Detection of Inconsistencies in Privacy Practices of Browser Extensions

Duc Bui, Brian Tang, and Kang G. Shin
The University of Michigan, Ann Arbor, MI, U.S.A.
{ducbui,bjaytang,kgshin}@umich.edu

***Abstract*—All major web browsers support extensions to provide additional functionalities and enhance users' browsing experience while the extensions can access and collect users' data during their web browsing. Although the web extensions inform users of their data practices via multiple forms of notices, prior work has overlooked the critical gap between the actual data practices and the published privacy notices of browser extensions. To fill this gap, we propose `ExtPrivA` that automatically detects the inconsistencies between browser extensions' data collection and their privacy disclosures. From the privacy policies and Dashboard disclosures, `ExtPrivA` extracts privacy statements to have a clear interpretation of the privacy practices of an extension. It emulates user interactions to trigger the extension's functionalities and analyzes the initiators of network requests to accurately extract the users' data transferred by the extension from the browser to external servers. Our end-to-end evaluation has shown `ExtPrivA` to detect inconsistencies between the privacy disclosures and data-collection behavior with an 85% precision. In a large-scale study of 47.2k extensions on the Chrome Web Store, we found 820 extensions with 1,290 flows that are inconsistent with their privacy statements. Even worse, we have found 525 pairs of contradictory privacy statements in the Dashboard disclosures and privacy policies of 360 extensions. These discrepancies between the privacy disclosures and the actual data-collection behavior are deemed as serious violations of the Store's policies. Our findings highlight the critical issues in the privacy disclosures of browser extensions that potentially mislead, and even pose high privacy risks to, end-users.**

I. Introduction

While web browser extensions have been widely used to extend the functionality, and enrich user experience, of web browsers, they pose significant privacy risks to the users. Billions of web browser users can easily install free extensions via extension stores. Due to their integration with web browsers, the extensions can collect highly sensitive data, such as personally identifiable information (PII) and any content that the users input to a web page [53]. These types of data can then be collected by the extensions themselves or transferred to unwanted/unknown/unauthorized third parties [19].

Major extension stores have strict requirements on extensions' privacy practices to reduce privacy risks for users [33, 35, 50]. For example, the Chrome Web Store requires extensions to provide privacy-practice disclosures via the developer Dashboard along with the privacy policies [36]. Privacy policies are free-form documents while Dashboard disclosures are based on a common template that is shared among all extensions on the Store. Fig. 1 shows an example of Dashboard privacy-practice disclosures.

Discrepancies between the different forms of privacy disclosures and extensions' behavior are a serious violation of the

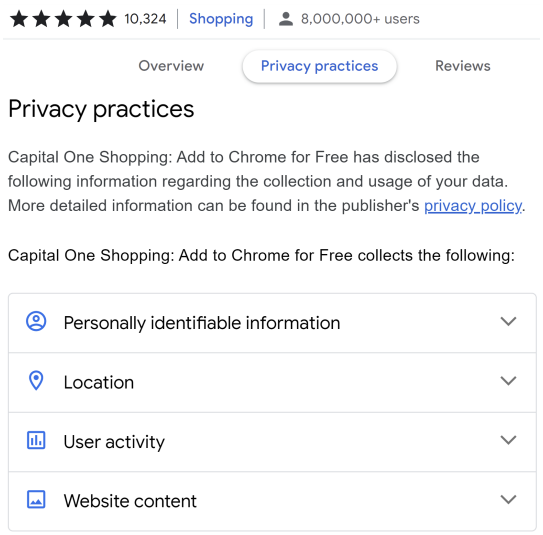

Fig. 1: Dashboard privacy disclosures of a Chrome extension.

Store's developer program policies [37]: *"Any discrepancies between the developer dashboard disclosures, your privacy policy, and the behavior of your item would be a violation of the Chrome Web Store's developer program policies. This can result in the suspension of all the items owned by the publisher, deactivation of the existing user-base, and ban of the entire publisher entity (including related accounts)."*

Because of the "non-discrepancy" requirements, data collection for potentially benign purposes may still violate an extension's privacy policy if the collected data is not disclosed in the policy. For example, if an extension claims *not to collect or use user data*, then it would violate the privacy disclosures even when the extension collects the users' location and keystrokes only for debugging and product-analytics purposes.

Prior work has largely overlooked the inconsistencies between the extensions' execution behavior and privacy policies. Due to their lack/inability of determining the legitimacy of data transfer, prior policy-agnostic detection techniques [16, 43, 60] can only analyze common malicious user-data leakage. For example, they can only detect obvious malicious behavior (such as uninstalling other extensions [43]) or check whether the privacy leakage is either accidental or intentional [60].

The main question we aim to answer is: *Can we automatically detect the inconsistencies of the actual data collection of a browser extension with its stated privacy practices?* We propose, `ExtPrivA`, an end-to-end system that extracts the stated privacy practices and performs a *fine-grained analysis*

of data flows to detect any inconsistencies between the actual data practices and the privacy disclosures of web browser extensions. Similar to software testing based on dynamic analysis [31, 32, 57], we aim to minimize false positives for (maximally) correct detection of inconsistencies. Specifically, ExtPrivA addresses the following 3 technical challenges:

a) TC1 – Detect contradictions of statements in heterogeneous privacy disclosures: Checking inconsistencies between privacy policies and actual data collection requires unambiguous interpretations of privacy disclosures, i.e., detecting any contradictions between the privacy statements. Different privacy-disclosure forms pose a significant challenge due to the differences in their definitions of data types (i.e., *ontologies*). We derived a formal representation of privacy statements from free-form privacy policies and template-based Dashboard disclosures. Finally, based on the extension Store's data-type specifications, we derived a unified ontology to leverage a state-of-the-art privacy analysis [13] to detect the contradictions between privacy policies and Dashboard disclosures.

b) TC2 – Extract actual data collection from extensions' behavior: Since extensions do not automatically execute their functionality while their data traffic only contains low-level key–values, ExtPrivA triggered an extension's functionality and inferred data types from its data traffic to extract its actual data-collection practices. ExtPrivA emulated user interactions on both real-world web pages and a *honeypage* to elicit the extensions' behavior that generated data traffic from the extensions to external servers. We developed a *request-initiator analysis* to isolate the data traffic initiated by extensions. Finally, ExtPrivA extracted data types from key–value pairs in HTTP(S) requests and URL query strings.

c) TC3 – Detect flow-to-policy inconsistencies: The differences between the semantic granularities of data flows and privacy statements (i.e., low and high level) make it challenging to analyze their relationship and check the (in)consistency. From the extension Store's developer policies, we extracted a data-object ontology used in the privacy-practice disclosures to analyze the relationship between data types in the flows and privacy statements. Finally, we established the consistency conditions between the data flows and the privacy statements represented in a formal model to detect their inconsistencies.

We evaluated the accuracy of ExtPrivA in extracting privacy statements or data flows, and end-to-end detection performance via the manual verification of two annotators. Our result shows a precision of higher than 90% in intermediate extractions and an end-to-end detection precision of 85%.

We analyzed the (in)consistencies of the privacy disclosures and data-collection behavior of 47,207 extensions that provide Dashboard disclosures on the Chrome Web Store. ExtPrivA identified 525 contradictions in the Dashboard disclosures and privacy policies of 360 extensions which made their privacy disclosures ambiguous. Finally, it found 820 extensions with 84.6M users experiencing 1,290 data flows that are inconsistent with their Dashboard disclosures.

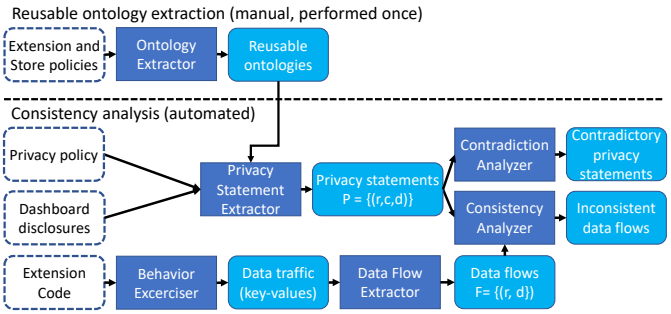

Fig. 2: ExtPrivA analysis pipeline.

This paper makes the following main contributions.

- A novel fine-grained analysis that detects the inconsistencies between the privacy disclosures and the actual data practices of web browser extensions. The analysis also identifies ambiguities in the privacy disclosures by detecting the contradictory privacy statements between free-form privacy policies and template-based Dashboard disclosures.
- An end-to-end automated framework, called ExtPrivA, that analyzes flow-to-policy (in)consistencies of browser extensions. It extracts privacy statements from the disclosed privacy practices (Section III), performs dynamic analysis to extract data traffic (Section IV), and extracts data flows from the transferred key-values (Section V). Finally, the system detects contradictory privacy statements and inconsistencies between the data flows and the privacy statements by using a formal model (Section VI). Our evaluation demonstrates that ExtPrivA detects contradictory statements with a 91.7% precision, and detects flow-to-policy inconsistencies with an 85% precision. Fig. 2 shows the analysis pipeline.
- A large-scale study of 47.2k extensions in the Chrome Web Store (Section VII). Despite the Store's strict vetting process, we still found a large number of extensions that had contradictory statements and flow-to-policy inconsistencies in their privacy disclosures and data-collection behavior, posing high privacy risks to millions of users. This finding highlights the critical issues of privacy-practice disclosures of browser extensions in practice.

II. Background

A. Extension-Platform Privacy Requirements

In addition to privacy policies, major extension stores require extensions to provide easy-to-read disclosures of their privacy practices to users. In particular, Google has required developers to declare the types of data their extensions collected via the developer dashboard since January 2021 [36]. Developers must also certify that they follow the Limited Use policy under which the transfer of user data to ad platforms, or for personalized advertising, is prohibited [35]. The Dashboard disclosure form is shown in Fig. 7 (Appendix C). In this paper, we use the term *privacy policy* to distinguish the free-form policy documents from the template-based *Dashboard privacy-practice disclosures* of the Chrome Web Store.

B. Extension Architecture

a) Web Browser Selection: We select Chrome as the representative extension architecture for further examination because of its widespread adoption. At the time of this writing, Google Chrome is the most popular browser. It constitutes 67% and Chromium-based browsers (Chrome, Edge, and Opera) constitute 79% of the desktop browser market share [61], a significantly larger share compared to other browsers (e.g., Firefox and Safari have less than 10% each). Furthermore, the Firefox and Safari browsers have adopted a cross-browser extension API, called *WebExtensions* [44, 52]. Therefore, the design principles of our analysis pipeline should apply to other non-Chrome browsers.

b) Main Components of Extensions: A Chrome browser extension comprises four main executable components: background scripts (or *background pages*), content scripts, web-accessible resources (WARs) and pop-up pages. A JSON manifest file declares the components and how they are executed with respect to a web page. Background scripts (Manifest V2) and service workers (Manifest V3) have no UI and are hidden from users. They have a lifetime independent of other user-facing web pages and can access privileged Chrome extension APIs. The scripts run asynchronously to handle events generated by other components and can communicate via message passing. Content scripts are injected into a web page and can read/modify the DOM tree which is inaccessible by the background pages [17]. The scripts can be configured to execute at either the start or end of DOM-loading. On the other hand, WAR resources' JavaScript is loaded and runs in the same context as the host pages. Pop-up pages execute upon a user's click on the extension icon to interact with users.

c) Extension Identification: An extension in a browser is uniquely identified by an extension ID that can be used to access the resources included in the extension's package. Resources of an extension, such as its content scripts, have URLs prefixed with *chrome-extension://<extension-id>*. The ID is generated randomly when the extension is loaded to the browser but can be made to be a fixed value by specifying a *key* value in the extension manifest [26].

d) Execution Entries: Since extensions are event-driven applications that have multiple entry points to trigger their functionality, the manifest file provides extensions with a means to statically declare their static entry points [27]. An extension can specify the URL patterns where the content scripts are executed when loading a web page from a pattern-matched URL. It can also implement event handlers for extension actions (i.e., mouse clicks on the extension icon on the browser menu bar) and the activation of the extension's context-menu items. Finally, extensions specify the URL-match patterns on which it has effect. These patterns are included in the *host permissions* and *web accessible resources* to restrict the web pages to which the extensions have access and the pages which can access the resources (e.g., JavaScript and CSS) included in the extension package, respectively.

e) Restriction of Network Access: To enhance the browser's security, only background scripts and pop-up pages bypass the same cross-origin resource sharing (CORS) policy and can send information to any servers without any restriction. By contrast, content scripts are subject to the CORS policy of the host web pages [23]. Unless the server side allows CORS requests, content scripts cannot directly request resources or send information to an arbitrary external server other than the origin of the currently visiting URL.

III. Analysis of Privacy-Practice Disclosures

The privacy statements of an extension comprises the statements from template-based Dashboard disclosures and free-form privacy policies. In this section, we describe the analysis and extraction of formal privacy statements from these two forms of privacy disclosures.

A. Privacy Statement Definition

To simplify the analysis of privacy policies, we limit the analysis to the statements about data collection, i.e., whether a receiver *r* collects a data type *d* or not, as formally defined next. Since privacy-practice disclosures specify a fixed policy for data-usage purposes, called a *Limited Use* policy [37], analysis on data-usage purposes can be done separately.

Definition III.1 (Privacy Statements)**.** *A privacy statement is a tuple $s = (r, c, d)$ where r is a receiver that collects or does not collect ($c \in \{collect, not\_collect\}$) a data type d.*

B. Analysis of Dashboard Disclosures

We extract privacy statements from an extension's Dashboard privacy-practice disclosures as follows. Let $D = \{d_i\}$ be the set of data types that the extension declares to collect and T be the set of all possible data types that an extension can declare. We assume that if an extension does not declare its collection of a data type $d_i \in T$, then it will not collect d_i. The set of data types U not collected by the extension is then derived by excluding the stated data types from T: $U = T \setminus D = \{d'_i | d'_i \in T \wedge d'_i \notin D\}$. From D and U, the following privacy statements will be created: $S = \{(r, collect, d_i) | d_i \in D\} \cup \{(r, not\_collect, d'_i) | d'_i \in U)\}$. For example, given the Dashboard disclosures in Fig. 1, where the extension states the collection of only 4 data types: PII, Location, User Activity, and Website Content, we have $D = \{PII, location, user\_activity, site\_content\}$ and the corresponding privacy statements are $S = S_c \cup S_n$ where $S_c = \{(extension, collect, d_i) | d_i \in D\}$ and $S_n = \{(extension, not\_collect, d'_i) | d'_i \in T \setminus D\}$.

At the time of writing this paper, the Chrome Web Store specifies a total of 9 data types that an extension can declare [36]. Therefore, `ExtPrivA` extracts a total of $|T| = 9$ privacy statements for each extension which comprise $|D|$ positive-sentiment statements (i.e., with a $collect$ action) for the declared data types D and $9 - |D|$ negative-sentiment statements (i.e., with a $not\_collect$ action) for the undeclared data types in the extension's Dashboard disclosures. Since the privacy-practice disclosures follow fixed declaration templates,

D is extracted from the disclosures using regular expressions. All data-type examples in the Chrome Web Store policies are listed in Table X and Fig. 7 (Appendix C).

C. Analysis of Free-form Privacy Policies

a) Privacy Statement Extraction: Given the privacy policy of an extension, ExtPrivA adopts PurPliance [13], a state-of-the-art privacy-policy analysis technique, to extract privacy statements from the sentences in the document. For each sentence, the parameters of privacy statements (data type, collection action and receiver) are determined by an NLP pipeline. The system first identifies the sharing-collection-and-use verbs in the sentence and then uses the semantic role labeling to extract the semantic arguments (e.g., subjects and objects) of each verb. The data types are extracted from the verbs' objects by a named entity recognition (NER) model.

Since the NLP pipeline was originally designed for Android apps, ExtPrivA addresses the following challenges to handle the differences between the privacy policies of Chrome extensions and Android apps.

b) Extension-Scope of Privacy Statements: While Android apps typically have dedicated privacy policies, many browser extensions are found to use generic privacy policies that cover the data practices shared by the web services developed by the same developer. Most privacy policies include statements for multiple platforms (such as websites, apps, and extensions). For example, "Capital One Shopping systems capture email-header data (sender, recipient, date and subject, not message bodies)" [55] applies only when users grant inbox access. However, the current sentence-based privacy-policy analysis techniques [7, 13] cannot distinguish the scope of each statement (i.e., whether the statement is about the website or the extension) owing to the lack of a holistic whole-document analysis. Therefore, we exclude statements that do not mention extensions to reduce false positives. In particular, we include only the sentences that contain the keyword "extension".

c) Extension Data Ontologies: Since the data-type ontologies modeling data types and their relationship of Android apps are different from the relationships of browser extensions, we augment them with the high- and low-level data types of the Web Store (Table X). Similar to the domain adaptation [46] in NLP, this addition is necessary because privacy policies of Android apps do not include certain extension-specific data types, such as Website Content and Web History.

D. Implementation Details

We extracted privacy statements from template-based Dashboard disclosures by parsing the Privacy-Practice page of each extension while we leveraged the open source PurPliance [13] to extract privacy statements from the privacy policies. For each extension, ExtPrivA parsed the overview page to obtain the URL of the privacy policy. The system then pre-processed and extracted the sentences from the policy using both rule-based methods and neural NLP models. The crawling and pre-processing of extensions' privacy policies are described in Appendix B.

IV. Analysis of Extension Execution

ExtPrivA analyzes the data collection of an extension in three main steps: 1) exercise the extension functionality, 2) extract the network traffic initiated by the extension, and 3) extract the data flows that comprise the receivers & data types from the raw data traffic. We describe the first two steps in the rest of this section while presenting the last step in Section V.

A. Triggering Extension Functionality

1) Candidate URL Extraction: Since extensions do not have access to all websites by default, ExtPrivA first identifies the URLs of the websites that an extension has access to. To generate these URLs, ExtPrivA analyzes the extension manifest and extracts the URLs patterns for the background scripts, content scripts and WAR resources declared in the *host_permissions* and *matches* keys in the manifest.

ExtPrivA generates a set of candidate URLs from each URL pattern. A pattern is first decomposed into 4 components, following the manifest format [25]: scheme, subdomain, domain, and path. ExtPrivA then synthesizes the candidate URLs that match the specified URL patterns by substituting wildcard components with common valid values such as *www* for a subdomain. For example, from *https://*.example.com/subpath/**, a candidate URL *https://www.example.com/subpath/* is generated. Inspired by Hulk [43], for those patterns that match unspecified domains and paths such as *<all_urls>* and *https://*/**, ExtPrivA selects top website domains in two categories, search and shopping, on which extensions commonly execute from the Tranco list [45]. These URLs are listed in Table VIII.

2) Test pages: To test the extensions, we use two types of web pages: real pages and a *honeypage*. The former is real-world web pages that are served either from the Internet and a web-page replay server. In contrast, the latter is a specially-crafted web page that is based on prior extension analysis work [64] and contains various HTML elements to trigger common functionality of extensions. The HTML elements include text and password *input* elements which use privacy-sensitive keywords (e.g., *username*, *name*, and *city*) for their HTML attributes (e.g., *id*, *name*, and *class*). Real pages are useful for extensions that execute based on the structure of websites where the *honeypage* cannot be replicated. For example, the *honeypage* cannot emulate real complex websites like Amazon shopping pages.

3) User Interaction Emulation: To trigger the data-collection functionality of extensions that operate upon user activities, we design interaction templates based on browser, mouse and keyboard actions. These actions are the main user-interaction categories expected by extensions to execute their functionality [58]. We re-implemented the interaction templates [58] and further customized them for specific websites. For example, a different element selector is used depending on whether the browser is accessing a Google search result page or an Amazon product page. ExtPrivA performs the following templates after a web page is fully loaded:

a) Text Selection and Mouse Actions: To elicit potential data collection on the text selected on a web page, ExtPrivA selects a word and activates the extension via the extension icons on the menu bar and/or the context menu. For example, a dictionary extension shows the definition of a selected word after the user selects the word, clicks the right mouse button and selects the extension icon on the context menu. Therefore, ExtPrivA performs mouse scrolling and clicking to select the text to trigger extensions that operate upon mouse events (like clicks and double-clicks). When the web page is the *honeypage* or a replayed page, the text of a fixed element is selected. Otherwise, ExtPrivA selects the first word of the <*body*> element that is expected to exist on any web page.

This interaction template already includes a click on the extension icon on the browser menu bar. For example, a shopping assistant shows the information of products on *amazon.com* only when the user clicks on its menu-bar icon. This interaction is called *extension action* (Manifest V3) or *browser action* (Manifest V2) and is one of the main entry points that trigger the functionality of extensions.

b) Keyboard Input and Form Submission: To trigger extension functionality that monitors keyboard events, ExtPrivA inputs a keyword into a form field on the *honeypage*, issues a copy command via *ctrl+c* and submits the input form to our server endpoint. For example, spelling checkers may monitor keyboard typing and suggest a correction. Inspired by the *bait* technique [1], ExtPrivA inputs a special value, called *bait*, to detect the extension's collection of keyboard input.

c) Interaction with New Tab Pages: To trigger the extensions that provide a customized new tab page, ExtPrivA opens a new tab and types in a keyword. For example, the *Infinity New Tab* extension opens a customizable tab that lets users enter a search term and shows the current weather forecast. This kind of extensions may collect user location and/or search terms without the user's awareness.

B. Data Traffic and Initiator Analysis

It is challenging to extract the data traffic originated from an extension because the HTTP requests sent from the browser do not differentiate between those sent by extensions and those sent by web pages. Therefore, ExtPrivA extracts extensions' data traffic by analyzing the request-initiator scripts and the HTTP Origin header as follows.

First, ExtPrivA leverages the call-stack information of script-initiated network requests provided by the network-activity inspection of Chrome's DevTools. In DevTools, an initiator of a network request can be one of 6 types such as a JavaScript script or HTML parser [20]. An extension's script, like other extension's resources, has its URL in the form of *chrome-extension://<extension-id>/path-to-script* (identifying extension IDs is described in Section IV-D3). Since the initiator information of a script contains call frames which include the initiated scripts' URLs in the call stacks, if a script URL is prefixed by a *chrome-extension* scheme and matches the extension ID, the traffic is initiated by the extension. While

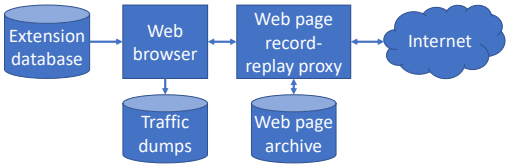

Fig. 3: ExtPrivA extension analysis testbed.

content scripts execute in web pages' contexts, DevTools captures the requests of content scripts. However, it indicates *chrome-extension://* initiators for content scripts but not for injected inline scripts.

Second, ExtPrivA utilizes the *Origin* HTTP request header which is non-programmatically modifiable to indicate the security contexts that cause the browser to initiate an HTTP request [51]. This header is set to *chrome-extensions://<extension-id>* if the request is initiated by an extension. External scripts in the background or pop-up pages of the extension do not have URLs with the chrome-extension scheme, and thus cannot be identified by using the call stacks in the script initiators. Using the Origin header can identify the requests initiated by such embedded external scripts.

C. Extraction of Key–Value Pairs

ExtPrivA parses HTTP requests in the extension traffic intercepted in the prior step into key–value pairs since structured responses are widely used by web services [21]. The key–value pairs are extracted from the sent cookies, URL query strings and request bodies of HTTP POST messages. In our dataset, while most of the traffic is plaintext, when encountering encoded traffic, ExtPrivA attempts to decode the data by using multiple rounds of Base64 decoding. This decoding is based on the technique used by Starov *et al.* [60]. However, ExtPrivA cannot extract data flows from the data traffic encrypted by extensions.

Unlike automatic data such as IP addresses as part of the IP protocol or information in the HTTP security headers, sending data via URL parameters or the POST body requires a significant effort to obtain and set the values correctly. In particular, obtaining and adding personal data to URL parameters require developers' time and effort, unlike the IP addresses that browsers automatically set. Therefore, the occurrences of these values in the transferred data are unlikely created by the extension developers by accident. To further reduce false positives of unintentional data leakage, we exclude key–value pairs in HTTP headers (other than the Cookie header) because the headers may include information automatically set by the browser rather than intentionally set by the extension.

We filtered out key–value pairs sent to the servers that had the same hostname as the currently visited web page because the web page had already collected the user's data. For example, if a user visits host *H=sub.example.com*, we exclude extensions' traffic to *H*. It is unclear whether the extensions are leaking data because the user already shared data with *H*. However, this filtering of same-host traffic does not create false positives and excludes only 0.81% (381/47,207) of extensions.

D. Implementation Details

1) Analysis Pipeline: ExtPrivA performs dynamic analysis and captures data traffic of the extensions using an analysis pipeline as shown in Fig. 3. Each extension is initially loaded to a clean browser instance that disables updates and other unnecessary background traffic such as user-metrics reporting to avoid noisy traffic, following the measurement procedure of Chromium telemetry framework [29]. The browser then records the traffic of the extensions and loads web pages via a web page record-replay proxy. The browser employs mechanisms to avoid bot detection that has been known to affect the real behavior of websites [11, 42].

ExtPrivA utilizes the Playwright browser automation tool [49] to drive an instance of the Chromium web browser. Extensions are loaded to browser instances that display to a virtual X11 frame buffer (Xvfb) as the browser does not support loading extensions in the headless mode. The keyboard keystrokes are sent to the browser instances via the X11 server using the *xdotool* [56]. To trigger an extension action (i.e., a click on the extension menu-bar icon), ExtPrivA instruments the manifest to set the shortcut keys to perform the extension actions because the browser automation tool can only interact with web pages' contents but not the interface of the browser.

To make the experiments reproducible, we employ a web page replay (WPR) proxy, a modified version of the Chromium WPR tool [22], to record, replay and passthrough network requests and responses. The WPR proxy replays website contents for reproducibility while allowing the browser extensions to communicate with the Internet to capture the extensions' realistic behavior. Since the WPR proxy passes through dynamic requests that had not been pre-recorded, it tests extensions on dynamic contents while replaying static content to avoid additional traffic to websites. In the record mode, the WPR proxy records the responses of web pages by using the browser with no installed extensions. In the replay mode, the proxy passes through requests which are not found in the WPR proxy's recorded request store. In particular, the most commonly visited web pages were recorded and replayed. We set up the browser to whitelist the SSL certificates for the WPR proxy to capture and replay encrypted HTTPS traffic.

2) Traffic Interception: To intercept the traffic generated by the JavaScript's XHR requests, ExtPrivA utilizes the Chrome DevTools Protocol (CDP) [24] to extract the network traffic from a web page to servers. ExtPrivA creates a CDP session via Playwright to send commands and receive events from the DevTools in the browser instance. Specifically, ExtPrivA enables network tracking functionality of the DevTools and extracts information from the network events. The request tracing information contains a *request initiator* which can be the DOM parser or a script. Since the browser treats a background page as a regular web page, ExtPrivA captures the network traffic of background pages of Chrome extensions separately.

3) Determine Extension IDs: To accurately extract the network traffic originating from an extension, ExtPrivA determines the extension's ID which is unique in the browser instance. The system adds a *key* value to the manifest to make the extension ID non-randomized [26]. ExtPrivA then extracts the extension ID from the preference configuration file in the browser's *user data directory* and also verifies the loaded extension path.

4) Testbed: To perform experiments in a large dataset of extensions, we create a distributed experimental framework to run the dynamic analysis on multiple machines. The testbed is replicated and run in identical and isolated environments. The framework is based on Docker Swarm [39] and the browser is started with arguments to make it run in the resource-constrained docker environments [48].

V. Data Flows

A. Data Flow Definition

Given the data traffic of an extension collected in Section IV, ExtPrivA extracts data flows that formally represent the data-collection behavior of the extension. A data flow is formalized in the following definition.

Definition V.1 (Data Flow)**.** *A data flow is a tuple $f = (r, d)$ where a receiver r receives a data object d.*

B. Extraction of Data Flows

1) Extraction of Data Types: We select data types and design a rule-based extractor as follows.

a) Data-Type Selection: Of the Store's 9 data types, we choose to extract 4 context-free data types whose meanings do not depend on their usage contexts: Website Content, Web History, Location and User Activity. It is challenging to extract context-sensitive data types because the lack of the server-side information makes it practically impossible to determine the ultimate usage purposes of the collected data. Moreover, these data types (e.g., PII and authentication information) are included in Website Content. For example, if a user enters a home address in a Google search box while extension E records every input to the search box, it is not possible to determine whether E intentionally collects home addresses (PII) or only website content by solely analyzing data traffic.

ExtPrivA extracts 11 low-level data types under the high-level data types as listed in Table I. Since the Store provides only several examples rather than an exhaustive list of low-level data types, we add the following examples for their privacy significance and relevance to our experiments. *Page URL* is one of the "browsing-related data", a definition of the Store for the Web History, and can be used to exactly determine the page that a user visited. Similarly, *Page Hostname* reveals a user's browsing habits while extensions frequently break a page URL into a hostname and a URL path before sending them to external servers. Finally, *Product ID* is considered separately for analyzing shopping-assisting extensions during user visits to shopping sites like *amazon* and *ebay*.

While adding the low-level data types widens the scope of the high-level types, we avoid any addition that makes the high-level data types overlap and become ambiguous. In particular, some low-level data types overlap (such as Page

URL and Page Hostname) but one low-level data type does not simultaneously fall into different high-level data types.

b) Extractor Design: The extraction of data types from a key-value pair is formulated as a classification problem. For each low-level data type, we create a classifier that determines whether the key-value contains the data type or not. To achieve low false positives (i.e., high precision), we design classifiers based on pattern-matching rules as follows.

To extract Website Content and Web History data types, `ExtPrivA` searches for the content and the URL of the currently visited web page in the transferred key-values. For example, if the traffic contains an exact match of the URL of the web page, the extension collects the currently visited URL or the Web History data type. Similarly, for certain websites, we search for an ID in the URL such as an item ID on amazon URLs (e.g., *amazon.com/dp/ABC* where the last part of the URL, *ABC*, is the item ID). Inspired by the *bait* technique [1], in addition to the existing website content, we search for the *bait* value contained in the *honeypage* in the traffic. The *bait* is selected to avoid collision with other common keywords in the traffic key-values so that its occurrence in the traffic indicates the collection of the Website Content.

To detect the collection of User Activity, we rely on API documentation and the *bait* technique. Specifically, we found that extensions utilized popular the Sentry monitoring library [40] to monitor the keyboard input and mouse clicks. In particular, "ui.click" and "ui.input" are used for a mouse click and keyboard input events, respectively. Furthermore, after `ExtPrivA` inputs a *bait* keyword W via keyboard, if only part of W, but not the whole W, exists in the traffic, we consider the extension monitored keystrokes.

c) Development of Data-Type Matching Rules: We follow the widely-used bootstrapping procedure in which the set of patterns is built iteratively with minimal human intervention [4, 41]. To create the seed matching patterns for a data type T, we first performed an exploratory study on the data traffic of the extensions that disclosed their collection of T. Using a set of patterns, we found a set of matching key-value pairs where we discovered the new patterns. The process is then repeated while retaining only the most reliable patterns after each iteration. The final patterns were found to change only slightly with carefully-tuned seeds [41] and are listed in Table I.

2) Extraction of Data Receivers: Given a data type extracted from a key-value pair, the *receiver* of the corresponding data flows is set to the *extension* that sent the data and the external server where the data is sent, regardless of the ownership of the external server. Because a key-value is transferred to an external server by the execution of an extension, the extension must first collect the data from the browser or web pages before sending it to the external server. The data types extracted by `ExtPrivA` (Table I) are dynamic data that require the execution of a script or API call to retrieve their values, rather than static/hard-coded data like an extension's version. For example, when an extension sends a user's mouse clicks to *google-analytics.com* for its development-analytics purposes, the extension is considered to collect the user activity even if it does not own the Google Analytics server.

High-level Type	Low-level Type	Matching Pattern
Web History	Page Title*	Exact match of page title
	Page URL	Exact match of page URL
	Page Hostname	Exact match of page hostname
Website Content	Hyperlink*	Hyperlinks in <a> elements
	Website Text*	*bait* text value
	Product ID	Product ID on shopping sites
Location	IP Address*	IP addresses of testbed servers
	Region*	<city_name>, <zip_code>
	GPS Coordinates*	Coordinates of testbed servers
User Activity	Mouse Click*	*ui.click* events
	Keystroke Logging*	*ui.input* events/partial *bait* input

TABLE I: List of the high-level and low-level data types supported by `ExtPrivA`. * marks the examples of low-level data types provided by the Chrome Web Store [36].

Even when an extension directly shares user data with third parties, it poses high privacy risks to users if the users are not aware of the collection of their data due to the execution of the extension. For example, when a translation extension transmits the user-selected text to an external spelling-checking service, the user needs to be aware of such data collection to avoid inadvertently selecting sensitive data, such as an email with a trade secret, to be sent to an external spell checker.

VI. Detection of Inconsistencies

A. Semantic Relationships

`ExtPrivA` detects the inconsistencies between an extension's actual data collection and its privacy-practice disclosures by analyzing the (in)consistencies between the extracted privacy statements (Section III) and data flows (Section V). As data flows and privacy statements are expressed in different terms and granularity, in order to check their (in)consistencies, `ExtPrivA` leverages ontologies of data types and receiving entities that represent the relationship between terms to perform logical comparisons between the statements and flows. An ontology o can be represented as a directed graph of data-type terms where each edge between two nodes x and y points from a more general term y to a more specific term x. For example, there is an edge from Website Content to Hyperlink data type. Inspired by prior work [7, 8, 13], the semantic relationships and consistency conditions are defined as follows.

Definition VI.1 (Semantic Equivalence)**.** *Two terms x and y are semantically equivalent in an ontology o, denoted as $x \equiv_o y$, if and only if they are synonyms in o.*

Definition VI.2 (Subsumptive Relationship)**.** *Two terms x and y have a subsumptive relationship (i.e., x has an "is-a" relationship with y) in an ontology o, denoted as $x \sqsubset_o y$, if there are a series of terms $x_1, x_2, \ldots, x_{n-1}$ ($n \in \mathbb{N}$ and $n \geq 1$) such as $x \sqsubset_o x_1$, $x_1 \sqsubset_o x_2$, ..., and $x_{n-1} \sqsubset_o y$. Similarly, $x \sqsubseteq_o y \Leftrightarrow x \equiv_o y \vee x \sqsubset_o y$.*

B. Privacy-Statement Contradictions

Two privacy statements are said to be *contradictory* if their data or receivers have subsumptive relationships with each other while the statements have opposite sentiments (positive vs. negative). For example, a contradiction occurs between "we do not collect your personal data" and "we may collect your location" because *location* subsumes under *personal data* while keeping the receivers the same. We leverage the logical contradiction rules in PolicyLint [7] to detect such contradictions and formalize them as follows.

Definition VI.3 (Policy Logical Contradiction)**.** *Two privacy statements* $(e_i, collect, d_k)$ *and* $(e_j, not\_collect, d_l)$ *are contradictory if* ($d_k \sqsubseteq_\delta d_l$ *or* $d_l \sqsubset_\delta d_k$) *and* $e_i \sqsubseteq_\epsilon e_j$ *in a data-type ontology* δ *and an entity ontology* ϵ*.*

A main challenge in detecting contradictions between Dashboard disclosures and privacy policies lies with the differences between their data-type ontologies that comprise the sets of data types and their subsumptive relationships. Specifically, the Dashboard data types are defined by the Chrome Web Store and follow narrower definitions than those used in the privacy policies that contain broader statements about the websites, services and extensions. For example, the term "personally identifiable information" in privacy policies includes "IP addresses" [47] while the Store's definition does not [37].

To resolve these differences and analyze privacy statements uniformly, we treat the collection of the data types in Dashboard disclosures as normal sentences so that they are comparable with the statements in the privacy-policy counterpart. For example, the collection of Location in Fig. 7 is treated as "we collect your location." Therefore, we use the privacy policies' ontologies that are broader than the Store's ontologies to analyze the privacy-statement tuples in both privacy policies and Dashboard disclosures. In particular, we add the data-type nodes and subsumptive-relationship edges in the Store's ontology graph into broader privacy-policy ontologies.

An advantage of this approach is that the unified ontologies can be used to detect the contradictions among the statements of the same privacy policies. While this approach excludes the generated negative-sentiment statements that require a complete declaration of all data types (Section III-B), ignoring these statements do not generate any false positives.

C. Flow-to-Policy Consistency

Definition VI.4 (Flow-Relevant Privacy Statement)**.** *A privacy statement* $s_f = (r_f, c, d_f)$ *is said to be relevant to a flow* $f = (r, d)$ *if and only if the flow's receiver and data object are subsumed under the corresponding terms of the statement, i.e.,* $r \sqsubseteq_\epsilon r_f$ *and* $d \sqsubseteq_\delta d_f$ *in an entity ontology* ϵ *and a data-type ontology* δ*.*

Definition VI.5 (Flow-to-Policy Consistency)**.** *A data flow* f *is said to be consistent with a set of privacy statements* $S = \{s\}$ *if and only if the set of flow-relevant privacy statements* $S_f \subset S$ *contains a positive-sentiment and no negative-sentiment privacy statement, i.e.,* $\exists s_f = (r_f, c, d_f) \in S_f$ *s.t.* $c = collect$ *and* $\nexists s'_f = (r'_f, c', d'_f) \in S_f$ *s.t.* $c' = not\_collect$*.*

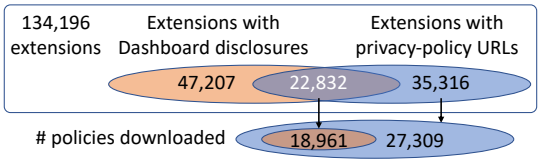

Fig. 4: Extension Dashboard disclosures and privacy policies.

Informally, given privacy disclosures that comprise a set of privacy statements, a flow is consistent with the disclosures if there is a positive-sentiment statement that states the collection of the data type in the flow while there is no negative-sentiment statement that describes the "non-collection" of the data. For example, a flow *f = (extension, selected text)*, where the *selected text* in the currently visiting web page is collected by the *extension*, has a relevant statement that "we collect the website content" because *website content* includes the *selected text* (i.e., *selected text* $\sqsubset$ *website content*) and $we \equiv extension$. The flow is then consistent with the disclosures if there is not any relevant statement that states otherwise.

A flow-to-policy inconsistency occurs when the Consistency Condition (Definition VI.5) is not satisfied. We classify the types of the inconsistencies into Correct Disclosure and Incorrect Disclosure. A Correct Disclosure occurs when the Consistency Condition holds and an Incorrect Disclosure happens if the condition does not hold. For example, a flow *(extension, selected text)* is inconsistent with privacy disclosures if there is a negative statement *(extension, not_collect, website content)*.

We focus on the inconsistencies between the extension behavior and the Dashboard disclosures since they follow the same extension-specific data-type ontologies defined by the Chrome Web Store. Comparing the data-collection behavior with the privacy policies requires resolving the semantic gap between the data types defined in the Store and the common policies (Section VI-B) while the flow extraction is designed based on the Store's data-type ontologies. Furthermore, the inconsistencies between the data flows and privacy-policy documents have already been studied before [7, 13]. Finally, because the complete list of data types is defined by the Store, this flow-to-policy consistency analysis utilizes the negative-sentiment privacy statements for the undeclared data types as described in Section III-B.

VII. Evaluation

We performed an in-depth analysis of the flow-to-policy inconsistencies of the extensions on the Chrome Web Store. Presented below are the experimental setup and results.

A. Extension Selection

We designed a crawler to collect extensions on the Chrome Web Store. By following the Store's *sitemaps* [34], the crawler systematically visited and extracted the source code and description of each extension. The data collection was done by a server located in the US on Feb 21, 2022, and took 18 hours to complete.

# Data Types	% Exts.
0	71.57
1	15.97
2	5.47
3	3.84
4	2.03
≥ 5	1.12

(a) Data types per extension.

Data Type	# Exts. (%)
Website Content	6,392 (13.54)
PII	4,545 (9.63)
User Activity	4,533 (9.60)
Authentication Info.	3,005 (6.37)
Location	2,562 (5.43)
Web History	2,549 (5.40)
Personal Comm.	929 (1.97)
Financial & Payment	509 (1.08)
Health Information	135 (0.29)

(b) Distribution of data types.

TABLE II: Data types of Dashboard disclosures.

Candidate URL	Total
<all_urls>	7841
https://www.youtube.com	1068
https://www.coolstart.com	1002
https://www.google.com	893
https://www.mystart.com	804
https://www.facebook.com	686
https://www.amazon.com	651
https://mail.google.com	646

TABLE III: Top candidate URLs.

# Flows	# Exts.
1	618
2	169
3	121
4	83
5	11
Total	1002

TABLE IV: # flows/extension.

Receiver Type	# Flows (Incons./Total)
First party	324/524
Analytics	265/314
Tracker	116/166
Ad network	50/51
Other	535/651
Total	1,290/1,706

TABLE V: # (inconsistent) flows per receiver type.

The total number of extensions collected is 134,196. There were 35,316 (26.32%) extensions providing a privacy-policy URL while 12,484 of them had no Dashboard disclosures. ExtPrivA downloaded and extracted plain text versions of the privacy policies of 27,309/35,316 (77.33%) extensions while the remaining policy URLs were inaccessible. Besides, a significant number of extensions, 74,505 (55.52%), provided neither Dashboard disclosures nor privacy-policy URLs. Fig. 4 shows the number of the privacy-disclosure types.

In the following experiments, we consider the 47,207 (35.18%) extensions that declared the Dashboard disclosures. The disclosures have been required for the publication of an extension on the Chrome Web Store since March 2021 [37]. We also observed a significant increase of extensions with Dashboard disclosures, from 31,839 extensions in a crawl in May 2021. Therefore, we assume that all extensions on the Store will gradually include Dashboard disclosures.

B. Policy and Flow Characterization

1) Dashboard Disclosures: The majority of extensions state not to collect any user data while a significant number of extensions state collection of only 1 data type. Of the extensions with Dashboard disclosures, 33,787/47,207 (71.57%) state that they do not collect or use any user data while 15.97% of the remaining extensions state the collection of only 1 data type. As shown in Table IIa, this kind of extensions (i.e., those that collect only 1 data type) is the most common.

For each data type, the number of extensions that declared the data collection is also small. The most common data type collected by the extensions is Website Content (13.5%) while the least common is Health Information (0.29%). Table IIb shows the distribution of the collected data types.

2) Privacy Policies: Of the 47,207 extensions with Dashboard disclosures, 22,832 (48.37%) contain privacy-policy URLs. From these URLs, 18,961 (83.05%) privacy policies were successfully downloaded. The system then extracted 8,012 extension-related privacy statements from 2,091 extensions' privacy policies. Because of the exclusion of the statements that do not mention browser extensions, ExtPrivA did not include the policies from the remaining extensions.

Of these privacy statements, 6,238 (77.86%) have a negative sentiment and 1,774 (22.14%) have a positive sentiment. 1,538 extension policies contain negative sentiment statements that discuss broad categories of data. Of the statements with a negative sentiment, the data object "personally identifiable information" or "PII" appears in 1,280 of these extensions. This high percentage highlights the significance of negative privacy statements as 83.22% (1,280/1,538) of the extensions contain a negative sentiment that excludes the collection of a broad data type.

C. Data Traffic and Flows

1) Experimental Setup: Given an extension E, ExtPrivA first identifies the candidate URLs to activate the extension's functionality (Section IV-A1). The system then visits each of the identified URLs in a clean browser instance with the extension E installed at each start-up while disabling other extensions to reduce execution and traffic noise. For each URL, the system visits a real page and a *honeypage*. If the URL has been recorded by the Web Page Replay proxy, the network requests are redirected to the proxy to reduce loads on the server side while improving the reproducibility of the experiments. Since the number of the candidate URLs can be large, for each extension, ExtPrivA visits the URLs until either all URLs or a maximum of 10 URLs are visited.

For each URL, the browser waits until the home pages are fully loaded by waiting until there are no network connections within a timeout of 5 seconds or a maximum of 30 seconds. Because the experimental servers used a fast Internet connection, we empirically found that these timeouts were sufficient to completely load most of the web pages. The page loading heuristics are commonly used in the empirical settings and provided as the default in the web browser automation tools [28, 49]. Finally, ExtPrivA interacts with the browser to activate the functionality of the extension (Section IV-A3). It is worth noting that an experiment does not raise false positives if the extension is not successfully loaded or its functionality is not activated. The analysis was performed on a cluster of 8 machines with 1.18TB of RAM in a university in the US and took 70 hours to complete.

2) Extension URL Patterns: From the 47,207 extensions that provide Dashboard disclosures, we extracted 129,218 candidate URLs on 28,618 domains. The distribution of the domains has a long tail with only 248 domains with a frequency greater than 100. The most common extracted domains are *yahoo.com* and *google.com* which involve a large number of country-specific subdomains for their services. The third most common domain is *coolstart.com* which hosts a new-tab page for numerous new-tab-customization extensions. Table III shows the most common extracted candidate URLs.

3) Extracted Key–Value Pairs: `ExtPrivA` activated the extensions' functionality, captured their network traffic and extracted 680,923 key–value pairs sent from 3,904 extensions to 3,280 unique hosts and 6,902 external server endpoints each of which is a combination of a host and a path. The most common host is *www.google-analytics.com* (80,171/680,923 (11.77%) key-value pairs). The high percentage of traffic to Google Analytics indicates its popularity among the extensions for data collection.

To activate an extension's functionality, `ExtPrivA` visited 5.1 candidate URLs on average (1.82 *SD*). The numbers of the unique web page URLs and website domains on which the extensions generated the data traffic are 1,532 and 1,381, respectively. The top website hostnames and domains are listed in Table IX (Appendix D).

4) Extracted Data Flows:

a) Flow Data Types: From the traffic key–value pairs, `ExtPrivA` extracted 1,706 unique data flows for the data types received by 1,002 extensions. Each extension collects 1.7 data types on average (1.04 *SD*). Table IV shows the number of data flows per extension.

The most common data types extracted from the extensions are the URLs and hostnames of the currently visited web pages, which are under the Web History high-level data type. Such data types are privacy-sensitive as they can be easily used to construct a user's web browsing habit. Table VI shows the distribution of the extracted data types over extensions.

b) Flow Receivers: We adopted PurPliance to determine the functionality of data-flow receivers based on extension descriptions and well-known advertising/analytics provider lists. First, rather than using Android package names, we extracted first-party domains as the domains of the privacy policy URL and publisher websites on each extension description web page. Second, we replaced PurPliance's mobile ad and tracking filters with those designed for websites [2] to identify online advertising networks and analytics providers. Third, to improve coverage further, if a host did not fall into these lists, we matched it with the 1Hosts Xtra list [10] to identify online trackers. Finally, if a receiver was not identified by these ad-filtering lists, we classified it as `Other`.

As shown in Table V, the most common receiver types are extensions' own hosts (first parties) and analytics providers. The first-party hosts have a long-tail distribution with 312 unique hosts for 524 flows. The most common first-party and analytics hosts are *bar.maxtrigger.com* (15/524 flows) and *www.google-analytics.com* (76/314 flows), respectively. Online trackers and ad networks are less common than analytics services. The most common tracker and ad network hosts are *sentry.io* (12/166 flows) and *adservice.google.com* (32/51 flows), respectively. The Other hosts that were unidentified by the lists of well-known ad/analytics services have a long-tail distribution, which comprises 163 hosts for 651 flows and includes service hosts such as the Google Cloud Translation end point *translate.googleapis.com*.

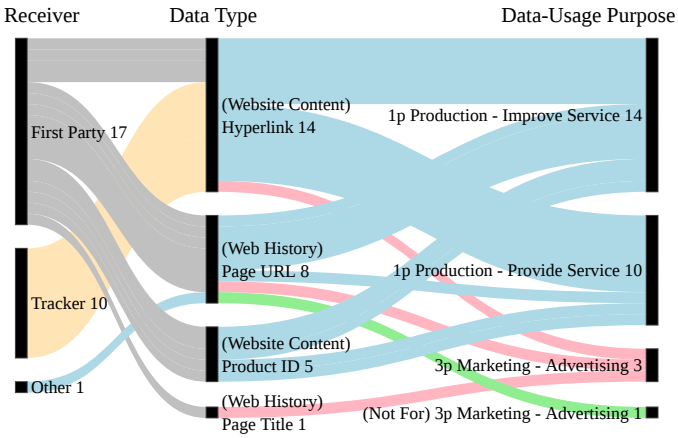

Fig. 5: Purposes of data collection. From the Data Type to Data-Usage Purpose, blue and red lines represent the usage of the first and third parties, respectively. Green lines indicate that a collected data type is not used for third-party purposes. 1p and 3p stand for the first and third parties, respectively. Each label contains the number of instances extracted.

c) Data-Collection Purposes: To understand the purposes of data collection, we identified the privacy statements that were relevant to consistent data flows based on Definitions VI.4 and VI.5, and extracted data-usage purposes from the statements using PurPliance. Of the 1,706 flows extracted in Section VII-C4a, we were able to find 20 privacy statements of 11 sentences with data-usage purposes which were relevant to 21 unique flows of 13 extensions. Since each flow may have multiple purposes, we expanded the flows to 28 flows so that each flow has exactly one usage purpose. The number of flows with specified purposes is not high because only part of privacy statements in a privacy policy specifies purposes (e.g., 25.8% of statements for Android apps [13]) while we even narrowed the statements down to only extension-related ones.

The results show that extensions primarily collected data for improving (14/28 flows) or providing services (10/28 flows) The most common data types are Website Content (Hyperlink and Product ID) and Web History (Page URLs of the currently visiting pages). Notably, three flows of an extension collected the data types for third-party advertising purposes by stating "We may share aggregate information about how our users use *www.valurank.com* or the extension with advertisers, business partners, sponsors, and other third parties" [62]. However, the Web Store's Limited Use policy prohibits any transfer of user data to advertisers [35]. The extracted purposes do not contain the full spectrum of data-usage purposes as when the entire privacy policy is analyzed. Fig. 5 shows the breakdown of the data-usage purposes.

To validate PurPliance's purpose extraction, two authors independently labeled the 11 extracted sentences using the purpose classes in PurPliance's purpose taxonomy. We identified 24 purposes of the 11 sentences and the purpose extraction had 95.00% (19/20) precision and 79.17% (19/24) recall. The precision is high, as PurPliance extraction uses strict rule-based matching, and is comparable to PurPliance's results [12].

Data Type		# Extensions
High-level	Low-level	(Inconsistent/Total)
Web History	Page URL	505/616
	Page Hostname	304/345
	Page Title	53/70
	Total	672/800
Website Content	Hyperlink	149/229
	Product ID	139/215
	Website Text	70/116
	Total	303/472
Location	IP Address	36/53
	Region	11/24
	GPS Coordinate	4/5
	Total	48/69
User Activity	Mouse Click	12/18
	Keystroke Logging	7/15
	Total	14/23
Total		820/1002

TABLE VI: Distribution of the extracted data types and detected inconsistencies. Each row reports # of extensions that have inconsistencies and ones that have data flows extracted.

D. Findings

1) Finding 1: A significant number of extensions fail to fully declare the data types that they collect from users in their privacy disclosures: `ExtPrivA` detected 820 extensions with more than 84.6M users that have a data flow inconsistent with their Dashboard disclosures. These extensions constitute 81.84% (820/1,002) extensions with an extracted data flow. The inconsistent data flows are 75.62% (1,290/1,706) of the total extracted flows. Each extension contains 1.57 inconsistent data flows and 103,190 users on average. This result underscores the incomplete privacy disclosures on the Chrome Web Store that pose high privacy risks to a large number of users.

While `ExtPrivA` detects inconsistencies in various types of extensions, the inconsistent extensions do not spread evenly over extension categories. The most common categories are Productivity and Shopping with 425 and 261 extensions, respectively. These categories tend to collect and use more data to customize and enhance web pages. For example, they include shopping price analytics, new-tab customization and translators. The distribution of the inconsistent extensions over the categories is shown in Fig. 6 and Table XII (Appendix F).

The first-parties and analytics services are the most common receivers: 45.66% (589/1,290) of the inconsistent flows involve data transfer to them (Table V). These two parties have high percentages of the top 2 categories, Productivity (37.70%) and Shopping (68.26%). The first-party is the most common receiver (43.69%) in the Shopping category. Fig. 6 shows the breakdown of data flows by receivers and extension categories.

2) Finding 2: Dashboard disclosures and privacy policies contain contradictory statements: `ExtPrivA` detected 525 pairs of contradictory privacy statements in the privacy policies of 17.22% (360/2,091) extensions that have extension-related statements (Section VII-B2). Each contradiction comprises one positive statement and one negative statement, either or both of which are from the privacy policies. Because Dashboard disclosures are template-based, there are no contradictions when both statements are from the Dashboard disclosures. The distribution of the contradiction types is shown on Table VII.

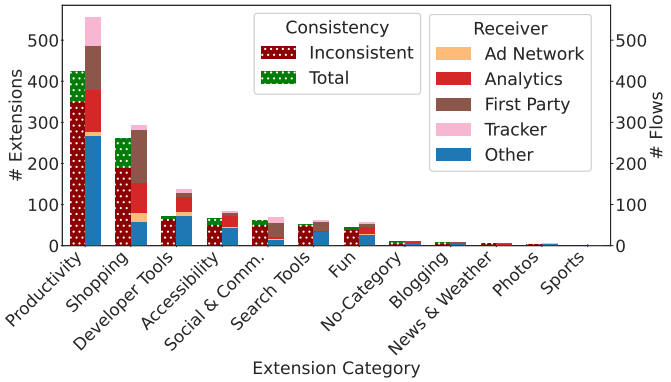

Fig. 6: Distribution of inconsistent extensions and flow-receiver types over extension categories.

Type	Positive Stmt.	Negative Stmt.	# Contradictions
1	Privacy policy	Dashboard	388
2	Dashboard	Privacy policy	73
3	Privacy policy	Privacy policy	64
	Total		525

TABLE VII: Number of contradictory pairs of privacy statements per statement type. *Stmt* stands for a privacy statement.

The most common contradiction type comprises one statement from the Dashboard disclosures that states the non-collection of any data and another statement from the privacy policy that claims the collection of certain data. This type constitutes 73.90% (388/525) of the detected contradictions. Such discrepancies between the privacy policies and Dashboard disclosures are a serious problem and can thus result in the suspension/removal of the extensions from the Store [37].

These extensions had a total of 27.3M users where 16 extensions have more than 100k users and 4 have more than 1M users. For example, AdBlock, that has more than 10M users, declared that it would not collect or use any user data on the Dashboard disclosures but its privacy policy stated that "when the AdBlock extension communicates with AdBlock servers, we receive the computer's IP address" [38]. The policy also mentioned that "after six months we will remove any identifying information such as IP address from our log files and databases," i.e., AdBlock databases record IP addresses.

There are multiple potential causes of the contradictions. First, the manual creation of these policies is error-prone, especially when the authors of the privacy policies and Dashboard disclosures are different. Second, the developers might reduce the number of declared data types in their Dashboard disclosures to reduce the time to publish the extension on the Chrome Web Store. At the same time, the privacy policy tends to contain a comprehensive description of data-collection practices to cover future use-cases.

3) Case Studies: As a case study, the *CapitalOneShopping* [14] extension (8M+ users), using its background

page, sent the full URLs of the currently visiting product pages of shopping websites (like *ebay* and *amazon*), the time of visits and a persistent user ID to its server at *track.capitaloneshopping.com*. The extension automatically collected these pieces of information every time we visited a new product page even without clicking the extension icon. While the information might be used for providing the extension's services, such as offering coupons, it is "browsing-related data" under the definition of the Web History data type and could be easily used to determine the browsing paths of users. Although the extension disclosed the collection of Web Content, it omits the actually-collected Web History from its Dashboard disclosures.

Similarly, the *SearchPreview* extension [54] (100k+ users) declared not to collect or use any user data but the data flows showed that the content of Google search result pages was transferred to the extension's server. The extension used the information to display the previews (thumbnails) of search results on Google search pages. In particular, the full URL of the Google search page and URLs of the search results were transferred to *searchpreview.de*. Since a Google search page URL includes the query term, browser and language information, the collected information is Website Content. However, such data collection was omitted from the Dashboard privacy disclosures.

4) Potential Root Causes: A direct cause of the flow-to-policy inconsistencies is that extensions tend to declare a limited number of user data types that they collect, effectively omitting their data collection practices. In particular, 56.95% (467/820) of the inconsistent extensions disclosed "no user data collection or use" in their Dashboard disclosures while the extensions still collected certain data types. In addition, declaring less collected data types might shorten the Store's security review and time of publishing extensions. For example, 71.57% (33,787/47,207) of extensions disclosed not to "collect or use any user data" while the other 15.97% disclosed to collect only one data type. Furthermore, the most common data types that are collected without declaration is Web History which occurred in 672/1,002 extensions. The extensions collected the URL or title of the currently visited web page while excluding them from the privacy disclosures. Omitting the sensitive browsing history data type might avoid being suspicious to users and increase their installation rate.

Similarly, the extensions commonly used analytics and monitoring libraries but might not fully configure the libraries to limit the data they collect. From the data traffic of extensions, we found that many extensions collect user data for analytic purposes yet failed to disclose them in the privacy disclosures. For example, 14% of the extracted flows were sent to Google Analytics or Sentry analytics libraries [40]. Therefore, we recommend the developers to minimize the data that is collected by the external analytics libraries and services.

E. Evaluation of Detection Performance

We now evaluate the performance of the extraction and consistency analysis. Since our goal is to minimize the false positives, we focus on the evaluation of system precision by verifying the correctness of randomly-selected samples. The verification was done manually by two PhD students with no less than 3 years of experience in user-privacy research. The annotators first agreed on a common annotation scheme/verification workflow, and then worked independently to verify the correctness of the system output. Finally, they held a follow-up meeting to reconcile the differences, if any. Since dynamic analysis cannot exercise all behaviors of an extension while making a ground-truth dataset of privacy policies and flows requires significant effort and time [41, 63], we leave the recall-rate evaluation as future work.

1) Performance of Contradiction Detection: To evaluate the performance of contradiction detection, we verified the pairs of the contradictory privacy statements detected by `ExtPrivA`. The annotators read the corresponding sentences of positive and negative statements to assess whether the extracted statements were indeed contradictory or not. Each privacy statement could be generated from different sentences in the privacy policies where the sentences were slightly different in grammar but expressed the same (non)collection of the same data types. When the sentences were ambiguous due to the lack of context, we traced them back to the extension's privacy-practice disclosures and privacy policies to fully understand the statements. Specifically, for each sentence, we identified the stated data types, receiving entities and whether each type was collected by each entity or not. Finally, we determined contradictions based on Definition VI.3.

The result shows that the detection achieves 91.7% precision. Of the 60 randomly selected statement pairs of 56 extensions, only 5 were false positives. Some of these false positives were due to the lack of cross-sentence analysis such as co-reference resolution. For example, a statement of the *#fastset For Social Media* extension privacy policy applied to another different extension of the same developer but such a mention could only be understood by reading the preceding sentences.

2) Accuracy of Data-Type Extraction: We verified the data types extracted from key-value pairs by attempting to understand the intention of the data traffic from the context which includes the web page being visited, the extension's description and external sources. In particular, we used the Chrome DevTools to obtain other key-values in the data traffic and traced them back to the request-initiated script to identify the data source of the key-values. Inspired by a prior mobile-app traffic analysis [41], we leveraged two properties of key-value pairs to infer the data types: 1) naming conventions indicate the data types of a key-value pair and 2) external knowledge such as the extension description can be a strong indication of data-collection purposes. For example, given *key=regionName* and *value=<city_name>*, the key-value pair likely represents the transfer of the user's geographical location.

We evaluated the extraction's precision on 330 randomly selected samples (30 samples per data type), showing a $\geq$93.3% precision. Since the extraction is based on strict matching rules, a match is likely a correct occurrence of the data

type in a key-value pair. One of the lowest precisions is the Hyperlink data type that indicates collection of the hyperlinks of the currently visited web page, because, in some cases, the links were inserted by the extensions, not the original web page content. The precision for each data type is provided in Table XI of Appendix E.

3) Accuracy of End-to-end (In)consistency Detection: Given a detected inconsistency, we attempted to reproduce the result by using only built-in tools of the browser to evaluate the end-to-end performance of the inconsistency detection. We installed the extension on a clean browser instance and captured the network traffic of the tab and the background pages via Chrome DevTools to reproduce the data-collection behavior of the extension. We read the privacy disclosures and verified the correctness of the privacy-statement extraction. Finally, we assessed whether the data-collection behavior violated the privacy disclosures or not.

Inconsistencies were detected with a precision of 85%. We were able to reproduce 51 of the randomly selected 60 detected inconsistencies of 59 extensions. Manual verification of each inconsistency took 15 minutes on average, so the two annotators spent 30 hours in total. Most of the false positives were due to a non-data-collection callback function of an extension script that was included as one of the initiators of the network traffic. For example, an extension installed an HTTP-request event handler to check the occurrence of a URL pattern in the HTTP requests even if it did not do any data collection.

VIII. Limitations and Future Work

While we aim to minimize false positives, `ExtPrivA` has some limitations in detecting inconsistencies. Although it supports widely used data types, `ExtPrivA` still does not support five *context-sensitive* data types including sensitive personal data. It uses simple heuristics to extract extension-related sentences and analyzes the policies on a sentence basis. Finally, it cannot trigger the data-collection behavior of the extensions that require sign-in. Discussed below are `ExtPrivA`'s limitations and our future work.

a) Applicability of Analysis Techniques: Detection of the inconsistencies between the privacy disclosures and the execution of extensions is critically important for all stakeholders in the web browser ecosystem. The removal of an extension from a marketplace would cause a substantial loss to the developers. The deviation of actual data collection and usage from the stated practices is not expected by the users and causes a loss of users' trust in the web browsers' privacy protection. Finally, the extension stores can leverage the analysis techniques/tools to audit and detect privacy breaches.

As an end-to-end framework, `ExtPrivA` can be easily integrated into the Chrome Web Store's vetting process or an IDE to help developers verify that their extensions operate consistently with their stated privacy policies. As extensions may use third-party libraries, it is hard and expensive for developers to check their (in)consistencies manually. A benefit of dynamic analysis is its ability to provide the inputs to reproduce the inconsistencies and facilitate the debugging process. Furthermore, even with an extensive vetting process, our results have shown the Chrome Web Store still misses the extensions that provide misleading privacy-practice disclosures. We plan to communicate our findings to the developers and Chrome Web Store to help them fix the inconsistencies in their extensions.

b) Analysis of Non-Chrome Browser Extensions: Since most of the `ExtPrivA` pipeline utilizes black-box analysis methods, it can be extended to detect the inconsistencies of non-Chrome web browsers — such as Firefox and Safari — extensions. First, the free-form privacy policies of non-Chrome browser extensions are not different from those of Chrome-based browsers. Extracting privacy policies only needs to handle the differences in the extension description web pages in different extension stores. Similarly, while the internal API of Chrome that extracts the initiators of network traffic does not automatically translate to Firefox and Safari, these browsers have equivalent APIs, such as the traffic initiator of Firefox network analysis [18]. Finally, the browser automation tool [49] of `ExtPrivA`'s testbed already supports multiple browsers. We leave the multi-browser support for Firefox and Safari extensions, which accounted for only 18% of the desktop browser market share (less than 10% each) [61], as our future work.

c) Detection of Contradictions in Privacy Disclosures: The existence of contradictions between the privacy policies and the Dashboard disclosures highlights the inadequacy of manual checking for the (in)consistencies of privacy disclosures. However, `ExtPrivA` cannot fully analyze the privacy policies due to the inherent limitation in determining the scope of policy statements, i.e., whether a sentence is about browser extensions or not. Some recent approaches [7, 13] still analyze privacy policies at the sentence level due to the lack of a holistic analysis of the entire documents. While solving this problem requires advances in both NLP and privacy-policy analysis, the recent ML models specialized in privacy policies [5] will help detect the contradictions more effectively.

d) Compliance of Data-usage Purposes: To comply with the Chrome Web Store developer policies, extension developers must agree with the Limited Use policy that prohibits the data types collected by extensions from being used or transferred to advertisers for advertising purposes [35]. Therefore, one can check the compliance with this policy by analyzing the data-usage purposes of the extensions. However, since determining the purposes of data collection without server-side information is still challenging [13, 41] and there is no prior work on analyzing the purposes of data usage for browser extensions, we leave this analysis as future work.

e) Dynamic Analysis and Log-in Extensions: Due to the limitations of dynamic analysis, `ExtPrivA` cannot exercise all execution paths of an extension to generate the transfer of all possible data types. Like software testing, it is non-trivial to generate inputs to completely activate all functionalities

of a sophisticated extension. Furthermore, supporting extensions that require login is challenging because the account registration process is complex and services frequently deploy bot-prevention to avoid automated account creation and detect fake identities. While recent techniques can generate input text for login Android apps [30], automatically performing account registration and login on web apps requires further advances in web page analysis and NLP. With improvements from the research in extension/JavaScript dynamic analysis [16, 43] and input generation [30], `ExtPrivA` can cover more data flows of each extension to detect more inconsistencies.

f) Extraction of Context-Sensitive Data Types: These types can be extracted from data flows by analyzing the semantics of data-collection contexts. In the future, we would like to support the data types by analyzing input fields (e.g., <*input*> attribute *type="password"* indicates password (authentication information)), performing static analysis on extension code (e.g., registration prompts in extensions' pop-up pages) and analyzing extension behavior after signing into websites.

IX. Related Work

A. Detection of Privacy Leakage

Researchers proposed various ways of examining the execution of web browser extensions to detect their privacy leakage, i.e., unexpected execution of privileged API and the flows of sensitive data to servers or disk storage. Hulk [43] introduced two ways to trigger malicious behavior, specially structured web pages called *honeypages*, and an event-handler triggering fuzzer, to detect affiliated fraud, credential theft, ad injection or replacement, and social network abuse. Starov *et al.* [60] analyzed and decoded network traffic of extensions to find the leakage of users' sensitive data, such as browsing history and search-engine queries. However, they did not determine whether such data collection violates the extensions' privacy policies or not.

Other researchers focus on JavaScript analysis techniques. ExtensionGuard [15], Mystique [16] and JTaint [64] presented JavaScript taint analysis schemes to detect the leakage of sensitive information in the data flows during the extension execution. Somé [59] analyzed the vulnerabilities in message passing interfaces of web extensions that can be exploited by web applications to access privileged browser APIs and sensitive user information. DoubleX [9] developed static analysis techniques to detect vulnerable internal data flows of an extension that can be exploited by attackers.

Another thread of research developed ML-based classifiers to classify whether a certain extension behavior is malicious or not. Aggarwal *et al.* [3] created a classifier based on Recurrent Neural Network (RNN) to classify whether a sequence of API calls indicates the stealing of sensitive user information or not. Zhao *et al.* [65, 66] attempted to determine the legitimacy of extensions' data flows based on the main functionality provided by the extensions.

None of the prior studies has analyzed the privacy policies of browser extensions to detect flow-to-policy inconsistencies. They rely on an expert analysis of the execution and network logs to determine whether the detected sensitive data transfer is malicious or not [16], but such a manual analysis greatly limits the scalability of the detection and the types of data leakage that can be detected. Furthermore, prior network-traffic-based analyses [43, 60] did not consider the receivers of the data traffic of web browser extensions, and hence may suffer from false positives where an extension sends user data to its servers to provide its functionality, not for malicious purposes. `ExtPrivA` avoids these limitations by analyzing privacy-practice disclosures and extracting data types/receivers in data traffic of extensions to determine the legitimacy of their data practices.

B. Analysis of Privacy Statements

Recently, researchers analyzed the statements in privacy policies of mobile apps and online services. PI-Extract [12] extracted fine-grained data types and collection/sharing actions performed thereon in website privacy policies. PolicyLint [7] detected contradictory policy statements in the privacy policies of mobile apps. However, none of these addressed the privacy statements and policies of browser extensions that require platform-specific interpretation and analysis.

C. Flow-to-policy Consistency Analysis

Inconsistencies between the privacy policies and the actual data collection of mobile apps have been the subject of recent research. Zimmeck *et al.* [67] conducted static analysis on Android apps to detect inconsistencies between their collected data types with those stated in the apps' policies. PoliCheck [8] improved the consistency analysis by considering the receivers in data flows from mobile apps to external receivers. PurPliance [13] modeled data-usage purposes to detect the inconsistencies in the data-usage purposes between the stated privacy statements and actual data collection of Android apps. However, none of the prior works have analyzed the flow-to-policy inconsistencies of browser extensions whose execution model is fundamentally different from mobile apps.

X. Conclusion

We have presented a novel system, `ExtPrivA`, to detect inconsistencies between the privacy disclosures and the actual data collection of browser extensions. `ExtPrivA` is an end-to-end system that performs a fine-grained analysis of data collection of browser extensions to detect their flow-to-policy inconsistencies. We used `ExtPrivA` to conduct a large-scale study of 47,207 extensions that provide Dashboard disclosures in the Chrome Web Store. Of these, `ExtPrivA` detected 1,290 inconsistent data flows of 820 extensions with more than 84.6M users. `ExtPrivA` has also detected 360 extensions that contain 525 pairs of contradictory privacy statements in their Dashboard disclosures and privacy policies. These findings highlight critical issues in the privacy notices of web extensions that may mislead users about their privacy practices. These findings will hopefully help all the involved parties remove/minimize such inconsistencies and enhance the users' privacy in the web browser ecosystem.

Acknowledgements

This work was supported in part by the ONR under Grant No. N00014-22-1-2622 and the ARO under Grant No. W911NF-21-1-0057.

References

[1] G. Acar, S. Englehardt, and A. Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. *Proceedings on Privacy Enhancing Technologies*, (4):220–238, 2020.

[2] AdGuard. AdGuard Ad Filters. 2022. URL: https://kb.adguard.com/en/general/adguard-ad-filters (visited on 07/24/2022).

[3] A. Aggarwal, B. Viswanath, L. Zhang, S. Kumar, A. Shah, and P. Kumaraguru. I spy with my little eye: analysis and detection of spying browser extensions. In *IEEE European Symposium on Security and Privacy (EuroS P)*, pages 47–61, 2018.

[4] E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94, 2000.

[5] W. Ahmad, J. Chi, T. Le, T. Norton, Y. Tian, and K.-W. Chang. Intent classification and slot filling for privacy policies. In *ACL*, 2021.

[6] E. AI. spaCy · industrial-strength natural language processing in python. 2020. URL: https://spacy.io/ (visited on 01/08/2021).

[7] 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 *28th USENIX Security Symposium (USENIX Security 19)*, pages 585–602, 2019.

[8] B. Andow, S. Y. Mahmud, J. Whitaker, W. Enck, B. Reaves, K. Singh, and S. Egelman. Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with POLICHECK. In *29th USENIX Security Symposium*, page 18, 2020.

[9] 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 ACM SIGSAC Conference on Computer and Communications Security*, page 16, 2021.

[10] badmojr. 1Hosts - World's most advanced general-purpose DNS filter-blocklists. 2022. URL: https://o0.pages.dev/ (visited on 07/17/2022).

[11] Berstend. Puppeteer-extra-plugin-stealth. npm. 2021. URL: https://www.npmjs.com/package/puppeteer-extra-plugin-stealth (visited on 06/30/2021).

[12] D. Bui, K. G. Shin, J.-M. Choi, and J. Shin. Automated extraction and presentation of data practices in privacy policies. *Proceedings on Privacy Enhancing Technologies*, (2), 2021.

[13] D. Bui, Y. Yao, K. G. Shin, J.-M. Choi, and J. Shin. Consistency analysis of data-usage purposes in mobile apps. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2021.

[14] capitaloneshopping.com. Capital one shopping: add to chrome for free. 2022. URL: https://chrome.google.com/webstore/detail/capital-one-shopping-add/nenlahapcbofgnanklpelkaejcehkggg (visited on 03/30/2022).

[15] W. Chang and S. Chen. ExtensionGuard: towards runtime browser extension information leakage detection. In *2016 IEEE Conference on Communications and Network Security (CNS)*, pages 154–162, 2016.

[16] Q. Chen and A. Kapravelos. Mystique: uncovering information leakage from browser extensions. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1687–1700, 2018.

[17] Chrome. Content scripts. Chrome Developers. 2019. URL: https://developer.chrome.com/docs/extensions/mv2/content_scripts/ (visited on 09/20/2021).

[18] M. Conley. These weeks in firefox: issue 70. Firefox Nightly News. 2020. URL: https://blog.nightly.mozilla.org/2020/03/03/these-weeks-in-firefox-issue-70 (visited on 03/26/2022).

[19] Detectify Labs. Chrome extensions - AKA total absence of privacy. Detectify Labs. 2015. URL: https://labs.detectify.com/2015/11/19/chrome-extensions-aka-total-absence-of-privacy/ (visited on 05/16/2021).

[20] C. Developers. Chrome DevTools protocol - network.initiator. 2022. URL: https://chromedevtools.github.io/devtools-protocol/tot/Network/#type-Initiator (visited on 03/24/2022).

[21] J. Freeman. What is JSON? a better format for data exchange. InfoWorld. 2019. URL: https://www.infoworld.com/article/3222851/what-is-json-a-better-format-for-data-exchange.html (visited on 03/15/2022).

[22] Google. Catapult performance tools. 2021. URL: https://chromium.googlesource.com/catapult/ (visited on 02/26/2021).

[23] Google. Changes to cross-origin requests in chrome extension content scripts. 2020. URL: https://www.chromium.org/Home/chromium-security/extension-content-script-fetches (visited on 09/20/2021).

[24] Google. Chrome DevTools protocol. 2021. URL: https://chromedevtools.github.io/devtools-protocol/ (visited on 09/18/2021).

[25] Google. Extension match patterns. Chrome Developers. 2021. URL: https://developer.chrome.com/docs/extensions/mv3/match_patterns/ (visited on 09/16/2021).

[26] Google. Manifest - key. Chrome Developers. 2018. URL: https://developer.chrome.com/docs/extensions/mv3/manifest/key/ (visited on 10/11/2021).

[27] Google. Overview of manifest v3. Chrome Developers. 2020. URL: https://developer.chrome.com/docs/

extensions / mv3 / intro / mv3 - overview/ (visited on 09/20/2021).

[28] Google Chrome DevTools Team. Puppeteer Tools for Web Developers. 2020. URL: https://pptr.dev/ (visited on 02/05/2020).

[29] Google Chromium. Catapult performance tools. 2021. URL: https://chromium.googlesource.com/catapult/ (visited on 02/26/2021).

[30] Y. He, L. Zhang, Z. Yang, Y. Cao, K. Lian, S. Li, W. Yang, Z. Zhang, M. Yang, Y. Zhang, and H. Duan. TextExerciser: feedback-driven text input exercising for android applications. In *IEEE Symposium on Security and Privacy (SP)*, pages 1071–1087, 2020.

[31] J. Huang, P. O. Meredith, and G. Rosu. Maximal sound predictive race detection with control flow abstraction. *ACM SIGPLAN Notices*, 49(6):337–348, 2014.

[32] Ilya Sergey. What does it mean for a program analysis to be sound? SIGPLAN Blog. 2019. URL: https://blog.sigplan.org/2019/08/07/what-does-it-mean-for-a-program-analysis-to-be-sound/ (visited on 10/03/2020).

[33] A. Inc. App store review guidelines - apple developer. 2021. URL: https://developer.apple.com/app-store/review/guidelines/#privacy (visited on 05/22/2021).

[34] G. Inc. Chrome web store sitemap. 2021. URL: https://chrome.google.com/webstore/sitemap (visited on 05/28/2021).

[35] G. Inc. Developer program policies. Chrome Developers. 2020. URL: https://developer.chrome.com/docs/webstore/program_policies/ (visited on 05/22/2021).

[36] G. Inc. Transparent privacy practices for chrome extensions. Chromium Blog. 2020. URL: https://blog.chromium.org/2020/11/transparent-privacy-practices.html (visited on 05/13/2021).

[37] G. Inc. Updated privacy policy & secure handling requirements. Chrome Developers. 2020. URL: https://developer.chrome.com/docs/webstore/user_data/ (visited on 05/20/2021).

[38] A. Inc. AdBlock privacy policy. 2022. URL: https://web.archive.org/web/20220318225253/https://getadblock.com/en/privacy/ (visited on 03/18/2022).

[39] D. Inc. Swarm mode. Docker Documentation. 2021. URL: https://docs.docker.com/engine/swarm/ (visited on 08/01/2021).

[40] F. S. Inc. Application monitoring and error tracking software. Sentry. 2022. URL: https://sentry.io/welcome/ (visited on 03/20/2022).

[41] H. Jin, M. Liu, K. Dodhia, Y. Li, G. Srivastava, M. Fredrikson, Y. Agarwal, and J. I. Hong. Why Are They Collecting My Data?: Inferring the Purposes of Network Traffic in Mobile Apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):173, 2018.

[42] J. Jueckstock, S. Sarker, P. Snyder, A. Beggs, P. Papadopoulos, M. Varvello, B. Livshits, and A. Kapravelos. Towards Realistic and Reproducible Web Crawl Measurements. In *Proceedings of the Web Conference*, pages 80–91, 2021.

[43] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 641–654, 2014.

[44] Kuba Suder. Meet Safari Web Extensions. WWDC NOTES. 2020. URL: https://www.wwdcnotes.com/notes/wwdc20/10665 (visited on 09/20/2021).

[45] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: a research-oriented top sites ranking hardened against manipulation. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, 2019.

[46] Q. Li. Literature survey: domain adaptation algorithms for natural language processing. *Department of Computer Science The Graduate Center, The City University of New York*:8–10, 2012.

[47] E. McCallister, T. Grance, and K. Scarfone. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII). Technical report NIST Special Publication (SP) 800-122, National Institute of Standards and Technology, 2010.

[48] Microsoft. Docker | playwright python. 2021. URL: https://playwright.dev/python/docs/docker/ (visited on 10/05/2021).

[49] Microsoft. Microsoft playwright python. 2020. URL: https://github.com/microsoft/playwright-python (visited on 12/18/2020).

[50] Mozilla. Add-on policies. Firefox Extension Workshop. 2019. URL: https://extensionworkshop.com/documentation/publish/add-on-policies/ (visited on 05/22/2021).

[51] Mozilla. Origin - HTTP request header. 2021. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin (visited on 09/27/2021).

[52] Mozilla. WebExtensions - MozillaWiki. 2018. URL: https://wiki.mozilla.org/WebExtensions (visited on 05/22/2021).

[53] R. Perrotta and F. Hao. Botnet in the browser: understanding threats caused by malicious browser extensions. *IEEE Security Privacy*, 16(4):66–81, 2018.

[54] searchpreview.de. SearchPreview. 2021. URL: https://chrome.google.com/webstore/detail/searchpreview/hcjdanpjacpeeppdjkppebobilhaglfo (visited on 08/01/2021).

[55] C. O. Shopping. Capital One Shopping Privacy Policy. 2022. URL: https://web.archive.org/web/20220724073640/https://capitaloneshopping.com/our-terms/privacy-policy.

[56] J. Sissel. Xdotool - x11 automation tool. 2021. URL: https://github.com/jordansissel/xdotool (visited on 09/26/2021).

[57] Y. Smaragdakis, J. Evans, C. Sadowski, J. Yi, and C. Flanagan. Sound predictive race detection in polynomial time. In *Proceedings of the 39th annual ACM*

SIGPLAN-SIGACT symposium on Principles of programming languages, pages 387–400, 2012.

[58] K. Solomos, P. Ilia, S. Karami, N. Nikiforakis, and J. Polakis. The dangers of human touch: fingerprinting browser extensions through user actions. In *Proceedings of the 31st USENIX Security Symposium*, 2022.

[59] D. F. Somé. EmPoWeb: empowering web applications with browser extensions. In *IEEE Symposium on Security and Privacy (SP)*, pages 227–245, 2019.

[60] O. Starov and N. Nikiforakis. Extended tracking powers: measuring the privacy diffusion enabled by browser extensions. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1481–1490, 2017.

[61] Statista. Desktop internet browser market share 2015-2021. Statista. 2022. URL: https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/ (visited on 03/16/2022).

[62] Valurank. Valurank Privacy Policy. 2022. URL: https://web.archive.org/web/20220530124253/https://valurank.com/privacypolicy.

[63] S. Wilson, F. Schaub, A. A. Dara, F. Liu, S. Cherivirala, P. Giovanni Leon, M. Schaarup Andersen, S. Zimmeck, K. M. Sathyendra, N. C. Russell, T. B. Norton, E. Hovy, J. Reidenberg, and N. Sadeh. The Creation and Analysis of a Website Privacy Policy Corpus. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1330–1340, 2016.

[64] M. Xie, J. Fu, J. He, C. Luo, and G. Peng. JTaint: finding privacy-leakage in chrome extensions. In *Information Security and Privacy*, pages 563–583, 2020.

[65] Y. Zhao, L. He, Z. Li, L. Yang, H. Dong, C. Li, and Y. Wang. Large-scale detection of privacy leaks for BAT browsers extensions in china. In *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 57–64, 2019.

[66] Y. Zhao, L. Yang, Z. Li, L. He, and Y. Zhang. Privacy model: detect privacy leakage for chinese browser extensions. *IEEE Access*, 2021.

[67] S. Zimmeck, Z. Wang, L. Zou, R. Iyengar, B. Liu, F. Schaub, S. Wilson, N. Sadeh, S. M. Bellovin, and J. Reidenberg. Automated Analysis of Privacy Requirements for Mobile Apps. In *Network and Distributed System Security Symposium*, 2017.

APPENDIX A
LIST OF TESTING URLS

The testing URLs used by `ExtPrivA` to generate candidate URLs are listed in Table VIII.

APPENDIX B
PRIVACY POLICY CRAWLING

To obtain the privacy policy documents, for each extension, `ExtPrivA` extracts the privacy policy URL from the extension's overview page. We use a clean instance of Chrome browser that fully executes JavaScript to extract privacy policies of dynamic web pages. `ExtPrivA` then extracts plain text from the HTML by using PolicyLint preprocessing tool [7]. The plain text is then segmented into sentences by using a transformer-based neural model *en_core_web_trf* included in the Spacy NLP library [6].

Category	URL
Search	https://www.google.com/search?q=statistics&hl=en https://www.bing.com/search?q=statistics
Shopping	https://www.amazon.com/gp/product/B085TFF7M1 https://www.amazon.com/dp/B07G7T3M6C https://www.ebay.com/itm/323879722346 https://www.aliexpress.com/item/4000901174719.html

TABLE VIII: Testing URLs for generating candidate URLs.

	Domain	# Exts.
1	amazon.com	2,243
2	google.com	1,924
3	ebay.com	1,899
4	aliexpress.com	1,845
5	bing.com	1,712
6	coolstart.com	268
7	youtube.com	34
8	taobao.com	26
9	linkedin.com	26
10	facebook.com	25
	Total	3,904

(a) Site domains.

	Hostname	# Exts.
1	www.amazon.com	2,238
2	www.ebay.com	1,899
3	www.aliexpress.com	1,844
4	www.google.com	1,817
5	www.bing.com	1,712
6	mail.google.com	86
7	www.youtube.com	33
8	inbox.google.com	30
9	accounts.google.com	27
10	www.linkedin.com	25
	Total	3,904

(b) Site hostnames.

TABLE IX: The top website domains/hostnames and the number of extensions that generated the network traffic. One website can be tested on multiple extensions.

APPENDIX C
LIST OF DATA TYPES ON CHROME WEB STORE

The list of data types used by the Chrome Web Store is shown in Table X and Fig. 7.

APPENDIX D
TESTED WEBSITES

The top website hostnames and domains that generated the network traffic are listed in Table IX.

APPENDIX E
PRECISION OF DATA TYPE EXTRACTION

The precision of data-type extraction is shown in Table XI.

APPENDIX F
DISTRIBUTION OF INCONSISTENT EXTENSIONS

Table XII shows the distribution of inconsistent extensions.

	Data Type	Example
1	Personally identifiable info.	Name, address, email address, age, identification number
2	Health information	Heart rate data, medical history, symptoms, diagnoses, procedures
3	Financial and payment info.	Transactions, credit card numbers, credit ratings, financial statements, payment history
4	Authentication information	Passwords, credentials, security question, personal identification number (PIN)
5	Personal communications	Emails, text or chat messages, social media posts, conference calls
6	Location	Region, IP address, GPS coordinates, information about things near the user's device
7	Web history	The list of web pages a user has visited, browsing-related data such as page title and time of visit
8	User activity	Network monitoring, clicks, mouse position, scroll, keystroke logging
9	Website content	Text, images, sounds, videos, hyperlinks

TABLE X: List of data-types and examples specified by the Chrome Web Store policies [36].

Data	# Flows	# Samples	Precision (%)
Page URL	7,262	30	100.00
Page Hostname	2,256	30	100.00
Product ID	1,302	30	100.00
Website Text	1,054	30	100.00
Hyperlink	786	30	93.33
Region	404	30	93.33
IP Address	396	30	100.00
Page Title	279	30	96.67
Mouse Click	133	30	100.00
GPS Coordinate	68	30	100.00
Keystroke Logging	59	30	100.00
Overall	13,999	330	99.37

TABLE XI: Precision of data-type extraction in data flows. The overall precision is a weighted average by the number of flows per data type.

Category	# Inconsistencies	% Inconsistencies	# Extensions
Productivity	557	43.18	351
Shopping	293	22.71	190
Developer Tools	138	10.70	66
Accessibility	84	6.51	52
Search Tools	62	4.81	51
Social & Communication	68	5.27	48
Fun	56	4.34	40
News & Weather	6	0.47	6
No-Category	10	0.78	6
Blogging	9	0.70	5
Photos	6	0.47	4
Sports	1	0.08	1
Total	1290	100.00	820

TABLE XII: Distribution of detected inconsistent extensions over category.

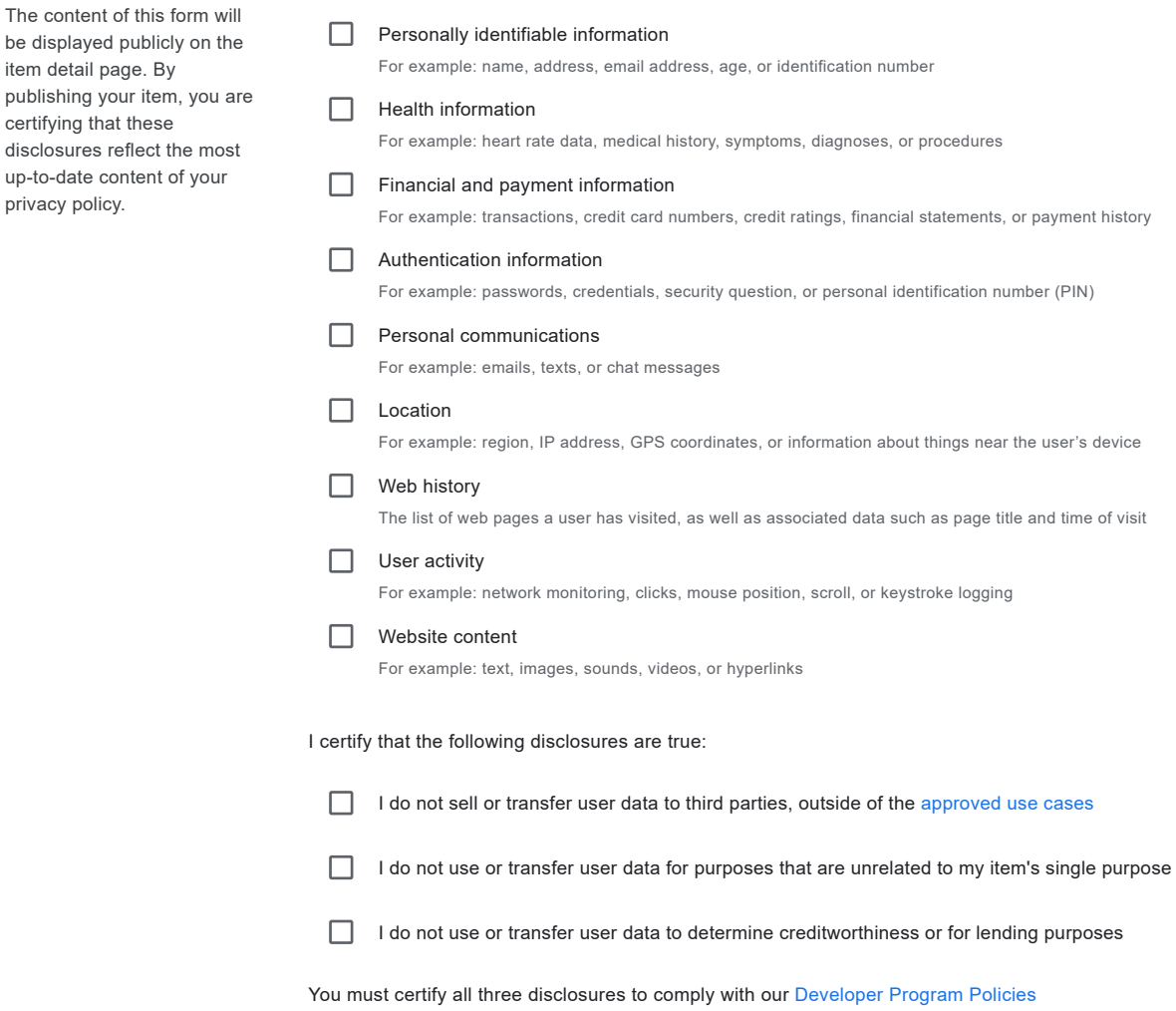

Fig. 7: Privacy-practice declaration on the Chrome Developer Dashboard.