Matcha: An IDE Plugin for Creating Accurate Privacy Nutrition Labels

TIANSHI LI, Northeastern University, USA
LORRIE FAITH CRANOR, Carnegie Mellon University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA

Apple and Google introduced their versions of privacy nutrition labels to the mobile app stores to better inform users of the apps' data practices. However, these labels are self-reported by developers and have been found to contain many inaccuracies due to misunderstandings of the label taxonomy. In this work, we present Matcha, an IDE plugin that uses automated code analysis to help developers create accurate Google Play data safety labels. Developers can benefit from Matcha's ability to detect user data accesses and transmissions while staying in control of the generated label by adding custom Java annotations and modifying an auto-generated XML specification. Our evaluation with 12 developers showed that Matcha helped our participants improved the accuracy of a label they created with Google's official tool for a real-world app they developed. We found that participants preferred Matcha for its accuracy benefits. Drawing on Matcha, we discuss general design recommendations for developer tools used to create accurate standardized privacy notices.

ACM Reference Format:
Tianshi Li, Lorrie Faith Cranor, Yuvraj Agarwal, and Jason I. Hong. 2024. Matcha: An IDE Plugin for Creating Accurate Privacy Nutrition Labels. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 1, Article 33 (March 2024), 38 pages. https://doi.org/10.1145/3643544

1 INTRODUCTION

Privacy nutrition labels are short, uniform, machine-readable notices that allow users to learn about how their data is collected and used at a glance [12, 21, 23]. Inspired by this idea, the two main mobile app stores, Apple app store and Google Play store, introduced a privacy label section in 2020 and 2022 respectively.[1] This section is displayed on the public page of each app and is self-reported by the app developers. As of August 26, 2022, 60% of apps on the Apple app store and 44% of apps on the Google Play store have filled out forms to create these labels. However, researchers [3, 24, 29, 30, 45] and consumer advocates [14] have raised numerous concerns about these labels given their current design. One key concern revolves around the accuracy of the labels, which could fundamentally undermine the entire effort if consumers lose confidence in the stated data practices.

Currently, developers alone are responsible for accurately creating the privacy label. However, recent research found that this seemingly straightforward task was very challenging for developers [29]. First, developers' understanding of their app's data practices may be incorrect or incomplete due to memory errors or unexpected data collection from third-party libraries [4, 5, 26, 34, 39]. Second, developers' misinterpretations of the label terminology could cause errors when they translate the understanding of the app to the privacy label [29].

[1]"App privacy details" on the Apple app store and "Data safety section" on the Google Play store

Authors' addresses: Tianshi Li, Northeastern University, Pittsburgh, USA, tia.li@northeastern.edu; Lorrie Faith Cranor, Carnegie Mellon University, Pittsburgh, USA, lorrie@cmu.edu; Yuvraj Agarwal, Carnegie Mellon University, Pittsburgh, USA, yuvraj@cs.cmu.edu; Jason I. Hong, Carnegie Mellon University, Pittsburgh, USA, jasonh@cs.cmu.edu.

2474-9567/2024/3-ART33
https://doi.org/10.1145/3643544

Using automated program or network analysis can provide insights into an app's data usage, but it can not replace developers in this process. The output of these analyses are not directly useful to end users and need to be converted to more high-level summaries of data practices. However, automated techniques for detecting data flows [2, 13] and categorizing them into concepts such as data types [19, 35] and purposes [20, 42] inevitably suffer from imperfect accuracy. Furthermore, after data leaves the device, it is infeasible to determine how data is stored, shared, or repurposed without access to the software backend. As the commercial privacy labels embrace a comprehensive design that also covers server-side data rentions and sharing, the developer's knowledge is needed to elucidate the detailed data usage and correct errors made by the automated analysis.

This paper explores a new design space for tools that can improve the accuracy of standardized privacy notices by leveraging the synergies between developers and automated analysis. We present Matcha, a plugin for the Android Studio IDE to help developers create accurate Google Play data safety labels. Matcha analyzes an app's codebase and provides suggestions about first-party code that accesses or transmits user data based on APIs and keywords. Matcha also automatically detects popular third-party SDKs that collect or share user data, and helps pre-fill part of the privacy labels based on the privacy information provided by the third-party developers. Then it asks developers to confirm or reject the suggestions by adding custom Java annotations and editing an auto-generated XML spec file for first-party and third-party data practices respectively. Finally, Matcha uses the annotations and XML to generate a CSV file that can be imported into the developer console to create the label.

The design of Matcha is inspired by prior research on privacy-enhancing IDE plugins, in which annotations has helped increase the developer's awareness of privacy issues and reinforce best practices [26], facilitate the documentation of data practices [26], and streamline the implementation of privacy features [28]. However, the annotation design of prior research has been relatively simple, requesting information in a more open-ended format. In contrast, the creation of privacy labels requires much more comprehensive and standardized information and can be subject to developers' misunderstandings and knowledge gaps [29]. With Matcha, we aim to investigate the following problem: How can we apply the annotation-based approach to help developers overcome the limitations in their capacities and create an *accurate* privacy label?

The design challenge lies in how to achieve a good balance between reducing developers' burden and soliciting accurate information from them. To achieve this goal, we first analyzed Google's data safety label to design the annotations and the XML spec that covers all the required information for generating the label. We then conducted preliminary tests for iterative design and found that developers' overconfidence and incorrect mental model of what the plugin can or cannot do made them reject correct suggestions by the plugin. Furthermore, we noticed that our participants generally lacked basic knowledge about Java annotations, which became another barrier to providing accurate information. These findings informed our final design of the Matcha IDE plugin.

Creating privacy labels for an app requires significant knowledge about its implementation. Therefore, we evaluated Matcha with 12 developers working on their own apps. Matcha helped 11 out of the 12 participants improve the accuracy of their data safety labels as compared to filling out forms on the Google Play developer console. Our analysis showed that Matcha was effective in addressing errors due to misunderstanding of the task of creating data safety labels, misunderstanding of the third-party libraries' data practices, forgetfulness, and misunderstanding of code behavior. All participants favored Matcha to the baseline due to improved label accuracy, user-friendly interface, learnability, and the educational benefit of learning more about their app's data practices. Drawing on our experiences, we discuss design recommendations for developer tools for creating accurate standardized privacy notices.

We make the following contributions:

- The design and implementation of Matcha, an IDE plugin for helping developers create accurate Google Play data safety labels. Our annotation-based approach to creating privacy labels the IDE plugin design can address the information overload and the misunderstandings of privacy label terms, resulting in improved accuracy of privacy labels. Our plugin and source code are available at https://matcha-ide.github.io

- Evaluation studies with Android developers working on their own real-world apps ($N = 12$), demonstrating the efficacy and usability of Matcha.
- Design recommendations for developer tools for creating accurate standardized privacy notices.

1.1 Matcha Use Case

The example below demonstrates the typical workflow for using Matcha to create a data safety label. Carol needs to create a data safety label for her app. She tries the default approach, which is to fill out forms on the Google Play developer console. However, the task is overwhelming and she is not sure whether she has answered all the questions correctly. Then she discovers Matcha and gives it a try.

Matcha analyzes the app's codebase and identifies API calls that access sensitive user data and send data off the device and asks her to add an annotation for each API call. Carol clicks on a detected API call (Figure 1A) and is navigated to the corresponding line of code in the code editor. She uses Matcha's quickfix (an IDE feature for repairing code issues, see Figure 1B) to add a `@DataAccess` annotations (Figure 1C). The API call accesses the search queries entered by the user, so she selects the data type "In App Search History" from a list of predefined options. Similarly, she checks another detected API call that sends the search queries out of the device and also uses the quickfix feature to add a `@DataTransmission` annotation that describes the "In App Search History" data flows from the source represented by the data access annotation to the sink represented by the transmission annotation and provides further information about why the data is being sent and how it is used after leaving the device (Figure 1D).

Finally she reviews the third-party SDKs' data practices detected by Matcha. Matcha informs her of data that is always collected and shared by the third-party SDKS integrated in her app, as well as, optional data collection and sharing which depends on her configuration of the SDKs. Carol reviews an XML file automatically generated by Matcha which details all the potential data collection and sharing of each detected SDK and the trigger conditions (Figure 1E). She considers the user name collection conditions of the Firebase Authentication SDK irrelevant and removes the corresponding `<data>` tag. After verifying all instances, she sets the attribute `verified` to `true` to indicate the completion status to Matcha.

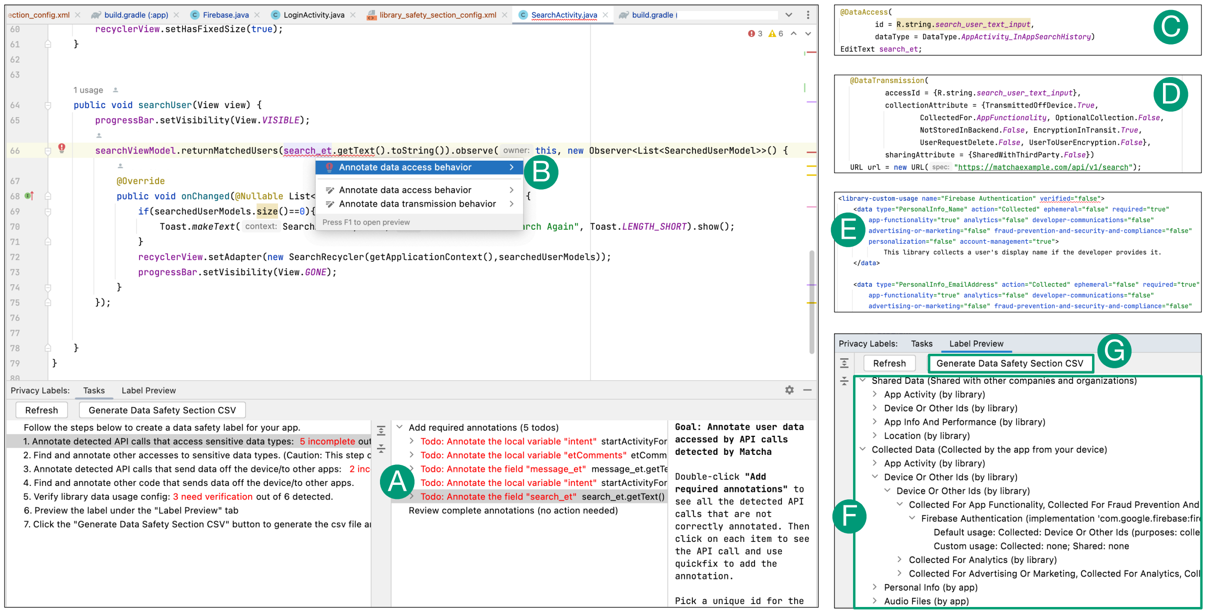
After providing all the required information, Carol opens the "Label Preview" view to see the resulting label (Figure 1F). She notices that her app both collects and shares data, while the sharing is only caused by the third-party SDKs in the app. She also learns that data accessed and processed on device does not need to be reported as *data collection* according to Google's definition of the term. She then clicks the "Generate Data Safety Section CSV" button (Figure 1G) to export the data safety label into a CSV, which she later uploads to the Google Play developer console to fulfill the requirement.

2 BACKGROUND AND RELATED WORK

2.1 Large-scale adoption of privacy nutrition labels: Opportunities and Challenges

Website privacy policies are notoriously long and difficult to read [33]. To tackle the problem, researchers proposed "privacy nutrition labels" more than a decade ago, to offer a clear, uniform, and succinct format for disclosing data usage. Many variants have been proposed for websites [21], mobile apps [23], and IoT devices [12]. Prior research has shown that standardized labels can help users find information about how their data is used faster [22], improve comprehension of privacy practices [22], and nudge consumers to make more privacy-conscious purchase choices [12, 23].

Apple introduced the App Privacy section to the Apple App Store in December 2020, marking the first large-scale deployment of privacy nutrition labels in real life. Google followed with the Data Safety section, their



```
        recyclerView.setHasFixedSize(true);
    }



    public void searchUser(View view) {
        progressBar.setVisibility(View.VISIBLE);

        searchViewModel.returnMatchedUsers(search_et.getText().toString()).observe( owner: this, new Observer<List<SearchedUserModel>>() {

            @Override
            public void onChanged(@Nullable List<                                                      {
                if(searchedUserModels.size()==0){
                    Toast.makeText( context: Search                                    rch Again", Toast.LENGTH_SHORT).show();
                }
                recyclerView.setAdapter(new SearchRecycler(getApplicationContext(),searchedUserModels));
                progressBar.setVisibility(View.GONE);
            }
        });


    }
}
```

Follow the steps below to create a data safety label for your app.
1. Annotate detected API calls that access sensitive data types: 5 incomplete out
2. Find and annotate other accesses to sensitive data types. (Caution: This step c
3. Annotate detected API calls that send data off the device/to other apps: 2 inc
4. Find and annotate other code that sends data off the device/to other apps.
5. Verify library data usage config: 3 need verification out of 6 detected.
6. Preview the label under the "Label Preview" tab
7. Click the "Generate Data Safety Section CSV" button to generate the csv file a

Add required annotations (5 todos)
- Todo: Annotate the local variable "intent" startActivityFor
- Todo: Annotate the local variable "etComments" etComm
- Todo: Annotate the field "message_et" message_et.getTe
- Todo: Annotate the local variable "intent" startActivityFor
- Todo: Annotate the field "search_et" search_et.getText()

Review complete annotations (no action needed)

```
Goal: Annotate user data
accessed by API calls
detected by Matcha

Double-click "Add
required annotations" to
see all the detected API
calls that are not
correctly annotated. Then
click on each item to see
the API call and use
quickfix to add the
annotation.

Pick a unique id for the
```

```
@DataAccess(
        id = R.string.search_user_text_input,
        dataType = DataType.AppActivity_InAppSearchHistory)
EditText search_et;
```

```
@DataTransmission(
        accessId = {R.string.search_user_text_input},
        collectionAttribute = {TransmittedOffDevice.True,
                CollectedFor.AppFunctionality, OptionalCollection.False,
                NotStoredInBackend.False, EncryptionInTransit.True,
                UserRequestDelete.False, UserToUserEncryption.False},
        sharingAttribute = {SharedWithThirdParty.False})
URL url = new URL( spec: "https://matchaexample.com/api/v1/search");
```

```
<library-custom-usage name="Firebase Authentication" verified="false">
    <data type="PersonalInfo_Name" action="Collected" ephemeral="false" required="true"
        app-functionality="true" analytics="false" developer-communications="false"
        advertising-or-marketing="false" fraud-prevention-and-security-and-compliance="false"
        personalization="false" account-management="true">
            This library collects a user's display name if the developer provides it.
    </data>

    <data type="PersonalInfo_EmailAddress" action="Collected" ephemeral="false" required="true"
        app-functionality="true" analytics="false" developer-communications="false"
        advertising-or-marketing="false" fraud-prevention-and-security-and-compliance="false"
```

- Shared Data (Shared with other companies and organizations)
  - App Activity (by library)
  - Device Or Other Ids (by library)
  - App Info And Performance (by library)
  - Location (by library)
- Collected Data (Collected by the app from your device)
  - App Activity (by library)
  - Device Or Other Ids (by library)
    - Device Or Other Ids (by library)
      - Collected For App Functionality, Collected For Fraud Prevention And
        - Firebase Authentication (implementation 'com.google.firebase:fir
          - Default usage: Collected: Device Or Other Ids (purposes: colle
          - Custom usage: Collected: none; Shared: none
      - Collected For Analytics (by library)
      - Collected For Advertising Or Marketing, Collected For Analytics, Coll
  - Personal Info (by app)
  - Audio Files (by app)

Fig. 1. An overview of the main features of Matcha. Matcha detects API calls that access user data and transmit data out of the app, as well as 3rd-party SDKs that collect and share user data. Then it guides developers to confirm, refine, or reject the suggestions by adding custom Java annotations and modifying an auto-generated XML file, which account for first-party and third-party data practices respectively. Finally, Matcha generates a CSV file that can be uploaded to create the safety label.

version of privacy nutrition labels, to the Google Play Store in May 2022. The introduction of privacy labels to the two major app stores has multiple potential benefits. First, users can directly gain a better understanding of an app's data use [22, 33]. Second, it gives developers a systematic and structured way to disclose their data practices to end users [29]. Third, the standardized and machine-readable format facilitates the research and deployment of novel formats of privacy notices to help users further synthesize, analyze, and compare app data practices [38].

However, researchers have identified numerous problems with these privacy labels. One issue is the prevalent inaccuracies in these labels. Balash et al. [3] found that many apps seemed likely to collect user data but did not declare any data collection in their labels. Li et al. [30] suggested that many Apple privacy labels may be outdated. Kollnig et al. [24] found that many apps used tracking libraries and sent data to known tracking domains but reported no data collected. Xiao et al. [44] found apps whose data flows were inconsistent with their privacy labels. Other work focuses on usability issues. Zhang et al. [45] interviewed 24 lay users about the Apple privacy labels and uncovered problems with the usability, understandability, and effectiveness of these labels. Li et al. [29] studied the usability and understandability of Apple privacy labels from the developer's perspective and identified many barriers that prevent developers from creating accurate privacy labels. These findings suggest that the accuracy and usability problems are interdependent.

We present Matcha, an IDE plugin, to improve the accuracy of privacy labels by addressing the usability and comprehension challenges for developers [29]. Although we focus on Google data safety labels because they support importing labels generated by external tools, we consider the developer-in-the-loop, machine-facilitated idea generalizable to other types of privacy nutrition labels and standardized privacy notices. We derive recommendations for designing developer tools in the same vein from in-depth studies on Matcha and discuss them at the end of the paper.

2.2 Challenges for developers to create accurate standardized privacy notices

Conceptually, the activity of creating a privacy nutrition label entails two steps. The developer needs to first establish a thorough and accurate understanding of how their app handles user data [38] and then translate it into a privacy label using the standard taxonomy. Unfortunately, research has shown that even if developers intend to create accurate privacy labels, they often encounter significant challenges in both steps.

First, prior work has discovered reasons leading to an inaccurate understanding of the app's data practices. Li et al. [26] found that developers may lose track of changes in data practices across different versions of their app and lack knowledge about data practices in code developed by other colleagues. Other research has revealed misunderstandings about the data practices of third-party libraries [4, 5, 34, 39], including developers being unaware of automatic data collection by libraries they include in their apps [5]. Third-party SDKs sometimes offer disclosures of their data practices, but developers are often unaware of these resources [29, 34, 39].

Second, it is difficult to synthesize data practices using standardized terms. Balebako et al. [6] tested both crowd workers' and privacy experts' ability to categorize data-sharing scenarios using a predefined taxonomy and found that participants' understanding of the concepts in the taxonomy varied greatly, even among experts. More specifically, Li et al. [29] studied how iOS developers created privacy labels for their apps and observed that developers frequently misinterpreted terms used in the privacy label such as "data collection".

In this work, we introduce Matcha to address these challenges. Matcha runs simple code analysis to identify first-party and third-party data practices, and provides scaffolding to help developers supply accurate information without spending time studying Google's definitions. Our developer studies showed the efficacy of Matcha in enhancing the accuracy of privacy labels, and also contributed further understandings of the types of errors that Matcha helped mitigate.

2.3 Developer tools for creating privacy notices

Some existing tools can help developers audit their data practices and indirectly help developers create privacy notices. Prior research has built information flow analyzers designed to detect malicious or unwanted information leaks from an app [2, 13, 16, 25, 37]. Google and Apple have introduced similar support in recent releases of their systems, such as the data access auditing APIs introduced in Android 11 for helping developers identify unexpected data accesses. However, most developers are unaware of and rarely use these expert features. Furthermore, these tools are not immediately useful for privacy label creation because it is difficult to align the detection results with the types of information that a privacy label needs.

Some tools directly help developers create privacy notices and privacy labels. Key examples are summarized in Table 1 and compared with Matcha among several dimensions. Notably, while some tools use program analysis to help identify data practices, none can fully automate this process, and all require developer input. The privacy labels of both iOS and Android include server-side data retention practices, data usage purposes, and complex exemption rules. Existing program analysis techniques, which focus on client-side data practices, fall short of detecting these types of information. Below we further introduce these tools and compare them with Matcha.

Before privacy nutrition labels are widely adopted, researchers have designed tools that leverage program analysis techniques to improve the creation of various types of privacy notices, such as privacy policies [26, 48] and in-app privacy notices [28]. These tools can not generate a privacy nutrition label, which requires a more comprehensive summary of the app's data practices in a predefined taxonomy that often does not match developers' intuitive understanding.

Tool/Resource Name	Privacy label creation	Address human Issues	Automated analysis	3rd-party info	IDE integration	Incremental update
Honeysuckle [28]	○	○	◓	○	●	●
Official Web Forms	●	○	○	○	○	○
Apple Privacy Manifest	◓	◓	○	●	●	○
Privacy Label Wiz [15]	●	◓	◓	◓	○	○
Privado.ai	●	◓	◓	●	○	○
Matcha	●	●	◓	●	●	●

Table 1. Summary of Matcha and other tools for creating privacy notices. We compare these tools along the following dimensions: Whether the tool helps with privacy label creation (*Privacy label creation*); Whether the tool helps address human-related issues (e.g., infomation overload, misunderstanding of privacy label terms) that can cause inaccurate privacy labels? (*Address human issues*); Whether the tool automatically detects data practices (*Automated analysis*); Whether the tool provides information for filling out privacy labels for third-party SDKs used in the app (*3rd-party info*); Whether the tool is integrated within the development environment so developers can have more context information and potentially work on the tasks during the development process (*IDE integration*); Whether the tool supports incremental update of the privacy notice/label rather than requiring developers to scan the entire codebase from scratch every time (*Incremental update*). ●: fully supported; ◓: partially supported; ○: not supported

Our work is directly inspired by and builds upon Coconut [26] and Honeysuckle [28], which are IDE plugins that detect sensitive data use and prompt developers to add annotations. Prior research has shown that adding annotations helped developers disclose more data practices when the privacy notice is written in a free form [26] and implement contextualized privacy UIs more efficiently [28]. However, the complex and standardized design poses more challenges to creating accurate privacy disclosures due to developers' misunderstanding and knowledge gaps [29]. Towards this end, we made substantial changes to the annotation design than prior work. The Matcha annotation design breaks down the disclosure required by privacy labels into fine-grained, easy-to-understand data practices, as embodied by the annotation fields. Our work for the first time shows that developers are able to add annotations based on code analysis results to provide precise information for creating the privacy labels.

With the introduction of the Apple and Google privacy labels, official tools are provided for creating the privacy labels in the format of web forms. These tools generate privacy labels based solely on developers' input and are subject to errors due to developers' misunderstanding of privacy label concepts and lack of knowledge about third-party SDKs' data practices. Matcha significantly address these issues by leveraging the synergies between annotation-based developer input and automated code analysis, as well as informing developers of the data practices of the detected third-party SDKs. In April 2023, Apple announced "Privacy Manifest", which is a property list that developers need to fill out in Xcode to describe the types of data collected by their app using the same taxonomy as privacy labels. Third-party SDKs need to provide their own privacy manifest files. Privacy manifests can be used as a reference to create privacy labels, potentially increasing developers' awareness of third-party SDKs. However, the fact that it is a separate requirement creates a barrier to adoption for privacy label creation.

There are other third-party tools for creating privacy nutrition labels for iOS or Android apps that use automated program analysis. For example, Privado.ai[2] is a commercial tool for creating a Google Play data safety label. Gardner et al. [15] presents Privacy Label Wiz, which is a web-based tool for creating an Apple privacy label and reports on the preliminary feedback from developers. Both Privado.ai and Privacy Label Wiz can detect

[2]https://www.privado.ai/data-safety-report

potential data types and use a wizard-like interface to guide developers in providing additional information for generating privacy labels. Both tools can detect third-party SDKs, and only Privado.ai can automatically fill out data practices of third-party SDKs. However, these tools were not designed to address the information overload or the privacy label term misunderstanding issues, as developers are still expected to read all the text-based guidelines and term definitions on their own. Conversely, Matcha has more design considerations like the annotation design, step-by-step task guidance, and quick-fixes to walk developers through this process and overcome these challenges. Another difference is that these tools show the code analysis results with limited contexts (i.e., no code snippets or only short code snippets), while the IDE integration of Matcha allows for ease and flexibility of reviewing the code context, which helps developers recall the context and provide accurate privacy label information. Furthermore, the design of Matcha has also taken the maintenance needs into account. The use of annotations allow Matcha to efficiently solicit more fine-grained information in context. This enables the incremental update of privacy labels. Specifically, the developer only needs to modify the annotations around the code that has changed in the new version, and then regenerate the label. This design eliminates the need to run the tools to scan the codebase and answer all the questions again from scratch, potentially addressing the issues of missing updates to the privacy labels [30].

In addition to the system design difference, our work also make research contributions by thoroughly evaluating our system and synthesizing design knowledge. We are the first to conduct in-depth studies to show that our tool (Matcha) can improve the accuracy of the privacy label for real-world apps created by the app developers. Our findings provide insights that can inform the design of future developer support for creating standardized privacy notices.

3 MATCHA DESIGN AND IMPLEMENTATION

In this section, we present our design goals, how our design fulfills these goals, improvements to the tool design based on preliminary studies, and our final design and implementation.

3.1 Design goals

We drew upon prior literature to inform our design goals. Li et al. [29] discovered that one obstacle to creating accurate Apple privacy labels was that developers needed to process a large amount of new information, including lengthy and complex definitions of terms like data collection. The similarities between the Apple and Google label filling process suggest that Android developers may also suffer from information overload issues. We argue that developers will benefit from more scaffolding, which leads to our first design goal:

D1 The privacy label questions should be deconstructed and situated within the context in which developers handle the specific code that deals with user data.

Developers know how their apps work, but prior research suggests they suffer from forgetfulness [26], lack of knowledge about other team members' code [26] and third-party SDK's data practices [5], and misinterpretations of terms in privacy labels [6, 29]. Hence, we set the second design goal:

D2 The tool should help developers overcome limitations in their ability to create accurate privacy labels.

Code analysis can be used to identify some data practices that need to be reported in the data safety label. However, it still has limitations. First, it mainly analyzes data practices within the client app and cannot answer how data is used after it leaves the device. Second, although algorithms exist to infer purposes of data use [20, 42] and data types that are not tied to a specific API [19, 35], they are not always accurate. This suggests that automated code analysis cannot be relied on completely, which leads to our last design goal.

D3 The tool should give developers control to refine or reject the automated analyses when they are inaccurate.

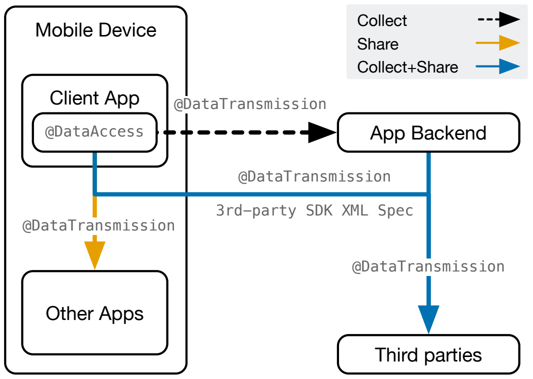


Fig. 2. An illustration of the mapping between the developer input (i.e., annotations and the SDK XML spec) and Google's definitions of data collection and sharing. By asking developers to add data access and transmission annotations rather than directly deal with the label-specific concepts, Matcha reduces errors due to misunderstanding of the label terms while keeping developers in control of the label.

To satisfy **D1**, we designed Matcha as an IDE plugin. We adopted the idea of using annotations to document data practices in code from prior work [26, 28], allowing developers to contribute their knowledge of the app's data practices by adding annotations. We divide the design of Matcha into two parts. In the first part, we focus on determining what types of information to solicit from developers and in what format. In the second part, we focus on the interaction design to help developers provide accurate information about their data use.

3.2 Developer Input Design

Prior work suggested that developers lack the time and ability to comprehend the label terms [29]. To address this issue, we designed the specific code format of developer input as a scaffolding for providing all the required information accurately.

3.2.1 Annotations for explicit data flows within apps. We design the `@DataAccess` and `@DataTransmission` annotations to indicate where data is accessed on device and transmitted to other apps or off the device, namely the sources and sinks of data flows. Figure 2 shows how Matcha translates the *data access* and *transmission* behaviors to the label terms *data collection* and *sharing*, which helps address the misinterpretations of the terms. Each `@DataAccess` describes the data types accessed by the app, and each `@DataTransmission` contains two sets of attributes covering the data use purposes and the special cases and exemptions of collection and sharing defined by Google. Developers can use `@NotPersonalDataAccess` and `@NotPersonalDataTransmission` to indicate no data is accessed or transmitted. Table 2 presents the annotation design.

3.2.2 XML spec for implicit data flows caused by SDKs. A library's data practices may depend on how the app uses it. For example, in the motivating example the Firebase Authentication library only collects the user's display name if the developer provides it. Since much of the configuration of library data use happens outside of the app, it is hard to determine a proper location for the annotation and check if the required annotation is added. Therefore, we design an XML file to allow developers to adjust the label based on their use of the library. In the XML, Matcha generates a `<library-custom-usage>` tag for each library and populates it with `<data>` tags that contain the collection and sharing conditions. Developers can either keep or remove each data tag based on the condition, and set the `verified` attribute to `true` to mark the configuration of a certain library as complete.

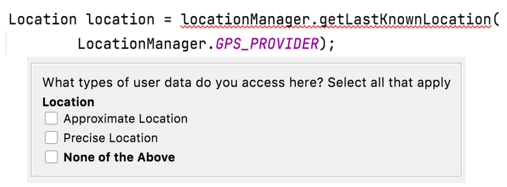
Table 2. The table shows the four annotations we designed for the task and their field members that hold different types of information needed for the label. More details about how `@DataTransmission` handles collection and sharing are in Appendix A.

Annotation	Fields	Note
`@DataAccess`	id	A unique ID defined by the developer to refer to this access when later annotating data transmissions.
	dataType	A list of Enum values of predefined data types accessed by the app and held in the annotated variable
`@NotPersonalDataAccess`	–	For explicitly indicating no personal data is accessed here. It is useful for dismissing an irrelevant data access suggestion.
`@DataTransmission`	accessId	A list of IDs indicating where the transmitted data is originally accessed. The IDs are previously defined in `@DataAccess`.
	collectionAttribute	A list of Enum values of collection-related information.
	sharingAttribute	A list of Enum values of sharing-related information.
`@NotPersonalDataTransmission`	–	For indicating no personal data is transmitted out here. It is useful for dismissing an irrelevant data transmission suggestion.

3.3 Preliminary Tests for Iterative Design

To achieve **D2**, we offer suggestions for data access and transmission and quickfixes for adding annotations based on code analysis. To achieve **D3**, we let developers have the final say, namely, they can ignore any suggestions, and the label creation only relies on the annotations and the XML spec that they can modify. This raises a question: *Are developers capable of correctly comprehending the suggestions and building on them to create an accurate data safety label?* We iteratively improved the design to achieve this goal by conducting IRB-approved preliminary tests with five developers on an initial prototype with basic support for adding annotations and editing the XML spec. The participants first created the label for their apps by Google's tool and then using Matcha. The interview script can be found in Appendix D. One researcher conducted a thematic analysis of the interview transcripts. Below, we summarize issues that emerged from the studies and how they informed the improvement of our system design.

3.3.1 Ignoring unexpected suggestions. Some developers appeared to be affected by confirmation bias [7], ignoring suggestions that did not match their expectations. For example, P3's app had a feature for sharing the user's high score, the game screenshot, and a short message provided by the user. This data falls under the "App activity" data category per Google's definition. However, P3 quickly added a `@NotPersonalDataAccess` annotation to dismiss Matcha's suggestion. He explained that "*I don't think high score is personal info*". This example shows that developers tended to place more trust in their understandings than the system's suggestion. This created obstacles to correcting the developer's misunderstandings. To address this problem, we consulted the guidelines for human-AI interaction [1] as building trust is the main goal of constructing efficient AI-infused systems. Specifically, we added proactive and contextualized guidance to help developers establish a clearer mental model of what Matcha can do and how Matcha's suggestions work.



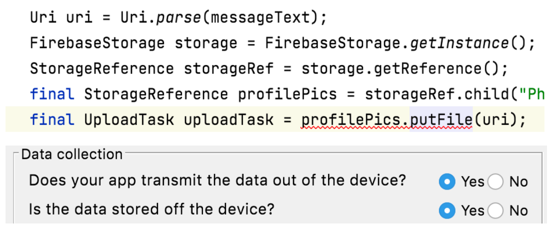
```
Location location = locationManager.getLastKnownLocation(
        LocationManager.GPS_PROVIDER);
```

What types of user data do you access here? Select all that apply

Location

- [ ] Approximate Location
- [ ] Precise Location
- [ ] **None of the Above**

(a) Data type options customized based on the API call.



```
Uri uri = Uri.parse(messageText);
FirebaseStorage storage = FirebaseStorage.getInstance();
StorageReference storageRef = storage.getReference();
final StorageReference profilePics = storageRef.child("Ph
final UploadTask uploadTask = profilePics.putFile(uri);
```

Data collection

Does your app transmit the data out of the device? [✓ Yes] No

Is the data stored off the device? [✓ Yes] No

(b) Transmission and storage pre-checked based on the API call.

Fig. 3. Examples of quickfix dialogs customized based on the API calls to avoid errors and improve learnability and usability.

3.3.2 Difficulty of handling Java annotations. Our participants faced challenges in adding the annotations due to unfamiliarity with the syntax. For example, Java annotations can only be added to specific code elements, such as a variable declaration. This troubled developers when they needed to manually add the annotation. Another example is when developers declare multiple variables altogether (e.g., `EditText nameText, ageText;`) they can not add different annotations to each variable. During our preliminary tests, we found that these seemingly trivial issues with annotations greatly hindered our participants' abilities to use our tool. Therefore, we optimized the support for adding annotations in the final version of Matcha to help developers automatically trace where to add the annotations and reformat their code.

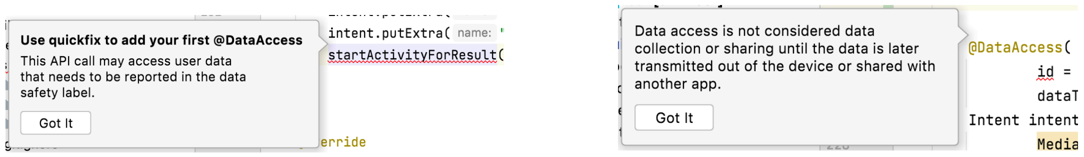
3.3.3 Error-prone direct editing of annotations. Some errors were introduced when developers edited annotations. Since our participants were first-time users of Matcha and unfamiliar with the Google data safety label design, they felt overwhelmed by the number of fields they needed to manually complete and the number of predefined values they can select from. Although direct editing may be more efficient for expert users, it was too error-prone and confusing for novices. Therefore, we provided a dialog for guiding the developer to fill out all the required information to generate the annotation.

3.4 Final Design of the IDE Plugin

We present the final design of Matcha, including the five tasks for creating the label and the main supporting features.

3.4.1 A five-step process. The Matcha label creation process consists of five steps. The first two are for adding data access annotations and the next two are for transmission annotations. In the first and the third steps, Matcha guides developers to do a precise API-based search, in which they annotate all the detected API calls that potentially access user data and send the data out of the app. In the second and the fourth steps, Matcha uses fuzzy keyword-based search to help developers detect more data types that are collected by the app but have not been annotated in the previous step. Adding annotations is voluntary for the second and the fourth steps, which means the developer only needs to add an annotation when they find the detected keywords relevant to the access to, and transmission of, user data. In the last step, the developer modifies the auto-generated XML spec file to adapt it based on their usage of the library.

3.4.2 Quickfixes for adding annotations. Matcha offers quickfixes to aid in the annotation creation (see Figure 1). The quickfix locates which variable to annotate for detected API calls. The developers can also use the quickfix to add additional annotations for any variables. The quickfix dialog can narrow down or pre-select options based on the detected API call. For example, for `LocationManager.getLastKnownLocation`, the available choices are only approximate location, precise location, and none of the above (Figure 3a). For a Firebase Cloud Storage API,



Use quickfix to add your first @DataAccess

This API call may access user data that needs to be reported in the data safety label.

Got It

```
intent.putExtra( name:
startActivityForResult(
erride
```

Data access is not considered data collection or sharing until the data is later transmitted out of the device or shared with another app.

Got It

```
@DataAccess(
    id =
    dataT
Intent intent
    Media
```

(a) A tooltip suggesting next action. (b) A tooltip explaining the generated annotation.

Fig. 4. Examples of the contextualized, proactive tooltips to offer just-in-time instructions and education about the annotations.

the "data being transmitted off the device" and "data being stored" options are automatically checked (Figure 3b). We note that users can still modify the annotations in any way they want, while using the dialog to constrain and validate the developer's input could both reduce the chance of accidentally making errors and make the tool easier to learn and use [36].

3.4.3 Contextualized and proactive guidance. To help developers understand what Matcha can do and how Matcha works, we designed just-in-time tooltips that pop up next to the related code when a certain type of Matcha suggestions is shown for the first time to give instructions on the expected actions (Figure 4a). Matcha also provides tooltips that are only informational, such as explaining what the annotations have to do with the creation of the safety label (Figure 4b). In addition, Matcha offers a systematic introduction of each step in a help panel (see Figure 1).

3.4.4 Label preview. To help developers better understand how the data safety label is generated, we design a label preview panel (see Figure 1). For each data type that is collected or shared, a note of "by library," "by app," or "by app and library" is provided, indicating the source of this data collection or sharing. After expanding each data type, it will show further information like the data collection or sharing purposes, as well as the related code links. These links allow the developer to check which annotations, custom library usage records in the XML file, or third-party libraries that always collect user data led to the generation of this particular data safety label entry.

3.5 Matcha System Implementation

The Matcha IDE plugin was developed using the IntelliJ Platform SDK[3]. We have released the plugin on JetBrain's official plugin store and open sourced it on GitHub[4].

Our API-based detection was built upon the code analysis subsystem of Coconut [26]. For data access, we augmented the Coconut API list with APIs from the official guidelines for this task[5]. For transmission, we kept the APIs about network requests from Coconut and added on-device sharing API calls for the sharing-only condition. Our final API list contains 91 data access APIs and 45 data transmission APIs.

We implemented the keyword search using the IntelliJ SDK's `findManager.findInProject` API. Our keyword list comes from three sources: (1) permissions mapped with specific data types in Google's official guideline[6], (2) keywords extracted from Google's data type definitions[7], and (3) keywords extracted from open-sourced Android apps using a TF-IDF algorithm. For the third approach, we selected 75 apps by searching for recently updated GitHub repos (in July 2020) that contain a Google Play link and declare dangerous permissions in the manifest. To identify the keyword candidates for the data types to be reported in the safety label, we tokenized the Java

[3]https://plugins.jetbrains.com/docs/intellij/welcome.html
[4]https://matcha-ide.github.io
[5]https://developer.android.com/guide/topics/data/collect-share
[6]https://developer.android.com/guide/topics/data/collect-share
[7]https://support.google.com/googleplay/android-developer/answer/10787469?hl=en#data_types

files from these projects and split the variables, treated each file as a document, and calculated the TF-IDF of each word per document. We then selected files containing the data access API calls and ranked words appearing in them based on the average TF-IDF values. Finally, a researcher reviewed the top results and incorporated words related to the data type into the final list. Matcha's keyword search feature is case-insensitive. Our final version covers 180 unique keywords (see Appendix B for details).

Matcha scans the app's build.gradle file to detect third-party SDKs and fills out the required data collection and sharing practices in the generated label. Because privacy labels (for both iOS and Android) heavily involve server-side data practices (such as retention and sharing), it is infeasible to automate the analysis of third-party SDKs by running static/dynamic program analysis. Therefore, the auto-generated privacy label information for third-party SDKs was obtained by one researcher manually coding the privacy label filling guidelines provided by third-party SDK developers (e.g., the documentation of the Firebase SDKs.[8]). We covered popular commercial SDKs included in the Google Play SDK Index[9]. We also added SDKs developed by Google that provided privacy label guidelines. The SDK list is dynamically loaded every time our plugin loads by requesting a JSON file hosted remotely, allowing the list to be updated without updating the entire plugin. Our final version for the study covers 58 unique third-party SDKs (see Appendix B for details). Prior research [9] has shown that over half of the sensitive data access by third-party libraries are from the most popular 30 libraries, which suggests that our current implementation could help developers cover a large portion of the data practices due to third-party SDK in their privacy labels.

For the SDKs that involve optional data collection and sharing practices, Matcha generates XML code to allow for further customization by the developer. For each detected SDK, Matcha generates a `<library-custom-usage>` tag with the initial value of the `verified` attribute set to false. Then it inserts `<data>` tags under the `<library-custom-usage>` to represent the data collection and sharing instances derived from the guidelines of the SDK.

4 MATCHA EVALUATION

4.1 Study Design Considerations

Evaluating interventions for improving the accuracy of a privacy label is difficult. Unlike many developer tools that can be evaluated with uniform tasks in well-controlled settings [11, 26, 28, 40, 41], a tool for creating the privacy label must be evaluated by developers who have adequate knowledge about the app. Otherwise, it becomes hard to eliminate the impact of lacking familiarity of the app. One potential method is to ask participants to develop an app with specific data practices and then create the label. However, even developing a small app can cost thousands to tens of thousands of dollars, making it too costly for research. Hence, we chose to ask participants to work on a real-world app they developed.

However, asking participants to work on their own apps is also challenging. First, we cannot obtain the ground truth of the data practices of these apps to verify the developers' answers. Second, since participants works on different apps, we cannot directly compare their performance, making a between-subjects study design unsuitable. Third, it is hard to recruit a large sample of participants who not only have developed an Android app but are also willing to install a plugin to analyze their code and let researchers look at their code.

Given these challenges, we conducted within-subjects, mixed-methods studies to gain quantitative and qualitative insights for this question: *How effective is Matcha in correcting the errors in a privacy label created by Google's official tool?* We observed how developers created the label using Google's tool and Matcha. We asked them to compare the two labels to measure the change in accuracy in the absence of absolute ground truth. The realistic setting placed higher requirement on our tool to work with arbitrary apps [17].

[8]https://firebase.google.com/docs/android/play-data-disclosure
[9]https://developer.android.com/distribute/sdk-index

4.2 Participants

12 developers participated in our study. We coded the data iteratively and stopped recruiting after reaching saturation in our qualitative analysis [18]. This sample size is consistent with evaluation studies of novel programming tools in prior work [11, 28, 31, 46, 47]. Most participants were recruited from freelancer websites, including eight from Freelancer[10] and one from Upwork.[11] The other three signed up after seeing advertisements on Twitter, Slack groups, or from personal connections. Our pre-screening survey asked for an Android app they developed and their role(s) in the development process, and had quiz questions about Android development.

Our participants came from eight countries and developed the apps either as part of their job, a hobby, or for a course project. Nine out of the 12 participants had experience in publishing apps on the Google Play store, though some chose not to use Google Play apps for our study due to NDA restrictions. Most of our participants have low familiarity with the task of creating privacy labels. Nine out of the 12 participants had heard about the Google privacy label as a developer or a user, while only three have created one. We also asked about their familiarity of iOS privacy labels, and only four have heard about it and two have created one. Our sample included six Google Play apps, with the most popular one having over one million downloads. We append details about each participant in Appendix E.

4.3 Study Procedure

We started the study by briefing the participant on the study goals and obtained their consent for audio and screen recording. Before the main tasks, we first gave a brief introduction to the Google Play data safety label. Then we asked the participants to introduce the app they had selected.

In the first task, the participant created the label for the selected app by filling out forms on the developer console. Participants logged into the console using an account we created for the study. We asked them to handle this task as they normally would and encouraged them to use any resources they would normally consult, except for the app's current label if available. In the second task, the participant created the label using Matcha. We first helped them download and install the plugin and then asked them to watch a short tutorial video (2 minutes 42 seconds) before working on the task. If they were not sure about how to fill in certain information, we asked them to answer based on their best understanding and explain their rationale. After creating the label, we asked them to import the CSV into the Google Play developer console to create the label. Participants were asked to think aloud during both tasks. We discuss the potential implications of a learning effect due to the two-task within-subjects study design along with other methodological limitations in Section 4.6.

The study ended with a brief semi-structured interview. We showed the discrepancies between the two labels and asked the participant to identify errors in either version and explained what caused the error. Then we asked which tool they preferred and why. The interview script is in Appendix D.

4.4 Ethics of the Study

The study was IRB-approved. Each interview took 1.5 to 2 hours. Each participant was compensated $70. During the interview, we allowed them to only share part of their screen to avoid showing identifiable information and skip any questions they did not feel comfortable answering and reassured them that it would not affect their compensation. We removed any information that could identify the participants, their apps, and their organizations before publishing the results.

[10]https://www.freelancer.com
[11]https://www.upwork.com

4.5 Qualitative Analysis Method

The first author coded the interview transcripts and the screen recordings. First, the author coded when developers added annotations and modified the XML spec entries. Then the author coded the verbal responses from the think-aloud process of the main tasks and the post-study interviews using a bottom-up open coding method [10]. The other two authors met with the first author weekly to discuss the findings and derive themes. The coding process was done with the software MAXQDA. We provide our complete codebook in Appendix F.

4.6 Methodological Limitations

Our study method has some limitations. First, using Matcha after using the developer console may result in a learning effect: the increased familiarity with the task might have contributed to the improvement in accuracy caused by Matcha. However, the process of using the two tools and the questions the developers answer are quite different, which suggests the learning effect may be small. Our qualitative analysis further delineates how Matcha improved the accuracy. Second, the identified errors are not exhaustive since the ground truth is not available. As such, the errors analyzed in the paper should not be interpreted as all possible errors. Third, our participants developed the app individually or in a small team, so our findings may not apply to developers who work in a big company. Fourth, our findings be subject to the social desirability bias, namely the participants may be more likely to express a preference for Matcha due to the financial compensation for participation. Hence, future field research is needed to investigate how developers perceive the tradeoff of time for accuracy in real life. Fifth, testing Matcha with different apps has inherent limitations as discussed in Section 4.1. One potential mitigating approach is to conduct the studies with multiple developers working on the same project. We decide to leave the exploration of this idea for future research.

5 RESULTS

Overall, we found that Matcha helped improve the accuracy of the safety labels. Both objective and subjective results suggest that Matcha was easy to learn and use. All participants preferred Matcha over Google's tool.

5.1 Matcha Improved Label Accuracy

All participants considered the Matcha version correct when reviewing the discrepancies between the two labels, except for F2, who correctly classified gender as "other personal information" in the baseline while then misclassifying it as "sexual orientation" using Matcha. The misclassification errors do not affect counting the data types and purposes and therefore do not affect our quantitative measurements. All participants except for F6 fixed some errors in their labels with Matcha. The proportion of participants who have fixed errors in their labels using Matcha was 11/12 (approximately 92%) with a Wilson confidence interval of (64.6%, 98.5%) at the 95% confidence level [32, 43]. Matcha helped report 1.8 times as many data types collected or shared by the app (92 vs. 52) and 3.0 times as many purposes for data collection and sharing (212 vs. 70) as compared to the baseline.

5.2 Types of Errors Fixed by Matcha

Our first analysis examines the errors corrected by Matcha in various aspects. Table 3 summarizes the errors.

5.2.1 Under-reporting vs. over-reporting. Matcha fixed many under-reporting errors (77%), which helped developers report more comprehensive data practices. We want to note that the Matcha labels also fixed under-reporting errors in the real-world labels of the six Google Play apps. Matcha also fixed some over-reporting errors (23%).

5.2.2 First-party vs. third-party. A significant fraction of errors corrected by Matcha were caused by third-party libraries that automatically collect or share data (78%) than by first-party code (22%). This was mostly due to the Firebase services used for functionality and analytics, as well as other advertising and utility libraries.

Table 3. Analysis of errors fixed by Matcha. The *Base.* column shows the number of data types and purposes reported using the developer console, and the *Add* and *Cut* columns show what Matcha helped add or remove as compared to the baseline version. Most fixed errors were under-reporting errors (more added than removed) caused by third-party libraries.

	Data Type			**Data Purpose**		
	Base.	Add	Cut	Base.	Add	Cut
1st-party collect	21	8	13	34	15	20
3rd-party collect	18	44	11	22	107	15
1st-party share	1	2	1	2	2	2
3rd-party share	12	20	9	12	64	9
Total	52	74	34	70	188	46

5.2.3 Fixed errors related to different data practices. More errors fixed by Matcha were related to data collection (70%) than data sharing (30%). However, we want to note that the improvement for data sharing might be more essential, because no data sharing was reported in baseline labels, whereas some data collection was already reported in baseline. This suggests developers had more severe awareness gaps regarding data sharing. Furthermore, sharing data with third parties is more sensitive [9], which means the Matcha labels can better inform users of the privacy risks.

5.3 Matcha Helped Tackle Challenges for Creating Accurate Privacy Labels

We identified four themes in participants' explanations of errors fixed by Matcha. We found that Matcha helped address common issues that can lead to misunderstanding of data practices and inaccurate privacy nutrition labels. [5, 26, 29].

5.3.1 Help tackle misunderstandings about data safety label (F2, F4, F5, F8, F9, F10, F11, F12). Matcha helped fix errors due to misunderstandings of the data safety label taxonomy. For example, F8 initially thought he should report some data as collected while the data was only used on device and therefore did not count as collection per Google's definition. Matcha explicitly asked whether the data is transmitted off the device, which resolved the problem by relieving the developer from translating low-level behaviors to the label terms. Interestingly, we observed that developers ignored unfamiliar data types in the baseline task. For example, F2 said "*I did not even think of 'other user-generated content.'*" Matcha correct these errors by having them focus on a specific API call and the data types related to the API.

5.3.2 Help reduce errors related to third-party libraries (F1, F2, F3, F5, F8, F9, F11). We found Matcha helped correct errors due to misperceptions about third-party libraries. Some developers did not consider these libraries when creating the label. For example, F3 felt the data collected by Firebase was "*collected by a different platform*" not part of his app. Some developers were unaware of data collected and shared by libraries. For example, F2 explained that "*I didn't know that the library (Firebase cloud storage) was doing that on its own behind the scenes.*"

Interestingly, Matcha has also helped developers who already had some expectation of the third-party data practices. For example, F1 said "*our data is sent to Firebase server, that's why I am selecting these*" as he thought out aloud when using the baseline tool. However, later Matcha revealed that Firebase collected more data then he expected. F11 searched the exact data practices of an advertising library used in his app, but couldn't find the specific guide provided by the library developer, so he referred to their privacy policy instead. The ambiguous wording of the privacy policy then caused errors in the first label that were later corrected by Matcha.

5.3.3 *Help reduce errors due to forgetfulness (F1, F2, F3, F6, F7, F10, F12).*

One common source of errors that Matcha helped fix is due to forgetfulness. They could simply be an oversight – "*it just escaped my mind*" (F3), or have deeper reasons. For example, F6 forgot the use of a third-party library and explained that "*I didn't actually recall that because I was not in charge of this part.*" F7 forgot he integrated the Admob library a long time ago. Both errors were caught by Matcha. Similar to earlier research [29], we found developers mostly answered questions from memory when using the developer console. Matcha's systematic review of data practices helped developers find and disclose more data types than they would have otherwise. For example, although F2 checked the Firebase database in the first task, he forgot certain tables and therefore missed certain collected data types. This was later fixed by Matcha.

5.3.4 *Help reduce errors due to unfamiliar APIs (F1, F2).*

Matcha even helped developers learn more about the behavior of unfamiliar APIs. For example, F2 searched for the `LocationManager.getLastKnownLocation` API online when adding annotations for the API, and therefore learned more about the precisions of the location data. F1 thought the Admob library could not obtain approximate location data because the app did not request location permissions. However, he later learned from Matcha that the Admob library used the user's IP address to derive the approximate location, which was not controlled by the permission system.

5.4 Perceived Benefits and Problems of Matcha

All participants preferred Matcha over the baseline due to four main benefits. We also discuss needs for future improvement.

5.4.1 *Benefit: Improved accuracy (F1, F2, F4, F7, F8, F10, F11).*

The primary benefit of using Matcha was the improved accuracy, which could outweigh the time cost. As stated by F2, "*Accuracy wise, I would prefer the tool Matcha...For an app that I'm going to publish on Google Play, I would use Matcha, because like, I emphasize accuracy over efficiency.*"

5.4.2 *Benefit: Ease of use (F2, F5, F6, F7, F9, F11, F12).*

Many participants considered Matcha easy to learn and use. F6 felt the quickfixes for adding annotations were "*pretty convenient*". F7 said "*I thought it might be complicated. But when I started a bit, it becomes easier to use*". F2 felt it would be easier if he added annotations as he coded the app.

We observed that those who devoted more effort towards providing accurate information in the initial task via the developer console tended to find more value in the ease of use of Matcha. For example, F11 tried to search for the third-party SDKs' data practices in the baseline task (as mentioned in Section 5.3.2). He expressed a preference for Matcha because, "*it is very easy, it saves a lot of time, and plus it is more accurate as compared to the Google Play console, which is very lengthy, and we have to read through all the options and then check the boxes, and we have to consult the documentation.*"

5.4.3 *Benefit: Informative tool (F2, F6, F5, F7, F8, F9).*

Many participants liked Matcha because it helped them learn a lot about their app and the data safety label. For example, F8 mentioned that, "*Before using your plugin, I was quite sure that I have submitted all the information that I'm getting, but after using the plugin, I am more knowledge about what's going on in my app.*"

5.4.4 *Benefit: Better engagement (F2, F6, F10).*

Some participants liked that Matcha contextualized all the questions around specific code, which better engaged them with this task than the developer console. F6 explained that:

> *The developer console does have everything written on it, but it's hard to actually relate that to your own code, because it's just a bunch of instructions. While the plugin could remind you of what you have written. For my example, I don't actually remember if I ever imported the WeChat SDK. I really don't remember that. And the console wouldn't actually remind me of anything.*

5.4.5 Benefit: Better flexibility (F3, F6, F12). Some participants considered the code-based label generation more flexible. F3 mentioned that he preferred the plugin because "*It gave me the flexibility I needed and made me feel like I was still doing development work. All I had to do was add annotations and it generated labels for me.*" F12 even thought of the benefits of Matcha's annotation-based method for future app updates: "*If I update the app to another version, it's easy to change the annotation and create a new CSV file.*"

5.4.6 Problem: Redundancy issues (F2, F6, F5, F8). Developers raised redundancy as a recurrent issue with Matcha, particularly in keyword search results. Some keywords were too generic and resulted in false positives. For example, F6 felt that the keyword "search" was not effective in detecting the data type "search history" because it was often used for unrelated purposes. Developers also noted the redundancy in the required developer input. We requested that developers add different annotations when making API calls that collect the same type of data, as they may serve different purposes. However, F2 complained about having to annotate `requestLocationUpdates` after annotating `getLastKnownLocation`, which collect the same data and for the same purpose in his use case. Future work may consider improving the design with more context-aware suggestions.

5.5 Efficiency of Matcha

Our analysis of developers' action traces offers insights into the efficiency and learning curve of Matcha.

5.5.1 Overall time performance comparison. It took our participants 30 minutes on average to complete the task using Matcha (std = 15 minutes), while it only took 9.8 minutes using the developer console (std = 9.3 minutes). Although developers were able to complete the task faster using the developer console, they did so at the cost of accuracy.

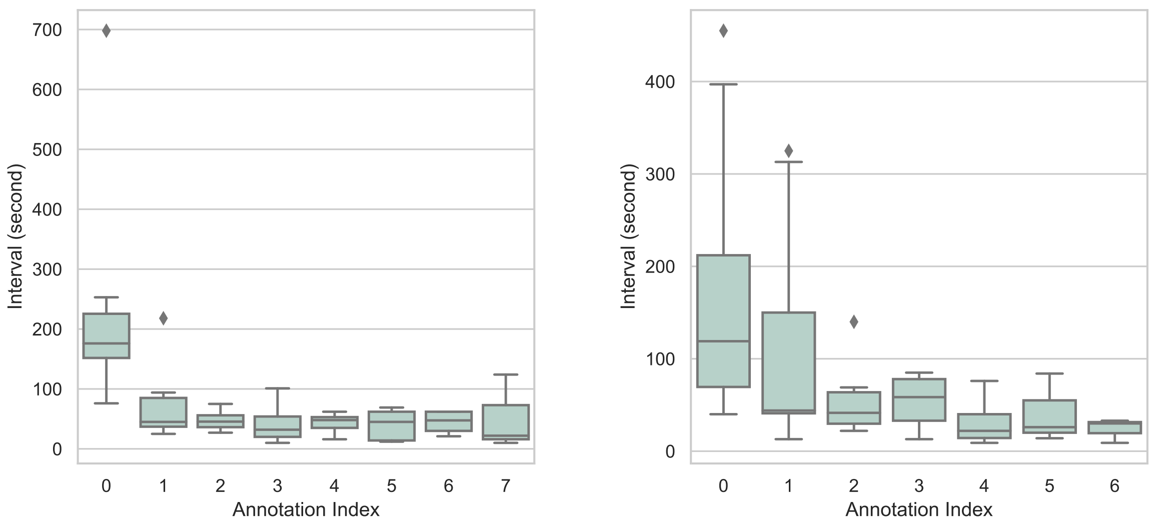
5.5.2 Time for adding annotations. We analyzed the time for adding privacy annotations, which is a novel task for developers. Each access annotation (`@DataAccess` or `@NotPersonalDataAccess`) took only 1.3 minutes on average to add (std = 1.6 minutes). Each transmission annotation (`@DataTransmission` or `@NotPersonalDataTransmission`) also took 1.3 minutes on average (std = 1.6 minutes). We further show in Figure 5 that the time of adding an annotation decreases over time, suggesting our participants' performance increased with practice.

5.6 Developers' Reactions to Suggestions

Finally, we analyze developers' reactions to the detected API calls and libraries. Matcha detected 10 access API calls and 6.3 transmission API calls on average per app, which led to 4.9 access annotations and 5.2 transmission annotations added by participants on average per app. Note that the number of required annotations are fewer than the detected API calls because multiple API calls can share one annotation.

Among the 59 access annotations, 19 were `@NotPersonalDataAccess`; among the 62 transmission annotations, 34 were `@NotPersonalDataTransmission`, which demonstrated the developers' abilities to identify and correct the false positives of Matcha suggestions. For example, the user password and files provided by the app were the two types of data most commonly labeled as `@NotPersonalDataAccess` in our study; and data stored locally on device, network request without user data, and data transmission practices that meet the exemption criteria were the common reasons for `@NotPersonalDataTransmission`.

Furthermore, F6, F7, and F10 each added a `@DataAccess` for a data type that was not covered by API call detection, showing the benefits of drawing on the developers' knowledge of the app to complement the API-based code analysis results.



(a) Access annotations (b) Transmission annotations

Fig. 5. Time to add an access and a transmission annotation. The indices represent the order of added annotations. For the reliability of the results, we only show the indices with at least three people's data. The two figures show that it took participants longer to add the first access annotation, while the time drastically decreased after the first two attempts and became stable afterward, suggesting an easy learning curve.

6 DISCUSSION

6.1 Developers and Code Analysis: Better Together

The improved label accuracy (Section 5.1 and Section 5.2) shows the efficacy of making developers and code analysis handle the part of work they are most effective at and benefit from each other. Meanwhile, using Matcha improved the participants' knowledge of their app's data practices and overcome misunderstandings in data safety labels (Section 5.4.3 and Section 5.3.1). It is useful to keep developers informed and involved during this task because their knowledge can complement and refine the code analysis results (Section 5.6). At present, developers are largely motivated by platform requirements to consider privacy, and their focus is largely limited to how to satisfy specific requirements [27]. Using Matcha showed a nice side effect to evoke more in-depth learning for privacy (Section 5.3.4), which might motivate and support further improvements in app privacy design.

6.2 Using Annotations as a Uniform Privacy Language for Developers

Instead of asking developers to directly describe their apps' data practices using the label terms, Matcha ask developers to add annotations and edit the XML spec, and let the tool translate them to the label terms (Figure 2). This not only mitigated errors due to developers' misinterpretations of label terms (Section 5.3.1), but also helped developers reflect on their data use (Section 5.4.4). Moreover, using annotations allows developers to contribute granular privacy information in a contextualized format, which can potentially be used to automatically generate various types of privacy notices to help relieve developers from the work of upgrading privacy interfaces [28].

Hence, the annotations and XML specification may have the potential to become a privacy domain-specific language for developers, bridging the low-level code behavior and the high-level user-facing disclosure.

6.3 Generalizability of Annotation-based Approach for Creating Privacy Labels

There are many scenarios beyond Android client apps written in Java that require improved developer support for creating accurate privacy notices. These include apps developed for other platforms (e.g., iOS, cross-platform web apps) and server-side code. We analyze the feasibility and cost of applying the annotation-based approach to other contexts in the following.

Matcha focuses on generating Google privacy labels, which follow a different design than the Apple privacy labels. The key concept in Apple privacy label is also "data collection". The term "collection" has a similar definition as in Google's context, which means the annotation design for gathering data collection related information can be easily adapted to the iOS context by coding the exemption rules and special considerations into new pre-defined values for the `collectionAttribute` field. The two platforms also have unique core concepts in their privacy label design, such as "data sharing" for Google, and "data linked to users" and "data used to track users." for Apple. As a result, the `sharingAttribute` field in Matcha annotation design should be correspondingly replaced with `linkingAttribute` and `trackingAttribute`. Specifically, the former is used to describe whether the data is identifiable or stored with other identifiable data, which will requests users to complement server-side data practices in a similar way as in Matcha. The latter is related to third-party advertising tracking and is directly associated with the use of the `AppTrackingTransparency` framework.

Then we discuss the technical feasibility of migrating the annotation-based approach to other contexts (e.g., iOS apps, cross-platform web apps.) We expect privacy annotations to be applicable to other languages, because many other languages also support developers to attach metadata to code in different ways. For example, C#, Python, and TypeScript support some level of customization for annotations or decorators similar to Java's annotations, so privacy annotations can be relatively easily migrated to apps written in these languages. For languages that do not have an official support for custom annotation or decorator yet (e.g., Swift), privacy annotations can be added in an alternative format such as free-form comments, or by extending the language syntax and writing a transpiler to do source-to-source translation.

Transferring privacy annotations to an individual programming language is relatively straightforward. However, a more sophisticated challenge arises when a real-world software system consists of multiple parts written in different languages. For example, the frontend website may be coded in TypeScript, while the backend server is coded in Python. As data may be transmitted across these subsystems, there needs to be a unified format to convert privacy annotations from different languages into intermediate files. These files can then be aggregated to generate a comprehensive picture of the data practices of the entire system. Aggregating privacy annotations across systems can further consolidate the annotation design and reduce errors, as some fields that previously needed to be manually filled can now be automatically inferred (e.g., server-side data retention practices, which are currently specified in the data transmission annotation as a collection attribute).

6.4 Limitations of the Annotation and Developer Tool Approach

Our work is focused on helping benign developers create accurate privacy labels by tackling their knowledge gaps and addressing common misunderstandings. However, we want to note that an intrinsic limitation of Matcha is that it can not prevent malicious developers from hiding their data practices from users. These malicious developers may intentionally aim to steal user data, or simply fear that users will be discouraged from installing the app. To address this issue, further research is needed to aid external parties (e.g., users, app marketplaces) in auditing the privacy labels. The auditing task is possible for popular third-party SDKs by cross checking the app's privacy label against the data practices disclosed by third-party SDK developers, who are more likely to be held accountable.

Fully automated auditing for less popular third-party SDKs and first-party data practices can be extremely challenging, because external parties are not able to audit how the data is used after it leaves the device. Nevertheless, the annotation approach potentially provides external parties with a method to partially vet the privacy labels. By requiring developers to disclose more intermediate data than is directly provided in Google's label creation form, the intermediate data (e.g., whether certain types of data flow from a source to a sink) can be vetted by dynamic analysis techniques. Overall, we envision that by designing standardized disclosure formats like annotations requiring developers to provide low-level, unambiguous data practices, we can enhance the auditability of privacy notices beyond existing methods such as privacy policies and privacy labels.

6.5 Design Implications for Developer Tools for Privacy

We have shown Matcha successfully helped developers improve the accuracy of their data safety label (Section 5.1 and Section 5.2), was easy to learn and use (Section 5.4.2 and Section 5.5.2), and was preferred by all study participants (Section 5.4). Below, we synthesize design recommendations for developer tools for creating standardized privacy notices.

6.5.1 Contextualize the task around code. When filling out forms on the developer console, developers rarely checked the code to verify their understanding. Moreover, developers had trouble systematically reviewing the code on their own. Matcha suggests that showing questions around the related code can help developers provide more accurate answers and learn more about their app (Section 5.4.4).

6.5.2 Provide scaffolding. Despite the promising benefits of making privacy information part of the code, it is difficult for developers to manually handle the task given the complexity of the required information and the difficulty of handling an unfamiliar syntax (i.e., the annotation). In Matcha, the use of the quickfix dialog helped solicit valid and accurate input and eased the learning curve. The dialog allows for more space for presenting the full questions in a structured format and verifies the developer's input before it is submitted. This method both provides sufficient guidance for novice users and flexibility for expert users.

6.5.3 From high-level tasks to low-level questions. Standardized privacy notices need to lump low-level practices into higher-level categories to improve the clarity of the notice to lay people. However, as developers often have misperceptions of the standard taxonomy [29], it is helpful to break down each high-level concept into lower-level questions that probe each aspect of the concept separately. For example, Matcha helped the developers correct their misunderstandings of "data collecion" and "data sharing" by separating data accesses and transmissions and asking about special cases and exemption conditions explicitly when the developer added transmission annotations (Section 5.3.1).

6.5.4 Use proactive guidance and actionable suggestions. One key challenge in creating accurate labels is that developers tend to be overconfident in their answers and unaware of their own errors and knowledge gaps (Section 3.3.1 and Section 5.4.3). To better engage the developers in this type of tasks, we used proactive guidance such as the just-in-time tooltips and the errors flagged in code for missing annotations (Figure 4). We also tried to break down the entire task into smaller actionable steps. Our suggestions of accesses and transmissions are grounded in these actionable steps to force the developer to interact with them and ponder on them.

6.5.5 Use precise and specific suggestions. A fundamental challenge in this type of task is to balance the recall and precision of the suggestions. Our study showed that the precise and specific API-call based suggestions were better received than more generic keyword-based suggestions (Section 5.4.6). Although it is still necessary to have something like the keyword-based detection that emphasizes a good recall rate, it would be more effective if the precision is also improved so the correct suggestions are not buried in a large volume of irrelevant suggestions.

6.6 Future Research Directions

In this work, we took the first step to design developer tools for creating accurate privacy nutrition labels. Below we discuss challenges that need to be addressed by future research.

6.6.1 Managing Third-Party SDK Label Information At Scale. In the long run, developers may want to obtain support for third-party SDKs that are currently not covered, and receive up-to-date suggestions as these SDKs update their data practices. However, there are two fundamental barriers: 1) some third-party SDK developers do not disclose their data practices for label creation tasks; 2) even for SDKs that do provide such information, the ad-hoc format of disclosure results in ambiguity and makes it difficult to automatically parse the information.

We propose two potential directions to tackle these barriers for future research. One idea, independent of the platform, is to automatically crawl and parse third-party SDK privacy label disclosures, and use a developer-sourcing approach to collectively vet, fix, and release the data as part of an open-source effort. For example, if a third-party SDK is detected, the tool may first check if there is a public resource associated with the SDK. If such a resource exists, the tool may retrieve it and compare it with the latest version to identify any changes in content. Then it can categorize these changes using the terminology of privacy label and request the developers to verify them. If such a resource can not be automatically detected, the tool may request the developer to search for the resource and provide a pointer. In this situation, third-party SDK developers only need to post and update the guidelines for fulfilling the platform disclosure requirements on their websites, just as they are currently doing.

Another idea relies on the platform to create a standardized format for third-party SDK developers to disclose their data practices, which can then be automatically integrated into the privacy nutrition label creation process. The closest real-world implementation of this concept is Apple's privacy manifests, introduced in 2023[12]. Apple requires third-party SDK developers to create privacy manifests, allowing downstream app developers to refer to these manifests when creating their privacy labels. Future research should explore methods to facilitate the creation and verification of accurate, fine-grained third-party disclosures, and a streamlined process to incorporate this information into user-facing disclosures, such as privacy labels.

6.6.2 Integrating advanced program analysis techniques. Our proof-of-concept prototype, built with simple code analysis techniques, already achieved substantial improvement. However, more precise suggestions can help address the redundancy issues mentioned by participants (Section 5.4.6). One idea is to enhance the keyword search with large programming models such as Codex [8]. In addition, future research can also study how to combine dynamic program analysis techniques to provide post-hoc data transmission monitoring and feedback.

6.6.3 Increasing incentives for improving accuracy. Matcha helped developers fix many under-reporting errors, but it can also make the apps look more invasive than apps with less accurate labels. To solve this problem, the platform should take actions to motivate developers to improve accuracy. For example, if Google provides some level of verification of the data practices and makes the results visible to both the developer and the end users, it can reward developers who honestly disclose their data practices in the privacy label.

6.6.4 Designing for other real-world challenges. Our study focused on creating a data safety label for a completed app, while future research should also explore other use scenarios. For example, future research may examine how developers add annotations while coding, as discussed in the methodological limitations (Section 4.6) and by our participants (Section 5.4.2). Furthermore, future work should study how to support multiple developers or even people in other roles to coordinate changes in data practices and properly encode them in the annotations.

[12]https://developer.apple.com/videos/play/wwdc2023/10060/

7 CONCLUSIONS

In this paper, we present Matcha, an IDE plugin that can help developers create an accurate data safety label. Matcha leverages automated code analysis to offer developers data use suggestions and allows developers control the label with annotations and an XML spec. In our studies, Matcha helped our participants improve their app's label accuracy. Matcha was perceived as easy to learn and use, and was preferred by all participants over Google's tool for the benefits of accuracy, better engagement and flexibility, and providing useful information. We discussed the design implications on developer tools for the creation of standardized privacy notices.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under Grant No. CNS-1801472, Innovators Network Foundation, and CMU CyLab Seed Funding. Tianshi Li was supported in part by the CMU CyLab Presidential Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. We thank the anonymous reviewers for their constructive feedback.

REFERENCES

[1] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N Bennett, Kori Inkpen, et al. Guidelines for human-ai interaction. In *Proceedings of the 2019 chi conference on human factors in computing systems*, pages 1–13, 2019.

[2] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49(6):259–269, June 2014. ISSN 1558-1160. doi: 10.1145/2666356.2594299. URL http://dx.doi.org/10.1145/2666356.2594299.

[3] David G Balash, Mir Masood Ali, Xiaoyuan Wu, Chris Kanich, and Adam J Aviv. Longitudinal analysis of privacy labels in the apple app store. *arXiv preprint arXiv:2206.02658*, 2022.

[4] Rebecca Balebako and Lorrie Cranor. Improving app privacy: Nudging app developers to protect user privacy. *IEEE Security & Privacy*, 12(4):55–58, July 2014. ISSN 1558-4046. doi: 10.1109/msp.2014.70. URL http://dx.doi.org/10.1109/msp.2014.70.

[5] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. The privacy and security behaviors of smartphone app developers. In *Proceedings 2014 Workshop on Usable Security*, USEC 2014. Internet Society, 2014. doi: 10.14722/usec.2014.23006. URL http://dx.doi.org/10.14722/usec.2014.23006.

[6] Rebecca Balebako, Richard Shay, and Lorrie Faith Cranor. Is your inseam a biometric? a case study on the role of usability studies in developing public policy. In *Proceedings 2014 Workshop on Usable Security*, USEC 2014. Internet Society, 2014. doi: 10.14722/usec.2014.23039. URL http://dx.doi.org/10.14722/usec.2014.23039.

[7] Souti Chattopadhyay, Nicholas Nelson, Audrey Au, Natalia Morales, Christopher Sanchez, Rahul Pandita, and Anita Sarma. A tale from the trenches: cognitive biases and software development: cognitive biases and software development. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20. ACM, June 2020. doi: 10.1145/3377811.3380330. URL http://dx.doi.org/10.1145/3377811.3380330.

[8] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[9] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones: Context-aware privacy management for smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):1–22, September 2017. ISSN 2474-9567. doi: 10.1145/3132029. URL http://dx.doi.org/10.1145/3132029.

[10] Robin Cooper. Decoding coding via the coding manual for qualitative researchers by johnny saldaña. *The Qualitative Report*, October 2016. ISSN 1052-0147. doi: 10.46743/2160-3715/2009.2856. URL http://dx.doi.org/10.46743/2160-3715/2009.2856.

[11] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20. ACM, April 2020. doi: 10.1145/3313831.3376442. URL http://dx.doi.org/10.1145/3313831.3376442.

[12] Pardis Emami-Naeini, Yuvraj Agarwal, Lorrie Faith Cranor, and Hanan Hibshi. Ask the experts: What should be on an iot privacy and security label? In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2020. doi: 10.1109/sp40000.2020.00043. URL http://dx.doi.org/10.1109/sp40000.2020.00043.

[13] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems*, 32(2):1–29, June 2014. ISSN 1557-7333. doi: 10.1145/2619091. URL http://dx.doi.org/10.1145/2619091.

[14] Geoffrey A. Fowler. iphone app privacy labels are a great idea, except when apple lets them deceive - the washington post. https://web.archive.org/web/20220630055538/https://www.washingtonpost.com/technology/2021/01/29/apple-privacy-nutrition-label/, 1 2021. (Accessed on 08/27/2022).

[15] Jack Gardner, Yuanyuan Feng, Kayla Reiman, Zhi Lin, Akshath Jain, and Norman Sadeh. Helping mobile application developers create accurate privacy labels. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, June 2022. doi: 10.1109/eurospw55150.2022.00028. URL http://dx.doi.org/10.1109/eurospw55150.2022.00028.

[16] Michael I. Gordon, Deokhwan Kim, Jeff Perkins, Limei Gilham, Nguyen Nguyen, and Martin Rinard. Information-flow analysis of android applications in droidsafe. In *Proceedings 2015 Network and Distributed System Security Symposium*, NDSS 2015. Internet Society, 2015. doi: 10.14722/ndss.2015.23089. URL http://dx.doi.org/10.14722/ndss.2015.23089.

[17] Philip Guo. Ten million users and ten years later: Python tutor's design guidelines for building scalable and sustainable research software in academia. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21. ACM, October 2021. doi: 10.1145/3472749.3474819. URL http://dx.doi.org/10.1145/3472749.3474819.

[18] Monique M. Hennink, Bonnie N. Kaiser, and Vincent C. Marconi. Code saturation versus meaning saturation: How many interviews are enough?: How many interviews are enough? *Qualitative Health Research*, 27(4):591–608, September 2016. ISSN 1552-7557. doi: 10.1177/1049732316665344. URL http://dx.doi.org/10.1177/1049732316665344.

[19] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. {SUPOR}: Precise and scalable sensitive user input detection for android apps. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 977–992, 2015.

[20] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. Why are they collecting my data?: Inferring the purposes of network traffic in mobile apps: Inferring the purposes of network traffic in mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–27, December 2018. ISSN 2474-9567. doi: 10.1145/3287051. URL http://dx.doi.org/10.1145/3287051.

[21] Patrick Gage Kelley, Joanna Bresee, Lorrie Faith Cranor, and Robert W. Reeder. A "nutrition label" for privacy. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09. ACM, July 2009. doi: 10.1145/1572532.1572538. URL http://dx.doi.org/10.1145/1572532.1572538.

[22] Patrick Gage Kelley, Lucian Cesca, Joanna Bresee, and Lorrie Faith Cranor. Standardizing privacy notices: an online study of the nutrition label approach: an online study of the nutrition label approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10. ACM, April 2010. doi: 10.1145/1753326.1753561. URL http://dx.doi.org/10.1145/1753326.1753561.

[23] Patrick Gage Kelley, Lorrie Faith Cranor, and Norman Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13. ACM, April 2013. doi: 10.1145/2470654.2466466. URL http://dx.doi.org/10.1145/2470654.2466466.

[24] Konrad Kollnig, Anastasia Shuba, Max Van Kleek, Reuben Binns, and Nigel Shadbolt. Goodbye tracking? impact of ios app tracking transparency and privacy labels. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22. ACM, June 2022. doi: 10.1145/3531146.3533116. URL http://dx.doi.org/10.1145/3531146.3533116.

[25] Li Li, Alexandre Bartel, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. I know what leaked in your pocket: uncovering privacy leaks on android apps with static taint analysis. *arXiv preprint arXiv:1404.7431*, 2014.

[26] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. Coconut: An ide plugin for developing privacy-friendly apps: An ide plugin for developing privacy-friendly apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2(4):1–35, December 2018. ISSN 2474-9567. doi: 10.1145/3287056. URL http://dx.doi.org/10.1145/3287056.

[27] Tianshi Li, Elizabeth Louie, Laura Dabbish, and Jason I. Hong. How developers talk about personal data and what it means for user privacy: A case study of a developer forum on reddit: A case study of a developer forum on reddit. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW3):1–28, January 2021. ISSN 2573-0142. doi: 10.1145/3432919. URL http://dx.doi.org/10.1145/3432919.

[28] Tianshi Li, Elijah B. Neundorfer, Yuvraj Agarwal, and Jason I. Hong. Honeysuckle: Annotation-guided code generation of in-app privacy notices: Annotation-guided code generation of in-app privacy notices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 5(3):1–27, September 2021. ISSN 2474-9567. doi: 10.1145/3478097. URL http://dx.doi.org/10.1145/3478097.

[29] Tianshi Li, Kayla Reiman, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I. Hong. Understanding challenges for developers to create accurate privacy nutrition labels. In *CHI Conference on Human Factors in Computing Systems*, CHI '22. ACM, April 2022. doi: 10.1145/3491102.3502012. URL http://dx.doi.org/10.1145/3491102.3502012.

[30] Yucheng Li, Deyuan Chen, Tianshi Li, Yuvraj Agarwal, Lorrie Faith Cranor, and Jason I. Hong. Understanding ios privacy nutrition labels: An exploratory large-scale analysis of app store data. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, CHI '22. ACM, April 2022. doi: 10.1145/3491101.3519739. URL http://dx.doi.org/10.1145/3491101.3519739.

[31] Michael Xieyang Liu, Andrew Kuznetsov, Yongsung Kim, Joseph Chee Chang, Aniket Kittur, and Brad A. Myers. Wigglite: Low-cost information collection and triage. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22. ACM, October 2022. doi: 10.1145/3526113.3545661. URL http://dx.doi.org/10.1145/3526113.3545661.

[32] Justin Lubin and Sarah E. Chasins. How statically-typed functional programmers write code. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–30, October 2021. ISSN 2475-1421. doi: 10.1145/3485532. URL http://dx.doi.org/10.1145/3485532.

[33] Aleecia M McDonald and Lorrie Faith Cranor. The cost of reading privacy policies. *I/S: A Journal of Law and Policy for the Information Society*, 4:543, 2008.

[34] Abraham H Mhaidli, Yixin Zou, and Florian Schaub. "we can't live without {Them!}" app developers' adoption of ad networks and their considerations of consumer risks. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 225–244, 2019.

[35] Yuhong Nan, Min Yang, Zhemin Yang, Shunfan Zhou, Guofei Gu, and XiaoFeng Wang. {UIPicker}:{User-Input} privacy identification in mobile applications. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 993–1008, 2015.

[36] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[37] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. In *Proceedings of the 22nd USENIX security symposium*, pages 543–558, 2013.

[38] Florian Schaub, Rebecca Balebako, Adam L. Durity, and Lorrie Faith Cranor. *A Design Space for Effective Privacy Notices**, page 365–393. Cambridge University Press. doi: 10.1017/9781316831960.021. URL http://dx.doi.org/10.1017/9781316831960.021.

[39] Mohammad Tahaei, Kopo M. Ramokapane, Tianshi Li, Jason I. Hong, and Awais Rashid. Charting app developers' journey through privacy regulation features in ad networks. *Proceedings on Privacy Enhancing Technologies*, 2022(3):33–56, July 2022. ISSN 2299-0984. doi: 10.56553/popets-2022-0061. URL http://dx.doi.org/10.56553/popets-2022-0061.

[40] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M. Drucker. Diff in the loop: Supporting data comparison in exploratory data analysis. In *CHI Conference on Human Factors in Computing Systems*, CHI '22. ACM, April 2022. doi: 10.1145/3491102.3502123. URL http://dx.doi.org/10.1145/3491102.3502123.

[41] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks. *ACM Transactions on Computer-Human Interaction*, 29(2):1–33, January 2022. ISSN 1557-7325. doi: 10.1145/3489465. URL http://dx.doi.org/10.1145/3489465.

[42] Haoyu Wang, Jason Hong, and Yao Guo. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15. ACM, September 2015. doi: 10.1145/2750858.2805833. URL http://dx.doi.org/10.1145/2750858.2805833.

[43] Edwin B. Wilson. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22 (158):209–212, June 1927. ISSN 1537-274X. doi: 10.1080/01621459.1927.10502953. URL http://dx.doi.org/10.1080/01621459.1927.10502953.

[44] Yue Xiao, Zhengyi Li, Yue Qin, Jiale Guan, Xiaolong Bai, Xiaojing Liao, and Luyi Xing. Lalaine: Measuring and characterizing non-compliance of apple privacy labels at scale. *arXiv preprint arXiv:2206.06274*, 2022.

[45] Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. How usable are ios app privacy labels? *Proceedings on Privacy Enhancing Technologies*, 2022(4):204–228, October 2022. ISSN 2299-0984. doi: 10.56553/popets-2022-0106. URL http://dx.doi.org/10.56553/popets-2022-0106.

[46] Tianyi Zhang, London Lowmanstone, Xinyu Wang, and Elena L. Glassman. Interactive program synthesis by augmented examples. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20. ACM, October 2020. doi: 10.1145/3379337.3415900. URL http://dx.doi.org/10.1145/3379337.3415900.

[47] Chengbo Zheng, Dakuo Wang, April Yi Wang, and Xiaojuan Ma. Telling stories from computational notebooks: Ai-assisted presentation slides creation for presenting data science work. In *CHI Conference on Human Factors in Computing Systems*, CHI '22. ACM, April 2022. doi: 10.1145/3491102.3517615. URL http://dx.doi.org/10.1145/3491102.3517615.

[48] Sebastian Zimmeck, Rafael Goldstein, and David Baraka. Privacyflash pro: Automating privacy policy generation for mobile apps. In *Proceedings 2021 Network and Distributed System Security Symposium*, NDSS 2021. Internet Society, 2021. doi: 10.14722/ndss.2021.24100. URL http://dx.doi.org/10.14722/ndss.2021.24100.

Table 4. The `collectionAttribute` field of the `@DataTransmission` annotation encodes the data collection information as a list of predefined attribute values. This table shows the groups of attributes that need to be completed in this field, as well as the corresponding collection questions and the exempt conditions of collection defined by Google.

Attribute name	Values	Original questions / Exempt conditions
`TransmittedOffDevice`	True or False	Is this data collected, shared, or both?
`NotStoredInBackend`	True or False	Is this data processed ephemerally?
`EncryptedInTransit`	True or False	Is all of the user data collected by your app encrypted in transit?
`OptionalCollection`	True or False	Is this data required for your app, or can users choose whether it's collected?
`UserToUserEncryption`	True or False	User data that is sent off device, but that is unreadable by you or anyone other than the sender and recipient as a result of end-to-end encryption does not need to be disclosed.
`CollectedFor`	Seven options: App functionality; Analytics; Developer communications; Advertising or marketing; Fraud prevention, Security and compliance; Personalization; Account Management	Why is this user data collected? Select all that apply.

A ANNOTATION DESIGN DETAILS

Table 4 and Table 5 summarize the design details of the `@DataTransmission` annotation.

Table 5. The `sharingAttribute` field of the `@DataTransmission` annotation encodes the data sharing information as a list of predefined attribute values. This table shows the groups of attributes that need to be completed in this field, as well as the corresponding sharing questions and the exempt conditions of sharing defined by Google.

Attribute name	Values	Original questions / Exempt conditions
`SharedWithThirdParty`	True or False	Is this data collected, shared, or both?
`OnlySharedWithServiceProviders`	True or False	Sharing is exempt if transferring user data to a "service provider" that processes it on behalf of the developer.
`OnlySharedForLegalPurposes`	True or False	Sharing is exempt if transferring user data for specific legal purposes, such as in response to a legal obligation or government requests.
`OnlyInitiatedByUser`	True or False	Sharing is exempt if transferring user data to a third party based on a specific user-initiated action, where the user reasonably expects the data to be shared.

`OnlyAfterGettingUserConsent`	True or False	Sharing is exempt if transferring user data to a third party based on a prominent in-app disclosure and consent that meets the requirements described in our User Data policy.
`OnlyTransferringAnonymousData`	True or False	Sharing is exempt if transferring user data that has been fully anonymized so that it can no longer be associated with an individual user.
`SharedFor`	Seven options: App functionality; Analytics; Developer communications; Advertising or marketing; Fraud prevention, Security and compliance; Personalization; Account Management	Why is this user data shared? Select all that apply.

B MATCHA IMPLEMENTATION DETAILS

This section summarizes details about the implementation of the Matcha IDE plugin. Table 7 and Table 6 presents the keywords used in Matcha to facilitate the detection of sensitive data access code. Table 8 presents all the SDKs that Matcha can detect and automatically generate the safety label for based on the SDK's open documentation about its data practices.

Table 6. Matcha keyword list (based on definitions and the keywords extracted from open-sourced projects that contain sensitive API calls). Matcha uses keyword search to complement the API-based detection of code that accesses sensitive user data.

Category	Data Type	Keywords
Personal Info	Name	name
	Email Address	email
	User ID	uid, user id
	Address	home address, city, country, zip code
	Phone Number	phone, default dialer
	Race and Ethnicity	race, ethnicity, african, indian, asian
	Political or Religious Beliefs	political, religious
	Sexual Orientation	sexual orientation, gay, lesbian, transgender, bisexual, queer
	Other Personal Info	birth, nationality, gender, male, female, non-binary, veteran
Financial Info	User Payment Info	credit card, billing, cvv, routing number, account number, bank
	Purchase History	purchase
	Credit Score	credit score
	Other Financial Info	salary, debt
Calendar	Calendar Events	calendar, attendee
Photos and Videos	Photos	photo, barcode, image, picture, media

	Videos	video, recording, media
Contacts	Contacts	contact, call history, interaction duration
Location	Approximate Location	location, city, country, ip address
	Precise Location	location, latitude, longitude
Health and Fitness	Health Info	health, medical, medicine, symptom, disease, doctor, physician, sleep, wellness, therapist, emergency, emergencies, period, pregnancy
	Fitness Info	fitness, exercise, workout, sport, diet, nutrition
Messages	Emails	email, sender, recipient, subject
	Sms or Mms	message, sms, mms, sender, recipient, subject
	In-App Messages	message, chat, reply, replies, comment, sender, recipient, subject
Device or Other IDs	Device or Other IDs	mac address, widevine, device id, instance id, app id, advertising id, fingerprint, user agent, unique id, token, AdvertisingId-Client
Files and Docs	Files and Docs	file, document, backup, restore, download, storage, media
Audio Files	Voice or Sound Recordings	voice, sound, recording
	Music Files	music, song
	Other User Audio Files	
App Activity	App Interactions	selected, visit number, view number, getItemAtPosition, getItemIdAtPosition, AccessibilityService, TextService, Instrumentation, shortcut
	Installed Apps	installed app
	In-App Search History	search
	Other User-Generated Content	bios, note, response
	Other User Activities	gameplay, dialog option
Web Browsing	Web Browsing History	browser, cookie, browser cache, browsing cache, search, web view
App Info and Performance	Crash Logs	crash, stack trace
	Diagnostics	ActivityManager, ApplicationErrorReport, ApplicationExitInfo, BatteryManager, Benchmark, Debug, HealthStats, Macrobenchmark, PowerManager, StrictMode, battery, loading time, latency, frame rate, diagnostics
	Other App Performance Data	performance

Table 7. Matcha keyword list (based on permissions). Matcha uses keyword search to complement the API-based detection of code that accesses sensitive user data.

Category	Data Type	Keywords
Personal Info	Name	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	Email Address	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	User ID	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS

	Address	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	Phone Number	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS, READ_CALL_LOG, READ_PHONE_NUMBERS, READ_PHONE_STATE, READ_SMS
	Race and Ethnicity	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	Political or Religious Beliefs	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	Sexual Orientation	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
	Other Personal Info	BIND_AUTOFILL_SERVICE, GET_ACCOUNTS
Financial Info	User Payment Info	BIND_AUTOFILL_SERVICE
	Purchase History	
	Credit Score	
	Other Financial Info	
Calendar	Calendar Events	READ_CALENDAR, WRITE_CALENDAR
Photos and Videos	Photos	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
	Videos	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
Contacts	Contacts	ACCEPT_HANDOVER, ADD_VOICEMAIL, ANSWER_PHONE_CALLS, CALL_PHONE, PROCESS_OUTGOING_CALLS, READ_CALL_LOG, READ_CONTACTS, READ_PHONE_NUMBERS, READ_PHONE_STATE, READ_SMS, RECEIVE_MMS, RECEIVE_SMS, RECEIVE_WAP_PUSH, SEND_SMS, WRITE_CONTACTS
Location	Approximate Location	ACCESS_COARSE_LOCATION, ACCESS_MEDIA_LOCATION
	Precise Location	ACCESS_FINE_LOCATION, ACCESS_MEDIA_LOCATION
Health and Fitness	Health Info	ACTIVITY_RECOGNITION, BODY_SENSORS
	Fitness Info	ACTIVITY_RECOGNITION, BODY_SENSORS
Messages	Emails	
	Sms or Mms	READ_SMS, RECEIVE_MMS, RECEIVE_SMS, RECEIVE_WAP_PUSH, SEND_SMS, WRITE_SMS
	In-App Messages	
Device or Other IDs	Device or Other IDs	AD_ID, READ_PRIVILEGED_PHONE_STATE
Files and Docs	Files and Docs	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, MANAGE_EXTERNAL_STORAGE
Audio Files	Voice or Sound Recordings	CAPTURE_AUDIO_OUTPUT, RECORD_AUDIO, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE

	Music Files	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
	Other User Audio Files	CAPTURE_AUDIO_OUTPUT, RECORD_AUDIO, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
App Activity	App Interactions	QUERY_ALL_PACKAGES
	Installed Apps	
	In-App Search History	
	Other User-Generated Content	
	Other User Activities	
Web Browsing	Web Browsing History	
App Info and Performance	Crash Logs	
	Diagnostics	BATTERY_STATS
	Other App Performance Data	

Table 8. Matcha third-party SDK list. Matcha can automatically detect 58 third-party SDKs and automatically fill out the data collection and sharing practices based on the SDK's documentation. The list is primarily curated based on the Google Play SDK Index and also contains a few SDKs developed by Google which also provided such type of documentation.

SDK Names	Maven ID Matching Pattern
AdMob	.*com.google.android.gms:play-services-ads.*|.*com.google.android.gms:play-services-ads-lite.*
Ironsource	.*com.ironsource.sdk:mediationsdk.*
Vungle	.*com.vungle:publisher-sdk-android.*
AppsFlyer	.*com.appsflyer:af-android-sdk.*
Adjust	.*com.adjust.sdk:adjust-android.* |.*com.android.installreferrer:installreferrer.* |.*com.adjust.sdk:adjust-android-webbridge.*
Chartboost	.*com.chartboost:chartboost-sdk.*
Tapjoy	.*com.tapjoy:tapjoy-android-sdk.*
Google Play Games Services	.*com.google.android.gms:play-services-games.*
Firebase Authentication	.*com.google.firebase:firebase-auth.*|.*com.google.firebase:firebase-auth-ktx.*
Firebase App Check	.*com.google.firebase:firebase-appcheck.*|.*com.google.firebase:firebase-appcheck-debug.*|.*com.google.firebase:firebase-appcheck-safetynet.* |.*com.google.firebase:firebase-appcheck-playintegrity.*
Firebase Cloud Firestore	.*com.google.firebase:firebase-firestore.*| .*com.google.firebase:firebase-firestore-ktx.*
Cloud Functions for Firebase	.*com.google.firebase:firebase-functions.*|.*com.google.firebase:firebase-functions-ktx.*
Firebase Cloud Messaging	.*com.google.firebase:firebase-messaging.*|.*com.google.firebase:firebase-messaging-ktx.*
Cloud Storage for Firebase	.*com.google.firebase:firebase-storage.*|.*com.google.firebase:firebase-storage-ktx.*
Crashlytics	.*com.google.firebase:firebase-crashlytics.*|.*com.google.firebase:firebase-crashlytics-ktx.*|.*com.google.firebase:firebase-crashlytics-ndk.*

Dynamic Links	.*com.google.firebase:firebase-dynamic-links.*|.*com.google.firebase:firebase-dynamic-links-ktx.*
Google Analytics	.*com.google.firebase:firebase-analytics.*|.*com.google.firebase:firebase-analytics-ktx.*
Firebase In-App Messaging	.*com.google.firebase:firebase-inappmessaging.*|.*com.google.firebase:firebase-inappmessaging-display.*|.*com.google.firebase:firebase-inappmessaging-ktx.*|.*com.google.firebase:firebase-inappmessaging-display-ktx.*
Firebase Installations	.*com.google.firebase:firebase-installations.*|.*com.google.firebase:firebase-installations-ktx.*
Firebase ML model downloader	.*com.google.firebase:firebase-ml-modeldownloader.*|.*com.google.firebase:firebase-ml-modeldownloader-ktx.*
Performance Monitoring	.*com.google.firebase:firebase-perf.*|.*com.google.firebase:firebase-perf-ktx.*
Realtime Database	.*com.google.firebase:firebase-database.*|.*com.google.firebase:firebase-database-ktx.*
Remote Config	.*com.google.firebase:firebase-config.*|.*com.google.firebase:firebase-config-ktx.*
RevenueCat	.*com.revenuecat.purchases:purchases.* |.*com.revenuecat.purchases:purchases-store-amazon.*
User Messaging Platform SDK	.*com.google.android.ump:user-messaging-platform.*
reCAPTCHA Enterprise	.*com.google.android.gms:play-services-recaptcha.*
ARCore	.*com.google.ar:core:.*
ML Kit	.*com.google.android.gms:play-services-mlkit-barcode-scanning.*|.*com.google.android.gms:play-services-mlkit-face-detection.*|.*com.google.android.gms:play-services-mlkit-image-labeling.*|.*com.google.android.gms:play-services-mlkit-image-labeling-custom.*|.*com.google.android.gms:play-services-mlkit-language-id.*|.*com.google.android.gms:play-services-mlkit-text-recognition.*|.*com.google.android.gms:play-services-code-scanner.*|.*com.google.mlkit:barcode-scanning.*|.*com.google.mlkit:camera.* |.*com.google.mlkit:digital-ink-recognition.*|.*com.google.mlkit:entity-extraction.*|.*com.google.mlkit:face-detection.*|.*com.google.mlkit:image-labeling.*|.*com.google.mlkit:image-labeling-custom.*|.*com.google.mlkit:language-id.*|.*com.google.mlkit:linkfirebase.* |.*com.google.mlkit:object-detection.*|.*com.google.mlkit:object-detection-custom.*|.*com.google.mlkit:playstore-dynamic-feature-support.*|.*com.google.mlkit:pose-detection.*|.*com.google.mlkit:pose-detection-accurate.* |.*com.google.mlkit:segmentation-selfie.*|.*com.google.mlkit:smart-reply.*|.*com.google.mlkit:text-recognition.*|.*com.google.mlkit:text-recognition-chinese.*|.*com.google.mlkit:text-recognition-devanagari.*|.*com.google.mlkit:text-recognition-japanese.*|.*com.google.mlkit:text-recognition-korean.*|.*com.google.mlkit:translate.*

Google Cast (cast-tv)	.*com.google.android.gms:play-services-cast-tv.*
Google Maps	.*com.google.android.gms:play-services-maps.*
Google Pay - Wallet SDK	.*com.google.android.gms:play-services-wallet.*
Google Pay - TapandPay SDK	.*com.google.android.gms:play-services-tapandpay.*
SafetyNet	.*com.google.android.gms:play-services-safetynet.*
Google Play Integrity	.*com.google.android.play:integrity.*
Snowplow Android Tracker	.*com.snowplowanalytics:snowplow-android-tracker.*
Kochava	.*com.kochava.base:tracker.*
Airship SDK	.*com.urbanairship.android:urbanairship-fcm.*|.*com.urbanairship.android:urbanairship-hms.*|.*com.urbanairship.android:urbanairship-message-center.* |.*com.urbanairship.android:urbanairship-adm.* |.*com.urbanairship.android:urbanairship-preference-center.* |.*com.urbanairship.android:urbanairship-automation.*
Appodeal SDK for Android	.*com.appodeal.ads:sdk.*
Apptentive	.*com.apptentive:apptentive-android.*
Branch	.*io.branch.sdk.android:library.*
Braze Android SDK	.*com.appboy:android-sdk-ui.*
Bugsnag	.*com.bugsnag:bugsnag-android.*
CleverTap Android SDK	.*com.clevertap.android:clevertap-android-sdk.*
Fyber Marketplace SDK	.*com.fyber:marketplace-sdk.*
HyprMX	.*com.hyprmx.android:HyprMX-SDK.*
Instabug	.*com.instabug.library:instabug.*
Interactive Media Ads (IMA) SDK	.*com.google.ads.interactivemedia.v3:interactivemedia.*
MoEngage Android SDK	.*com.moengage:moe-android-sdk.*
Ogury SDK	.*co.ogury:ogury-sdk.*
Pangle Ad SDK	.*com.pangle.global:ads-sdk.*
Pollfish	.*com.pollfish:pollfish-googleplay.*
PubMatic OpenWrap SDK	.*com.pubmatic.sdk:openwrap.*
Singular SDK	.*com.singular.sdk:singular_sdk.*
Smaato NextGen SDK	.*com.smaato.android.sdk:smaato-sdk.*|.*com.smaato.android.sdk:smaato-sdk-rewarded-ads.*|.*com.smaato.android.sdk:smaato-sdk-banner.*|.*com.smaato.android.sdk:smaato-sdk-interstitial.*
Start.io (Formerly StartApp)	.*com.startapp:inapp-sdk.*
Taboola SDK	.*com.taboola:android-sdk.*
Verve Group HyBid SDK (formerly PubNative)	.*net.pubnative:hybid.sdk.*

C PRE-STUDY SURVEY

Thank you for agreeing to participate in this CMU study on creating the Google Play data safety section. We look forward to interviewing you.

For this interview study, we will ask you to create the data safety section for one Android app that we selected from your recent Android apps. The selected app has been sent to you. If you are not sure which app to report on, please message us to ask.

In this pre-study survey, we would like to ask a few questions about you and the selected app. At the end of the survey, you will see a scheduling link where you can make a booking for our interview.

(1) What is your participant ID for this study? (The ID was sent to you.)

(2) What is the Google Play link to the app that you will report on? (The selected app was sent to you. If your app is not on Google Play, just provide the name of the app that you entered in the screening survey.)

(3) Please confirm that your app is mainly developed in Java (not other languages/frameworks such as Kotlin, Unity, Flutter, Cordova etc.)
- Yes, my app is mainly developed in Java
- No, my app is developed in other languages/frameworks

(4) Which option best describes this Android app?
- Commercial project
- Research project
- Course project
- Hobby Project
- Other [Free form response expected]

(5) (If the Q4 answer is commercial project) How many employees work in the company that developed this app?
- 1-4
- 5-9
- 10-19
- 20-49
- 50-99
- 100-249
- 250-499
- 500-999
- 1,000 or more

(6) Is this an individual-developed app or a group-developed app?
- individual
- group

(7) (if the Q6 answer is individual) How many people participated in the development of this app (including app design, mobile app, and server side development) ?
- 2-5
- 6-10
- 11-20
- more than 20

(8) Which of these roles describe your job for developing this app? (Please select all that apply)
- Mobile App Developer
- Backend Developer
- Data Scientist and Analyst
- Designer
- Project Manager
- Security Engineer
- Privacy Engineer
- Quality Assurance Analyst
- Other roles (please specify) [Free form response expected]

(9) Note that during the study, you will try out an Android Studio plugin and use it to generate the data safety label for your selected app. Therefore, please make sure to install Android Studio and have your app's source code readily available on the machine you use for the interview. Since our plugin may guide you to add annotations in your app's code, we recommend you to either create a copy of your app or commit all changes before the study. We need to collect the data safety labels you created during the study solely for research purposes, and we will not collect other data about your app. Feel free to remove the annotations and uninstall the plugin after the study. I have read and understood the requirements above and still want to participate in this study. If you have any concerns, please contact me before submitting the survey.
- Yes
- No

(10) What is the version of your Android Studio?

(11) Are you a professional Software Developer, i.e. software development is the major component of your job?
- Yes
- No

(12) Did you major in computer science or related fields in school?
- Yes
- No

(13) What is your gender?
- Man
- Woman
- Non-binary/third gender
- Prefer not to answer

(14) What is your age group?
- 18-24
- 25-34
- 35-44
- 45-54
- 55-64
- 65+
- Prefer not to answer

(15) In which country do you currently reside?

D INTERVIEW SCRIPT

D.1 Introduction

Thanks for agreeing to participate in our study. First, I need to read our standard introduction, as required by our study protocol.

Our group at CMU has been doing research for many years on tools for developers. We are currently working on a research project about the Google Play safety labels, which is a new feature of the Google Play store that shows details of Android apps to end users. Android developers are now required to provide the privacy details for their apps by answering certain questions about data collection and sharing. The general goal of our research is to learn about how Android developers accomplish this task to help us improve a developer tool we design and build to streamline this task.

We understand that you have developed an app named [the app name]. We would like to have you complete the task of creating a safety label for the selected app using different methods. We expect the entire study session to take approximately 90 minutes, though timing may vary depending on the complexity of the app. The study involves two tasks. In the first task, we will ask you to create the label on the Google Play developer console. Then we will ask you to create the label again using a different method. Finally, we will ask some follow-up questions regarding the labels you created during the study, how you perceive certain concepts, and whether you encountered any difficulty during the process. Since we want to observe how you completed this task, we would like you to share your screen during the interview. We need to record both the audio and the screen during the entire interview solely for analysis purposes. We will use Zoom to make the recordings. Only researchers in our group working on this project will have access to the recordings. The interviews will be transcribed automatically by Zoom and we may include parts of the transcripts in our research papers that do not identify you, your app, or your organization.

In the second task, we would like you to try out an Android Studio plugin developed by our lab. We would like you to install the plugin on your Android Studio and then open the source code of the selected app in this Android Studio. The plugin will guide you to create a csv file that you can import into the Google Play developer console to complete the safety label requirement. We will ask you to send us the csv file for analysis purposes. The plugin will not collect any other information about your app and you can either choose to keep it installed or remove it after the study.

Do you have the latest version of Android Studio IDE installed? Some features of the plugin may not work well if you're not using the latest version of Android Studio. [Proceed after getting their affirmative answer]

Do you have the source code prepared on this machine? Is it OK to install the plugin on your Android Studio? [Proceed after getting their affirmative answer]

And in the second task/later part of the study, the plugin may potentially guide you to make some slight modifications to your app, such as adding annotations and adding a configuration file. We highly recommend you to make a copy of your source code or commit all the previous changes before the study. Is this OK with you? [Proceed after getting their affirmative answer]

Your participation is entirely voluntary and you may quit the study at any time. If you don't feel comfortable answering a question, feel free to skip it and it will not affect your compensation. You must be 18 or older to participate in this study. You will be compensated $70 for participating. The interview will be conducted remotely through the computer. Since the interview will be recorded, it is important that you be in a private room, and not in an open-space cubicle, for example. These recordings may be stored on protected computers at CMU and on Zoom, with transcripts potentially edited using a service called Otter. There are no expected risks or benefits to you for participating, beyond the benefits of helping improve the understanding of privacy labels in general and helping you improve the accuracy of your label.

This study was approved by the Institutional Review Board (IRB) at CMU. We will not identify you, your app, or your organization in any publications that come out of this research without your written permission.

Is that all OK? If yes, please sign the consent form.

Is it OK if I record the interview? [Start recording after receiving their positive answer]

D.2 Background Questions

Now I'll introduce some background about the Google data safety section, which is the main topic we're discussing today. I prepared some slides and I'm gonna share my screen.

(The slides contain screenshots of Google Play safety labels. After showing the slides, ask the following questions)

- Have you heard about them before?
- Have you created any of these labels before?
- Have you heard about the iOS privacy label before? If so, have you created it for any iOS apps?

Before we get started, I'd love to learn a little bit more about your app. Can you briefly tell me:

- What's the app designed for?
- What was your role in the development?
- Is it still under active development?

D.3 Task 1: Use Google Play Developer Console to Create the Label

Verbal instruction: Now I'll introduce today's first task. I'd like you to log into the google play developer console using a test account provided by us, and create a label of the selected app. The label should accurately represent the data practices of the selected app.

Please handle this task as you normally would and take as long as you need. You are welcome to look at any documentation you would normally consult, except for the app's privacy label if it's available. In order for us to see any resources you use, please either share your full screen or open any additional resources in the same window where you're completing the task.

If you need a resource that is not currently available or would ordinarily ask somebody for help, please say aloud what resources you would use and who you would usually contact.

Please try to keep thinking aloud during this process. Basically that means tell me whatever comes to your mind when working on this task, such as say your thought process aloud or voice any questions or comments you have. When you think you're done, just let me know.

D.4 Task 2: Use Matcha to Create the Label

Verbal instruction: Our lab developed an Android Studio plugin to help you create the safety label in a semi-automated way. In the second task, you will create the label again using this tool. Now let me help you install the plugin and set up the environment.

[After installing the plugin] I have prepared a short video introducing how to use this plugin. I'll send the link to you. Could you play it from your end? This video contains some sound. Please let me know if the sound doesn't work correctly on your end.

[After the tutorial] Do you have any questions?

[After answering their questions] Now let's go back to the IDE. Similar to the first task, please try to keep thinking aloud during this process. Basically that means tell me whatever comes to your mind when working on this task, such as say your thought process aloud or voice any questions or comments you have.

D.5 Post-Study Interview

Now I would like to compare the two safety labels that you just created. For the first label you just created on the developer console, please open the developer console to show the preview. For the second label created with Matcha, please switch to the "label preview" tab in the IDE plugin. Put the two windows side by side so we can compare the results. We're anticipating there may be some discrepancies.

Before we compare the results, I want to reassure you that the goal of this study is not to measure your ability, and discussing these discrepancies will help us understand the effectiveness of our tool and identify challenges developers may encounter when handling this task, so please don't be shy in noting any inaccuracies in either label. Your perspective is really helpful, and no identifying information will be shared about you, your app, or your company, in our report.

When I go through each discrepancy instance between the two versions, could you tell me;

- Which version do you think is more accurate?
- What do you think could possibly cause the difference between the two privacy labels?

Next, I'd like to ask a few questions about your experience using the two tools.

How is the experience of using the Google Play developer console and Matcha? Which one do you prefer? Why?

What do you think about using Matcha to generate privacy labels in general?

[Showing the key features of Matcha using a few slides] Could you tell me which are the three features of Matcha that you felt the most useful and why?

We hope to deploy this tool in the future and would like feedback to help us improve the design and implementation. Is there anything that we can improve in this tool that can make you more likely to install and use it? Any other thoughts to share? We'll continue working on Matcha, if you have any friends that might be interested in it, let us know!

E PARTICIPANT OVERVIEW

Table 9 provides a detailed overview of the background of each participant and their app selected for the study.

F QUALITATIVE ANALYSIS CODE BOOK

We present the final code book of our qualitative analysis in Table 10.

Table 9. Participant Overview. Our sample features a good sample of developers and apps across several dimensions, including participant's geographic location (*Location*), app development purpose (*Purpose*), app development team size (*Team Size*), app downloads (*Downloads*), the current data safety label on Google Play (Current label), and participant's role(s) in the development team (*Participant's Role(s) In Team*). Nine out of the 12 participants had prior experience in publishing apps on the Google Play store (Play). The app development purposes involve four options, covering situations when the participant developed the app as part of their job (*Job*), as part of their hobby (*Hobby*), for a course project (*Course*), and for a research project (*Research*).

ID	Play	Location	Purpose	Team Size	Downloads	Current label	Participant's Role(s) in Team
F1	yes	Pakistan	Job	2-5	1M+	No data shared, 4 data types in 3 categories collected (App activity, App info and performance, and Device or other IDs)	Mobile App Developer, Designer
F2	no	U.S.	Course	2-5	Not Play	N/A	Mobile App & Backend Developer, Designer
F3	no	Nigeria	Hobby	1	Not Play	N/A	Mobile App & Backend Developer, Designer, Project Manager
F4	yes	Ukraine	Hobby	1	Not Play	N/A	Mobile App Developer
F5	yes	Georgia	Job	2-5	50K+	No data shared, 6 data types in 4 categories collected (Personal info, Photos and videos, App activity, and Device or other IDs)	Mobile App Developer
F6	no	U.S.	Course	2-5	Not Play	N/A	Mobile App & Backend Developer
F7	yes	Pakistan	Job	1	Not Play	N/A	Mobile App Developer
F8	yes	Pakistan	Job	1	100+	No data shared, no data collected	Mobile App Developer
F9	yes	Bangladesh	Hobby	1	500+	Not provided	Mobile App Developer
F10	yes	Egypt	Course	1	Not Play	N/A	Mobile App Developer
F11	yes	Pakistan	Job	1	100K+	Not provided	Mobile App Developer
F12	yes	India	Job	1	100K	Not provided	Mobile App Developer

Table 10. The complete codebook of our qualitative analysis of the interview recordings

Theme	Code	Memo	Example
Cause of error	Forgetfulness	The developer mentioned they forgot something about their apps, such as libraries integrated in the app or a feature implemented in the app.	“Sorry, I forgot about this section. It was for users to add text to their photos” (F7)
	Library	The developer mentioned misunderstanding related to third-party libraries.	“This one is more precise, because I did not know that the library was was doing it on its own behind the screen. So that’s why I did not put this information.” (F2)
	Misunderstanding about the task	The developer mentioned they did not understand something related to the data safety label creation task.	“I did not know that I need to report out other user generated content.” (F2)
	lack technical knowledge	The developer mentioned something that demonstrated their misunderstanding about technical concepts.	“I’m not sure if that is if I’m actually using the precise location are like an approximate location. That’s where I’m confused.” (F2)
Comment on Matcha	prefer Matcha - informative	The developer mentioned Matcha helped them learn useful information.	“I never care about data collection and I don’t even look at what we do. So I think I learned a lot from this” (F5)
	prefer Matcha - better flexibility	The developer mentioned Matcha gave them better flexibility in the label creation process.	“I want it because it gives me the flexibility and it give me the feeling of a developer’s mindset.” (F3)
	prefer Matcha - better engagement	The developer mentioned Matcha better involved them in the task.	“It’s a lot more involved process when you’re using Matcha than Google Play console.” (F2)
	prefer Matcha - accuracy	The developer mentioned Matcha improved the label accuracy.	“I prefer the plugin because it can search the privacy leak for developers.” (F4)
	prefer Matcha - easy to use	The developer mentioned Matcha was easy to learn and use.	“I think the quickfix for the annotation is pretty convenient” (F6)
	issue - redundancy	The developer complained that several tasks felt repetitive and unnecessary to them.	“Like for some strange reason I think, it looks for the word search, but isn’t search really common?” (F6)