Analyzing Excessive Permission Requests in Google Workspace Add-ons

Liuhuo Wan[1], Chuan Yan[1], Mark Huasong Meng[2], Kailong Wang[3], and Haoyu Wang[3]

[1] The University of Queensland, Australia
[2] National University of Singapore, Singapore
[3] Huazhong University of Science and Technology, China

Abstract. In the digital era, business collaboration platforms have become pivotal in facilitating seamless remote work and virtual team interactions. These platforms, typified by Google Workspace, offer an integrated suite of tools (such as Google Docs, Slides, and Calendar) that significantly enhance business operations. They often extend their functionality through the integration of third-party applications, known as "add-ons". Google Workspace exemplifies this trend, blending traditional business solutions with advanced, add-on-driven capabilities. While this greatly augments productivity and collaboration for online personal or team work, concerns about the excessive use of data and permissions have been raised by both users and legislators, as add-ons can utilize the granted permissions to access and manipulate files managed by business collaboration platforms.
In this work, we propose an end-to-end approach to automatically detecting excessive permissions among add-ons. It advocates *purpose limitation* that the requested permissions of the add-on should be for its specific functionality and in compliance with the actual needs in fulfilling the functionality. Our approach utilizes a hybrid analysis to detect excessive permissions, including analysis of the add-on's runtime behavior and source code, and state-of-the-art language processing techniques for textual artifact interpretation. This approach can serve the users, developers and store operators as an efficient and practical detection mechanism for excessive permissions. We conduct a large-scale diagnostic evaluation on 3,756 add-ons, revealing that almost half of existing add-ons contain issues of excessive permissions. We further investigate the root cause of excessive permissions and provide insights to stakeholders. Our work should raise the awareness of add-on users, service providers, and platform operators, and encourage them to implement solutions that restrict the excessive permissions in practice.

1 Introduction

Business collaboration platforms represent a complex web-based application paradigm, epitomizing the concept of a "super platform". Platforms like Google Workspace [8] and Zoom [17] exemplify this model by hosting a diverse array of third-party applications, referred to as "add-ons". These add-ons integrate deeply with the platform's core services, offering users a seamless and efficient experience. Google Workspace, in particular, is a prime example of this ecosystem, providing a suite of applications that cater to various business needs. Notable add-ons within Google Workspace include Auto-LaTeX Equations [2] for Google Docs, facilitating the insertion of mathematical

equations, and No Phishing for Gmail, enhancing email security [13]. The escalating demand for remote work, online education, and virtual collaboration has significantly propelled the adoption and evolution of these platforms, establishing them as mature ecosystems in the contemporary digital landscape.

The add-on runs with a strong dependence on the host platform. It is typically executed as an extension without its own process address space, requesting resources through the APIs provided by the host. Since cross-domain data and control flow are involved, the OAuth protocol [14] has been commonly adopted for performing multi-party authorization among the add-on, the host and the user. More specifically, developers simply declare the requested permissions through the OAuth scope fields, in a similar manner to the Android's manifest file. As illustrated by the example shown in Figure 1, the add-on declares three permissions, including the ability to view email messages, send emails and view user's spreadsheets, inside `oauthScope` in the `appsscript.json` file. Despite the convenience for authorization, the coarse-grained permission management inevitably introduces potentially over-privileged or excessive permissions [23, 43]. As a result, dishonest developers could inadvertently request more permissions than those required by the functionality of their add-ons, rendering ill-purposed exploitation possible. This clearly violates the principle of least privilege, and may also cross the boundary for user data protection drawn by the strictly implemented and enforced privacy regulations/laws around the world [3, 7].

As efforts towards a better understanding of the underlying permission management risks [44] in business collaboration platforms, several empirical studies have focused on examining the inconsistencies and weaknesses in their permission authorization, such as privacy violation [22, 25, 27, 36], unprompted permission authorization [47, 48], and phishing [21]. They attribute the user data leakage to loose permission scopes [21, 48], without touching on the essential causes of excessive permission request issues, such as coarse-grained bundled permissions.

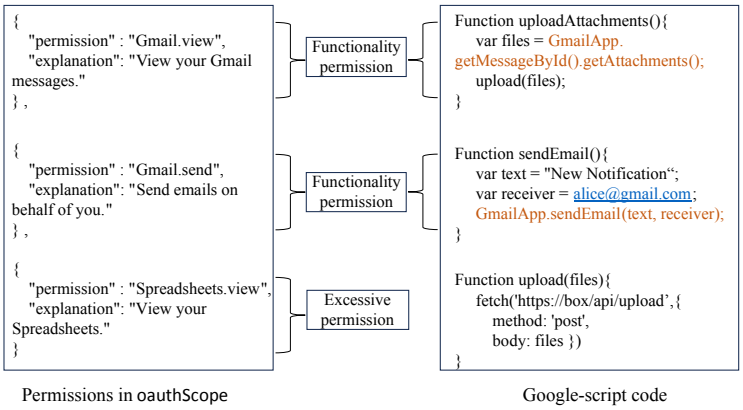

Fig. 1: An example of permission requests from an add-on

Our work. This work introduces a novel, scalable end-to-end approach to automatically detect issues of excessive permissions among Google Workspace add-ons. The objective is to provide an efficient and practical solution for evaluating excessive permissions for various stakeholders, including users, add-on developers, and the platform operators. With our approach, users can better understand excessive permissions before installing the add-on, and platform operators can scan for potential excessive permissions to enhance the permission and privacy compliance of their system. For add-on developers, this approach can mitigate the threats of excessive permissions before releasing their products.

We implement our approach as PEDEC, a *permission excess detector* for Google Workspace add-ons. PEDEC consists of two major components, the *functionality extractor* and the *excessive permission detector*. First, due to the partial availability of source code for these add-ons (**Challenge #1**), the functionality extractor extracts all functionality-related materials, including the interacted DOM, through web testing. We acknowledge that web testing has limitations and may not cover all paths. To supplement, we include functionality-related materials such as descriptions, tutorials, or demonstration videos. For the partially available add-ons with source code (228), we develop a lightweight static analysis tool to extract their functionality-related host APIs. Second, the excessive permission detector checks the consistency of functionality permissions and requested permissions to detect excessive permissions. The functionality permissions, sourced primarily from the materials extracted by the functionality extractor, primarily comprise natural language descriptions with varying format and quality among add-ons (**Challenge #2**). To address this, we employ the latest natural language inference (NLI) model to infer the functionality-related (or unrelated) permissions.

To understand the current status of excessive permissions among add-ons, our proposed approach is utilized to comprehensively screen 3,756 add-ons in the Google Workspace Marketplace [12]. The results show that a significant number (2,091) of add-ons have excessive permission issues.

Contributions. To the best of our knowledge, we are the first to diagnose the excessive permission issues of add-ons from Google Workspace at scale. In this work, we particularly focus on Google Workspace due to its popularity, which constitutes a substantial and representative market share of around 70% [8] in business collaboration platforms. In summary, this work mainly contributes to the following aspects.

- **Understanding the excessive permission issues for add-ons hosted on Google Workspace**. We first formally formulate and study the permission excessive issue, which has been a key cause for private data mismanagement in add-ons.
- **A systematic assessment approach.** We propose an automatic tool named PEDEC to detect excessive permissions among add-ons. We implement a prototype of PEDEC that checks permission compliance based on a hybrid analysis. The ground-truth evaluation on 100 add-ons shows that PEDEC achieves a 100% TNR (true negative rate) and 92% TPR (true positive rate) for detecting the excessive permission issue.
- **Revealing prevalence of the excessive permission issue in real-world Google Workspace add-ons**. Our large-scale study on Google Workspace reveals that permission management of add-ons is problematic (Section 4). Our investigation reveals that bundled permission declaration is the major cause of such an issue. Our

findings should raise an alert to the users, and encourage add-on developers and platform operators to redesign their interfaces.

2 A Running Example And Definitions

2.1 A Running Example

We delineate the overarching workflow of the Google Workspace add-on as follows. The user interacts with the add-on client on their computer or mobile device. Their request is sent to the workspace server as an *event*, which is then handled by a corresponding *event handler* provided by the add-on developer. The event handler serves the request, and during this process, it invokes APIs provided by the workspace to access the user's data managed by the workspace. The obtained data may be sent back to the add-on server for further processing in some computation-intensive tasks. After processing the request, the workspace server sends a response back to the add-on client and updates the user's client page.

For example, when the user clicks the "save attachments" button (shown in the sidebar of Figure 2), the event is transmitted to the Google server, triggering the event handler of the Box add-on, this event handler calls the `GmailApp.getMessageById().getAttachments()` to retrieve attachments of the current Gmail message and uploads them to Box cloud storage.

2.2 Problem Definition

PEDEC focuses on addressing the issue of permission excess, specifically determining whether the add-on unnecessarily requests functionality-irrelevant permissions. To facilitate the understanding of our work, we formally define the target problem.

Definition 1 (Functionality Permissions). *Permissions must be granted to the add-on for it to execute its functionality correctly, denoted as $\mathbb{P}_{fun}$. The functionality of the add-on can be reflected by its source code, runtime behaviour, and textual description, and we use $\mathbb{P}_s$, $\mathbb{P}_r$, and $\mathbb{P}_d$ to indicate the permission sets that are associated with each of them, respectively. Therefore, we have $\mathbb{P}_{fun} = \mathbb{P}_s \cup \mathbb{P}_r \cup \mathbb{P}_d$.*

Definition 2 (Requested Permissions). *The set of permissions that the add-on requests through `oauthScope` field is called requested permissions, denoted as $\mathbb{P}_{req}$. They are specified by developers (the left side of Figure 1) and vetted by platform operators.*

Definition 3 (Excessive Permissions). *We name $\hat{\mathbb{P}} = \mathbb{P}_{req} - \mathbb{P}_{fun}$ as excessive permissions. Intuitively, $\hat{\mathbb{P}}$ is defined as the set of permissions that can be removed from $\mathbb{P}_{req}$ without affecting the normal functionality of the add-on. These permissions are deemed excessive because they are unnecessary for the add-on to execute its intended functionality, and their inclusion raises security and privacy concerns for the users and their data.*

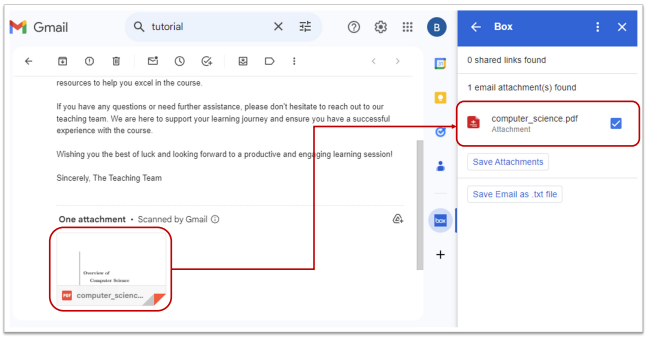

Fig. 2: Example of the Box add-on

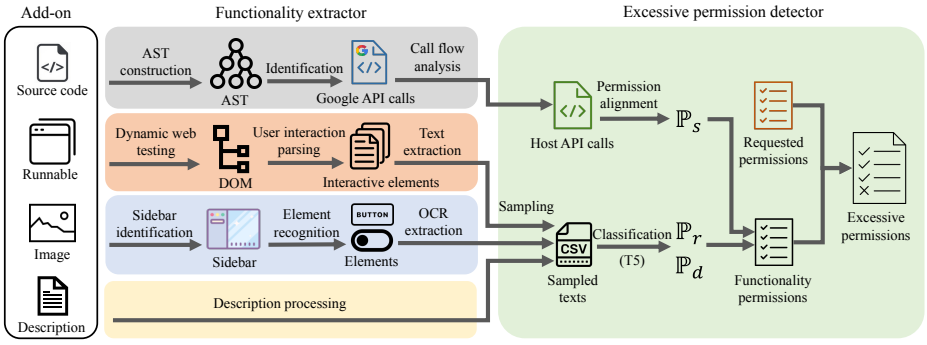

Fig. 3: Workflow of PEDec

3 Our Approach

3.1 Approach Overview

We propose a fully automatic analysis approach, leveraging techniques of hybrid functionality analysis and state-of-the-art natural language inference and Text-To-Text-Transfer-Transformer (T5) models. The overall architecture of our approach is depicted in Figure 3, which composes the following components.

Functionality Extractor. This component aims to extract functionality-related materials. The Document Object Model (DOM) tree of a dynamically generated HTML file provides an accurate hierarchical representation of the available operations on the add-on interface, allowing us to infer permissions related to the add-on's functionality (**addressing Challenge #1**). For the partially available add-ons with source code, similar to Android, specific actions or functions performed through API calls can reveal fine-grained permissions.

Excessive Permission Detector. The second component is an excessive permission detector that compares the functionality permissions based on extracted materials (**Definition 1**) with the declared permissions (**Definition 2**) of add-ons. However, the detection of excessive permissions remains a challenge. Previous efforts [20,23,26,42,47]

have approached this issue using applications' counterparts [41, 42, 49] - applications that share similar functionality - as a reference. However, this approach is not suitable for Google Workspace, given the limited number of add-ons (3,756) compared with other problem settings (millions of mobile and browser applications). Therefore, we propose a text-based technique to extract permission-related information (**addressing Challenge #2**) from application pages, images, and description (to be detailed in Section 3.3).

3.2 Functionality Extractor

Dynamic Analysis for Element Discovery PEDEC extracts functionality-related materials, such as interacted DOM trees, through dynamic web testing for the majority of add-ons in the absence of source code. While web testing has limitations in covering paths requiring semantic-sensitive input, we supplement functionality-related materials with descriptions and tutorial videos provided by add-on developers. These materials serve as guidelines for new users, outlining the supported functionality of add-ons.

Dynamic Web Testing. We perform dynamic web testing on add-ons to trigger a wide range of behaviours and HTML elements, deriving the functionality permissions required by them. For example, buttons related to creating or deleting user documents imply permissions related to document management. To this end, we develop a tester based on Selenium [15] to automatically simulate user interactions. In greater detail, the tester follows a step-by-step approach. Initially, it installs the add-on, initiates its host application (e.g., Gmail), and subsequently activates the add-on to conduct a thorough scan of all user-interactive elements on the interface, such as buttons and input fields. Following that, it sequentially interacts with these elements, mimicking the user's perspective in a usage scenario. To thoroughly explore these pages, the tester employs the depth-first search (DFS) algorithm. All dynamically generated HTML files are recorded at the same time.

To expedite the dynamic analysis process, we restrict the maximum HTML page search depth to ten, based on our observation from a small-scale empirical study that 100% of sampled add-ons (500 randomly sampled) fall within this limit. Previous studies also reveal that add-on developers aim to optimize user experience by constricting the complexity of their products [39, 45, 48].

Enhancing Interaction during Testing. The effectiveness of dynamic interaction is intrinsically limited in scenarios where logical dependencies among events are crucial. For example, certain add-ons require users to set the names of columns in the spreadsheet before launching specific functions. Basic and straightforward dynamic interactions may fail to capture such subtle relationships [32], resulting in problems like the inability to navigate to the next page. Additionally, some add-ons have complex functions that demand domain knowledge (i.e., insert math equations into a Google Doc) and cannot be triggered by simple interaction rules. While manual efforts might address these issues, they are unsuitable for large-scale analysis. As a mitigation strategy, we introduce a novel performance enhancement technique that incorporates add-ons' tutorials and demonstration videos/images.

We start by retrieving all videos and images from the homepage of each add-on. Next, we employ the openCV [10] library to divide the video into image sequences, sampling one frame per second, and filter out identical or similar images. Subsequently, we utilize

the YOLO object recognition method to locate the sidebar of add-on and its interactive elements from the images. Finally, we apply Optical Character Recognition (OCR) to identify the text description for the target interactive elements.

Locating Add-on Sidebar. Figure 2 illustrates how the add-on appears as a sidebar in Gmail, enhancing the functionality of the host application. To accurately recognize the add-on in the derived images, we utilize the state-of-the-art object detection algorithm YOLOv8 [28]. To train the model, we manually label 150 images collected from the Google Workspace Marketplace and split them into training (80%), validation (15%), and test (5%) datasets. Our model achieves a 100% accuracy rate with a total processing and prediction time of less than 7ms. We have publicly shared our dataset[4] for further research.

Locating Interactive Elements. To build our model dataset, we utilize HTML pages successfully triggered during **Dynamic Web Testing** and extract the attributes of interactive elements using Selenium. This dataset includes fundamental details about interactive elements in HTML pages, such as type (e.g., button, input field), explanatory text, and coordinates. Additionally, we convert HTML pages into screenshots using selenium, associating element attributes as the ground truth of each screenshot.

We gather a total of 3,553 different HTML pages containing 18,195 interactive elements during **Dynamic Web Testing**. Using the same experimental settings as in the sidebar location, we achieve a high accuracy of 92%.

Code Analysis for API Call Identification The add-on is developed in Google Script, based on JavaScript, and incorporates Host API specifications for interacting with Google services. While it shares syntax similarities with JavaScript, it introduces host-related features that pose critical challenges to traditional static analyzers. Our goal is to identify **ONLY** the Host API calls that the add-on requires for its functionality, e.g., `GmailApp.getActiveThread().getEmails()` shown in Figure 4. Traditional analysis (e.g., control flow and data flow analysis) of JavaScript code becomes complicated and ineffective for add-ons. Therefore, we use syntax-level parsing to track the call chain in the add-on's script code.

The workflow of the code analysis is summarized in Algorithm 1. First, we leverage esprima [6] to convert all Google Script code of the add-on into Abstract Syntax Trees (ASTs), which offer a clear view of function and class hierarchy. Second, we identify entry-point classes[5] of Host APIs [9] from the ASTs and record all relevant variables (as shown in line 5-12 in Algorithm 1). We track both intra-function (lines 9-12) and inter-function data flow (lines 7-8) to determine all related Google API calls, which serve as references for its $\mathbb{P}_s$.

Intra-function Call Flow. We record variables related to the entry-point classes as local variables within the function (line 10 in Algorithm 1), and then traverse all statements and analyze the `CallExpression` or `AssignExpression` to determine whether any value propagation (e.g., line 1, 9 in Figure 4) or function call occurs (e.g., line 2, 3, 10 in Figure 4). For complex statements (e.g., line 3, 6 in Figure 4), we apply depth-first search (DFS) of the AST to identify the call chain.

[4] https://app.roboflow.com/addon/sidebar

[5] Seven are available, each representing a Workspace app, e.g., DocumentApp and DriveApp.

```
attachment = GmailApp.getMessageById().getAttachments();
attachment.upload('Google Drive Folder');
attachment.getMetadata().getSender().getEmail()

func current_active_thread(){
    return GmailApp.getActiveThread()
}

emails = current_active_thread().getEmails()
emails.delete()
```

Fig. 4: An example of script invoking Google APIs

Inter-function Call Flow. We record the variables that appear in the return statement of a function as global variables along with the function name (line 7-8 in Algorithm 1, line 5-7 in Figure 4). Next, we traverse all statements in other functions, including the main function, to check whether the recorded function name (line 9 in Figure 4) is called. This enables us to identify the flow of call chain between functions in the Google Script code and track the usage of global variables, allowing us to capture dependencies among functions and identify the Google APIs used across functions.

3.3 Excessive Permission Detector

Following the definition of excessive permissions in Section 2.2, we need to explicitly determine the requested and functionality permissions respectively.
Requested Permissions. The requested permissions are listed on the homepage of each add-on, following Google's permission restrictions [5] as per the standard and can be accurately parsed.
Functionality Permissions. The functionality permissions include permissions identified from source code and runtime behaviours of add-ons. The detected APIs from source code also exhibit no ambiguity by referring to Google developer documentation [9], since Google lists the required permission for each API. Unfortunately, determining permissions for runtime behaviours is challenging due to the heterogeneity of our analysis (i.e., testing and interaction enhancement).

As a result, it is not practical to simply detect excessive permissions through well-defined techniques such as keywords [32, 43] or rule-based [18] matching (which may lead to high false positives) or human inspection (which is difficult to scale). To address the challenge in runtime permission detection, we employ a natural language inference method to infer the functionality permissions existing in the extracted materials. We first construct a manually labelled dataset for runtime permissions as the ground truth and train a classifier for accurate permission prediction at runtime.

Data for Determining Runtime Permissions The key challenge in this pertains to the dataset construction, as only a part of the dynamic analysis results provides permission-

Algorithm 1 Code analysis for API calls

Input: Source code C
Output: Array of permission-related $APIs$
1: $AST \leftarrow parse(C), G_{API}Global \leftarrow dict\{\}, G_{API}Local \leftarrow dict\{\}$
2: $G_{API}Entry \leftarrow [CalendarApp, DocumentApp, DriveApp, FormApp, GmailApp, SlidesApp, SpreadsheetApp]$
3: **function** PROC_FUNCTION()
4: **for** $stmt$ in AST **do**
5: $API, G\_related \leftarrow$ GOOGLE_RELATED_API($stmt$)
6: **if** $G\_related = True$ **then**
7: **if** $stmt.type = Return$ **then**
8: $G_{API}Global[function] \leftarrow API$
9: **if** $stmt.type \in Variable$ **then**
10: $G_{API}Local[variable] \leftarrow API$
11: **if** $stmt.type \in Expression$ **then**
12: $APIs \leftarrow APIs \cup API$
13: $APIs \leftarrow APIs \cup G_{API}Local$
14: $G_{API}Local \leftarrow \{\}$
15: **function** GOOGLE_RELATED_API($stmt$)
16: Get API_calls of $stmt$
17: $name \leftarrow API\_calls, G\_related \leftarrow False$
18: **if** $name \in (G_{API}Entry \vee G_{API}Global \vee G_{API}Local)$ **then**
19: $G\_related \leftarrow True$
20: **return** $name, G\_related$

relevant context. For example, many DOM pages are permission-irrelevant, such as add-ons settings, advertisements, and membership purchases. The unevenly distributed permissions further exacerbate the issue, with some permissions being more commonly seen than others. In this work, we leverage the self-descriptive nature of triggered HTML pages and images, which contain contextual and semantic significance to runtime permissions, as discussed by Miniukovich [39]. Based on the technique detailed in Section 3.2, we convert images and HTML pages derived in dynamic analysis into natural language description texts. We also include functionality descriptions along with them for complementary explanations as some dynamically triggered pages might not be self-explanatory. To further preserve the special contextual meaning of DOM elements such as buttons, inputs, and radios, we apply an auxiliary processing method using simple rules [30]. For example, a button named "send an email" would be converted to "This button helps you send an email", while an input with the description "email address" would be converted to "You need to provide an email address to this application".

We rely on the Natural Language Inference (NLI) [34] to determine if the derived texts entail specific permissions. This involves deciding whether a natural language hypothesis can be inferred (i.e., true/entailment) or not (i.e., false/contradiction or undetermined/neutral) from a given premise. For example, consider a premise that states, "This application helps you to edit cell values in your spreadsheet and send an email notification to your collaborators". A hypothesis claiming, "This application can send

emails on your behalf" would be labelled as entailment. As an illustration, we provide some of the used permission hypotheses in Table 1. These hypotheses are derived from Google's official developer documentation without any modification [5]. We utilize the pre-trained NLI model used by a recent study [26]. This model is trained on MultiNLI [12], a dataset comprising 433K sentence pairs from various domains, making it suitable for our functionality-diverse add-ons. We query the NLI model with each pair of add-on's functionality description (i.e., premise) and its requested permission (i.e., hypothesis), then record the returned response from this model (i.e., entailment, neural, or contradiction). After filtering out neutral labels, this results in 7,722 pairs (premise and hypothesis) labelled as entailment or contradiction.

Table 1: Permission hypotheses

Scope	**Hypotheses**
drive.readonly	This add-on can see all of your Google drive files.
	This add-on can download all of your Google drive files.
drive.file	This add-on can see only the specific Google drive files used with itself.
	This add-on an edit only the specific Google drive files used with itself.
	This add-on can create only the specific Google drive files used with itself.
	This add-on can delete only the specific Google drive files used with itself.
drive	This add-on can see all of your Google drive files.
	This add-on can edit all of your Google drive files.
	This add-on can create all of your Google drive files.
	This add-on can delete all of your Google drive files.
spreadsheets.readonly	This add-on can see all your spreadsheets.
spreadsheets	This add-on can see all of your spreadsheets.
	This add-on can edit all of your spreadsheets.
	This add-on can create all of your spreadsheets.
	This add-on can delete all of your spreadsheets.
userinfo.email	This add-on can see your primary Google account email address.
userinfo.profile	This add-on can see your personal info that is publicly available.
gmail.settings.sharing	This add-on can manage your sensitive mail settings, including who can manage your mail.
gmail.send	This add-on can send email on your behalf.
gmail.insert	This add-on can add emails to your Gmail mailbox.

Runtime Permission Detection Due to the uneven distribution of the hypotheses (representing permissions) in the constructed dataset, some permissions receive more entailment or contradiction labels than others. To prevent bias towards any specific permission, we randomly sampled 40 instances (half entailment and half contradiction) for each permission. This results in 800 pairs for manual labelling. To avoid ambiguity in the prediction results, we only label permission as entailment (indicating non-excessive) or contradiction (indicating excessive), avoiding assigning neutral labels.

We proceed with the manual verification of the description for each involved add-on, followed by installation and interaction with it to label the functionality permissions and excessive permissions. The corpus is independently labelled by two authors, both with a relevant computer science background. A tiny proportion of 0.37% (3 out of 800) of hypotheses received different labels from them. In cases of disagreement, they would discuss together to make the final decision. The dataset is split into three parts: 80% for training, 15% for validation, and 5% for testing. We use a large language model named

Text-To-Text Transfer Transformer (abbreviated as T5), known for its good performance in such inference tasks [26], and compare its performance with other mainstream models, including DistilBERT [11] and GPT-based models [4].

- **Fine-tuned T5 model**: We utilize the pre-trained T5-base [16] model and fine-tune it on our labelled corpus. Our model achieves an accuracy of 86.9%.
- **DistilBERT**: We evaluate the DistilBERT [11] model trained on the GLUE dataset and fine-tuned on our corpus. While BERT [29] has been a well-known large language model from 2018 to 2022 (before the rise of ChatGPT), our testing shows that it achieves an accuracy of 82.8%, which is lower than our T5 model.
- **ChatGPT**: We also evaluate the performance of ChatGPT, one of the most popular GPT models. However, as ChatGPT is a black-box model, we apply prompt engineering [33] to guide it. To avoid bias, we use the same prompt as T5 and choose GPT-3.5 [4] as the base model, which is widely used. Despite this, it achieves an accuracy of only 65.6%, possibly due to the domain-specific nature of our task, which may not be well understood by GPT-3.5, as it primarily focuses on question-answering tasks.

4 Evaluation

To evaluate PEDec, we have collected a large-scale dataset of publicly available Google Workspace add-ons (Section 4.1). Following this, we scrutinized the dataset for excessive permission issues, focusing on three key research questions (RQs).

- **RQ1.** What is the performance of PEDec in detecting excessive permissions for add-ons (refer to Section 4.2)?
- **RQ2.** What is the proportion of add-ons that requested excessive permissions, as reported by PEDec (Section 4.3)?
- **RQ3.** What are the root causes for excessive permissions (Section 4.4)?

RQ1 assesses the reliability and effectiveness of PEDec, while **RQ2** provides insights into the compliance of add-ons. Finally, **RQ3** is essential for developers of Google Workspace and add-ons to address the issue of excessive permission requests.

4.1 Dataset Overview

We employ a web crawler to gather data from the Google Workspace Marketplace. Due to the absence of a comprehensive list of add-ons, we leverage its search functionality, utilizing the 10,000 most common English words provided by Google [1] as keywords. Following a de-duplication process, we acquire a total of 3,756 add-ons. Among the acquired add-ons, 123 add-ons are excluded as they are tailored for non-English users. Additional 228 add-ons are identified with publicly accessible source code post-installation in Google developer mode, consequently selected for our code analysis. The majority of the add-ons (1,414) are developed for Google Spreadsheets, with a smaller number designed for Google Forms, as detailed in Table 2. Notably, two-thirds of the add-ons (2,515) are available for free, while 239 add-ons offer free trials, and 198 add-ons feature premium services.

Table 2: Dataset overview

Feature	Category	Number of add-ons
Host	Google drive	681
	Gmail	430
	Google calendar	167
	Google doc	566
	Google sheets	1414
	Google slides	274
	Google forms	171
	Google classroom	12
	No Host	408
Charge	Free	3285
	Paid	34
	Free trial	239
	Free charge paid features	198

Table 3: Performance over sets of `oauthScope` permissions

Permission set	TP	TN	FP	FN	TPR	TNR	Accuracy
See	16	80	3	1	94%	96%	96%
Edit	15	79	3	3	83%	96%	94%
Create	37	58	2	3	93%	97%	95%
Delete	42	56	1	1	98%	98%	98%
Manage	21	71	1	7	75%	99%	92%
Total	**131**	**344**	**10**	**15**	**90%**	**97%**	**95%**

4.2 RQ1: PEDec Accuracy

Ground Truth Annotation. Given the absence of prior work and benchmarks, we manually curate a benchmark dataset. We randomly select 100 add-ons from our collected dataset and engage three annotators with computer science backgrounds—comprising one final-year undergraduate, one master's student, and one PhD student. To standardize the assessment of excessive permissions, we formulate explicit labelling instructions [6], drawing inspiration from established definitions in the literature and common practices [20, 26]. The instructions include definitions for excessive permissions and functionality permissions, aiming to address individual perceptions. To clarify nuanced cases, such as permissions on all files or currently interacted files, we provide illustrative examples. Throughout the annotation process, annotators adhere to the guidelines, install and interact with add-ons using our testing account, and identify instances of excessive permissions. In disagreements, annotators and we collectively resolve differences, seeking input from a security expert with eight years of experience (2015-) as needed. The Kappa score for annotator agreement is 0.844, signifying nearly perfect agreement (above 0.81) according to established criteria [38].

PEDec Performance. We initiate our analysis with the 100 annotated add-ons. In Table 3, we present the true positive (TP), false positive (FP), true negative (TN), and false negative (FN) rates of PEDec. We define the excessive permission as positive and the non-excessive permission as negative. Thus, TP denotes that the excessive permission

[6] Available at https://github.com/CoTestAccount/third-party-application

is detected by both PEDEC and human annotators, while FP indicates that PEDEC marks the permission as excessive, whereas human annotators determine it as non-excessive.

PEDEC achieves a promising performance, we evaluate PEDEC's performance on each positive and negative case using the top-5 [7] requested permission sets. As shown in Table 3, PEDEC achieves an average of 90% TPR and 97% TNR. Notably, for the highly sensitive permission like *delete*, PEDEC can achieve 98% TPR and 98% TNR. The performance of PEDEC on the add-on level is presented in Table 4. It showcases the ability to accurately identify add-ons with (or without) excessive permissions, achieving a high accuracy of 95%.

Table 4: Overall performance of PEDEC on Ground Truth

TP	TN	FP	FN	TPR	TNR	Accuracy
61	34	0	5	92%	100%	95%

We conduct a manual inspection to investigate the low TPR associated with the "manage" permission, as illustrated in Table 3. Discussions with annotators reveal that the complexity lies in the compound nature of "manage", encompassing a series of permissions that vary across different host applications. Further scrutiny of the Google Workspace developer page [5] unveils that "manage" includes permissions like "modify" and "share with other users spreadsheets that this application has been installed in". While we did sample "manage" hypotheses in Section 3.3 and assigned the correct label as training data, our model struggles to capture such fine-grained scopes without additional information. As depicted in Table 3, "manage" exhibits a relatively low TPR compared to others. A potential solution involves splitting the compound permission into a series of permissions, as demonstrated in Table 5, necessitating additional manual effort for corpus relabeling.

API Call Identification Performance. We additionally evaluate the accuracy of our lightweight code analysis tool. We randomly sample 20% (50 out of 228) of add-ons and perform a manual inspection of the code analysis performance. Leveraging Google's detailed mapping from each Host API to the required permissions in the Google Developer Documentation [9], we employ the fine-grained Host API Coverage (detector/ground-truth) as a metric to gauge PEDEC's effectiveness in capturing all functionality-related Host API.

PEDEC fails to detect API calls in four add-ons. After investigating through source code screening, we discover that two add-ons employ code obfuscation, rendering PEDEC designed for readable source code ineffective. Additionally, two add-ons utilize outdated APIs no longer supported by Google. For the remaining 46 add-ons, the API coverage stands at 100%.

4.3 RQ2: Landscape of Add-ons

We then apply PEDEC to analyze all collected add-ons. It identifies that 56% (2,091 out of 3,756) of them have excessive permissions. Table 7 breaks down the service with excessive permissions across different installation ranges and particular permissions. The most common (40%, 1,484 out of 3,756) excessive permission is *delete*, where the

[7] The remaining one is "send', comprising only one record.

Table 5: Manage under different scenarios

Manage permission	**Compound permissions**
Manage spreadsheets that this add-on has been installed in.	Modify spreadsheets that this add-on has been installed in. Share with other users spreadsheets that this add-on has been installed in.
Manage your forms in Google drive.	Create new forms. Modify existing forms. Submit form responses. Process form responses. Share forms with others.
Manage forms that this add-on has been installed in	Modify forms that this add-on has been installed in. Share with other users forms that this add-on has been installed in. Process responses to forms that this add-on has been installed in.

Table 6: Co-occurrence analysis of excessive permission set

Co-occurrence	**See**	**Edit**	**Create**	**Delete**	**Manage**
See	772	587	635	660	130
Edit	-	631	611	622	113
Create	-	-	1257	1220	172
Delete	-	-	-	1484	186
Manage	-	-	-	-	603

Co-occurrence	**Sheets**	**Slides**	**Docs**	**Form**	**Gmail**
Sheets	1204	101	229	87	16
Slides	-	246	101	29	4
Docs	-	-	532	52	5
Form	-	-	-	158	4
Gmail	-	-	-	-	45

Table 7: Proportion of excessive permission requested by add-ons. “E” represents “Excessive” (i.e., the number of add-ons that contain excessive permissions), “N” represents “Non-Excessive” (i.e., the number of add-ons that contain no excessive permissions) ,and “T” represents “Total” (i.e., the total number of add-ons)

App	**Add-ons' installation distribution**									**Add-ons with excessive permissions on the `oauthScope` set**								
	0-1000			1000-1 Million			1 Million+			See		Edit		Create		Delete		Manage	
	E	N	E/T	E	N	E/T	E	N	E/T	E	E/T	E	E/T	E	E/T	E	E/T	E	E/T
Gmail	39	127	0.23	65	178	0.27	8	13	0.38	71	0.17	25	0.06	35	0.08	55	0.13	17	0.04
Google Docs	111	35	0.76	259	90	0.74	56	15	0.79	150	0.27	128	0.23	275	0.49	290	0.51	157	0.28
Google Sheets	417	142	0.75	598	173	0.78	68	16	0.81	353	0.25	306	0.22	713	0.51	835	0.59	342	0.24
Google Slides	41	14	0.75	149	27	0.85	36	7	0.84	102	0.37	78	0.28	156	0.57	163	0.59	82	0.30
Google Forms	11	2	0.85	84	30	0.74	35	9	0.80	44	0.26	44	0.26	70	0.41	73	0.43	99	0.58
Google Calendar	27	43	0.39	40	50	0.44	1	6	0.14	29	0.17	11	0.07	12	0.07	41	0.24	7	0.04
Google Drive	38	50	0.43	153	432	0.26	26	68	0.28	150	0.20	91	0.12	127	0.17	136	0.18	36	0.05
No Host	47	88	0.35	137	12	0.92	12	17	0.41	92	0.23	52	0.13	59	0.14	114	0.28	11	0.03

add-on does not provide delete files or folder functionality but requests this permission. We observe that the issue of excessive permissions is quite prevalent at 17% for editing files, 16% for managing (including sharing) files, and 20% for seeing files. Despite its prevalence, excessive permissions remain a significant issue in many widely-used add-ons, particularly those with over one million installations. Table 7 shows that add-ons designed for Google Docs, Sheets, Slides, and Forms have a higher proportion with excessive permissions, considering their rich functionality. We conclude the root cause in **RQ3** as an exploration and alert for developers of Google Workspace and add-ons.

We conduct an analysis of the distribution of co-occurrence of detected excessive permissions, as depicted in Table 6. To streamline the presentation, we exclusively showcase the upper triangular matrix due to its symmetrical nature. It is noteworthy that when the permission "edit" is present, "delete" is identified as an excessive permission in nearly 98.6% of cases (622 out of 631). Although "delete" is the most frequently detected excessive permission, it often co-occurs with "edit" (41.9%, 622 out of 1484), "manage" (12.5%), and "see" (44.5%). Additionally, a noteworthy observation is that a substantial portion of add-ons designed for Google Sheets are flagged for excessive permissions, frequently requesting such permissions for Docs (19.0%, 229 out of 1204), Slides (8.4%), and Gmail (1.3%).

4.4 RQ3: Root Causes (RCs) and Case Study

After discovering that approximately half of the add-ons request excessive permissions, we delve deeper into their Root Causes (RCs) to gain a comprehensive understanding of excessive permissions. We randomly select and manually inspect 50 (out of 2,091) services with excessive permissions. Based on the types of RCs, we categorize them into the following three groups. The first two types stem from permission management issues on the platform operator's side (i.e., Google in our study), while the last type is associated with poor implementation practices from the add-on developers. Each type of RC is elucidated through detailed example.

RC1: Permission Bundle (30 add-ons). When permissions are grouped into bundles, there is an increased risk of excessive permissions. Specifically, the add-on has to request the entire group of permissions even if only part of them is functionality-required. This all-or-nothing implementation of permissions contributes to security threats in Google Workspace [21]. This Root Cause (RC) is the primary factor behind excessive permissions in popular add-ons like Box (more than one million installations, 1M+ for short hereafter), DocuSign eSignature (2M+), and Auto-LaTex Equations (10M+). Based on their types, the permission bundle can be further classified into two categories: *bundled operations* and *bundled objects*. MathType (10M+) functions as a math equations and chemical formulas editor for Google Docs. The correct permissions for its functionality are "see" and "modify". However, the issue arises as "see" and "modify", and "share" permissions are bundled into one group. This bundling requires MathType to request the excessive "share" permission, even though it is not strictly necessary for its functionality.

RC2: Coarse-grained Permission Management (20 add-ons). Google's current permission management system employs a general and broad permission coverage, inadvertently exposing more data than intended by the developers. Zoom for Gmail (12M+) enables users to respond to meeting schedules. Only the email title is needed for setting

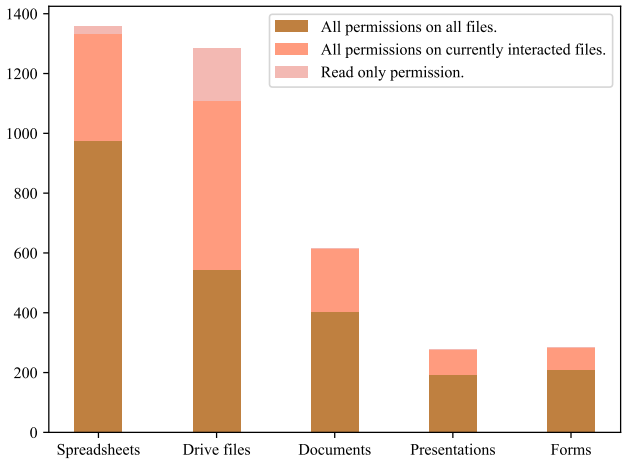

Fig. 5: Distribution of permission scope

the meeting topic, and the remaining content (i.e., body content, attachment) is non-relevant to the functionality. However, Google provides no fine-grained permission to exclusively view the email title. By requesting the "see" permission, Zoom can access all the content of emails.

RC3: Extra Permissions Requested by Add-ons (12 add-ons). There is a tendency for the add-ons to frequently request excessive permissions under a loosely managed permission scheme [47]. As depicted in Figure 5, around two-thirds of add-ons request "All permissions on all files" instead of the more restricted permission option of "All permissions on currently interacted files". Bjorn's NAV-thumbs (607) offers features for navigation in Google Slides. While it requires permission to "see and edit all your Google Slides presentations" for its functionality, it also requests excessive permissions permissions such as "see, edit, create and delete all your Google Docs documents", raising privacy and security concerns. It is alarming that such requested permissions passed the vetting process.

5 Limitations and Discussion

5.1 Limitations

Quality of Tutorial Images and Videos. Developers are required to supply demonstration images, but the use of low-quality images may adversely affect PEDEC performance. Our evaluation indicates that developers of popular services with a large user base tend to offer high-quality images and comprehensive demonstration videos. In contrast, developers with a smaller user base may not provide clear tutorials.

NLP Performance. PEDEC relies on the functionality materials and employs the Natural Language Inference (NLI) and T5 methods to extract excessive permissions. However,

there is still a failure rate of 13% due to inherent limitation of Natural Language Processing (NLP) techniques.

5.2 Discussion

PEDEC heavily relies on dynamic analysis of runtime behavior and NLP techniques to infer excessive permissions. Each time the add-ons receive a new update, the tool must be rerun to check for excessive permissions. In the future, we plan to explore strategies that only consider the new update to re-detect excessive permissions. Additionally, since Google frequently updates their host API documentation, PEDEC should be able to capture such updates in a timely manner.

6 Related Work

Static Code Analysis. To assess the security of software permissions and safeguard user data, many researchers explore static code analysis [19] and API call graphs. Taint analysis is widely employed in add-ons for mobile apps, identifying numerous permission escalation [21, 45]. Given our emphasis on Host API call detection, existing call-graph or data-flow analysis designed for JavaScript, as demonstrated in previous research [32], are unsuitable for our tasks. In our work, we employ a lightweight API call analysis to recognize all functionality-related Host API calls.

Dynamic Web Testing and Enhancement. To ensure the quality and usability of web applications, many researchers explore dynamic testing in both black-box and white-box contexts [31–33, 35, 37]. Given that human interaction serves as the primary communication between web applications and users [39], the most common automated testing methods [40] are based on user profiles (e.g., name, gender and license number) or scenarios (e.g., shopping websites). Gao et al. [24] investigate the ecosystem of account registration bots. Their experiments suggest that the most prevalent human verification methods, including SMS, CAPTCHA, and IP restrictions, can be successfully bypassed by anti-human verification services. However, owing to the complexity and customization of modern web applications, dynamic automated GUI testing tools [33] that simulate human operations (e.g., login, click or scroll) cannot guarantee the triggering of all web paths. We acknowledge this inherent limitation and utilize tutorial images/videos as supplements.

Excessive Permission Detection in Android. Many studies have analyzed excessive permissions in Android apps [41, 42, 46, 49]. Whyper [41] and Autocog [42] leverage the app's description and counterpart to infer permissions that are irrelevant to functionality. Pscout [19] and Taintmini [45] construct call graphs and track sensitive data flow to detect excessive permissions. These approaches require access to source codes or a large app base for accuracy and reliability in clustering app functionality. However, addressing the domain-specific challenges existing in Google Workspace is necessary due to the limited size and closed nature of the data.

7 Conclusion

The add-on enhances customization options but also introduces new security concerns related to excessive permissions. They may request access to more permissions than necessary, potentially putting user's data at risk. It is important to carefully review the permissions requested by the add-on and consider whether they are necessary for its functionality. To our best knowledge, no prior work has provided such automated checks for excessive permissions of the add-on that may lead to security issues, and our work addresses this gap. In this paper, we design a tool called PEDEC to automatically check whether the add-on requests excessive permissions. Ground truth evaluation shows that PEDEC performs well, achieving 100% TNR and 92% TPR in checking excessive permissions.

Acknowledgement

We would like to thank anonymous reviewers for improving this manuscript. This research has been partially supported by the University of Queensland and the Huazhong University of Science and Technology.

References

1. 10,000 most common English words Repo. https://github.com/first20hours/google-10000-english. Online; Accessed: 2023.
2. Auto-Latex Equation. https://workspace.google.com/marketplace/app/autolatex_equations/850293439076. Online; Accessed: 2023.
3. California Consumer Privacy Act. https://oag.ca.gov/privacy/ccpa. Online; Accessed: 2023.
4. ChatGPT fine tune reference. https://platform.openai.com/docs/api-reference/fine-tunes. Online; Accessed: 2023.
5. Developer Reference: OAuth 2.0 scopes. https://developers.google.com/identity/protocols/oauth2/scopes. Online; Accessed: 2023.
6. Esprima. https://esprima.org. Online; Accessed: 2023.
7. General Data Protection Regulation . https://gdpr-info.eu/. Online; Accessed: 2023.
8. Google Workspace Market Share. https://6sense.com/tech/office-suites/google-workspace-market-share. Online; Accessed: 2023.
9. Google Workspace Reference Overview. https://developers.google.com/apps-script/reference/. Online; Accessed: 2023.
10. Homepage of OpenCV. https://opencv.org/. Online; Accessed: 2023.
11. Hugging Face DistilBERT. https://huggingface.co/docs/transformers/model_doc/distilbert. Online; Accessed: 2023.
12. HuggingFace: roberta-large-mnli model. https://huggingface.co/roberta-large-mnli. Online; Accessed: 2023.
13. No Phishing! https://workspace.google.com/marketplace/app/no_phishing/149706744667. Online; Accessed: 2023.
14. OAuth 2.0 Rich Authorization Requests . https://datatracker.ietf.org/doc/html/draft-ietf-oauth-rar-12. Online; Accessed: 2023.
15. Selenium Dev. https://www.selenium.dev/. Online; Accessed: 2023.

16. T5 base. `https://huggingface.co/t5-base`. Online; Accessed: 2023.
17. Zoom Apps. `https://marketplace.zoom.us/`. Online; Accessed: 2023.
18. Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. Policylint: Investigating internal privacy policy contradictions on google play. In *USENIX Security Symposium*, pages 585–602, 2019.
19. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. Pscout: analyzing the android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228, 2012.
20. David G Balash, Xiaoyuan Wu, Miles Grant, Irwin Reyes, and Adam J Aviv. Security and privacy perceptions of {Third-Party} application access for google accounts. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3397–3414, 2022.
21. Yunang Chen, Yue Gao, Nick Ceccio, Rahul Chatterjee, Kassem Fawaz, and Earlence Fernandes. Experimental security analysis of the app model in business collaboration platforms. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2011–2028, 2022.
22. Jide Edu, Cliona Mulligan, Fabio Pierazzi, Jason Polakis, Guillermo Suarez-Tangil, and Jose Such. Exploring the security and privacy risks of chatbots in messaging services. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 581–588, 2022.
23. Zheran Fang, Weili Han, and Yingjiu Li. Permission based android security: Issues and countermeasures. *computers & security*, 43:205–218, 2014.
24. Yuhao Gao, Guoai Xu, Li Li, Xiapu Luo, Chenyu Wang, and Yulei Sui. Demystifying the underground ecosystem of account registration bots. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 897–909, 2022.
25. Hamza Harkous and Karl Aberer. " if you can't beat them, join them" a usability approach to interdependent privacy in cloud apps. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 127–138, 2017.
26. Hamza Harkous, Sai Teja Peddinti, Rishabh Khandelwal, Animesh Srivastava, and Nina Taft. Hark: A deep learning system for navigating privacy feedback at scale. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2469–2486. IEEE, 2022.
27. Louis Jannett, Vladislav Mladenov, Christian Mainka, and Jörg Schwenk. Distinct: Identity theft using in-browser communications in dual-window single sign-on. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1553–1567, 2022.
28. Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, 2023.
29. Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2, 2019.
30. Rishabh Khandelwal, Thomas Linden, Hamza Harkous, and Kassem Fawaz. {PriSEC}: A privacy settings enforcement controller. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 465–482, 2021.
31. Yuxi Ling, Yun Hao, Yuyan Wang, Kailong Wang, Guangdong Bai, and Jin Song Dong. Essential or excessive? mindaext: Measuring data minimization practices among browser extensions.
32. 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 *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2022.
33. Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. Fill in the blank: Context-aware automated text input generation for mobile gui testing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1355–1367. IEEE, 2023.

34. Bill MacCartney. *Natural language inference*. Stanford University, 2009.
35. Kulani Mahadewa, Kailong Wang, Guangdong Bai, Ling Shi, Yan Liu, Jin Song Dong, and Zhenkai Liang. Scrutinizing implementations of smart home integrations. *IEEE Transactions on Software Engineering*, 47(12):2667–2683, 2019.
36. Kulani Mahadewa, Yanjun Zhang, Guangdong Bai, Lei Bu, Zhiqiang Zuo, Dileepa Fernando, Zhenkai Liang, and Jin Song Dong. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 2–15, 2021.
37. Kulani Tharaka Mahadewa, Kailong Wang, Guangdong Bai, Ling Shi, Jin Song Dong, and Zhenkai Liang. Homescan: Scrutinizing implementations of smart home integrations. In *2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 21–30. IEEE, 2018.
38. Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia medica*, 22(3):276–282, 2012.
39. Aliaksei Miniukovich, Antonella De Angeli, Simone Sulpizio, and Paola Venuti. Design guidelines for web readability. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, pages 285–296, 2017.
40. Olfa Nasraoui, Maha Soliman, Esin Saka, Antonio Badia, and Richard Germain. A web usage mining framework for mining evolving user profiles in dynamic web sites. *IEEE transactions on knowledge and data engineering*, 20(2):202–215, 2007.
41. Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. Whyper: Towards automating risk assessment of mobile applications. In *USENIX Security Symposium*, volume 2013. Washington, DC, 2013.
42. Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. Autocog: Measuring the description-to-permission fidelity in android applications. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1354–1365, 2014.
43. Liuhuo Wan, Kailong Wang, Kulani Tharaka Mahadewa, Haoyu Wang, and Guangdong Bai. Don't bite off more than you can chew: Investigating excessive permission requests in trigger-action integrations. In *Proceedings of the ACM Web Conference 2024*, 2024.
44. Liuhuo Wan, Kailong Wang, Haoyu Wang, and Guangdong Bai. Is it safe to share your files? an empirical security analysis of google workspace. In *Proceedings of the ACM Web Conference 2024*, 2024.
45. Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 932–944. IEEE, 2023.
46. Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. Malwhiteout: Reducing label errors in android malware detection. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–13, 2022.
47. Mingming Zha, J Wang, et al. Hazard integrated: Understanding security risks in app extensions to team chat systems. In *Network and Distributed Systems Security (NDSS) Symposium*, 2022.
48. Jianyi Zhang, Leixin Yang, Yuyang Han, Zixiao Xiang, and Xiali Hei. A small leak will sink many ships: Vulnerabilities related to mini-programs permissions. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 595–606. IEEE, 2023.
49. Lu Zhou, Chengyongxiao Wei, Tong Zhu, Guoxing Chen, Xiaokuan Zhang, Suguo Du, Hui Cao, and Haojin Zhu. {POLICYCOMP}: Counterpart comparison of privacy policies uncovers overbroad personal data collection practices. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1073–1090, 2023.