

Essential or Excessive? MINDAEXT: Measuring Data Minimization Practices among Browser Extensions

Yuxi Ling*, Yun Hao*, Yuyan Wang[‡], Kailong Wang^{§§}, Guangdong Bai[†] and Jin Song Dong*

* National University of Singapore

[†] University of Queensland

[‡] Fujian Normal University

[§] Huazhong University of Science and Technology

Abstract—Since browser extensions are prevalingly executed in the background to enable extra functionalities and enhance the user experience for web browsers, the potential over-collection of personal data beyond the necessity for given purposes is always ignored by ordinary users. Existing privacy regulations, such as the principle of *Data Minimization* in GDPR, have provided the criteria that only directly *relevant* and *necessary* data for specified *purposes* should be collected. Various tools have made efforts to examine the compliance of data minimization and its equivalent in different application domains. To our knowledge, in the area of browser extensions, there is still a gap between the general data minimization principle and precisely defined extension behaviors. We propose MINDAEXT, a framework that takes one step further to automatically examine end-to-end *data minimization* practices in browser extensions by description text analysis and hybrid program analysis techniques.

In our large-scale measurement, covering around 200K extensions collected in October 2023, we find that 38.0% of extensions are likely to collect private user data outside their essential functionality scopes. They are distributed across all categories, exhibiting distinct patterns of the target data types. Our evaluation shows that MINDAEXT can detect the data over-collection with a precision of 74.3%.

I. INTRODUCTION

Recent privacy regulations, such as General Data Protection Regulation (GDPR) [43] in the EU and the California Privacy Rights Act (CPRA) [30] in the US, have defined a set of principles on personal data processing to provide a legal basis. In particular, *data minimization* is the principle against the excessive data processing issue, requiring the collection of personal information to be “*directly relevant and necessary to accomplish a specified purpose*”.

In response to the data minimization principle, major extension platforms (e.g., Chrome and Firefox) have enacted to eliminate excessive data collection by requiring developers to access “*the least amount of data*” [16]. Unfortunately, this requirement is hardly sufficient and practical to ensure compliance from them, as there is a lack of clear and actionable steps throughout the compliance auditing process. In

this context, it is necessary to give an end-to-end analysis technique that analyzes the application directly and determines whether excessive personal data collection exists, required by *data minimization* principle. However, the boundary of data minimization is vague, and it is hard to define precisely which personal data is *relevant* and *necessary* to the purpose/functionality of browser extensions. Some attempts [8], [14], [37], [13], [15], [1] have been published to give some steps forward, defined or examined data minimization practices in various application scenarios, such as Mobile Applications [1], Trigger-Action Platform [8], and etc. But none of these works provide an end-to-end solution to examine the compliance of *data minimization* in browser extensions.

Challenges and contributions. Analyzing the compliance of the data minimization principle among browser extensions poses several challenges.

The first challenge is determining the set of minimized personal data essential to each extension’s purpose/functionality. Due to the ambiguity in the definition of minimization, POLICYCOMP [1] first attempts to automate the identification of minimized personal data for general applications in various purposes/functionality. There is still no objective and well-recognized standard to draw a clear boundary between the essential and excessive data for a particular extension.

The Second challenge is identifying the set of personal data collected by extensions. Both static and dynamic analysis, like data flow and network traffic analysis, have limitations in large-scale or precise studies [49]. The precision is limited by consensual challenges, such as the path explosion in data flow analysis and data identification in network traffic analysis.

To meet these challenges, we mainly contribute to the following aspects:

- **An end-to-end automatic approach for data minimization practice analysis.** We propose an automatic end-to-end analysis approach, MINDAEXT, which consists of two components: a *minimized data inferer* and a *collected data analyzer*. *Minimized data inferer* extracts a set of minimized personal data (MPD) essential based on the purpose while *collected data analyzer* extracts a set of

^{§§}The main work from Kailong Wang was done when he worked at National University of Singapore

collected personal data (CPD) for each extension in practice. By comparing the difference between MPD set and CPD set, it flags the excessive data the extension collects. MINDAEXT provides a reference in privacy law compliance auditing, instead of legality judgment.

- **A practical definition to data minimization.** We leverage a multi-source approach to identify an extension’s MPD set based on extension description and references from counterparts/similar extensions. In the description analysis, we compare recent popular NLP models (i.e. LSTM, BERT, GPT-4.0, and GPT-3.5) and choose the best performer, BERT, to identify MPD from description texts. In counterpart reference, we inherit the core idea from POLICYCOMP [1] that the common data types collected by most counterpart extensions have a higher possibility of being MPD. We identify counterpart extensions from description text similarity. Based on a threshold, we extract the common collected data types of these counterparts to MPD set. A final MPD set is the union set of these two sources.
- **A manually labeled corpus** For the analysis of descriptions, we construct a corpus MINDA-128 [33] with 2421 sentences manually labeled by 12 data types. This corpus is used to train NLP models to identify data types mentioned by the description text and examine the accuracy of GPT-3.5 in our scenario.
- **A large-scale measurement.** We deploy MINDAEXT and present a large-scale data minimization practices analysis on 142,756 extensions available in the Chrome Web Store and Firefox Browser Add-ons collected in October 2023. Our results reveal the status quo of the data minimization practices among mainstream browser extensions in terms of the distribution among different categories, excessively collected data types, code features, etc. We found that 38.0% of the examined extensions are likely to be non-compliant with GDPR’s data minimization principle.

Ethical considerations. Any output from MINDAEXT only provides a reference to lawyers to help audit the compliance of data minimization. Because interpretation of *Data Minimization* from developers, service providers, and even platform operators could be inconsistent and subjective. The judgment of compliance with data minimization must involve lawyers’ efforts. Furthermore, we partially anonymize the results to avoid any potential legal disputes with service providers and platform operators.

Roadmap. This paper is organized as follows. Sec. II introduces related works. Sec. III shows a toy example and formal definition. Sec. IV gives an overview design of MINDAEXT, while Sec. V and Sec. VI detail the methodology of constructing MPD and CPD set. Sec. VII evaluates MINDAEXT and presents the result of large-scale analysis. From Sec. VIII to Sec. XI are the discussion and conclusions.

II. BACKGROUND AND RELATED WORK

In this section, we review the state-of-the-art works in analyzing data minimization regulatory compliance and privacy analysis for web and mobile applications.

A. Analysis of Privacy Compliance

Many efforts have been put into privacy analysis in web applications [34], [28], [23] and Android applications [46], [39], [20], [21], [45], [11]. In more detail, Slavin et al. [39] design and implement a privacy violation checking framework to semi-automatically check the inconsistency between the privacy policy and actual practice of Android apps. Yu et al. [46] develop a framework named PPChecker, analyzing the trustworthiness of privacy policies by NLP and program analysis to check the incorrectness, incompleteness and inconsistency in privacy policies. Zimmeck et al. [49] define private data practices and achieve a large-scale privacy compliance analysis in mobile apps. Liu et al. [26] investigate privacy compliance among analytics libraries using static and dynamic analysis. Nguyen et al. [29] reveal that more than 20K apps send out personal user data before or without consent. More recently, Wang et al. [44] present a static analysis tool, APER, to detect asynchronous runtime permission issues in Android Apps, and investigate the status quo of those permission issues in the Google Play Store.

Due to the current popularity of browser extensions, their privacy issues have been brought to the public horizon. Ling et al. [25] and Bui et al. [6] conduct similar works in parallel, studying the compliance between the privacy statements and actual practices of browser extensions at scale. Compared to our work, their works mainly focus on checking the consistency between privacy policy and actual behavior without touching on specific data principles such as *data minimization*.

B. Analysis of Data Minimization Compliance

With increased emphasis on privacy, many works have been targeting different aspects of privacy compliance, such as that in privacy policy [4], [42], [7] and privacy-sensitive practices [47], [48], [2]. Since GDPR defined the principle of *data minimization*, several works [5], [1], [3], [35], [38], [32] have targeted for defining criteria for data minimization in a wide range, covering from formal definition to complex application design. In particular, Basin et al. [5] propose a formal method to audit a data collection’s adherence to its purpose. Instead of defining the necessity of data, Basin et al. solve it in a reverse way by determining unnecessary data: check whether data that has been collected is used. If not, it is unnecessary data and violates data minimization. Chen et al. [8] propose *minTAP*, an automatic minimized data checker, to reduce the over-privilege in function attributes. The core insight is similar to checking unnecessary data: if the absence of data does not affect the normal execution of the program, it violates data minimization. Any absence of minimized data could make certain functions not being completed. However, the approach proposed by *minTAP* is not portable to a general application, such as mobile applications and browser extensions, because

TABLE I
DATA TYPES AND DESCRIPTIONS

Data Types	Abbr.	Description
User Profile Information	UPI	personally identifiable information (e.g., Email)
Health Information	HI	health condition and medical records
Financial and Payment Information	FPI	payment information and records
Authentication Information	AI	account credentials
Personal Communication Information	PCI	individual communications with other parties
Location Information	LI	geolocation location in any form (e.g., GPS)
Web History Information	WHI	list of web pages a user has visited
User Activity Information	UAI	interactions with the browser (e.g., clicks)
Website Content Information	WCI	rendered content from websites
Device Information	DI	hardware settings/configurations
Privacy-related Settings	PS	privacy-related browser configurations
File System	FS	local file system

TABLE II
IDENTIFIED DATA COLLECTION FOR *Ext A*

Sources		Data Types
MPD		website content, user activity, authentication information, personal information and privacy-related settings
CPD	Static Features	user activity, location information, and website content
	Dynamic Features	authentication information, personal information, financial and payment information(optional), website content

it is hard to determine whether the lack of certain data affects the functionality of extensions.

Zhou et al. [1] propose POLICYCOMP and take the first attempt to find out minimized data for an application directly. They believe that data used by most of its counterparts has a higher possibility of being the necessary data. Counterparts are a set of similar applications with similar functionalities. They calculate the likelihood of data being necessary based on the proportion of it in counterparts. Compared to our work, they mainly focus on the compliance of data collection declarations in privacy policies without checking actual practices. We reference their method as one part of the MPD construction and complement it with data collection practice analysis to further study data minimization fulfillment in the real world.

III. A SAMPLE DATA MINIMIZATION ANALYSIS AND PROBLEM FORMALIZATION

Given an extension, two challenges exist in analyzing its data minimization practices, as mentioned in Section I: 1) What are the extension-specific minimized personal data (MPD)? 2) What are the actual collected personal data (CPD) from extensions? In this section, we first illustrate our solution using a sample analysis of an extension *Ext A*¹ from the Chrome Web Store. Then, we formalize the data minimization problem.

A. Running Example: *Ext A*

We consider 11 types of personal data (as listed in Table I) where nine are given by Chrome [17], and three are additional privacy-related data types (i.e., **DI**, **PS**, and **FS**) identified from our manual inspection.

MPD extraction. The MPD is constructed from the extension’s description and counterpart references. For the description text analysis, as shown in Figure 1, a sentence is highlighted if it is relevant to the target data types and

¹Extension ID:lokxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxmoci (the real ID is partially anonymized to avoid legal disputes), installs: 400K+

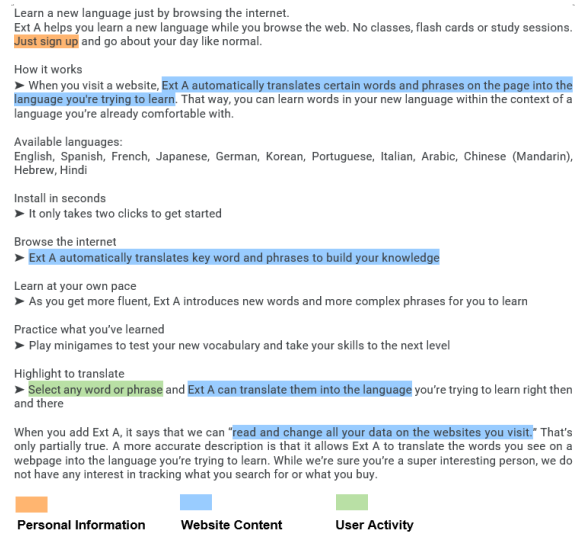


Fig. 1. Description of *Ext A*

vice versa. In this way, we get the first part of MPD. For counterpart references, it is inspired by POLICYCOMP [1] that the common data types collected by most counterparts should be adequate, necessary, and relevant to the purpose. We reference 10 of its counterparts’ data collection practices in this example. In particular, the 10 counterparts are selected from the “*Related*” section in Chrome Web Store, containing officially recommended extensions with similar functions. Note that referring “*Related*” recommendation is not included in MINDAEXT and is only used in this toy example to simplify the selection of counterparts and better illustrate the main approach. We identify CPD in these counterparts manually. Then, we extract data types collected by the vast majority (i.e., over 8 out of 10) of counterparts as the second part of MPD for *Ext A*. The union of the above two sets constructs the final MPD for *Ext A*, as listed in Table II.

CPD extraction. CPD is constructed from static and dynamic features by examining the source code and interacting with UIs. Note that the extraction of counterpart CPD above is the same as here. In the static features, we check all the requested APIs related to data collection behavior. For example, in Figure 2(a), the `navigator.geolocation` API indicates access to location information. In the dynamic features, we interact with the extension, click all the buttons, monitor network traffic, and record generated HTML files during the process. We capture data collection behavior through user inputs and network transfer. For example, a registration page indicates the collection of personal information. The CPD set for *Ext A* is shown in Table II.

Excessive data collection. By comparing MPD with actual practices CPD, we find a violation of the data minimization principle in *Ext A*. It excessively collects users’ *location information* in the background JavaScript code captured by static feature analysis. In our manual confirmation, it is irrelevant to any functionality indicated in the description text or referred to in counterparts.

Moreover, the location information and two other data




```
Zone.__load_patch("geolocation", (e => {e.navigator.&&e.navigator.geolocation
(e.navigator.geolocation, ["getCurrentPosition", "watchPosition"])
```

(a) Code snippet of *ExtA background.chunk.js* file

Privacy practices

_____ has disclosed the following information regarding the collection and usage of your data. More detailed information can be found in the developer's [privacy policy](#).

_____ collects the following:

-  Personally identifiable information
-  User activity
-  Website content

(b) Declaration of privacy practices of *Ext A* from Chrome Web Store

Fig. 2. Data Collection Practices of *Ext A*

types (i.e. authentication and payment information) are not declared in its *Privacy Practices* on the Chrome Web Store, as shown in Figure 2 (b). This is a common issue alerted by recent research [25], [6] that *Privacy Practices* declared by developers are inconsistent with actual behavior. But this inconsistency is not the main issue this paper aims to check.

B. Problem Formalization

To further facilitate the understanding of data minimization analysis, we formally define the target problem in the context of general applications, including browser extensions.

A Minimized data set is defined based on the purpose, specifically the functionality.

Definition 1: The minimized data collection of a functionality f is defined as a set, $MIN(f) = \{d_1, \dots, d_k\}$, with k data types. $MIN(f)$ is the minimum data for f to achieve the purpose.

Definition 2: Given an extension ext , the functionality of ext is a set $F(ext) = \{f_1, f_2, \dots, f_p\}$ with the arity p . The set of minimized personal data (MPD) for the extension is defined as the union set of minimized data types from $F(ext)$:

$$MPD = \bigcup_{i=1}^p MIN(f_i)$$

Definition 3: Given an extension ext , the actual data collection is a set, $CPD = \{d_1, \dots, d_k\}$, with the arity k .

Definition 4: (Compliance of Data Minimization) Given sets MPD and CPD of an extension, the data minimization principle is satisfied if $CPD \subseteq MPD$. Otherwise, it is violated. Intuitively, if an extension collects more data than its minimized data set, we consider the principle to be violated.

IV. MINDAEXT OVERVIEW

The workflow of MINDAEXT is presented in Figure 3. It extracts MPD and CPD respectively from *minimized data inferer* and *collected data analyzer*, solving the two main challenges we discussed in Sec I.

Minimized data inferer This module provides a reference of MPD for an extension in two aspects: 1) extracting minimized data types $MIN(f)$ for each functionality indicated by its description text analysis and 2) referring to the data collection behaviors of counterpart extensions. Firstly, we leverage natural language processing (NLP) models to identify the functionality and label pre-defined minimized data types indicated by the functionality for each extension from its description text. Particularly, we train BERT and BiLSTM and compare them with GPT-3.5. To train NLP models or examine results from GPT-3.5, we label a corpus, MINDA-128 with 2421 sentences. And we select the model with the best performance, BERT, in MINDAEXT. Details in Sec V-A.

However, extension developers do not always describe all the functionality in the description. They are also not responsible to do that. In this case, fully relying on the description text to extract all the functionality from extensions is insufficient. We include counterpart analysis to supplement the MPD set. For each extension, we calculate the text-similarity between its description text and others. Similar to POLICYCMP [1], we rank the text similarity and identify the most similar 20 counterpart extensions. Then, we define the essential index to evaluate the likelihood of the data from these 20 counterparts being necessary. For the data that essential index is greater than 0, we collect them into the MPD set. Details will be in Sec. V-B.

Note that, unlike related works [1], [7], [2] that take privacy policies as an important source to infer functionality, we adopt description texts instead of privacy policies or other extension-specific information for two main reasons. 1) Firstly, the browser extension community lacks management of privacy policies. Only 26.53% extensions provide a privacy policy in the Chrome Web Store, which is a consensus by both Ling et al. [25] and Bui et al. [6]. 2) Secondly, privacy policies in the browser extension ecosystem are of poor quality. Among 40k available privacy policies, we found 54.64% of them are duplicated with other products. In this case, little unique information regarding functionality is available in the extensions' privacy policies. A detailed analysis of descriptions and privacy policies is available in Section VII-C.

Despite privacy policies, other extension meta information, such as extension name, rating, download, reviews, and category, is available in online stores but too coarse-grained to infer unique extensions' functionality. Ultimately, we decided to use description text to perform a more reliable analysis towards more extensions. Because almost every extension has a description text briefly introducing its functions and features ranging from several words to multiple paragraphs. This allows us to analyze the MPD in 73.46% of extensions that do not provide privacy policies. Moreover, we include counterpart

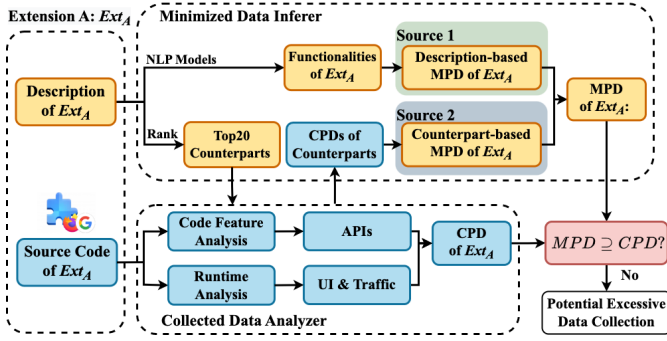


Fig. 3. MINDAEXT Overview

analysis as an important supplement source to cover the shortage of the description. Details will be in Section V.

Collected data analyzer With MPD that defines which data an extension should collect, we next identify which data an extension really collects, i.e. CPD. This module utilizes the following program analysis to extract CPD in two aspects: 1) code feature analysis on privacy-related APIs in source code, and 2) runtime analysis on dynamically loaded UI pages and network traffic. Firstly, we conduct code feature analysis to identify obvious data collection behavior by simple API matching. However, some code is generated dynamically, and some data types are not collected by API (such as personal information). In this case, we have runtime analysis to construct another half of the CPD. We execute the extension, imitate user actions, and capture data collection behavior. Details will be in Sec. VI. We note the absence of data flow analysis after API identification. However, as far as we know, conducting data flow analysis in JavaScript code is tedious, less precise, and might cause false negatives, which is not what we expect. We will discuss this in Sec. VIII.

Excessive data collection identification The last step is to compare MPD with CPD. If CPD is the subset of MPD, the extension is likely to be compliant with the data minimization principle. Otherwise, there is potential excessive data collection behavior.

V. MINIMIZED DATA INFERER

The *minimized data inferer* of MINDAEXT aims to determine MPD. As privacy policies and other meta-information suffer from the shortcomings we discussed above (Sec IV), our analysis is based on its description text and counterparts. MPD for each extension is the union set of the following two parts:

A. Description-based MPD

The first part is to extract MPD from the description text directly. Considering the impressive improvement made in the large language models (LLMs), we are facing the choice of using either general NLP models (e.g. LSTM [36] and BERT [9]) or LLMs (e.g. GPT-3.5 [31] and GPT-4.0 [31]). Both paths require a corpus to train the model or evaluate the result. In this case, we annotate a corpus, named MINDA-128,

compare the performance among LSTM, BERT, and GPT-3.5, and embed the model with the best performance into MINDAEXT.

1) *MINDA-128 Corpus*: To train NLP models and evaluate the accuracy of LLMs, we first construct the corpus, MINDA-128, containing 2421 manually labeled sentences from 128 extension descriptions.

Preprocessing description texts. We first randomly assemble 128 extensions, with at least 10 extensions from each of the 11 extension categories in the Chrome Web Store, and the descriptions in high quality (e.g., at least five sentences; no non-English word). This selection is to ensure sufficient coverage of extensions exhibiting distinct features across various functionalities. Then, we divide each description text into individual sentences using the Natural Language Toolkit (NLTK [40]) and remove all non-ASCII characters.

Corpus annotation. For each sentence, we manually assign a label mapping its implied functionality to one of the data types listed in Table I. For example, we label the sentence “Install the extension and sign up.” as **UPI**, as it suggests the requirement to collect user profiles for registration. During the annotation process, each sentence is independently labeled by three of the co-authors. It takes an average of 4 minutes for each annotator to label one sentence. If three annotators assign the same label to one sentence, the label for this sentence is confirmed. In case of any inconsistent labeling, we would initiate a discussion, and further consult a privacy law specialist and an experienced software developer (with over five years in the industry) to reach a consensus over the final label. The distribution of annotated sentences under each data type is shown in Table III. There are 13 labels in total; 12 of them correspond to the 12 types of data we target, and the last label, **NA**, indicates that the sentence cannot refer to any data type. MINDA-128 is released in the repository [33] to facilitate reproduction and future research.

2) *Description-based MPD extraction*: With the corpus, MINDA-128, the next step is to have an NLP model. Considering the imbalance and limited size of MINDA-128 in 13 labels as shown in Table III, we adopt k-fold cross-validation to train BiLSTM and BERT, with the value k in 10.

Moreover, we designed a conversation template to convert MINDA-128 into dialogue format, used as the testing set for GPT. Specifically, we fill each sentence from MINDA-128 into the following conversation template: “Base on the description text: [fill_sentence_here]. Which of the following 12 data types are likely to be collected by the extension: (1) user profile information (2) health information ... (12) file system? “. The detailed 12 data types are shown in Table I but omitted here. However, GPT always answers the question in a long paragraph, which makes it difficult to extract results directly. In this case, we append the following constraint sentences to the end of the conversation template: “Only answer ‘yes’ or ‘no’ for each data type. And Only one type is collected in maximum“. Therefore, GPT will respond with a formatted answer, like “1. User profile information: No“, which can easily be mapped to origin corpus labels. The performance is

TABLE III
LABEL DISTRIBUTION IN MINDA-128

Labels	UPI	HI	FPI	AI	PCI	LI	WHI	UAI	WCI	DI	PS	FS	NA
No. of Samples	21	44	25	42	85	32	20	250	284	37	107	91	1383
Percentage	0.87	1.82	1.03	1.73	3.51	1.32	0.83	10.33	11.73	1.53	4.42	3.76	57.13

detailed in Sec. VII-B.

B. Counterpart-based MPD

The second part is to infer MPD from counterpart extensions' practices.

1) *Counterpart identification*: Firstly, we identify counterpart extensions with similar functionality or purpose to the target extension. We found that the description introduces the basic functionality of the extensions in most of the case. Thus, similar descriptions imply similar functionality of extensions. So, we calculate description similarity between each pair of two extensions, rank the similarity scores, and take the top 20 as the counterparts. For the calculation of description similarity, we deploy the same approach as previous works [1], [19], [22]. In the preprocessing stage, we remove all stop words, non-ASCII characters, numerals, HTML tags, URLs, and email addresses. Then, we apply stemming to all word tokens to extract word roots. After that, we use a well-trained Latent Dirichlet Allocation (LDA) model provided by GENSIM to calculate the probability of a description being in a certain topic. We set the number of topics to 30, as suggested by both Gorla et al. [19] and Jiang et al. [22]. In this case, we could construct a vector consisting of 30 probability values for each description. By calculating the cosine similarity of two vectors, we can quantify the semantic similarity of descriptions from two extensions. Then, we identify the top 20 counterparts by ranking the similarity score. The reason for selecting the top 20 is in Sec. VII-B.

2) *Counterpart-based MPD extraction*: In this step, we extract MPD from data types that counterparts collect. Here, we introduce the *essential index*, $I(d, E)$, to calculate the necessity of a data type d for an extension E , as shown in Eq. 1. In this formula, CPD_k is the CPD of the k -th counterpart extracted by *collected data analyzer* (will be detailed in Sec VI) and σ is the essential threshold. If the essential index satisfies $I(d, E) > \sigma$, we include the data type d to the MPD set of the extension E . The key insight is to calculate the proportion of counterparts that collect the target data type and compare it with the essential threshold to determine whether it should be included in MPD set of the extension. Here, we set σ to 0.5. The selection of σ value is discussed in Sec. VII-B.

$$I(d, E) = \frac{1}{20} \sum_{k=1, \dots, 20} \begin{cases} 1, & \text{if } d \in CPD_k \\ 0, & \text{otherwise} \end{cases} - \sigma \quad (1)$$

VI. COLLECTED DATA ANALYZER

This section introduces *collected data analyzer* that identifies the data collected by the extension in practice, i.e. CPD. *collected data analyzer* get the union set of CPD from code feature analysis and runtime behavior analysis.

A. Code Feature Analysis

The structure of extension source code is similar to the front end of web applications. A special *json* file, *manifest.json*, declares basic information about the extension, including the name, version number, permissions, domains, etc. The remaining file includes HTML, CCS, Javascript code, images, etc.

1) *API extraction*: We identify the CPD by checking the usage of relevant browser APIs. To construct the target API list, we refer to the official documentation from Chrome and Firefox to identify the privacy-related APIs. Then, we map those APIs to a specific relevant user data type. For example, the calling of *browser.history.getVisits()* implies a potential collection of data **WHI**. In summary, we identified 36 Chrome APIs and 27 Firefox APIs from 126 developer APIs. The complete mapping list from API to data type is available on the repository [33]. Then, we use the tool *esprima* [10] to parse JavaScript code into Abstract Syntax Trees (ASTs), which provide a clear form of the code logic and the function hierarchy. Then, we traverse the ASTs using depth-first search (DFS) and identify all nodes in the type of *CallExpression* related to function calls. We check the function name of these nodes. If it matches with any API in the target API list, we add associated personal data types to the CPD set. Note that we only account for the APIs that call *get()* related functions, like *browser.action.get()*.

B. Dynamic Runtime Analysis

Another half supplement of CPD is from dynamic runtime analysis. Since some code can be dynamically generated from JavaScript code, simple static analysis might not capture all CPD. In this section, we conduct a UI analysis to detect the data collected from user inputs and a traffic analysis to detect the data transmitted from the network traffic.

1) *UI analysis*: We design and implement a lightweight UI analysis that simulates user actions and traverses extension UIs by depth-first search. For each UI, we interact with all page elements and observe if any CPD is requested. We implement our tool using the well-known web automation library *Selenium* [41].

Automatic installation and initialization. In the beginning, our runtime analysis tool can automatically install an extension by specifying the path of its source files

in the *Driver Options* provided by *Selenium*. This simulates the user-conducted installation process. Afterward, our tool starts the extension (i.e., clicks on the extension icon on the right side of the browser’s toolbar) and gets the extension’s initial pop-up page through the paths “*chrome-extension://[ext_id]/[popup_page]*” and “*moz-extension://[dynamic_u” “uid]/[popup_page]*” for Chrome and Firefox extensions respectively. In particular, we extract the *dynamic UUID* for Firefox extensions from the browser’s memory by analyzing the page “*about:config*” and the field of “*extensions.webextensions.uuids*”.

Traversing extension behaviors. With the setup of the initial page, we then simulate realistic user interactions. We tend to trigger as many actions/functions/new pages as possible. Based on breadth-first search (BFS), we interact with all interactive elements (i.e., buttons, links, forms, drop-down menus, etc.) Note that we use the hash value derived from a concatenated string consisting of the URL and DOM element list as the identifier of a page state. During the traverse, we maintain a set of hash values representing unique page states it has traversed. By checking the hash set after each action, we can easily know whether the current status has been reached previously. If it is duplicated, we prune the following traverse to save time and prevent loops.

The automatic page traversing works for most extensions with simple UIs. However, it could be stopped by the scenarios that login or registration is mandatory. Some extensions even separate the registration procedure into a series of UI pages. To handle such cases, we define six typical registration and login action templates and prepare some pre-defined testing account information (i.e. email address, username, and password). The analysis follows our templates, automatically registers testing accounts, logins into the account, and utilizes the aforementioned method to traverse the extension behavior. If extra verification is required, such as email link click and CAPTCHA, we resort to manual efforts. In total, we found that 868 extensions require registration/login requirements, while 149 extensions require human verification.

CPD identification. In this step, we analyze the HTML file of each unique page state to identify CPD that extensions request from the user. We extract the attributes of input-related (e.g., `<input>`, `<fieldset>`) and operation-related (e.g., `<button>`, `<select>`) tags. We further project each to a specific data type by matching the string with the target keywords. The full list of keywords is available in the repository [33].

2) *Network Traffic Analysis:* During the extension page navigation, we intercept the browser network traffic in the background using *MitMProxy* [27]. To ensure the traffic is generated by the target extension, we only test one extension at one time in a plain browser without any other process running in the background. To confirm a request’s source, we adopt the same approach from Bui et al. [6] that sets the HTTP Origin header of each request to *chrome-extension://ext_id* or *moz-extension://ext_id* for Chrome and Firefox extensions respectively. For the captured HTTP(S) requests, we extract

all the URLs, key-value pairs in query strings, request bodies, and POST request forms. Then, we identify the collected and transmitted data types by the same keyword-matching rules as what is used in UI analysis.

VII. EVALUATION

In this section, we evaluate the accuracy and effectiveness of MINDAEXT and perform a large-scale analysis to check the compliance of data minimization practices among browser extensions. Our analysis targets the following two research questions (RQs):

- **RQ1:** What is the performance of MINDAEXT in analyzing data minimization practices? (Section VII-B)
- **RQ2:** What is the status quo of data minimization enforcement? (Section VII-C)

A. Data Collection

In this work, we target extensions on Chrome and Firefox extension stores, covering over 85% worldwide market share [24]. We use *Sitemaps* [18], [12] to obtain complete lists of available extensions from official stores. The total number of extensions is 196,599, with 167,577 Chrome and 29,022 Firefox extensions respectively. After deriving the extension list, we deploy our extension source code crawler on an AWS instance equipped with 4 vCPUs, 32GB RAM, and 6TB storage. We started the crawler in October 2023, and the collection process was completed within two days. Then, we remove extensions without description, with non-English descriptions, or unavailable source code. Finally, we managed to automatically download the complete data (i.e., source code, description texts, etc.) from 122,587 Chrome extensions (205GB of data) and 20,169 Firefox extensions (15 GB of data), on which we base our subsequent analyses.

B. RQ1: Performance of MINDAEXT

As mentioned in Section V, MPD consists of description-based and counterpart-based information. In this RQ, we compare the quality between descriptions and privacy policies. Then, we present the results for each MPD component separately.

1) *Descriptions and Privacy Policies:* From sitemap traversing, we collected 167,577 unique Chrome extension IDs in October 2023. However, 91.38% (153,139 out of 167,577) are downloadable and still alive in the extension store. Among those alive extensions, 99.53% (152,412 out of 153,139) provide description texts. The distribution of description length is shown in Table IV. 93.87% of descriptions are longer than 10 words, and 59.48% are longer than 50 words, which is sufficient to conduct a useful MPD extraction. The distribution of languages these descriptions are written in is shown in Figure 4. We calculate the logarithmic value with base 10 for the number of extensions in each language. 50 languages are observed in descriptions, while English is the dominant language, accounting for 80.02% (122,587 out of 153,189). The top 5 languages are English, Vietnamese, Russian, Japanese, and Chinese.

TABLE IV
LENGTH OF DESCRIPTION TEXTS

	0-10 words	11-50 words	51-100 words	101-200 words	≥ 201 words
# of Ext	9,339	52,419	39,082	31,942	19,630

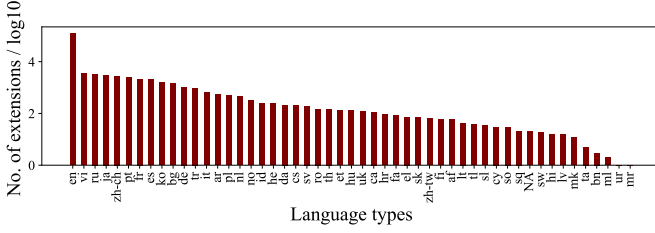


Fig. 4. Distribution of Language Types

Among those alive extensions, 26.54% (40,644 out of 153,139) provide privacy policies. However, there are only 23,230 (out of 40,644) unique privacy policies. The most interesting finding is that <https://trawellcat.com> is the most repeated privacy policy link, which has been used in 796 extensions. But it is not linked to any privacy policy content. Moreover, there is an individual developer who provides one privacy policy for all 341 extensions he released in the Chrome Web Store. The distribution of privacy policy repetition times is shown in Table V. Only 45.36% (18,438 out of 40,644) privacy policies are specialized for one extension. Other communal privacy policies cannot always describe an extension’s functionality and purpose precisely. Considering the limited amount of privacy policy and widespread repetitions, the identification of MPD in MINDAEXT is from descriptions instead of the privacy policy.

2) *Description-based MPD*: The accuracy of MPD identification mainly relies on the performance of NLP models. We compare the performance of BiLSTM, BERT, GPT-3.5, and GPT-4.0 with regard to precision, average weighted recall rate, and F1 score, as shown in Table VIII. We find that BERT performs the best. So BERT is embedded into MINDAEXT at last and used in large-scale analysis. The performance of BiLSTM and BERT is limited by the size of the training data set, MINDA-128. However, two GPT models give the worst performance in this classification task. During the experiment, we found that we could not force GPT to annotate the data with only one label, even though we add the sentence, “*Only one data type in maximum.*”, into the conversation template. We only provide data type names to GPT, which can be improved by giving detailed explanations of each data type in the conversation template. And answers from GPT are not always consistent, which can be improved by executing multiple times and getting the average result. We leave these

TABLE V
DISTRIBUTION OF PRIVACY POLICY REPETITIONS

Repetitions	1 time	2 times	3-5 times	6-10 times
# of PP	18,438	2,655	1,487	375
Repetitions	11-20 times	21-50 times	51-100 times	≥ 101 times
# of PP	194	53	17	11

TABLE VI
ACCURACY OF COUNTERPART-BASED CPD UNDER DIFFERENT COUNTERPART NUMBERS AND ESSENTIAL CONSTANT

σ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Top 10	0.3	0.2	0.3	0.5	0.5	0.4	0.3	0.4
Top 20	0.5	0.6	0.5	0.5	0.6	0.4	0.6	0.3
Top 30	0.3	0.4	0.5	0.5	0.5	0.3	0.4	0.4

TABLE VII
NUMBER OF UNIQUE PAGES

# of Ext	Without UI	With UI			
		1 Page	2-5 Pages	6-10 Pages	≥ 11 Pages
Chrome	47,452	45,043	26,498	2,152	1,443
Firefox	16,081	2,350	1,608	116	13
Total	63,533	47,393	28,106	2,268	1,456

improvement tasks in GPT for future work.

3) *Counterpart-based MPD*: Following the approach detailed in Section V-B, we select the top 20 counterparts based on description text similarity. Then, we determine the MPD set based on the essential index. In essential index $I(d_j, C)$ (given by Equation 1), essential constant σ is set to 0.50. We randomly select 10 extensions and manually annotate MPD as the baseline. We test the accuracy of counterpart-based MPD under different combinations of counterpart numbers and essential constants. We choose the best-performing combination: top 20 counterparts and σ at 0.5. We note the baseline is too small; our chosen values might be unreliable. Considering the validity check of the final analysis result (to be discussed soon) is good enough, we leave the task of finding a more reliable counterpart number and σ value for future work.

4) *CPD Extraction*: CPD consists of two parts: code feature analysis and runtime analysis. In runtime analysis, we find that 44.5% extensions (63,533 out of 142,756) do not have any UI, as shown in Table VII. Because some extensions would only be active in the background when visiting certain URLs. These URLs are the host URLs that must be declared in *manifest.json*. In such cases, we open the host URL in the browser, click the extension icon, and rely on the network traffic to detect CPD. However, this method is still not comprehensive enough to trigger all the extensions. For example, we cannot trigger the action that the extension inserts as an option in the right mouse-click menu. As far as we know, no existing work provides a complete action template list to cover all cases. We leave this to future work.

5) *MPD and CPD Validity Check*: Getting the MPD and CPD of an extension, we evaluate the overall accuracy of MINDAEXT. We randomly sample 50 Chrome and 50 Firefox non-overlapping (i.e., descriptions not included in the corpus) extensions from the dataset. To confirm the accuracy of their MPD and CPD from MINDAEXT, we need to construct a baseline for those 100 extensions. Three co-authors annotated MPD and CPD for each extension by reading the description from the extension store, downloading the extension to understand its functionality, and observing any data collection behavior in the testing. Any inconsistency between baselines

TABLE VIII
NLP CLASSIFICATION RESULTS ON MINDA-128

Algorithms	Precision	Recall	F1	Algorithms	Precision	Recall	F1
GPT-3.5	0.53	0.55	0.52	Bi-LSTM	0.54	0.62	0.57
GPT-4.0	0.53	0.54	0.52	BERT	0.62	0.65	0.59

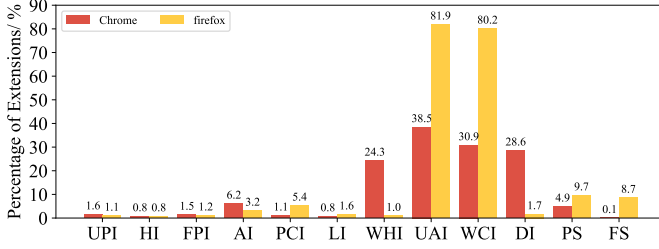


Fig. 5. Percentage of Extensions v.s. Data Types

from three co-authors will invoke a discussion. With the baseline, we compare it with the MPD and CPD. For the accuracy of MPD, 53% exactly match the baseline, 87% contain two mismatches at most, and 98% contain five mismatches at most. For the accuracy of CPD, 65% match the baseline, 80% include the baseline but contains false positives. For the overall accuracy of examining data minimization, out of the 35 extensions with actual data minimization violations, MINDAEXT detects 22 (62.9%) of them (exactly match in each data type). If we focus on whether MINDAEXT labels an extension as compliant or incompliant correctly (the violation data types might be different), MINDAEXT detects 26 (74.3%) of them, with 6 (9.23%) false positives.

C. RQ2: Enforcement Status of Data Minimization

We apply MINDAEXT to systematically analyze the status quo of the data minimization practices implemented for each extension. To this end, we present the results in each step, followed by the data minimization compliance analysis and feature relevance analysis.

1) *Identified MPD*: The distribution of extensions that contain each data type in MPD is shown in Figure 5. We find that UAI and WCI are the most necessary data types that more than 30% extensions have in Chrome and over 80% in Firefox. This finding coincides with the stereotype of extension’s functionality that facilitates user experience when browsing general websites. Because UAI and WCI are essential data types in this scenario for extensions to detect and react to specific user actions and website contents. We also find that PCI and FS are more popular among Firefox extensions than Chrome. And WHI and DI are popular in Chrome extensions but not in Firefox. The number of data types included by MPD in each platform is shown in Figure 6. We will discuss this later with the identified CPD.

2) *Identified CPD*: we follow the technique in Section VI to detect data collection practices from code feature analysis (API) and runtime analysis (UI and Traffic). The collection of each data type found in each analysis step is shown in Table IX. Empty cells represent the data that cannot be detected in this step. For example, UI analysis cannot find

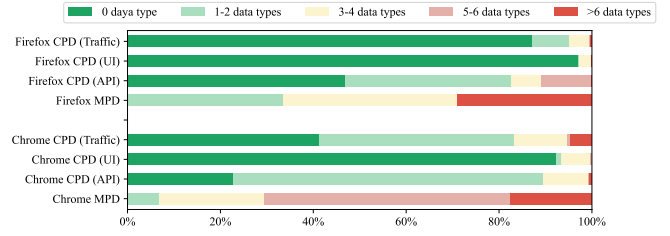


Fig. 6. Number of Data Types Collected in MPD and CPD

TABLE IX
DISTRIBUTION OF DATA TYPES COLLECTED

Data Types	Chrome Extension			Firefox Extension		
	Code	Runtime		Code	Runtime	
	API	UI	Traffic	API	UI	Traffic
UPI	–	4,673	15,819	–	229	190
HI	–	158	586	–	16	21
FPI	–	125	366	–	8	22
AI	3,063	4,809	19,148	195	322	875
PCI	–	1,301	1,712	–	77	101
LI	387	162	407	32	14	18
WHI	2,732	–	–	471	–	–
UAI	77,806	–	–	15,342	–	–
WCI	79,304	–	–	14,744	–	–
DI	4,170	226	65,024	948	25	75
PS	5,665	277	577	1,258	22	33
FS	30	232	236	1	27	10

“–” represents data is unavailable in this step.

WCI, UAI, WCI, but can capture most user inputs (i.e., UPI and AI). The union of the three CPD subsets covers all data types. The distribution of CPD against the number of data types is presented in Figure 6. We find that the overall number of data types from CPD is less than MPD. On the platform level, Chrome extensions are inclined to require more types of data than Firefox.

3) *Status Quo of Data Minimization Compliance*: We compare MPD with its CPD for each extension to obtain the potential compliance status. We identified two possible states: compliance (CL), where CPD perfectly matches or is the subset of MPD; otherwise, incompliance (ICL), where CPD contains data beyond MPD. The compliance status in total and in different numbers of data types in CPD are shown in Table X. Overall, we identified 39.89% (48,863 out of 122,587) ICL extensions in Chrome, 26.69% (5,383 out of 20,169) in Firefox, and 38.00% (54,246 out of 142,756) in total. Out of them, over half abuse more than two types of personal data. A high proportion of incompliant extensions collect more than 5 data types. The proportion of excessive data collection in different data types is shown in Figure 7. We observe that **UAI**, **WCI**, and **DI** are top targets for Chrome and Firefox extensions. And excessive collected data types distributed differently in two platforms. For example, **UPI** and **AI** are excessively collected among 15.9% and 16.4% of Chrome extensions, but only a few exist in Firefox extensions.

4) *Feature Relevance in Data Minimization Compliance*: We identify some simple but interesting characteristics in incompliant extensions that users and privacy regulation audi-

TABLE X
(IN)COMPLIANCE STATUS VERSUS NUMBER OF DATA TYPES IN CPD

Platform	Status	Total	0	1	2	3	4	5+
Chrome	CL	115,289	23,190	31,724	38,530	19,150	2,035	660
	ICL	48,863	0	6,882	13,097	14,092	5934	8,858
Firefox	CL	14,786	5,547	3,899	5,132	203	5	0
	ICL	5,383	0	1,137	1,965	1,773	402	106

CL: Compliant; ICL: InCompliant

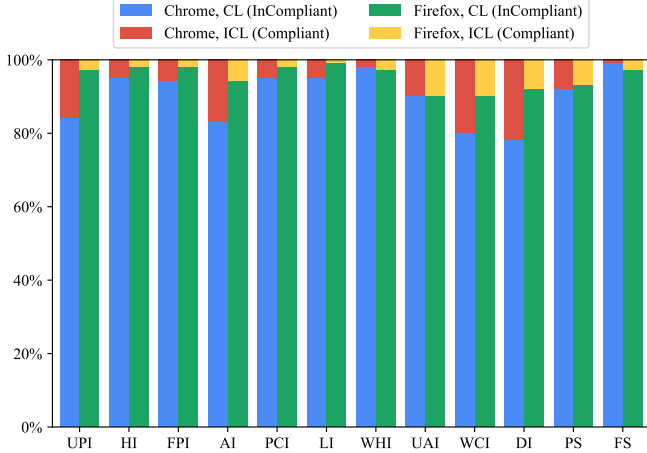


Fig. 7. Data Minimization (In)Compliance vs. Data Types

tors can refer to. We propose 4 features that users always notice when browsing extension stores, covering extension size, length of the description, ratings, and downloads, as listed in Table XI. We use Odds Ratio (OR) to measure each feature’s relevance between the complete sets of compliant and in-compliant extensions. The larger the OR (higher than 1), the stronger the positive correlation, and the smaller the OR (lower than 1), the stronger the negative correlation. Considering the OR is based on two events, but the value of our features is continuous, we have to find a threshold to separate each feature into two statuses (e.g. Rating > 4.8 and Rating ≤ 4.8). We calculate the OR using different thresholds and select the one yielding the most obvious correlation.

The OR for each feature with one threshold is shown in Table XI. We find that extensions with smaller sizes, shorter description sentences, and fewer downloads have higher relevance to excessive data collection behavior. However, the rating cannot imply the compliance of extensions under the data minimization requirement. It is likely that ratings mainly indicate what users care about, like the completeness of functionality, instead of the hidden excessive data collection.

VIII. DISCUSSION AND LIMITATION

A. Implications

The analysis results reveal the wide gap between the data minimization principle mandated by the privacy regulations and browser extension data collection practices, highlighting the previously neglected critical risk of excessive user data collection and abuse. The prevalent violations advocate further efforts towards efficient approaches for examining such compliance issues from end to end. Toward data minimization compliance, precisely defining the qualitative minimization concept is a complicated task, hindered by subjective inter-

TABLE XI
ODDS RATIO VALUES OF EXTENSION FEATURES

Platform	Meta Features			
	Ext Size (KB)	No. of Sentences in Description	Ratings	No. of Downloads
Chrome	$<30,000$	<20	>4.8	$<5,000$
	1.69	1.47	1.02	1.18
Firefox	$<10,000$	<15	>4	$<10,000$
	1.69	5.31	1.06	1.33

pretations of terms, such as “*necessary*”, “*relevant*”, and “*adequate*”. MINDAEXT takes one step forward to automate MPD identification for general applications, especially for browser extensions. MINDAEXT cannot provide law verdicts but provides references for legislators and store operators to facilitate the compliance auditing process.

B. False positives

The MPD from *minimized data inferer* is an under-approximation of the real MPD, because it cannot cover the functionality never mentioned in the description or other extension meta information. And CPD from *collected data analyzer* is an over-approximation, because we include the data type into CPD once the API is detected. Sometimes, privacy-related APIs might not be executed or reachable eventually, or the data might only be accessed but not transferred to the server side. We did not include data flow analysis, which might cause false negatives. We utilize false positives from this step. With the over-approximated CPD and under-approximated MPD, MINDAEXT maintains a high recall rate in examining data minimization practices. We do not want MINDAEXT to miss any possible excessive data collection. The current accuracy is acceptable (see Sec VII-B). Based on it, we can reduce false positives in future work.

C. Counterpart Assumption

The correctness of counterpart reference is based on the assumption that most extensions comply with the data minimization principle. If most extensions violate the regulation, collecting excessive data, counterpart-based MPD will also include such excessive data. Our results show that the assumption holds in the current browser extension ecosystem.

D. Transferability

Adaptability for different privacy laws. MINDAEXT can be applied to other privacy laws that contain the same data minimization requirements, such as GDPR, CPRA, and PIPL.

Adaptability for different usage scenarios. MINDAEXT can be extended for examining the data minimization against other applications, such as mobile and IoT-related applications. The extraction of MPD will remain the same if description texts are provided with high quality. However, the extraction of CPD will require modification in code features and runtime analysis. Because the API list, programming language, and tool for dynamic testing might have changed. Nevertheless, the basic idea and approach can be adapted to other applications.

IX. CONCLUSION

Towards better data minimization compliance among extensions, we propose MINDAEXT for an automatic end-to-end examination of this data collection principle in browser extensions. MINDAEXT determines the minimized data set necessary for its functionality based on description analysis and counterpart reference. Then, MINDAEXT leverages code feature analysis and runtime analysis to derive the data collection practices for comparison. Based on our large-scale study of over 142K browser extensions, we have identified prevalent possible excessive data collection in 54,246 extensions (38.0%). Our work unveils the previously neglected user privacy threat, highlighting the necessity for service providers and platform operators to enact better data minimization compliance.

X. DATA AVAILABILITY

We open-source MinDaExt implementations on our anonymous repository [33], including the extension crawler, code feature analysis, runtime analysis, and the corpus MINDA-128. In addition, we also release the intermediate analysis results as mentioned throughout the paper.

XI. ACKNOWLEDGE

We are grateful to the anonymous reviewers for their valuable and detailed comments. The first author wishes to thank Nan Jiang for her feedback on the draft of this paper, Fuman Xie and Ruiping Liu for their encouraging and insightful discussions. This work is partially supported by National University of Singapore under the funding [A-8000596-00-00] (11.8 - 5.31); and the University of Queensland under the Global Strategy and Partnerships Seed Funding and Australian Research Council Discovery Projects under DP240103068.

REFERENCES

- [1] POLICYCOMP: Counterpart comparison of privacy policies uncovers overbroad personal data collection practices. In *USENIX Security*, 2023.
- [2] Benjamin Andow, Samin Yaseer Mahmud, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Serge Egelman. *Actions Speak Louder than Words: Entity-Sensitive Privacy Policy and Data Flow Analysis with POLICHECK*. 2020.
- [3] Thibaud Antignac, David Sands, and Gerardo Schneider. Data minimization: a language-based approach. In *IFIP SEC*, 2017.
- [4] Vanessa Ayala-Rivera and Liliana Pasquale. The Grace Period Has Ended: An Approach to Operationalize GDPR Requirements. In *RE*, pages 136–146, 2018.
- [5] David Basin, Søren Debois, and Thomas Hildebrandt. On purpose and by necessity: compliance under the gdpr. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 20–37. Springer, 2018.
- [6] Duc Bui, Brian Tang, and Kang G. Shin. Detection of Inconsistencies in Privacy Practices of Browser Extensions. In *S&P*, 2023.
- [7] Cheng Chang, Huaxin Li, Yichi Zhang, Suguo Du, Hui Cao, and Haojin Zhu. Automated and Personalized Privacy Policy Extraction Under GDPR Consideration. In *WASA*, 2019.
- [8] Yunang Chen, Mohannad Alhanahnah, Andrei Sabelfeld, Rahul Chatterjee, and Earlece Fernandes. Practical Data Access Minimization in Trigger-Action Platforms. In *USENIX Security*, 2022.
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [10] Esprima. <https://esprima.org>, 2022. Accessed: 2022-11.
- [11] Ming Fan, Le Yu, Sen Chen, Hao Zhou, Xiapu Luo, Shuyue Li, Yang Liu, Jun Liu, and Ting Liu. An Empirical Evaluation of GDPR Compliance Violations in Android mHealth Apps. In *ISSRE*, 2020.
- [12] Firefox. <https://addons.mozilla.org/sitemap.xml>, 2023. Accessed: 2022-12.
- [13] Gemma Galdon Clavell, Mariano Martín Zamorano, Carlos Castillo, Oliver Smith, and Aleksandar Matic. *Auditing Algorithms: On Lessons Learned and the Risks of Data Minimization*, page 265–271. 2020.
- [14] Giuseppe Garofalo, Tim Van hamme, Davy Preuveneers, and Wouter Joosen. A Siamese Adversarial Anonymizer for Data Minimization in Biometric Applications. In *EuroS&PW*, pages 334–343, 2020.
- [15] Abigail Goldstein, Gilad Ezov, Ron Shmelkin, Micha Moffie, and Ariel Farkash. Data Minimization for GDPR Compliance in Machine Learning Models. *AI and Ethics*, pages 1–15, 2021.
- [16] Google. Project strobe: Updates to our user data policy. <https://blog.chromium.org/2019/07/project-strobe-updates.html>, 2019. Accessed: 2022-11.
- [17] Google. Updated privacy policy & secure handling requirements. https://developer.chrome.com/docs/webstore/user_data, 2022. Accessed: 2022-11.
- [18] Google. <https://chrome.google.com/webstore/sitemap>, 2023. Accessed: 2022-12.
- [19] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. Checking App Behavior Against App Descriptions. In *ICSE*, 2014.
- [20] Majid Hatamian, Nurul Momen, Lothar Fritsch, and Kai Rannenberg. A Multilateral Privacy Impact Analysis Method for Android Apps. In *Annual Privacy Forum*, 2019.
- [21] Kris Heid and Jens Heider. Automated, Dynamic Android App Vulnerability and Privacy Leak Analysis: Design Considerations, Required Components and Available Tools. In *EICC*, 2021.
- [22] He Jiang, Jingxuan Zhang, Xiaochen Li, Zhilei Ren, David Lo, Xindong Wu, and Zhongxuan Luo. Recommending new features from mobile app descriptions. *ACM Trans. Softw. Eng. Methodol*, 28(4), 2019.
- [23] Rishabh Khandelwal, Thomas Linden, Hamza Harkous, and Kassem Fawaz. PriSEC: A Privacy Settings Enforcement Controller. In *USENIX Security*, pages 465–482, 2021.
- [24] Kinsta. Global Desktop Browser Market Share for 2022. <https://kinsta.com/browser-market-share/>, 2022. Accessed: 2022-11.
- [25] Yuxi Ling, Kailong Wang, Guangdong Bai, Haoyu Wang, and Jin Song Dong. Are They Toeing the Line? Diagnosing Privacy Compliance Violations among Browser Extensions. In *ASE*, 2022.
- [26] Xing Liu, Jiqiang Liu, Sencun Zhu, Wei Wang, and Xiangliang Zhang. Privacy Risk Analysis and Mitigation of Analytics Libraries in the Android Ecosystem. *IEEE TMC*, 2019.
- [27] mitmproxy. <https://mitmproxy.org>, 2022. Accessed: 2022-11.
- [28] David Monschein and Oliver P. Waldhorst. SPCAuth: Scalable and Privacy-Preserving Continuous Authentication for Web Applications. In *IEEE LCN*, pages 281–286, 2021.
- [29] Trung Tin Nguyen, Michael Backes, Ninja Marnau, and Ben Stock. Share First, Ask Later (or Never?) Studying Violations of {GDPR’s} Explicit Consent in Android Apps. In *USENIX Security*, 2021.
- [30] State of California Department of Justice Office of the Attorney General. California consumer privacy act. <https://oag.ca.gov/privacy/ccpa>, 2018. Accessed: 2022-11.
- [31] OpenAI. <https://platform.openai.com/docs/models>, 2023. Accessed: 2023-10.
- [32] Srinivas Pinisetty, Thibaud Antignac, David Sands, and Gerardo Schneider. Monitoring Data Minimization. *arXiv preprint*, 2018.
- [33] Repository for Analyzer, Dataset and Corpus. <https://github.com/YuxiLing/MinDaExt>, 2022. Accessed: 2022-11.
- [34] Iskander Sanchez-Rola, Davide Balzarotti, and Igor Santos. Baking-Timer: Privacy Analysis of Server-Side Request Processing Time. In *ACSAC*, page 478–488, 2019.
- [35] Gerardo Schneider. Is privacy by construction possible? In *ISO/ISA Symposium*, 2018.
- [36] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [37] Awanthika Senarath and Nalin Asanka Gamagedara Arachchilage. A Aata Minimization Model for Embedding Privacy into Software Systems. *Computers & Security*, 87:101605, 2019.
- [38] Laurens Sion, Dimitri Van Landuyt, and Wouter Joosen. An Overview of Runtime Data Protection Enforcement Approaches. In *EuroS&PW*, 2021.

- [39] Rocky Slavin, Xiaoyin Wang, Mitra Bokaei Hosseini, James Hester, Ram Krishnan, Jaspreet Bhatia, Travis D Breaux, and Jianwei Niu. Toward A Framework for Detecting Privacy Policy Violations in Android Application Code. In *ICSE*, 2016.
- [40] Sphinx. Natural Language Toolkit. <https://www.nltk.org>, 2023. Accessed: 2023-02.
- [41] The Selenium Project. <https://www.selenium.dev>, 2022. Accessed: 2022-11.
- [42] Jake Tom, Eduard Sing, and Raimundas Matulevičius. Conceptual representation of the gdpr: Model and application directions. In Jelena Zdravkovic, Jānis Grabis, Selmin Nurcan, and Janis Stirna, editors, *BIR*, pages 18–28, 2018.
- [43] European Union. General data protection regulation. <https://gdpr-info.eu>, 2016. Accessed: 2022-11.
- [44] Sinan Wang, Yibo Wang, Xian Zhan, Ying Wang, Yepang Liu, Xiapu Luo, and Shing-Chi Cheung. Aper: Evolution-Aware Runtime Permission Misuse Detection for Android Apps. *arXiv preprint*, 2022.
- [45] Le Yu. Identifying privacy issues in mobile apps via synthesizing static analysis and NLP. *PhD Thesis of Hong Kong Polytechnic University*.
- [46] Le Yu, Xiapu Luo, Jiachi Chen, Hao Zhou, Tao Zhang, Henry Chang, and Hareton KN Leung. Ppchecker: Towards Accessing the Trustworthiness of Android Apps' Privacy Policies. *IEEE TSE*, 2018.
- [47] Le Yu, Tao Zhang, Xiapu Luo, Lei Xue, and Henry Chang. Toward Automatically Generating Privacy Policy for Android Apps. *TIFS*, 12(4):865–880, 2017.
- [48] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. PrivacyFlash Pro: Automating Privacy Policy Generation for Mobile Apps. In *NDSS*, 2021.
- [49] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel Reidenberg, N. Cameron Russell, and Norman Sadeh. MAPS: Scaling Privacy Compliance Analysis to A Million Apps. Number 3, pages 66–86, 2019.