: 20.0% of malware extensions changed their author names (as opposed to 11.0% overall) and 15.3% changed their extension names (as opposed to 9.1% overall). Comeback extensions also have higher changes than overall: 17.0% changed author names and 19.4% changed extension names. Malware comeback extensions show the highest rates of change across all categories: 32.3% change author names, 27.7% change extension names, and 90.8% receive version updates.

To characterise the *magnitude* of these changes, we compute (i) Levenshtein distances and (ii) cosine similarities between consecutive author and extension names. These statistics are computed only for extensions that exhibit at least one change; extensions with no changes are excluded from the analysis.

1) Levenshtein Distances: Table IX reports statistics for author name distances across different extension categories. Malware extensions that change their author names exhibit a higher mean distance (16.55) and median distance (15) compared to all extensions (mean 13.07, median 12), meaning that when malicious extensions change author names, they tend to make larger changes. Comeback extensions show similar mean distances (13.23) to the overall mean.

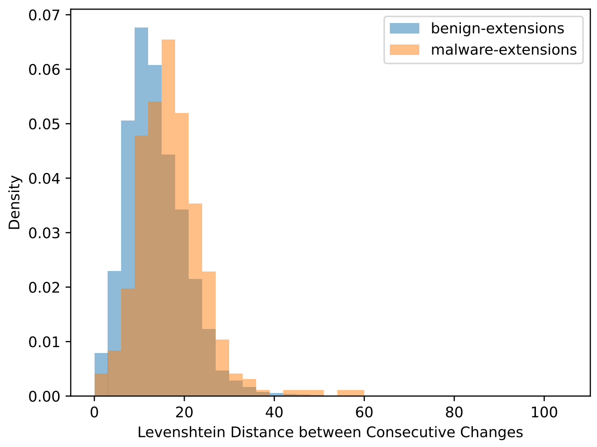


Fig. 8. Distribution of author name Levenshtein distances between consecutive versions for benign and malware Chrome extensions. The malware distribution is clearly shifted to the right, indicating that malware extensions tend to make larger changes than benign extensions.

TABLE IX
AUTHOR NAME LEVENSHTEIN DISTANCE STATISTICS FOR CHROME EXTENSIONS WHEN AUTHOR NAMES CHANGE.

	Max	Min	Mean	Med	SD	N
All Extensions	108	1	13.07	12	6.69	28,139
Malware	108	1	16.55	15	8.95	321
Comeback	100	1	13.23	12	7.35	3,682
Benign	100	1	13.03	12	6.65	27,818
Malware Comeback	46	1	15.55	13	8.49	33

In contrast, when extensions change their *extension names*, malware extensions exhibit slightly smaller character-level changes on average than the overall population (mean 13.67 vs. 15.36; same median of 13). This suggests that malware name changes may more often involve modest edits (e.g., small rebrandings) rather than complete renaming, while author changes tend to be more substantial.

To summarize, malware extensions are more likely to change their details over time, and the changes in author names are more significant than the changes in extension names. Figure 8 shows a visual comparison of the distributions for malware and benign extensions on the CWS.

2) Similarity Among Embeddings: To complement the character-level Levenshtein distance analysis, we also examine semantic similarity using sentence embeddings. A sentence embedding is a representation of a short string in vector form, such that semantically similar sentences result in sim-

TABLE X
COSINE SIMILARITY STATISTICS FOR AUTHOR NAME EMBEDDINGS WHEN AUTHOR NAMES CHANGE IN CHROME EXTENSIONS.

	Max	Min	Mean	Med	SD	N
All Extensions	1.0	-0.11	0.35	0.26	0.25	24,337
Malware	1.0	-0.06	0.28	0.20	0.22	255
Comeback	1.0	-0.11	0.37	0.27	0.26	3,205
Malware Comeback	0.93	0.03	0.36	0.29	0.24	22

ilar vectors. We use the SentenceTransformers [15] library's `all-MiniLM-L6-v2` model to produce embeddings.

We compute sentence embeddings for extension names and author names. To reduce superficial variation, we normalise names by lowercasing, stripping surrounding white space, and replacing hyphens with spaces. We then compute cosine similarities between consecutive embeddings, providing a measure of semantic similarity that captures meaning beyond character-level differences.

Table X presents cosine similarity statistics for author name changes across different extension categories. Malware extensions show a lower mean similarity (0.28) compared to all extensions (0.35), meaning that when malicious extensions change author names, the new names are semantically more distinct. Comeback extensions show a slightly higher mean similarity (0.37) than the overall population.

Extension names show much higher semantic similarity overall (mean 0.69) compared to author names, with a median of 0.75, indicating that most extension name changes maintain semantic similarity. Malware extensions show a similar mean similarity (0.69) with a median of 0.78. The embedding-based analysis matches the character-level analysis, confirming that malware extensions are more likely to change their details over time. This is consistent with our findings in Section V-A.

VI. LIMITATIONS

Throughout this paper, we rely on ChromeStats' counterpart detection, which is based on proprietary heuristics. This approach may be susceptible to targeted detail changes by attackers, leading to false negatives. This means our numbers are a lower bound. While false positives are also possible in principle, we did not observe any in our experiments. In addition, we use ChromeStats for "first/last seen" dates, removal dates, and (where available) removal reasons. Since ChromeStats collects data daily from the CWS and EAS, events occurring within a 24-hour interval may be missed, and we may only observe the final state in that window.

VII. RELATED WORK

Prior research has examined the problem of malware within the extension ecosystem and has proposed approaches for detecting suspicious browser extensions. Hsu et al. [34] leveraged fuzzy hashes (with ssdeep) to identify duplicate or near-duplicate extensions in the CWS, revealing extensive abuse of copied code for malicious purposes. Other studies have explored syntactic similarity analysis, such as AST-based n-gram models, to identify malware extensions and track malicious code included across extension updates [41], [43]. These techniques demonstrate that code similarity analysis can reveal coordinated malware campaigns and repeated abuse patterns.

Additionally, extension metadata has been used by Picazo-Sanchez et al. [42] to detect malicious extensions. They combined an analysis of download patterns from the CWS with code similarity techniques to unearth malicious extensions. Dynamic analysis methods have also been employed in the hunt for malicious browser extensions. Chen and Kapravelos [26] proposed modifications to Chromium's V8 engine to support taint tracking. Other approaches include examining the network traffic of an extension [37], [45], instrumenting an extension's APIs [23], and forcing the execution of conditionally injected code [40]. Previous work has also considered vulnerable extensions. Fass et al. [31] and Somé [44] presented static analysis techniques to detect vulnerabilities in extensions, while Yu et al. [46] leveraged abstract interpretation. While the works above focused on the CWS, in 2010, Bandhakavi et al. conducted the first study of vulnerable extensions in Firefox [21]. Compared with the works above, all focusing on a single extensions store, we present the first cross-store analysis to understand and detect malware extensions across stores.

VIII. CONCLUSION

Browser extension stores are independent entities with separate governance structures. Lack of cross-store cooperation results in exploitable gaps: threats detected and removed on one platform can persist on another. Our analysis identified 11 confirmed malicious Edge extensions that remained active for years on average after their Chrome counterparts was removed, illustrating that independent enforcement results in persistence of known threats across different stores.

We further investigate the prevalence of extension detail changes as an evasion strategy. We find that malicious extensions change identifying details more frequently than benign extensions, both across stores and within a store over time. This divergence can undermine counterpart mapping and complicate longitudinal tracking, and it also provides a measurable behavioural signal that may be useful for malware detection and triage.

Finally, the comeback phenomenon underscores the complexity and inconsistency of current enforcement practices. The reappearance of extensions previously removed for malware, as well as the large-scale, tightly timed removal–comeback cycles observed on the CWS, suggest that removals, comebacks, and removal reasons may be influenced by shifting criteria, automation, and appeal or review processes. Taken together, these findings highlight that independent store governance can fragment security outcomes.

Overall, our results suggest that without systematic cross-store communication and coordinated responses, each store remains susceptible to threats that have already been identified elsewhere. Enabling reliable sharing of malware signals (e.g. counterpart links and enforcement actions) across extension ecosystems would reduce the opportunity for malicious developers to exploit store isolation and would improve user protection across platforms.

ACKNOWLEDGMENT

We would like to thank ChromeStats [20], specifically Hao Nguyen, for providing us with extensions, their metadata, and additional information or support upon request. We also thank the Edge team for their reactivity and for having promptly removed from the EAS the malicious extensions we uncovered and reported. Finally, we thank the anonymous MADWeb reviewers for their constructive reviews and helpful feedback.

REFERENCES

[1] acorn - npm. https://www.npmjs.com/package/acorn. Accessed: 2025-11-10.

[2] Add-ons for firefox. https://addons.mozilla.org/en-US/firefox/. Accessed: 2025-12-06.

[3] Adobe acrobat: Pdf edit, convert, sign tools - chrome web store. https://chromewebstore.google.com/detail/adobe-acrobat-pdf-edit-co/efaidnbmnnnibpcajpcglclefindmkaj. Accessed: 2025-12-11.

[4] Adobe acrobat: Pdf edit, convert, sign tools - microsoft edge addons. https://microsoftedge.microsoft.com/addons/detail/adobe-acrobat-pdf-edit-/elhekieabhbkpmcefcoobjddigjcaadp. Accessed: 2025-12-11.

[5] Browser market share worldwide. https://gs.statcounter.com/browser-market-share/. Accessed: 2025-12-10.

[6] Chrome extension statistics. https://chrome-stats.com/chrome/stats. Accessed: 2025-12-10.

[7] Chrome extension statistics. https://chrome-stats.com/edge/stats. Accessed: 2025-12-10.

[8] Chrome extension statistics. https://chrome-stats.com/firefox/stats. Accessed: 2025-12-10.

[9] Chrome incompatibilities. https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Chrome_incompatibilities. Accessed: 2025-12-06.

[10] Chrome web store. https://chromewebstore.google.com/. Accessed: 2025-12-06.

[11] Google analytics. https://marketingplatform.google.com/about/analytics/. Accessed: 2025-12-17.

[12] Microsoft edge add-ons. https://microsoftedge.microsoft.com/addons/Microsoft-Edge-Extensions-Home. Accessed: 2025-12-06.

[13] Prettier · opinionated code formatter · prettier. https://prettier.io/. Accessed: 2025-11-10.

[14] Retire.js. https://retirejs.github.io/retire.js/. Accessed: 2025-11-10.

[15] Sentencetransformers documentation. https://www.sbert.net/. Accessed: 2025-12-02.

[16] ssdeep 3.4. https://pypi.org/project/ssdeep. Accessed: 2025-12-01.

[17] Steps to publish a microsoft edge extension. https://learn.microsoft.com/en-us/microsoft-edge/extensions/publish/publish-extension. Accessed: 2025-12-11.

[18] Steps to publish a microsoft edge extension. https://learn.microsoft.com/en-us/microsoft-edge/extensions/developer-guide/best-practices. Accessed: 2025-12-11.

[19] Supported apis for microsoft edge extensions. https://learn.microsoft.com/en-us/microsoft-edge/extensions/developer-guide/api-support. Accessed: 2025-12-06.

[20] Track and analyze chrome extensions / mobile apps — chrome-stats. https://chrome-stats.com/. Accessed: 2025-11-10.

[21] Vex: Vetting browser extensions for security vulnerabilities. In *USENIX 2010*, Proceedings of the 19th USENIX Security Symposium, 2010.

[22] Video downloader extension: Universal xss. https://project-zero.issues.chromium.org/issues/42450610, 2018. Accessed: 2025-12-03.

[23] Shubham Agarwal. Helping or hindering? how browser extensions undermine security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 23–37, New York, NY, USA, 2022. Association for Computing Machinery.

[24] Mohammad M. Ahmadpanah, Matías F. Gobbi, Daniel Hedin, Johannes Kinder, and Andrei Sabelfeld. Codex: Contextual flow tracking for browser extensions. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy*, CODASPY '25, page 6–17, New York, NY, USA, 2025. Association for Computing Machinery.

[25] Martin Brinkmann. Hoverzoom extension malware controversy. https://www.ghacks.net/2013/12/26/hoverzooms-malware-controversy-imagus-alternative/, 2013. Accessed: 2024-12-26.

[26] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1687–1700, New York, NY, USA, 2018. Association for Computing Machinery.

[27] chrome. Content scripts. https://developer.chrome.com/docs/extensions/develop/concepts/content-scripts, Accessed on 2025-08-26.

[28] chrome. Service worker overview. https://developer.chrome.com/docs/workbox/service-worker-overview, Accessed on 2025-08-26.

[29] Catalin Cimpanu. "particle" chrome extension sold to new dev who immediately turns it into adware. https://www.bleepingcomputer.com/news/security/-particle-chrome-extension-sold-to-new-dev-who-immediately-turns-it-into-adware/, 2017. Accessed: 2024-12-30.

[30] Mohan Dhawan and Vinod Ganapathy. Analyzing information flow in javascript-based browser extensions. In *2009 Annual Computer Security Applications Conference*, pages 382–391, 2009.

[31] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. Doublex: Statically detecting vulnerable data flows in browser extensions at scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 1789–1804, New York, NY, USA, 2021. Association for Computing Machinery.

[32] Google LLC. Chrome web store review process. https://developer.chrome.com/docs/webstore/review-process/, 2024. Accessed: 2025-01-15.

[33] Google Security Blog. Staying safe with chrome extensions. https://security.googleblog.com/2024/06/staying-safe-with-chrome-extensions.html, June 2024. Accessed: 2025-01-15.

[34] Sheryl Hsu, Manda Tran, and Aurore Fass. What is in the chrome web store? In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, page 785–798, New York, NY, USA, 2024. Association for Computing Machinery.

[35] Thomas Hütter, Nikolaus Augsten, Christoph M. Kirsch, Michael J. Carey, and Chen Li. Jedi: These aren't the json documents you're looking for... In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1584–1597, New York, NY, USA, 2022. Association for Computing Machinery.

[36] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 579–593, Washington, D.C., August 2015. USENIX Association.

[37] Alexandros Kapravelos, Chris Grier, Neha Chachra, Chris Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting Malicious Behavior in Browser Extensions. In *Proceedings of the USENIX Security Symposium*. USENIX, 2014.

[38] Jesse Kornblum. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation*, 3:91–97, 2006. The Proceedings of the 6th Annual Digital Forensic Research Workshop (DFRWS '06).

[39] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

[40] Nikolaos Pantelaios and Alexandros Kapravelos. FV8: A forced execution JavaScript engine for detecting evasive techniques. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3747–3764, Philadelphia, PA, August 2024. USENIX Association.

[41] Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. You've changed: Detecting malicious browser extensions through their update deltas. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 477–491, New York, NY, USA, 2020. Association for Computing Machinery.

[42] Pablo Picazo-Sanchez, Benjamin Eriksson, and Andrei Sabelfeld. No signal left to chance: Driving browser extension analysis by download patterns. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 896–910, New York, NY, USA, 2022. Association for Computing Machinery.

[43] Ben Rosenzweig, Valentino Dalla Valle, Giovanni Apruzzese, and Aurore Fass. It's not easy: Applying supervised machine learning to detect malicious extensions in the chrome web store. *ACM Trans. Web*, October 2025. Just Accepted.

[44] Dolière Francis Somé. Empoweb: Empowering web applications with browser extensions. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 227–245, 2019.

[45] Michael Weissbacher, Enrico Mariconti, Guillermo Suarez-Tangil, Gianluca Stringhini, William Robertson, and Engin Kirda. Ex-ray: Detection of history-leaking browser extensions. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC '17, page 590–602, New York, NY, USA, 2017. Association for Computing Machinery.

[46] Jianjia Yu, Song Li, Junmin Zhu, and Yinzhi Cao. Coco: Efficient browser extension vulnerability detection via coverage-guided, concurrent abstract interpretation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 2441–2455, New York, NY, USA, 2023. Association for Computing Machinery.

[47] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18:1245–1262, 12 1989.

Appendix

TABLE XI
List of Web APIs and Chrome Extension APIs

APIs from [41]
addEventListener
bookmarks.getTree
chrome.storage.local.set
chrome.tabs.executeScript
cookies.getAll
createElement
document.cookie
document.createElement
document.write
downloads.download
googleTag.defineSlot.addService
history.getVisits
history.search
innerHTML
localStorage.getItem
localStorage.setItem
management.getAll
runtime.connect
runtime.onConnect.addListener
runtime.onMessage.addListener
runtime.sendMessage
storage.local.get
storage.local.set
storage.sync.get
storage.sync.set
tabs.executeScript
tabs.sendMessage
window.addEventListener
window.location.replace
xhr
XMLHttpRequest

TABLE XII
APIs empirically observed to be common in malicious extensions (disjoint from Table XI)

Empirically observed APIs
Array.forEach
Array.indexOf
ChildNode
chrome.i18n.getMessage
chrome.runtime.id
chrome.runtime.onMessage
chrome.runtime.sendMessage
chrome.tabs.create
chrome.tabs.query
chrome.tabs.sendMessage
clearInterval
clearTimeout
Date.getTime
Document.body
Document.createElement
Document.documentElement
Document.getElementsByTagName
Element.addEventListener
Element.classList
Element.setAttribute
encodeURIComponent
EventTarget
JSON.parse
JSON.stringify
localStorage
Math.floor
Math.max
Math.random
Math.round
navigator
Node.appendChild
Number.toString
ParentNode.querySelectorAll
parseFloat
parseInt
PropertyDescriptor.get
Readonly
setInterval
setTimeout
Storage.getItem
Storage.removeItem
Storage.setItem
String.concat
String.indexOf
String.match
String.replace
String.slice
String.toLowerCase
Text
TypeError
Window.getComputedStyle

TABLE XIII
Extensions reported to the EAS.

Extension ID (Edge)	Removed after Report	Removal Date
afbegdjlledbjnldkkealalienoigmcp	Yes	2025-12-17
agnnmkflejgoopnfdmpoobimddgbpnnc	No	-
fkfilbkmmgekgajhflamglahakhalhch	Yes	2025-12-16
hapenfmacflbepnbfcfghbgggeebapbk	Yes	2025-12-16
jgloaibpggbddobnljeccnignhehmjoo	No	-
nalpccphogjleedccmoeoaekcijfldbm	Yes	2025-12-16
opdmopbnppchdodekidepknfoepbbggp	Yes	2025-12-17