# Honeysuckle: Annotation-Guided Code Generation of In-App Privacy Notices

TIANSHI LI, Carnegie Mellon University, USA
ELIJAH B. NEUNDORFER, Columbus State University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA

In-app privacy notices can help smartphone users make informed privacy decisions. However, they are rarely used in real-world apps, since developers often lack the knowledge, time, and resources to design and implement them well. We present `Honeysuckle`, a programming tool that helps Android developers build in-app privacy notices using an annotation-based code generation approach facilitated by an IDE plugin, a build system plugin, and a library. We conducted a within-subjects study with 12 Android developers to evaluate `Honeysuckle`. Each participant was asked to implement privacy notices for two popular open-source apps using the `Honeysuckle` library as a baseline as well as the annotation-based approach. Our results show that the annotation-based approach helps developers accomplish the task faster with significantly lower cognitive load. Developers preferred the annotation-based approach over the library approach because it was much easier to learn and use and allowed developers to achieve various types of privacy notices using a unified code format, which can enhance code readability and benefit team collaboration.

CCS Concepts: • **Security and privacy → Human and societal aspects of security and privacy**; • **Human-centered computing → Human computer interaction (HCI)**; • **Software and its engineering → Integrated and visual development environments**.

Additional Key Words and Phrases: Privacy, Privacy Notice, Transparency, Android Development, Java Annotation, Programming/Development Support, IDE Plugin, Code Generation

**ACM Reference Format:**
Tianshi Li, Elijah B. Neundorfer, Yuvraj Agarwal, and Jason I. Hong. 2021. Honeysuckle: Annotation-Guided Code Generation of In-App Privacy Notices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 5, 3, Article 112 (September 2021), 27 pages.
https://doi.org/10.1145/3478097

## 1 INTRODUCTION

Effective privacy notices are essential for helping users make informed privacy decisions. In the context of mobile apps, usable privacy researchers have advocated various in-app privacy notices that are concise, informative, and user-friendly [7, 47]. This body of work, along with increasing consumer concerns about privacy and an encouraging trend in recent privacy laws (e.g., GDPR) and policies (e.g., Google Developer Policies), has led to smartphone operating systems introducing many privacy features such as runtime permissions, background location access alerts, and visible indicators of ongoing audio recording.

Authors' addresses: Tianshi Li, Carnegie Mellon University, Pittsburgh, USA, tianshil@cs.cmu.edu; Elijah B. Neundorfer, Columbus State University, Columbus, USA, elijah.neundorfer@gmail.com; Yuvraj Agarwal, Carnegie Mellon University, Pittsburgh, USA, yuvraj@cs.cmu.edu; Jason I. Hong, Carnegie Mellon University, Pittsburgh, USA, jasonh@cs.cmu.edu.

2474-9567/2021/9-ART112
https://doi.org/10.1145/3478097

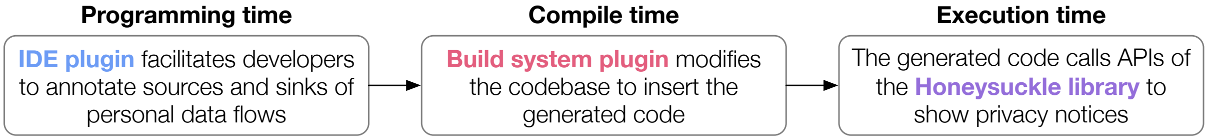
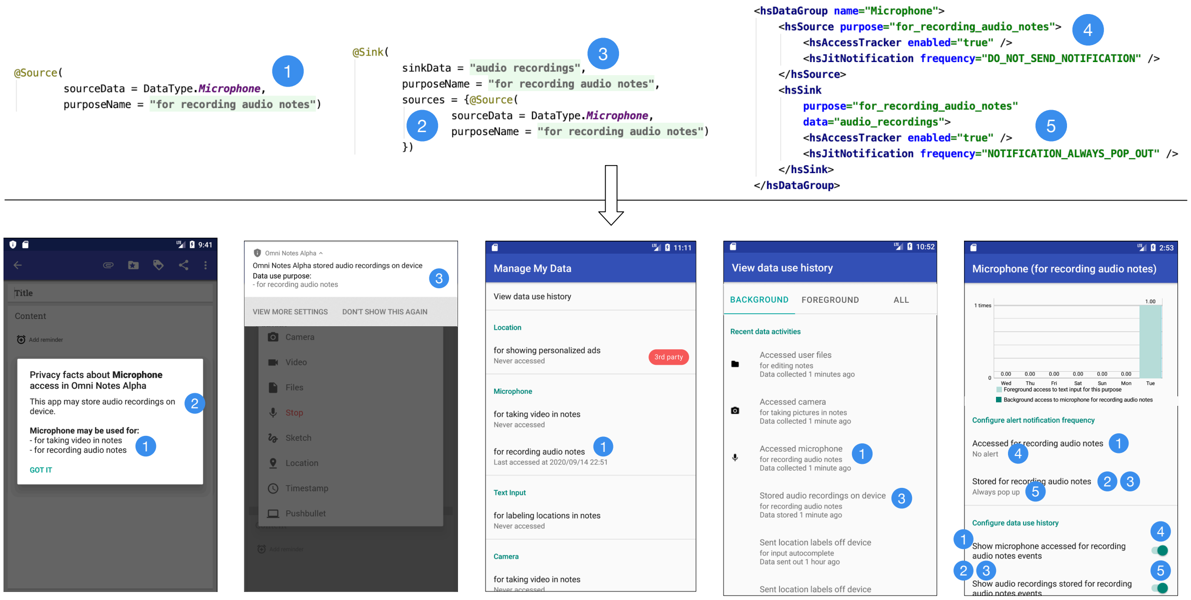

Programming time
IDE plugin facilitates developers to annotate sources and sinks of personal data flows
Compile time
Build system plugin modifies the codebase to insert the generated code
Execution time
The generated code calls APIs of the Honeysuckle library to show privacy notices

Fig. 1. Honeysuckle consists of three components: an IDE plugin, a build system plugin, and a privacy notice library. As illustrated by the flowchart, the three components each play a role at the programming time, the compile time, and the execution time of an app's life cycle to help developers implement in-app privacy notices. The IDE plugin facilitates developers to annotate the code, and the annotation will later be used to generate privacy notice code. The generated code will be inserted into the app during compile time by our build system plugin. Finally, the generated code calls APIs provided by our library to show privacy notices during runtime.

However, while system features like these can increase data transparency to some extent, they are often coarse-grained [56] and limited to data types predefined by the operating system [3, 4]. As such, this leaves much of the responsibility of providing effective in-app privacy notices on app developers' shoulders, which is a difficult task for several reasons. First, it has been repeatedly found that developers tend to focus more on security factors over aspects of privacy such as data transparency [23, 30, 48]. Second, developers' incomplete understanding of the concept of "privacy" may create inherent barriers to the adoption of techniques like in-app privacy notices. Third, the design and implementation of effective in-app privacy notices can be a challenging task even for developers who care about transparency issues. For instance, developers do not always know all data practices of their apps, especially when the app behavior gets complicated and involves the use of third-party libraries [12, 50], or when the data practices are not well documented [29]. Fourth, developers have limited knowledge about design choices and best practices for privacy notices, which has led to uninformative or even misleading ones in real-world apps [36, 42]. Lastly, there is little developer support to help developers keep pace with ever-changing legal and policy requirements, and systematically improve their apps' data transparency [30].

In this paper, we present Honeysuckle, a developer tool consisting of a plugin for the Android Studio IDE, a plugin for the Gradle build system, and a privacy notice library (Figure 1). Combined, these components jointly support a new approach of implementing in-app privacy notices, namely *annotation-based generation of privacy notices*. Instead of treating each privacy notice as a separate feature to implement, Honeysuckle only requires developers to (a) use Java's existing annotation feature to annotate *each data source and sink* with descriptions of data practices, e.g. declaring the purpose of data use; and (b) optionally modify a configuration file to fine-tune the frequency and visibility of the privacy notices. Honeysuckle can then transparently generate code at compile time that links to our privacy notice library, implementing the privacy notice UIs based on these privacy annotations, the configuration file, and code analysis results. Figure 2 illustrates how annotations are used to generate privacy notices and demonstrates a list of example privacy notices that can be generated using Honeysuckle.

Our novel annotation-based approach is grounded in three key design considerations that can benefit both developers and users. First, it gives developers control over the generated UI to provide context information and configure the behaviors of privacy notices at a relatively fine granularity. Second, it reduces the implementation and maintenance cost of in-app privacy notices by minimizing redundant information and providing alerts for inconsistency. Third, our IDE plugin offers features that can help developers track and annotate all places that collect, store, and transmit sensitive user data (i.e., data sources and sinks), which improves the accuracy and comprehensiveness of information conveyed in the privacy notices. Overall, we believe Honeysuckle demonstrates a viable path forward to reducing the barriers to adopting in-app privacy notices while also improving the consistency of in-app privacy notices across apps by different developers.



```
@Source(
        sourceData = DataType.Microphone,
        purposeName = "for recording audio notes")
```

```
@Sink(
        sinkData = "audio recordings",
        purposeName = "for recording audio notes",
        sources = {@Source(
                sourceData = DataType.Microphone,
                purposeName = "for recording audio notes")
        })
```

```
<hsDataGroup name="Microphone">
    <hsSource purpose="for_recording_audio_notes">
        <hsAccessTracker enabled="true" />
        <hsJitNotification frequency="DO_NOT_SEND_NOTIFICATION" />
    </hsSource>
    <hsSink
        purpose="for_recording_audio_notes"
        data="audio_recordings">
        <hsAccessTracker enabled="true" />
        <hsJitNotification frequency="NOTIFICATION_ALWAYS_POP_OUT" />
    </hsSink>
</hsDataGroup>
```

**Privacy facts about Microphone access in Omni Notes Alpha**

This app may store audio recordings on device.

**Microphone may be used for:**
- for taking video in notes
- for recording audio notes

Omni Notes Alpha

Omni Notes Alpha stored audio recordings on device
Data use purpose:
- for recording audio notes

Manage My Data

View data use history

Location
for showing personalized ads
Never accessed
3rd party

Microphone
for taking video in notes
Never accessed
for recording audio notes
Last accessed at 2020/09/14 22:51

Text Input
for labeling locations in notes
Never accessed

Camera
for taking video in notes
Never accessed

View data use history

Recent data activities

Accessed user files
for editing notes
Data collected 1 minutes ago

Accessed camera
for taking pictures in notes
Data collected 1 minute ago

Accessed microphone
for recording audio notes
Data collected 1 minute ago

Stored audio recordings on device
for recording audio notes
Data stored 1 minute ago

Sent location labels off device
for input autocomplete
Data sent out 1 hour ago

Sent location labels off device

Microphone (for recording audio notes)

1 times
0
1.00
0.00
0.00
0.00
0.00
0.00
0.00
Wed
Thu
Fri
Sat
Sun
Mon
Tue
Foreground access to text input for this purpose
Background access to microphone for recording audio notes

Configure alert notification frequency

Accessed for recording audio notes
No alert

Stored for recording audio notes
Always pop up

Configure data use history

Show microphone accessed for recording audio notes events

Show audio recordings stored for recording audio notes events

a) Privacy facts shown before permission requests

b) Just-in-time notifications

c) App-level summary of personal data use (3rd-party highlighted)

d) Data use history

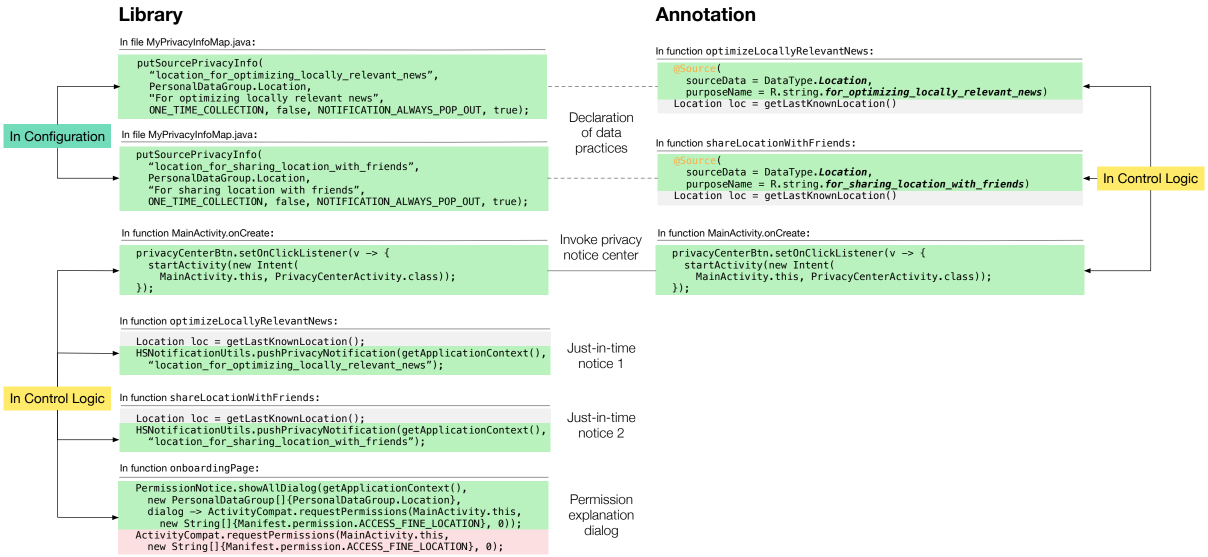
e) For each purpose: detailed data usage and in-app notice settings

Fig. 2. The top row shows the annotations and UI configuration XML that developers need to add or modify to generate privacy notices. The bottom row shows the five privacy notice features supported by Honeysuckle ("a" through "e"). We use the same number (1 through 5) to indicate which part of the annotation is used to populate which part of the generated UI, and how the configuration XML guides the presentation of the UI. The design of all these features were inspired by proposals from usable privacy research and privacy regulations. These examples demonstrate the breadth of privacy notices that Honeysuckle can generate with *one annotation* for each data source and sink.

To evaluate Honeysuckle, we conducted a within-subjects programming study with 12 Android developers. During the study, each participant was asked to implement and integrate in-app privacy notices (as demonstrated in Figure 2) for two popular open-source apps, a notification manager app with over 1M installs and a note-taking app with over 100K installs. A challenge, however, is that the current state of the art is to build these privacy notices manually, which would not be a good baseline for comparison since it clearly takes a great deal of time and energy to design and implement these notices. That is, we felt that we would not learn much against offering no support. Instead, we decided to compare our annotation-based approach (a relatively unfamiliar approach for participants) against simply using our own privacy notice library. We felt that this approach would be a reasonable approximation of how developers or large organizations might provide reusable privacy notices.

Our results show that the annotation-based approach proposed by Honeysuckle significantly reduced the time and the cognitive load of implementing fine-grained, contextualized privacy notices for the two open source apps than using our library directly. Within the 40-minute allocated time, all 12 developers completed all the required privacy notices using the annotation-based approach, while only 2 developers completed the same task using the library. All participants considered the annotation-based approach much easier to use and learn than the library-based approach because the clear step-by-step guidance provided by our Honeysuckle IDE plugin effectively reduced the learning curve, the use of quick-fixes and code generation allowed developers to write much less code, and the use of annotations reduced redundancies and made the code easier to comprehend.

This paper makes the following contributions:



Library

In file MyPrivacyInfoMap.java:

```
putSourcePrivacyInfo(
  “location_for_optimizing_locally_relevant_news”,
  PersonalDataGroup.Location,
  "For optimizing locally relevant news”,
  ONE_TIME_COLLECTION, false, NOTIFICATION_ALWAYS_POP_OUT, true);
```

In file MyPrivacyInfoMap.java:

```
putSourcePrivacyInfo(
  “location_for_sharing_location_with_friends”,
  PersonalDataGroup.Location,
  “For sharing location with friends”,
  ONE_TIME_COLLECTION, false, NOTIFICATION_ALWAYS_POP_OUT, true);
```

In function MainActivity.onCreate:

```
privacyCenterBtn.setOnClickListener(v -> {
  startActivity(new Intent(
    MainActivity.this, PrivacyCenterActivity.class));
});
```

In function optimizeLocallyRelevantNews:

```
Location loc = getLastKnownLocation();
HSNotificationUtils.pushPrivacyNotification(getApplicationContext(),
  “location_for_optimizing_locally_relevant_news”);
```

In function shareLocationWithFriends:

```
Location loc = getLastKnownLocation();
HSNotificationUtils.pushPrivacyNotification(getApplicationContext(),
  “location_for_sharing_location_with_friends”);
```

In function onboardingPage:

```
PermissionNotice.showAllDialog(getApplicationContext(),
  new PersonalDataGroup[]{PersonalDataGroup.Location},
  dialog -> ActivityCompat.requestPermissions(MainActivity.this,
    new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 0));
ActivityCompat.requestPermissions(MainActivity.this,
  new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 0);
```

Annotation

In function optimizeLocallyRelevantNews:

```
@Source(
  sourceData = DataType.Location,
  purposeName = R.string.for_optimizing_locally_relevant_news)
Location loc = getLastKnownLocation()
```

In function shareLocationWithFriends:

```
@Source(
  sourceData = DataType.Location,
  purposeName = R.string.for_sharing_location_with_friends)
Location loc = getLastKnownLocation()
```

In function MainActivity.onCreate:

```
privacyCenterBtn.setOnClickListener(v -> {
  startActivity(new Intent(
    MainActivity.this, PrivacyCenterActivity.class));
});
```

Fig. 3. An illustration of code changes required to implement the same in-app privacy notices for two location features using the library and the annotation-based method. In the code, green highlights indicate lines that were added; red indicates lines that are removed; grey indicates lines that are unchanged. Between the two conditions, the dashed lines connect code that conceptually serves the same goal, and the solid line connects the same code. We can see that using annotations substantially reduces the amount of code that developer needs to write. The annotation-based approach also supports a more uniform format while the library-based approach requires developers to handle low-level implementation tasks using different APIs and declare data practices in a different place from the actual sources and sinks (outside the control logic).

- We present the design and implementation of Honeysuckle, an Android developer tool that offers *annotation-based generation of privacy notices*. Honeysuckle consists of an IDE (Android Studio) plugin, a build system (Gradle) plugin, and a privacy notice library.
- We conducted a controlled, within-subjects programming study to evaluate these two approaches of implementing in-app privacy notices supported by Honeysuckle. Our results show that the annotation-based approach helped developers build in-app privacy notices faster with significantly lower cognitive load than manually implementing and integrating them using a library.

## 1.1 Motivating example: Library vs. Annotation

Below, we offer two scenarios to help convey the intuition of how Honeysuckle's two approaches for in-app privacy notices work. Bao is an Android developer working on a social media app and has just finished two features using location data. The first feature silently accesses the latest GPS location when the user refreshes the news feed, to optimize the relevance of the results based on the current location (mainly implemented in the function optimizeLocallyRelevantNews). The second feature allows users to view their current location in a map and send the location to selected friends.

She aims to implement three types of in-app privacy notices to enhance data transparency, as recommended by the usable privacy literature. First, she wants to show a dialog that explains both purposes before location permission requests are made, to help users make informed decisions [33, 53, 56]. Second, she wants to show just-in-time notifications when the app accesses location data to notify users about the data collection behavior and the corresponding purpose [47]. Lastly, she wants to implement a privacy notice center to provide an overview of all data practices and the purposes of this app, and allow users to configure the notice visibility and frequency based on their preferences [47]. Figure 2 shows the reference designs of these privacy notices.

One straightforward way to implement in-app privacy notices is to treat different types of privacy notices as different features and handle them individually. Hence, in the first scenario, we demonstrate how Bao can implement the three target features using the `Honeysuckle` library APIs. The "Library" column of Figure 3 illustrates the main code changes that Bao needs to make. She needs to manually track code that handles location data collection and location permission requests and modify the code to implement the privacy notices. She needs to declare information about the data practices such as the purpose in a separate file from the actual data source or sink. She also needs to choose the proper APIs for different types of privacy notices (e.g., `HSNotificationUtils.pushPrivacyNotification` and `PermissionNotice.showAllDialog` in this example), and to integrate the privacy notice center into the app. This entire process involves a lot of manual work. Informally, we observe that while this library approach is feasible, there will be challenges in terms of implementation and maintenance for apps with more data practices, especially if the types of required privacy notices change [1, 30]. We also re-iterate that currently for Android, there is no support for building these kinds of privacy notices.

The second scenario demonstrates an alternative approach, namely shifting from "implementing the features" to "declaring the data practices". This time Bao only needs to declare the data practices by adding annotations to code that handles the collection of location data in different contexts and manually integrate the privacy center so she can decide where to embed it. `Honeysuckle` can then automatically generate other code for in-app privacy notices (which still uses our library under the hood) and insert it at the right place in the source code. This method is a variation of the classic idea of "model-based user interface design" [41], which aimed to reduce developers' workload by having them focus on *what* rather than *how*. In addition, our `Honeysuckle` IDE plugin runs an always-on code analysis to detect code signatures of APIs that may be related to the collection, storage, and transmission of sensitive user data, so as to prompt developers to add these annotations in real time and provide quick-fixes to streamline the task. This can further reduce the burden of manually tracking all data practices and may improve the coverage and accuracy of the in-app privacy notices accordingly.

## 2 RELATED WORK

Studying and solving privacy issues in mobile apps is a longstanding topic in UbiComp research. Although most work has been focused on building and studying user-facing systems to protect privacy [15, 28, 40, 54, 55], we also noticed an emerging interest in studying developer's experiences and building usable developer tools to improve app privacy [29, 31]. Our work is focused on designing developer tools to facilitate the adoption of in-app privacy notices, so we review the design space of in-app privacy notices, detail challenges facing developers regarding in-app privacy notices and discuss other methods that help developers build privacy-friendly apps.

### 2.1 Design space of in-app privacy notices

Research in usable privacy has argued that effective privacy notices should be concise, contextualized, and emphasize unexpected data practices [22, 33, 43, 47]. In this paper, we use the term in-app privacy notices to refer to notices that are well-integrated into the app's interaction flow and customized based on a deep understanding of the app's design and data practices.

However, designing effective in-app privacy notices is not a trivial task. Several common issues need to be taken into account to achieve a usable and useful design. For example, designers need to reduce the notice complexity by using concise description and plain language, mainly for two reasons. First, prior research has found that it takes a great deal of time to read through lengthy privacy policies [38] and understand the data practices and corresponding implications [20]. Second, the limited screen space constrains the amount of information that can be conveyed [47]. Furthermore, designers need to grapple with notice fatigue [46, 47] and habituation [27, 46, 47] by finding a balance between increasing transparency and decreasing disruption. A common approach to addressing these issues is to use multi-layered notices [47] that can first present data practices that are less expected by users initially and further present more information on demand [33]. However, there is no one-size-fits-all definition of unexpected data practices, and designers need to analyze their own app design and conduct user studies to understand the perceptions of different audiences [32, 34].

Prior research has proposed many designs of in-app privacy notices such as just-in-time privacy notices [16] and privacy dashboards that show the access frequency of specific data [7, 11]. By synthesizing privacy notice designs in research proposals and real-world systems, Schaub et al. [47] presented a general framework to outline the design space of effective privacy notices for multiple platforms, including mobile apps. Their framework consists of four dimensions: timing, channel, modality, and control. For each dimension, they proposed multiple design opportunities to achieve the key requirements of effective privacy notices, including making the notice simple, using multi-layered notices, emphasizing unexpected data practices, and providing contextual information.

In our work, we present the `Honeysuckle` system that supports the implementation of a set of in-app privacy notices that draw on design recommendations of prior research. Inspired by the issues and requirements mentioned above, our key design goals include allowing developers to be able to adopt the best practices without becoming an expert in designing in-app privacy notices and without having to add or modify a large amount of code.

### 2.2 Challenges facing developers regarding the adoption of in-app privacy notices

Currently, developers are responsible for implementing in-app privacy notices with proper content, in proper formats, and with proper timing. However, prior research has found that developers often lack awareness, knowledge, and motivation to comply with privacy requirements [23, 29, 30, 48], which leads to several challenges to the adoption of in-app privacy notices.

The first challenge is developers' lack of awareness and knowledge regarding the importance of data transparency and how to use in-app privacy notices to improve transparency. Sheth et al. [48] conducted an online survey with developers and found that they preferred technical measures like data anonymization to mitigate privacy risks, perceiving privacy policies as less effective. Li et al. [29] interviewed Android developers and found most of them do not know the best privacy practices listed in the official Android documentation. Liu et al. [36] analyzed the custom rationale messages of permission requests in Android apps and found that a large portion of these messages merely paraphrased the permission request, which corroborated the findings of the interview study. Altogether, these findings suggest that many developers may not have enough awareness of the benefits of in-app privacy notices nor knowledge of how to design them.

The second challenge is in having a comprehensive and precise understanding of data practices across the entire app. Schaub et al. [47] clearly indicated that the first step in designing effective privacy notices is to thoroughly understand the data practices of the app. However, this goal can be hard for developers, even for apps that they participated in developing. One reason is that implementation details are not documented on time [29, 35]. Another is that developers are not always aware of the behaviors of third-party libraries, which might automatically collect user data [12, 50]. Overall, not addressing these issues may result in uninformative or even misleading privacy notices that are harmful to users' privacy.

The third challenge is that implementing in-app privacy notices increases implementation and maintenance costs. Although the design of in-app privacy notices has been studied a lot, there is little research in the implementation of these in-app privacy notices. There are also few developer tools to facilitate implementation, other than libraries that help handle permission requests required by the operating system [2] or comply with recent privacy laws such as requesting for consent before data collection [5]. Furthermore, the currently available developer tools have narrow purposes, such as "providing the consent dialog required by GDPR for the Google

Admob library", which offers little help with maintaining an app if there are new privacy laws or app store policies to comply with. In fact, recent research examining an Android developer subreddit found that developers complained about this type of unnecessary cost and had negative attitudes towards complying with privacy requirements [30].

In our work, we present `Honeysuckle` as a tool to help developers with the design and implementation of in-app privacy notices. The design of `Honeysuckle` is largely driven by the goal to address the three challenges above. In particular, our annotation-based code generation allows developers who are not experts in privacy to follow best practices in privacy notice design because those best practices are already embodied in the generated interfaces. `Honeysuckle` also saves on implementation and maintenance work because low-level implementation details are hidden from developers, with some details that can be customized using annotations and a configuration file. Furthermore, our IDE plugin can automatically scan all potential sources and sinks and prompt developers to annotate them to ensure comprehensive coverage in the generated privacy notices.

## 2.3 Auto-generated in-app privacy notices in current mobile apps

An orthogonal approach to improving data transparency of mobile apps is to use auto-generated in-app privacy notices, while developers are not involved in the creation of these privacy notices at all. The most widely available auto-generated privacy notices are privacy features provided by the operating system. For example, in iOS, an icon will show up on the notification bar when audio or location recording is ongoing. Another feature is to alert users if an app just copied data from their clipboard. The system also occasionally reminds users of background location access of a certain app [3]. Although these features do improve transparency, the lack of customization of what information is displayed and when and how it is displayed makes them inherently insufficient to accommodate the varying privacy preferences among different people [32] and under different contexts [33].

Researchers and practitioners have also explored building privacy managers as third-party apps. Examples include monitoring data collection and push notifications to alert users about a data access [6, 15], and monitoring network traffic to alert users about potential leaks of sensitive personal data [45, 49, 50]. Despite the benefits of offering better privacy protection in a plug-and-play manner, these privacy manager apps also have limitations, including 1) performance degradation caused by the overhead of dynamic program analysis or running an VPN to intercept all network traffic, 2) imposing extra privacy risks when allowing a third-party app to monitor data flows and intercept network traffic, 3) automated analysis has limited ability in inferring fine-grained data usage information, and 4) potential false positives and false negatives due to app behavior inference.

We consider that `Honeysuckle` can offer complementary benefits and work in parallel with system-generated privacy notices and third-party privacy managers. First, developers can have more control over `Honeysuckle`-generated privacy notices than system-generated notices. Second, since `Honeysuckle` modifies the code at compile time, `Honeysuckle`-generated privacy notices are integrated as a native part of the app, leading to a minimal impact on performance and no additional privacy risks. Third, having developers annotate data practices can provide more precise descriptions about the rationales than inferring them.

## 2.4 Using annotations to help developers enhance privacy

Annotations are a form of syntactic metadata supported in many programming languages, such as Java and Kotlin. As part of our work we extended Java annotations specifically for privacy. Here, we review existing developer tools and research that also use annotations for privacy, to highlight where our work differs.

Annotations, in general, are non-functional code which can help with tasks such as code generation and static analysis [57]. Android developers have created libraries using annotation-based code generation to help handle permissions, such as the popular open-source library PermissionsDispatcher [2]. However these projects do not provide help beyond handling Android permission requests. Ernst et al. [19] studied how to use annotations to increase the accuracy of static analysis, though this work was more for finding malicious code rather than improving transparency of app data practices for end users.

Annotations have also been used in IDEs to provide better privacy-related feedback to developers while programming. Li et al. [29] presented Coconut, an IDE plugin which can detect sensitive API calls and request developers to annotate these API calls. These annotations could be used by the IDE to provide suggestions to improve privacy handling, e.g. suggesting alternative APIs, and to offer a privacy panel in the IDE that makes it easy to see all uses of sensitive data in one place. The primary difference with `Honeysuckle` is that Coconut does not offer any support for in-app privacy notices. There are also secondary differences, e.g. `Honeysuckle`'s annotations are designed to capture more sophisticated data practices (e.g., data flows) and code patterns (e.g., handling sensitive API calls in helper functions that can be used for multiple purposes), and `Honeysuckle` has a different system architecture to support its functions, including a UI configuration file that developers can customize and a Gradle plugin for inserting generated code.

## 3 HONEYSUCKLE DESIGN AND IMPLEMENTATION

To help developers who are not privacy experts implement in-app privacy notices following the best practices, we designed and implemented `Honeysuckle`. Our design is mainly driven by the three goals:

**D1** Give developers control over the privacy notice UI.
**D2** Reduce the implementation and maintenance cost of privacy notices.
**D3** Increase the accuracy and coverage of privacy notices regarding the actual app behavior.

**D1** is a basic requirement as developers need to have sufficient flexibility to customize the generated interfaces to tailor to the app behavior. In addition, this design goal is also crucial for leveraging developers' knowledge about their apps to generate in-app privacy notices that can accommodate different users' privacy preferences under different contexts as discussed in Section 2.3. **D2** is motivated by the implementation and maintenance cost challenge as explained in Section 2.2. Since developers usually treat privacy as a secondary concern [10, 29], minimizing the workload of implementing privacy features is an important requirement for voluntary adoption. **D3** is motivated by the challenges of developers lacking awareness and knowledge about privacy and often not having a precise and up-to-date understanding of their own apps' data practices as discussed in Section 2.2.

`Honeysuckle` consists of three components: an Android Studio plugin, a Gradle plugin, and a privacy notice library. Developers can use the `Honeysuckle` library alone to implement privacy notices in a conventional way, which gives developers control (**D1 ✓**), albeit at the cost of still significant implementation and maintenance effort (**D2 ✗**). Developers may also not be able to keep track of all data practices due to challenges detailed in Section 2.2, which can result in inaccurate or incomprehensive privacy notices (**D3 ✗**)

Alternatively, developers can use the entire `Honeysuckle` system to annotate the code and modify a configuration file to generate code that calls the library to create in-app privacy notices. This method greatly reduces the implementation and maintenance effort (**D2 ✓**) and provides the same control over the generated UI. (**D1 ✓**). Furthermore, our `Honeysuckle` Android Studio plugin provides features that help increase the accuracy and coverage of data practices declared in the annotations to improve the generated privacy notices (**D3 ✓**).

In the following, we first present the design of the in-app privacy notices supported by `Honeysuckle`. Then we describe how we designed `Honeysuckle` to support the two approaches mentioned above to implement these in-app privacy notices. We describe the implementation of the main modules of `Honeysuckle` at the end.

### 3.1 Privacy notice designs and rationales

We present the design of the in-app privacy notices supported by `Honeysuckle` and explain their relationship with best practices advocated by prior literature. Our design decisions concern two aspects: timing and content.

*3.1.1 Timing.* According to Schaub et al. [47], the timing of these notices is a key consideration. We designed three types of privacy notices corresponding to three timing options described in that paper: *at setup*, *just in time*, and *on demand*.

- **T1** *at setup*: A dialog shown before permission requests that proactively explains the purposes of the permission request and the related data egress and retention behavior (Figure 2 (a)).
- **T2** *just in time*: A just-in-time notification that appears immediately after the data is accessed or sent off the device by the app (Figure 2 (b)).
- **T3** *on demand*: A privacy center that contain a hierarchical overview of all sensitive data practices in this app. Unlike the previous two types, the privacy center shows up on demand, so it can contain more in-depth information about the apps' data practices (Figure 2 (c)(e)(f)).

*3.1.2 Content.* As discussed in Section 2.1, privacy notices should reduce potential notification fatigue and habituation by highlighting unexpected and sensitive data practices. Our design is based on recommendations of prior research, common practices of real-world systems, and requirements of privacy laws. We detail each design and the supporting literature below.

- **I1** Data use purpose: Prior research has recommended apps to explicitly explain their purposes of data use to users especially before permission requests to help them make informed decisions and also feel more comfortable with sharing data with the app [33, 53, 56]. These recommendations were later adopted by the Android official documentation about best practices regarding permission requests [18].
- **I2** Third-party data use: Research has showed that 3rd-party data use causes most privacy concerns because they are unlikely to be expected by users and are often used for purposes such as targeted advertising that do not provide direct benefits to users [15, 51]. Many privacy laws and app store policies now require developers to explicitly specify data sharing with third parties in privacy policies.
- **I3** Data egress and retention: Data egress and retention can both lead to higher secondary data use and data breach risks. Researchers have proposed program analysis techniques to automatically detect [21] and alert users about these sensitive data practices [37].
- **I4** Background access: Background data access is likely unexpected by the user, and therefore deserves more explicit alerts [43, 47] For example, both iOS and Android allow users to only grant permission to location collection in foreground. iOS also occasionally alerts users about background location collection.
- **I5** Data access frequency and time: Prior research has demonstrated that showing the when and how frequently an app has accessed sensitive data can nudge users to improve their awareness of sensitive data practices and take more action to protect their privacy [7, 11].

## 3.2 `Honeysuckle` Library-Only Mode

We designed a library for implementing all the proposed features. Figure 3 shows an example of how to use the library to implement in-app privacy notices for two location features in a social media app. Developers need to use three different APIs to implement privacy notices shown at different times.

For **T1**, developers need to replace all permission requests with `PermissionNotice.showAllDialog` function calls to show the permission explanation dialog.

For **T2**, developers call the function `HSNotificationUtils.pushPrivacyNotification` after all data sources and sinks to show a just-in-time notification when the data is collected or sent off the device. A string identifier of the current data usage needs to be passed as a parameter to retrieve the purpose of the current data usage and configurations about how to show the privacy notice declared in a separate file (see the `MyPrivacyInfoMap.java` in Figure 3 as an example; details will be provided later).

For **T3**, developers call the system API `startActivity` and pass `PrivacyCenterActivity.class` as a parameter to open the privacy center. `PrivacyCenterActivity` is an activity class defined by the `Honeysuckle` library.

It can automatically load data practices declared in the configuration file and populate the privacy center with these data practices at runtime. Note that developers need to manually call this API both when using the library alone and when using annotations to generate privacy notices, because developers should be able to determine where to embed the entry point of the privacy center in their apps.

In addition to the three APIs, developers also need to declare what information will be presented in the privacy notices (based on information types listed in Section 3.1.2) and default configurations about how to show the privacy notices for each source and sink using the `putSourcePrivacyInfo` and `putSinkPrivacyInfo` APIs. Each API call creates and registers an instance in a *global privacy information map* that is used to populate privacy notices at runtime, so the first parameter of the API call needs to be a unique identifier for retrieving the instance. Developers need to pass five parameters to `putSourcePrivacyInfo`: the data type by choosing a predefined option, the purpose of data use, whether the data is used by a third-party library, whether the data will be stored or sent off the device, and whether it is a one-off, periodic, or continuous data collection. For each data sink, developers need to provide all of the parameters above, as well as a list of the identifiers of all the sources of the data flow to the sink, and a string that describes the type of data that will leave the sink node. Our `Honeysuckle` library can automatically determine whether the app collects sensitive data in the background, hence developers do not need to handle this themselves.

Regarding how and which type of privacy notices is shown, developers need to pass two parameters. One is an enum parameter that determines how frequently the just-in-time notification will pop up (e.g., one possible option is `NOTIFICATION_ALWAYS_POP_OUT` as demonstrated in Figure 3). The other is a boolean parameter that indicates whether to record the timestamps of a data collection, data retention, or data egress activity and show them in the privacy notice center (see Figure 2 (c) (d) and (e)).

## 3.3 `Honeysuckle` Annotation-Based Code Generation Mode

Although the `Honeysuckle` library can help generate in-app privacy notices, it comes at a significant development cost due to all the repetitive code that has to be added. For instance, developers need to explicitly specify information already encoded in the source and sink API such as the data type, whether it is a third-party API, and whether it is a one-off or periodic data access. Furthermore, using the `Honeysuckle` library alone leads to higher maintenance effort since the code that handles privacy UI is scattered across the project. These limitations suggest that the library approach may not effectively achieve design goal **D2**. In the meanwhile, using the the `Honeysuckle` library alone may also not satisfy the design goal **D3** because developers may not recall all the data practices of their app and not keep up-to-date documentation [12, 29].

To address this problem, we propose *annotation-based code generation*. The key idea is to move from having developers directly implement in-app privacy notices to having them declare relevant privacy-related information using *custom Java annotations and a UI XML configuration*. Our *build system plugin* can then automatically generate code that implements in-app privacy notices based on that information at compile time.

To help developers add annotations that cover all sources and sinks and include accurate information (addressing **D3**), we designed the `Honeysuckle` *IDE plugin* to give developers real-time feedback to help them identify missing annotations or annotations with errors. Our IDE plugin also provides several automation features, such as quick-fixes to annotation errors and automatically generating a default UI XML configuration for each source and sink. These features further streamline the actions developers need to do at programming time, thus helping with **D2**.

### 3.3.1 Annotation and UI XML configuration.

We put the information related to data practices in annotations that are directly embedded in the control logic right next to the corresponding data sources and sinks and put the information about the UI behavior in a separate UI XML configuration file (see examples in Figure 2). The annotations and the UI XML configuration together serve the same goal of the privacy information map in the library mode (Section 3.2), while the new method leverages the software design principle of separation of concerns to result in code that is easier to read, maintain, and analyze.

Table 1. `@Source` and `@Sink` annotation definition in `Honeysuckle`.

| Annotation type | Annotation fields |
|---|---|
| @Source | sourceData (DataType enum value)<br>purposeName (int, String resource ID) |
| @Sink | sources (@Source list)<br>sinkData (int, String resource ID)<br>purposeName (int, String resource ID) |

Table 1 demonstrates the definition of the two main annotations: `@Source` and `@Sink`. While programming, developers need to annotate the sources and sinks of the sensitive data flows in their app using these two annotations. The two annotations *only* need the purposes of data use and the connection between sources and sinks from developers, because other information in privacy notices can be automatically inferred and inserted into the generated code at compile time.

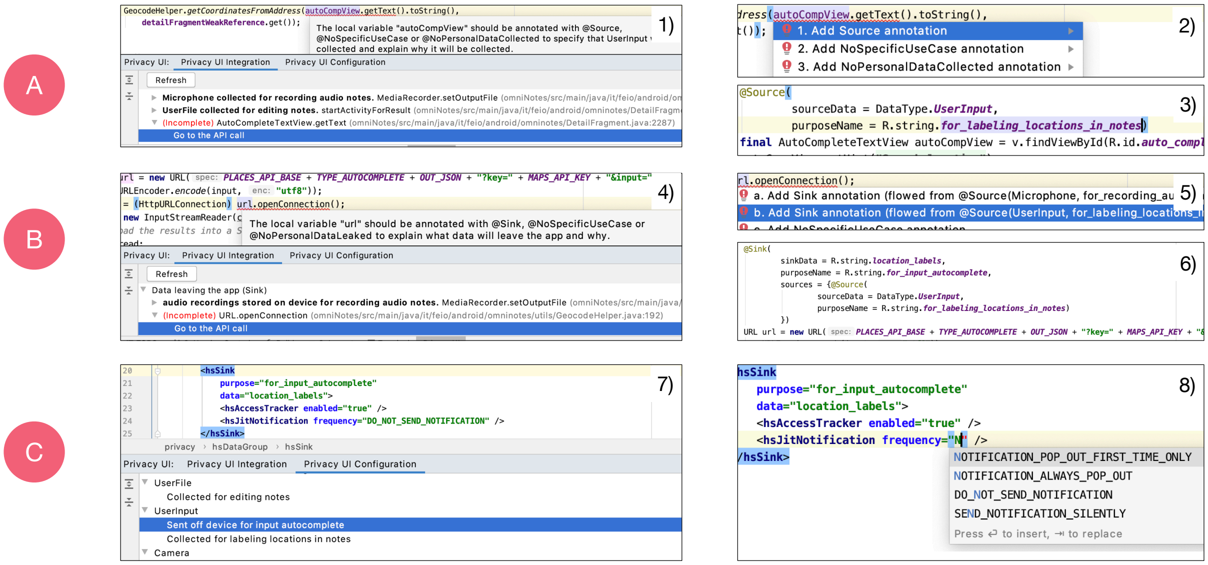
We design the UI XML configuration to provide developers with some options to control how to show the privacy notices at the granularity of `<data type, purpose>` pairs. Note that multiple API calls can be used collectively to achieve the same purpose. In our terminology, they are considered as the same sources and sinks.

*3.3.2* `Honeysuckle` *IDE plugin.* The two main design goals of the IDE plugin are to help improve the accuracy and coverage of annotations and streamline the editing of annotations and the UI XML configuration. To achieve this goal, our IDE plugin provides developers with step-by-step guidance to walk through a usual workflow for implementing in-app privacy notices using the annotation-based approach as demonstrated in Figure 4.

This workflow can be roughly divided into three steps (corresponding to A, B, C in Figure 4), although there is no strict order restriction. In the first two steps, developers can see all the source and sink API calls that remain to be annotated marked in code and listed in the "Privacy UI Integration" panel (Figure 4 (1) and (4)). Then they can use the quick-fixes to add a partially filled `@Source` (or `@Sink`) annotation to the source (or sink) code. There are three design considerations for the quick-fixes that we want to highlight. First, for the `purposeName` field, our quick-fix will generate a "for_" prefix to nudge developers to use expressions like *"for labeling locations in notes"* that provide concrete explanations of the data use purpose rather than simply paraphrase generic permission requests [36]. This consideration is also explained to developers in error messages for missing annotations. Second, for each sink, we have multiple quick-fixes that each generate a `@Sink` annotation with an existing `@Source` annotation filled in the `sources` field (Table 1). We also have quick-fixes to add more `@Source` annotations to `@Sink` annotations. Third, since false positives are inevitable in automatically detected sources [26] and sinks [8], we allow developers to use `NoPersonalDataCollected` to explicitly indicate that sensitive user data is not collected at this point to suppress the alerts. However, there is a risk that developers may provide false information and we discuss potential mitigating approaches in Section 5.4.1.

In the third step, developers can use the "Privacy UI Configuration" panel to examine the current UI XML configuration in the panel or in the original code. To minimize the code that developers need to write, our IDE plugin automatically updates the configuration file if new sources and sinks have been added. Therefore, developers only need to modify this file if they want to change certain options. Furthermore, code auto-completion is provided when developers edit the configuration file (Figure 4 (8)).

In addition to the usual workflow, our IDE plugin also runs code inspection continuously to help developers identify and fix invalid or outdated annotations and UI configuration entries (Figure 5). To raise developers'



```
GeocodeHelper.getCoordinatesFromAddress(autoCompView.getText().toString(),
    detailFragmentWeakReference.get());
```

The local variable "autoCompView" should be annotated with @Source, @NoSpecificUseCase or @NoPersonalDataCollected to specify that UserInput collected and explain why it will be collected.

- ▶ **Microphone collected for recording audio notes.** MediaRecorder.setOutputFile (omniNotes/src/main/java/it/feio/android/om
- ▶ **UserFile collected for editing notes.** startActivityForResult (omniNotes/src/main/java/it/feio/android/omninotes/DetailFragm
- ▼ (Incomplete) AutoCompleteTextView.getText (omniNotes/src/main/java/it/feio/android/omninotes/DetailFragment.java:2287)

```
dress(autoCompView.getText().toString(),
t());
```

1. Add Source annotation ▶
2. Add NoSpecificUseCase annotation ▶
3. Add NoPersonalDataCollected annotation ▶

```
@Source(
        sourceData = DataType.UserInput,
        purposeName = R.string.for_labeling_locations_in_notes)
final AutoCompleteTextView autoCompView = v.findViewById(R.id.auto_compl
```

```
url = new URL( spec: PLACES_API_BASE + TYPE_AUTOCOMPLETE + OUT_JSON + "?key=" + MAPS_API_KEY + "&input="
URLEncoder.encode(input,  enc: "utf8"));
= (HttpURLConnection) url.openConnection();
new InputStreamReader(c
oad the results into a S
read:
```

The local variable "url" should be annotated with @Sink, @NoSpecificUseCase or @NoPersonalDataLeaked to explain what data will leave the app and why.

- ▼ Data leaving the app (Sink)
  - ▶ **audio recordings stored on device for recording audio notes.** MediaRecorder.setOutputFile (omniNotes/src/main/java/
  - ▼ (Incomplete) URL.openConnection (omniNotes/src/main/java/it/feio/android/omninotes/utils/GeocodeHelper.java:192)

```
rl.openConnection();
```

a. Add Sink annotation (flowed from @Source(Microphone, for_recording_au
b. Add Sink annotation (flowed from @Source(UserInput, for_labeling_locations_i
c. Add NoSpecificUseCase annotation

```
@Sink(
        sinkData = R.string.location_labels,
        purposeName = R.string.for_input_autocomplete,
        sources = {@Source(
                sourceData = DataType.UserInput,
                purposeName = R.string.for_labeling_locations_in_notes)
        })
URL url = new URL( spec: PLACES_API_BASE + TYPE_AUTOCOMPLETE + OUT_JSON + "?key=" + MAPS_API_KEY + "&
```

```
<hsSink
    purpose="for_input_autocomplete"
    data="location_labels">
    <hsAccessTracker enabled="true" />
    <hsJitNotification frequency="DO_NOT_SEND_NOTIFICATION" />
</hsSink>
```

- ▼ UserFile
  - Collected for editing notes
- ▼ UserInput
  - Sent off device for input autocomplete
  - Collected for labeling locations in notes
- ▼ Camera

```
hsSink
    purpose="for_input_autocomplete"
    data="location_labels">
    <hsAccessTracker enabled="true" />
    <hsJitNotification frequency="N" />
/hsSink>
```

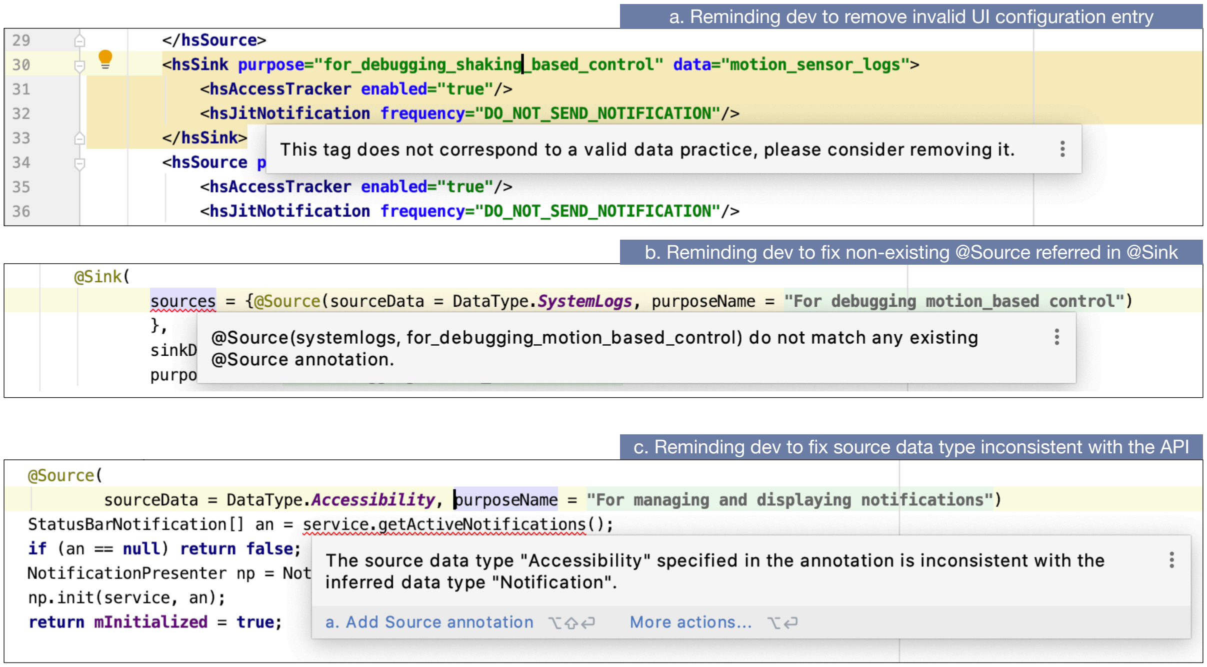
NOTIFICATION_POP_OUT_FIRST_TIME_ONLY
NOTIFICATION_ALWAYS_POP_OUT
DO_NOT_SEND_NOTIFICATION
SEND_NOTIFICATION_SILENTLY
Press ⏎ to insert, ⇥ to replace

Fig. 4. This figure demonstrates the main features of the Honeysuckle IDE plugin via an example use case. First, the app developer opens the Privacy UI Integration panel (A-1) to see what code accesses sensitive user data (i.e., data sources) as detected by the plugin. Then she can use the quick-fix "Add Source annotation" (A-2) to add an annotation for each of these sources and complete required fields in the annotation (e.g., purposeName) (A-3) to resolve the "(incomplete)" errors. Similarly, she needs to annotate "sinks" that may store or send sensitive user data off device (B-4) with the help of quick-fixes (B-5). To indicate how data flows, she also needs to explicitly specify all sources (represented by the source annotations) that may reach this sink (B-6). Lastly, she can modify an XML configuration file automatically generated by the IDE plugin to fine tune the UI behavior (C-7). For example, she can adjust whether to show JIT notification for each source and sink to avoid overwhelming users with unimportant notifications (C-8).

awareness, annotations that contain these errors will be both marked in the code and indicated in the two panels. To achieve a balance between convenience and control, we only show the errors and provide some quick-fixes to help resolve the errors, thus relying on developers to take the initiative to make the actual changes.

During the development of Honeysuckle, we observed that some developers may create helper functions that are often called under different contexts for different purposes. In this situation, directly annotating the source or sink in the helper function will have to conflate the different purposes and therefore fail to provide contextualized privacy notices. In the following, we use an example of a writeToFile helper function (Figure 6) to demonstrate how our Honeysuckle IDE plugin help handle this task using a new annotation: @NoSpecificUseCase.

Our method is analogous to a Java method throwing an exception and require the caller to handle things. When a developer chooses to use the @NoSpecificUseCase annotation to indicate that the original sensitive sink API can be used for multiple purposes in different situations, the IDE expands the sink API list to also include the helper function itself. The IDE plugin then analyzes which parameter of the helper function can reach the initial sink target variable (text), and identifies the second parameter as the new target variable of the helper function. Then the developer can either use the quick-fixes to navigate to the helper function calls or refresh the "Privacy UI Integration" to reveal additional tasks due to the addition of the helper function as a new sink. This can be a recursive process in which developers can keep adding multiple @NoSpecificUseCase until a specific use case is


a. Reminding dev to remove invalid UI configuration entry

```
</hsSource>
<hsSink purpose="for_debugging_shaking_based_control" data="motion_sensor_logs">
    <hsAccessTracker enabled="true"/>
    <hsJitNotification frequency="DO_NOT_SEND_NOTIFICATION"/>
</hsSink>
<hsSource p
    <hsAccessTracker enabled="true"/>
    <hsJitNotification frequency="DO_NOT_SEND_NOTIFICATION"/>
```

This tag does not correspond to a valid data practice, please consider removing it.

b. Reminding dev to fix non-existing @Source referred in @Sink

```
    @Sink(
            sources = {@Source(sourceData = DataType.SystemLogs, purposeName = "For debugging motion_based control")
            },
            sinkD
            purpo
```

@Source(systemlogs, for_debugging_motion_based_control) do not match any existing @Source annotation.

c. Reminding dev to fix source data type inconsistent with the API

```
@Source(
        sourceData = DataType.Accessibility, purposeName = "For managing and displaying notifications")
StatusBarNotification[] an = service.getActiveNotifications();
if (an == null) return false;
NotificationPresenter np = Not
np.init(service, an);
return mInitialized = true;
```

The source data type "Accessibility" specified in the annotation is inconsistent with the inferred data type "Notification".

Fig. 5. This figure demonstrates three types of auto-checks that can be conducted by the Honeysuckle IDE plugin. The general design principle is that the IDE plugin only aims to provide timely reminders to developers about annotations and UI configuration entries that may be invalid or outdated, and expects the developers to take the initiative to fix the issue.

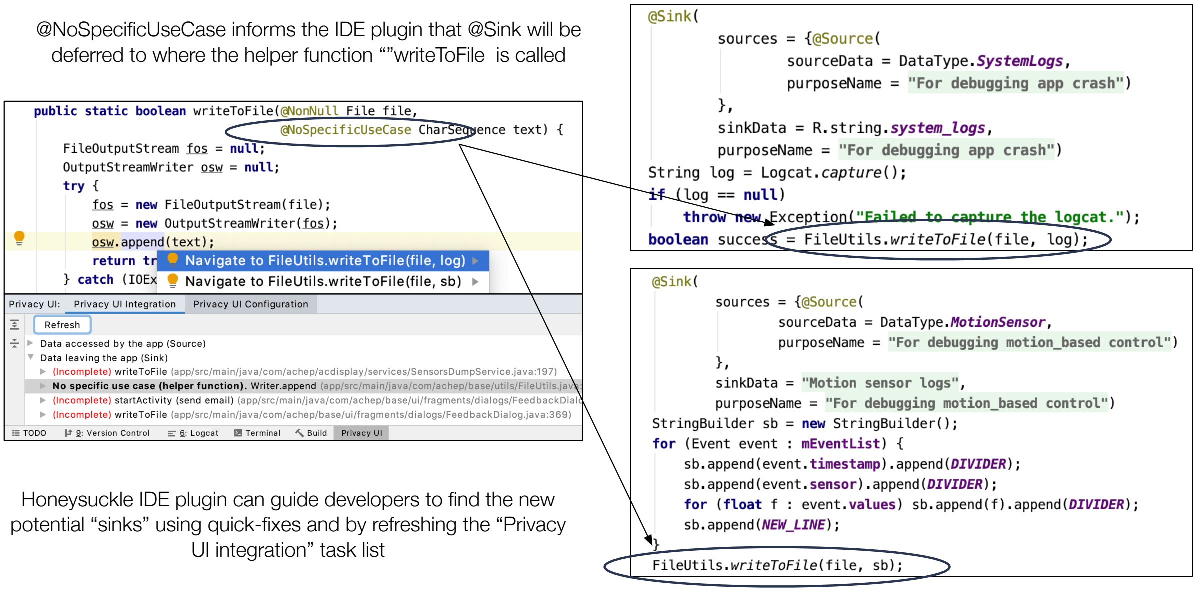
available. For @Source annotations, the process is similar except that the return value of the helper function will be treated as the new target variable.

*3.3.3 Honeysuckle Build System Plugin.* As alluded earlier, our build system plugin can generate code for providing in-app privacy notices using the Honeysuckle library based on the annotations, the UI XML configuration, and the code analysis results. A specific design goal of our build system plugin is that when generating the in-app privacy notice code, there should be as little impact on the programming, debugging, and execution of other code as possible. One related design choice is that the mapping between the line number and code should remain the same after modification, so that developers can debug the app with the source code as usual. Towards this end, Honeysuckle appends all the additional code for each source and sink in the same line.

## 3.4 Honeysuckle Implementation

Our system consists of three main modules: an IDE plugin (implemented for Android Studio), a build system plugin (implemented for Gradle), and an Android library for creating in-app privacy notices. Figure 7 presents four main functionalities of Honeysuckle and we introduce the implementation for each of them below.

*3.4.1 IDE plugin: Code Analysis.* The code analysis subsystem of Honeysuckle was built on top of the analysis engine of Coconut [29], which supports real-time API call detection based on method signature matching. We select 568 source APIs and 76 sink APIs using the same criteria as in Coconut and feed their method signatures into the system. Source APIs are defined as APIs that directly trigger sensitive data collection and sink APIs are APIs that store or send data out of the phone. First-party APIs were selected by reviewing the Android



@NoSpecificUseCase informs the IDE plugin that @Sink will be deferred to where the helper function “”writeToFile is called

```
public static boolean writeToFile(@NonNull File file,
                                  @NoSpecificUseCase CharSequence text) {
    FileOutputStream fos = null;
    OutputStreamWriter osw = null;
    try {
        fos = new FileOutputStream(file);
        osw = new OutputStreamWriter(fos);
        osw.append(text);
        return tr
    } catch (IOEx
```

Navigate to FileUtils.writeToFile(file, log)
Navigate to FileUtils.writeToFile(file, sb)
Privacy UI:
Privacy UI Integration
Privacy UI Configuration
Refresh

Data accessed by the app (Source)
Data leaving the app (Sink)
(Incomplete) writeToFile (app/src/main/java/com/achep/acdisplay/services/SensorsDumpService.java:197)
**No specific use case (helper function).** Writer.append (app/src/main/java/com/achep/base/utils/FileUtils.java
(Incomplete) startActivity (send email) (app/src/main/java/com/achep/base/ui/fragments/dialogs/FeedbackDial
(Incomplete) writeToFile (app/src/main/java/com/achep/base/ui/fragments/dialogs/FeedbackDialog.java:369)

Honeysuckle IDE plugin can guide developers to find the new potential “sinks” using quick-fixes and by refreshing the “Privacy UI integration” task list

```
@Sink(
        sources = {@Source(
                sourceData = DataType.SystemLogs,
                purposeName = "For debugging app crash")
        },
        sinkData = R.string.system_logs,
        purposeName = "For debugging app crash")
String log = Logcat.capture();
if (log == null)
    throw new Exception("Failed to capture the logcat.");
boolean success = FileUtils.writeToFile(file, log);
```

```
@Sink(
        sources = {@Source(
                sourceData = DataType.MotionSensor,
                purposeName = "For debugging motion_based control")
        },
        sinkData = "Motion sensor logs",
        purposeName = "For debugging motion_based control")
StringBuilder sb = new StringBuilder();
for (Event event : mEventList) {
    sb.append(event.timestamp).append(DIVIDER);
    sb.append(event.sensor).append(DIVIDER);
    for (float f : event.values) sb.append(f).append(DIVIDER);
    sb.append(NEW_LINE);
}
FileUtils.writeToFile(file, sb);
```

Fig. 6. Annotation and IDE design example: Sometimes an app might have helper functions to collect, store, or transmit data, which leads to a tricky case that one sensitive API call can be used for multiple purposes. In this example, the app creates a helper function “writeToFile” that calls a system API “osw.append(text)” to write data to files. However, because the helper function is called from two different contexts which store different data for different purposes, directly adding a @Sink annotation to the “text” variable is not sufficient for distinguishing between these two contexts when the app is running. Therefore, we propose a new annotation @NoSpecificUseCase to allow developers to provide a cue to Honeysuckle to skip the @Sink requirement for the current level of the function call stack, so developers can annotate @Sink for helper function calls to distinguish data use in different contexts.

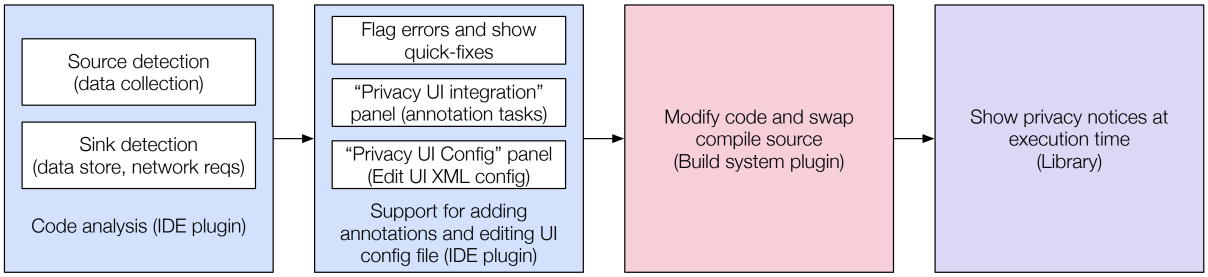

Source detection
(data collection)
Sink detection
(data store, network reqs)
Code analysis (IDE plugin)
Flag errors and show quick-fixes
“Privacy UI integration” panel (annotation tasks)
“Privacy UI Config” panel (Edit UI XML config)
Support for adding annotations and editing UI config file (IDE plugin)
Modify code and swap compile source
(Build system plugin)
Show privacy notices at execution time
(Library)

Fig. 7. Honeysuckle system overview. The three colors correspond to the three component: the IDE plugin, the build system plugin, and the library.

documentation and Third-party APIs were selected using stacktraces of third-party libraries collected by Chitkara et al. [15]. Honeysuckle covers 18 sensitive data types, two of which were adapted from Coconut (location and unique id) and the other 16 types were added by us. The sensitive data types are listed in the appendices (Table 2).

To evaluate the code analysis subsystem, we selected 75 open-sourced Android apps on GitHub by searching for recently updated repos (by July 2020) that contain a Google Play link and declare dangerous permissions in the manifest. We cross-checked the API calls detected by Honeysuckle against the API calls detected by doing a method signature search on the Smali code[1] of these apps and obtained the same results. This shows that Honeysuckle can accurately detect all known source and sink APIs. Furthermore, Honeysuckle detected 23 sensitive API calls per app on average (*median* = 12, *std* = 40) and detected at least one sensitive API call for 70 out of the 75 apps. This suggests that although we did not aim to optimize for coverage, Honeysuckle can already detect a sizable number of sensitive API calls so developers do not need to manually keep track of them. We discuss the limitations of Honeysuckle code analysis and potential enhancement in more depth in Section 5.3.

*3.4.2 IDE plugin: Support for Adding Annotations and Editing UI configuration.* Our IDE plugin runs custom code inspections continuously using the API BaseJavaLocalInspectionTool to maintain a record of all the current data practices, annotations, and UI XML confifuration. With this information, We implemented the "Privacy UI integration" panel and the "Privacy UI Config" panel using the ToolWindowFactory API. The quick-fixes are implemented using the LocalQuickFix API. All the above APIs are provided by the IntelliJ Platform SDK.

Regarding the feature to automatically generate UI XML configuration for new sources and sinks, we chose to initiate the modification of the configuration file only after the user clicks the tab to switch to the "Privacy UI Configuration" panel. This method can help avoid generating configuration for intermediate editing results.

*3.4.3 Build System Plugin.* We chose to develop a plugin for Gradle, which is the most widely used build system for Android development. We implemented custom Gradle tasks and then packaged the tasks into a plugin that can be applied in any Android project by adding one line of code similar to importing a library[2].

As the app starts compiling, our Gradle plugin will send a compile start signal to a local server run in the IDE plugin so IDE plugin can dump the code analysis results. Then the Gradle plugin inserts privacy notice UI code in the project at proper places based on the sensitive API calls detected by the IDE plugin, the annotations provided by developers for these API calls, and the UI configuration file. After finishing compilation, the source code will be reverted back to the original version that only contains the annotations but not the privacy notice UI code.

*3.4.4 Library.* The main functionality of our Honeysuckle library is to provide APIs for different types of in-app privacy notices. The permission explanation dialog is implemented using the AlertDialog system API. The just-in-time notification is implemented as an Android notification using the system API. The privacy center is mainly implemented as a Preference UI, which is a common way for Android apps to implement app settings. We used an open-source library MPAndroidChart[3] to implement the data access frequency diagram in the privacy center. We also defined our custom Java annotations (e.g., Table 1) in the Honeysuckle library.

## 4 HONEYSUCKLE EVALUATION: COMPARING THE ANNOTATION AND LIBRARY APPROACH

To evaluate the effectiveness of the two methods supported by Honeysuckle in helping developers implement fine-grained, in-app privacy notices for real-world projects, we conduct a within-subjects programming study with 12 Android developers to examine the following research questions:

**RQ1** What are their effect on developers' time performance in implementing privacy notices?
**RQ2** What are their effect on developers' cognitive load of implementing privacy notices?
**RQ3** What are their effect on developers' perceived usefulness and usability?
**RQ4** How do developers think about the two methods?

[1] Smali is a human-readable intermediate code format generated by decompiling an app apk which expands every API call to its complete method signature.
[2] Developing Custom Gradle Plugins. https://docs.gradle.org/current/userguide/custom_plugins.html
[3] https://github.com/PhilJay/MPAndroidChart

## 4.1 Participants

Out of the 12 participants, 11 were freelance Android developers recruited from a freelancing platform (Upwork). The other participant signed up for the study after seeing our recruiting ads posted on Twitter. In a pre-study survey, we asked participants to provide demographic information. 3 participants self-identified as female, and 9 as male. The average age is 31.3 ($std = 9.49, min = 22, max = 56$) years old.

We also requested our participants to provide their background in Android development. Our participants reported 4.3 years of Android development experience on average ($std = 1.5, min = 2, max = 7$). 8 of 12 participants self-reported that they have released Android apps on an app store (e.g., Google Play, F-Droid). One participant answered that his most popular app achieved 10M+ installs, another participant 1M+ installs, and a third participant 10K+ installs. All participants considered themselves familiar with Java and Android Studio.

Lastly, we asked them about their prior experience in handling private data and using Java annotations in general. 11 out of 12 self-reported that they have created Android apps that access users' personal data (e.g., geolocation, device ID), with the other participant answering "I'm not sure". However, when choosing from a list of different types of privacy notices, most participants reported only creating privacy policies and requesting system permissions to provide privacy notices. This finding supports our working assumption that developers rarely adopt other types of privacy notices for their apps. Regarding the use of Java annotations, all participants have at least used some simple features such as adding `@SuppressLint` to clear IDE warnings/errors. 3 out of 12 participants reported that they have previously used annotations for code generation.

## 4.2 Control and test conditions

The test condition of our study is to use `Honeysuckle` to modify a real-world app to implement the five types of example privacy notices. The control condition is to use the library we developed for `Honeysuckle` to generate the example privacy notices introduced in Section 3.2. To minimize the carryover effect in within-subjects study design, we chose two popular open-source apps, AcDisplay[4] (a notification manager app) and OmniNotes[5] (a note-taking app) and ask each participant to work on different projects using the two methods. Both projects use Java as the main programming language and are of similar complexity. OmniNotes contains around 231 Java files and 21714 lines of Java code; AcDisplay contains around 331 Java files and 59489 lines of Java code. Regarding sensitive data use, OmniNotes contains 10 sources and 2 sinks, and AcDisplay contains 13 sources and 4 sinks.

Note that we chose to use the library as the baseline condition instead of asking developers to achieve the task without any support, because otherwise UI design and implementation issues would clearly dominate the amount of the time on task and it would be unequal comparison with annotations and do not generate insightful results. We searched for Android libraries that support developers to implement these three in-app privacy notices but to no avail, likely because there has yet to be sufficient need to drive the development of such libraries. Hence we asked participants to use the `Honeysuckle` library in the library condition, which is also what our generated code calls to show the privacy notices, to provide a relatively equal comparison between the two approaches.

We sequentially assigned participants to the following groups based on their signup order:

**1** modify AcDisplay using annotations + modify OmniNotes using library
**2** modify OmniNotes using library + modify AcDisplay using annotations
**3** modify OmniNotes using annotations + modify AcDisplay using library
**4** modify AcDisplay using library + modify OmniNotes using annotations

[4] https://github.com/AChep/AcDisplay
[5] https://github.com/federicoiosue/Omni-Notes

## 4.3 Procedure

We conducted our study remotely via Zoom because our participants resided in multiple locations. After the participant signed the consent form, the experimenter turned on screen recording to record the entire session for further analysis. The entire study took around 2.5 hours and mainly consisted of two programming tasks (corresponding to the two conditions). Developers had at most 40 minutes for each programming task, and then spent 5 minutes to fill out a NASA-TLX questionnaire [25] to measure the cognitive load and a Technology Acceptance Model Scale (TAMS) questionnaire [17] to measure the perceived usefulness and usability of our tool. For the two programming tasks, developers modified different apps to implement privacy notices using different methods. Since we wanted to evaluate the programming methods rather than examine developers code comprehension skills, we provided documentation of the apps' data practices containing all the information needed to make the privacy notices so the developer could focus on using the provided tools to achieve the goal.

Before each programming task, the experimenter used a toy app as an example to teach the participant how to use annotations and the library to achieve the task. Specifically, the experimenter first implemented privacy notices for half of the sources and sinks in the toy app, and then asked the participant to complete the rest of privacy notices. During this process, the experimenter answered all the questions the participant had regarding the method of the current condition. For the library condition, we ensured that developers had access to the example app code during the main programming task so they could refer to them when needed.

After completing the two programming tasks and filling out the surveys, the experimenter asked several follow-up questions regarding their thoughts about the two methods, such as which part of the tool was hard to use, what could be improved, and what they thought about the generated privacy notice UIs. Upon completion, each participant was compensated with 75 USD. This study was approved by our university's IRB.

## 4.4 Study environment

We set up a virtual workstation using a VNC server on a Google Cloud virtual machine, so all participants can work on the programming tasks in this virtual workstation. The experimenter can view and interact with the same interfaces during the study to do live coding during the warm up tasks and to provide guidance to help developers quickly get used to this new environment.

## 4.5 Quantitative results (RQ1-3)

We present the results to the first three research questions below.

*4.5.1 Time on Task (RQ1).* All developers completed the task in the `Honeysuckle` condition in the allotted time, while only 2 developers (P7 and P9) completed the task when using the library. Furthermore, for these 2 developers, it took them longer time to finish when using the library than using the annotation. P7 used 22 minutes to finished the task using the annotation and 37 minutes using the library. P9 used 14 minutes to finished the task using the annotation and 27 minutes using the library.

*4.5.2 Cognitive load (RQ2).* We analyzed the six dimensions of cognitive load measured by NASA-TLX on a scale of 1 to 100 (lower is better) [25]. `Honeysuckle` achieved a statistically significant reduction in cognitive load using a paired t-test ($p < .001$, $t=5.09$). Figure 8 shows that `Honeysuckle` helped reduce the cognitive load across all six dimensions, especially for mental load and perceived success.

*4.5.3 Usefulness and usability (RQ3).* We analyzed the perceived usability and usefulness of `Honeysuckle` and the privacy notice library measured using TAMS on a scale of 1 to 7 (lower is better, 1=extremely likely, 7=extremely unlikely) [17]. We then calculated the sum of all ratings in the usability subscale and the usefulness subscale and compare them between the two conditions. For both usability and usefulness, `Honeysuckle` received significantly better (lower) ratings under paired t-tests (for usability, $p = .02$, $t=2.76$; for usefulness, $p = .02$, $t=2.67$). The mean

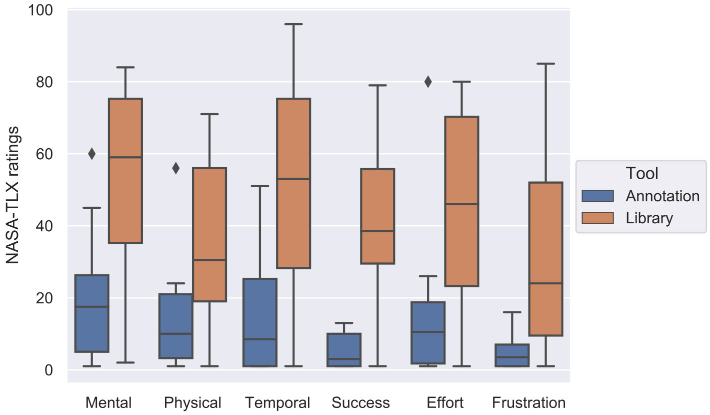


Fig. 8. NASA-TLX ratings of the Annotation and Library conditions (lower is better). Using the annotation-based approach of Honeysuckle significantly reduces the overall cognitive load as compared to using the library to implement the same runtime privacy notices. The error bars represent the 95% confidence intervals.

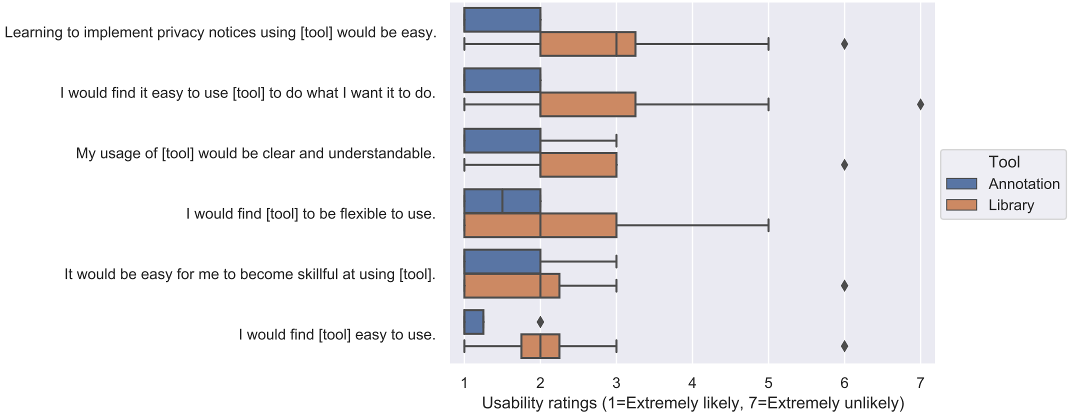


Fig. 9. Perceived usability ratings of the annotation-based approach and the library (lower is better, 1=extremely likely, 7=extremely unlikely). Overall, the annotation-based approach was perceived as more usable than the library. The error bars represent the 95% confidence intervals.

usability rating of Honeysuckle is 1.45 and the mean usability rating of the library is 2.56. The mean usefulness rating of Honeysuckle is 1.27 and the mean usefulness rating of the library is 2.30. Overall, developers perceived both tools to be useful and usable, and perceived Honeysuckle to be more useful and usable than the library.

## 4.6 Qualitative feedback (RQ4)

One researcher transcribed the interviews, conducted thematic analysis following previous guidelines [14], and held regular meetings with other authors to review the analysis process and discuss the findings. The identified themes are categorized as *feedback on programming methods* or *feedback on the generated privacy notices*.

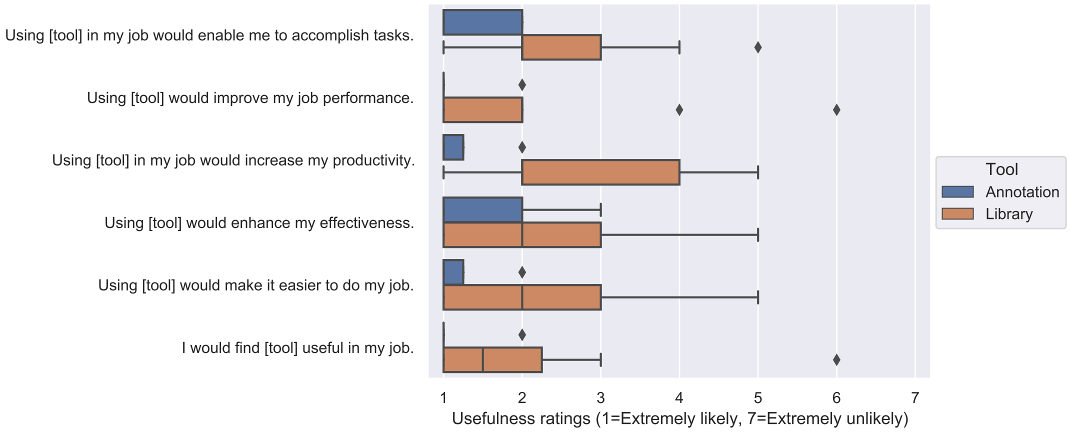


Fig. 10. Perceived usefulness ratings of the annotation-based approach and the library (lower is better, 1=extremely likely, 7=extremely unlikely). Overall, the annotation-based approach was perceived as more useful than the library. The error bars represent the 95% confidence intervals.

*4.6.1 Feedback on programming methods.* When comparing the two methods, all participants preferred the annotation method a lot than the library method. Without being asked, P2, P7, P10, P11 said they hoped to see a public release of `Honeysuckle` so they can use it for their own projects. For example, P7 said that *"I hope any of these can be launched. Anyone of them would be cool, but my preference would be the plugin."* This echos the fact that there is very limited developer support for building in-app privacy notices and suggests both the `Honeysuckle` library and annotation can improve app's transparency when also making developer's life a lot easier. Below, we present themes emerged from participants' explanations of their preferences.

**Productivity.** Six participants (P1, P3, P4, P7, P11, P12) mentioned that they felt a lot more productive when using the annotation method than the library. Specifically, they felt using annotations allowed them to write less code and achieve the task faster. For example, P4 mentioned using annotations could *"avoid a lot of parameters for calling the APIs"*, which is because our annotation design reduces a lot of redundancies by automatically generating multiple types of notices using one annotation. Developers also praised the automated source and sink detection and quick-fixes offered by `Honeysuckle` IDE plugin. For example, P3 mentioned *"Just being able to go through some automated steps makes it more efficient to add annotations"*.

**Mental Load.** Nine participants (P1, P2, P4, P7-P12) said that using annotations helped reduce the mental load in many ways. For example, P8 loved the automated suggestions offered by the IDE plugin because *"It simply tells me where I need to add annotations and I don't need to consciously take effort to do that"*. On the contrary, using the library *"felt very tedious"* (P2) and they found it challenging to *"remember what API to use"* (P7). Moreover, using the library requires the developer to *"manually track many things at the same time"*, including naming the sources and remembering the names when defining connections between sources and sinks in the privacy info map (P9). When using annotations, this task is made much easier by the quick-fixes of the IDE plugin.

**Learnability and Usability.** Nine participants (P2, P4-P7, P9-P12) discussed the learnability and usability of the two methods. Some people found the annotation tool easy to learn. For example, P11 said *"The tool is new to me, but I didn't need to think much to figure out what to do. The plugin has made it very simple."* Conversely, some participants shared a slightly different feeling that *"the annotations may have slightly steeper learning curve, but after getting the hang of it, using it is way easier"* (P7). The difficulty was mainly in understanding how to specify the source in a sink annotation (P2, P10) and handle helper functions (P2, P5, P10). This result echos the fact that only three out of the twelve participants had used annotations for code generation before the study. However, the unfamiliarity with annotations did not seem to hinder the usage of annotations for privacy notice generation. P4 even said *"Normally I don't like annotations, but for this task I like it. It's very easy to use."*

**Code Comprehension.** Four participants (P1, P7, P8, P11) mentioned that using annotations made the code *"organized"* (P1) and *"very easy to read"* (P8). P8 further explained that *"all the configuration goes to the XML, so the annotation only keeps simple information"*.

**Teamwork.** P4 and P9 explicitly discussed different benefits of using annotations and the IDE plugin for teamwork. P4 liked the quick-fixes for generating annotation templates because it helps everyone in a team code in the same format. P9 was thinking about the situation when developers working in a larger group that have a separate privacy team. He explained that *"The IDE automatically analyzes all the sources and sinks, so the privacy team can focus on the privacy work such as checking if privacy notices have been implemented for all of them."*

*4.6.2 Feedback on the generated privacy notices.* When asked about their favorite design among the three privacy notices as a user, eight participants picked the privacy center (P1, P3, P5-P9, P12), three picked the permission notice (P4, P10, P11), and only one picked the just-in-time notification (P2). Participants preferred the privacy center the most because it contained more information about the app's data practices and it was available to the user at any time. They also generally found the permission notice and the just-in-time notification useful, for they use a *"streamlined format"* (P2) and provide *"finer-grained detail than what you get with regular permissions"* (P3).

Some people also touched upon potential issues about these privacy notices, categorized into two themes:

**Notification overload problem.** P3 and P5 expressed their concerns about the notification overload problem specifically for the just-in-time notice design. P3 elaborated her concerns using the Starbucks app as an example,

> *"I use the Starbucks app, and I guarantee you it'll be blurting at you all the time some privacy notifications, but would that improve my experience? I'm not sure because I kind of know at the back of my head that they're probably doing that and I guess I'm okay with they know where I am".*

Similarly, P5 said *"receiving many notifications from an app can be annoying from a user perspective"*. We have considered this issue when designing `Honeysuckle` and allow developers to control default visibility for each just-in-time notification to balance user experience and privacy. However, there are still issues such as developers can only make static decisions so the generated UI can not accommodate to different privacy concerns of different users. In Section 5.4.2, we discuss potential ways to further improve privacy using the annotations.

**Factors that impede adoption.** Some participants also explicitly discussed or implied factors that may prevent them from using `Honeysuckle` in real life. First, although many participants have been familiar with system permissions, in-app privacy notice was quite a foreign concept to them. For example, P4 was confused when first seeing the permission notice and asked *"Is this extra dialog useful?"* P3 mentioned that she had worked with many clients but had never encountered a requirement of implementing in-app privacy notices. Second, currently `Honeysuckle` has to rely on developer's input for some fields in the annotation such as purposes and can not automatically verify them, causing the risk of generating misleading privacy notices if the developer does not provide accurate information. P1 said *"You can not ensure that the developer will be an honest developer and he will tell you his true intentions of the data"*. However, we want to note that this is a general challenge facing any kind of privacy notices. We discuss potential mitigation methods in Section 5.4.1.

## 5 DISCUSSION

We now discuss the benefits of using the `Honeysuckle` system for annotation-based privacy notice generation, the limitations of `Honeysuckle`, and other opportunities to improve data transparency and control using annotations.

### 5.1 Annotations as a Bridge to Connect Users and Developers

Previously, Coconut [29] demonstrated that annotations (and other metadata with similar attributes) can help educate developers and nudge them towards adopting best practices for privacy in their apps. In this work, we demonstrate that annotations are also effective in helping developers generate useful and usable privacy notices.

There are many reasons why annotation-based privacy notice generation can address common challenges regarding the implementation of in-app privacy notices. Since annotations are added next to sensitive API calls, developers do not need to explicitly specify any context, which saves on a lot of effort. There is a clear relationship between the annotation and the generated interfaces, which helps with code maintenance and collaboration for teamwork. Our IDE plugin can automatically detect data sources and sinks and remind developers to add annotations, which reduces developers' cognitive load and increases the coverage and accuracy of the information declared in annotations. We carefully designed `Honeysuckle` to handle complicated scenarios in real-world apps, such as having multiple API calls serving the same purpose and having the same API call serving different purposes. More importantly, our approach allows developers to build effective privacy notices without having to be a privacy expert, since the generated interfaces automatically follow the best practices.

Our quantitative lab study results demonstrated that the annotation-based approach consistently outperformed the library-based approach in terms of time on task, cognitive load and perceived usefulness and usability to developers. We also showed in our qualitative analysis that participants felt adding annotations was a better approach than using the library. They preferred using annotations because it improved productivity, reduced mental load, was easier to learn and use, made code easier to comprehend, and could provide benefits for teamwork. Overall, adding annotations was perceived as a simple task and only led to a small extra effort, because the annotation and UI configuration XML design allowed them to write much less code and the IDE plugin offered sufficient suggestions and quick-fixes to streamline the task. They felt the `Honeysuckle` library more complicated to use and required more work than annotations, but still considered it useful since no such tool exists either. Note that both approaches were new to developers who participated in our study. Only three out of the twelve participants had previously used annotations for any kind of code generation, though all participants picked up the annotation-based approach and finished the task much faster than the library-based approach.

In a broader sense, some of our design considerations may increase the likelihood for `Honeysuckle` to be adopted, based on lessons learned about why model-based user interface design (MBUID) [39, 52] did not achieve widespread adoption. Instead of solving the generic UI implementation issue, we focus on privacy notice generation specifically and try to minimize the implications on regular app development tasks. Furthermore, given the low adoption of in-app privacy notices, developers likely have not established preferences for building their own interfaces, one of the barriers that prevented the adoption of MBUID [52].

### 5.2 Increasing Developers' Incentives to Improve Privacy

Many privacy-related requirements currently in place for Android and Google Play impose many restrictions on what data developers can use [30]. Conversely, we argue that requiring developers to create more privacy notices to improve transparency will cause little to no impact on the apps' functionality, which makes developers potentially more amenable to them. Furthermore, by making it easier to create privacy notices, we may be able to nudge developers who care about users' privacy to adopt more effective designs.

As suggested in prior research [13, 33], providing more detailed privacy notices for legitimate data practices is likely to not only improve users' perceived comprehension and control of their privacy, but also make users more comfortable with allowing the app to access their data and trust the app more. Nowadays, there seems to be a growing interest in both academia and industry to build applications that can provide benefits to users and preserve privacy at the same time, but the common technical approach is still to minimize data practices, such as using differential privacy, federated learning, and on-device training and inference. With the current permission system, there is still a gap between users' perceptions and the actual data practices [30], and our tool can potentially address this problem and make developers' effort in protecting user privacy more visible to users.

## 5.3 Limitations of Honeysuckle

We chose a relatively simple code analysis method for Honeysuckle implementation, which is to search for all API calls that match a predefined source and sink API list. Although the code analysis subsystem is largely adapted from Coconut [29] and is not our main contribution, we want to discuss limitations in the current implementation and point out potential mitigation methods. First, the detected sinks may not actually store or send *sensitive data* off the device. Although we allow developers to manually add annotations to mark non-sensitive sinks, this process can be tedious and error-prone, especially for teamwork. Given that researchers have built tools to statically analyze data flows of compiled Android apps [8], a potential solution is to investigate how to conduct in-IDE data flow analysis over the source code and give developers hints and warnings for more efficient and accurate annotation filling. Second, our sensitive API list does not guarantee a full coverage. To improve the coverage, we can integrate existing methods [9, 44] to automatically identify source and sink APIs for different versions of Android. For third-party libraries, Honeysuckle solely relies on a curated list of popular third-party libraries. Since prior work suggests the usage of third-party libraries in Android apps follows a long-tail distribution [15], a potential solution is to combine a predefined third-party API list to cover the majority of cases with on-demand analysis to handle edge cases. Furthermore, Honeysuckle does not support customizing the styles and themes of the generated UI. This feature is straightforward to implement by allowing developers to specify their own style and theme XMLs [6] to Honeysuckle.

## 5.4 Future work: Going further from Honeysuckle and privacy notice generation

Despite the promising benefits, a developer tool for generating privacy notices alone can not solve all challenges facing the adoption of privacy notices and privacy by design in general. Therefore, we propose some follow-up work ideas to further improve privacy based on Honeysuckle and annotation-based privacy notice generation.

*5.4.1 Engaging other stakeholders to enforce and stimulate accurate privacy notice.* Honeysuckle relies on developers to provide accurate input to generate useful privacy notices. This could lead to the issue of "dishonest developers", which was also brought up by our participants (Section 4.6.2). Hence, we discuss potential methods to enforce or stimulate accurate privacy notices beyond building developer tools. First, the internal privacy auditing teams or the app store need to check if developers have provided correct information in their annotations. As we have alluded to earlier, to supplement the free-form purpose string, we can add another annotation field that requires developers to declare a purpose category, using a set of predefined options. Then app reviewers may be able to infer the purpose category to cross check the annotations. An alternative approach is to generating code to log, synthesize, and help users visualize what data is collected, retained, or sent off the device. The rationale here is to give users, privacy researchers, and consumer advocates the ability to cross check what data practices are declared by developers against actual data practices. Similar ideas can be found in both research and commercial systems. For example, the iOS background location history dialog renders the location traces collected by the app on a map along with a rationale string provided by the developer. Harbach et al. [24] demonstrated visualization design using personal examples to improve the communication of privacy risks.

*5.4.2 Other opportunities of improving transparency and control using annotations.* Currently, Honeysuckle works as a standalone tool, but we envision that privacy annotations, the Honeysuckle IDE plugin, and build system plugin could be more tightly integrated into the Android ecosystem. For example, annotations can be embedded into compiled apps, which would let the operating system generate unified interfaces for all apps and

[6] https://developer.android.com/guide/topics/ui/look-and-feel/themes

allow users to manage global settings about the appearances and frequency of notices, as well as when and where to see notices to avoid notification overload. As another example, having annotations embedded into compiled apps could make it easier to check the behavior of apps by app stores, researchers, and consumer advocates. By expanding the kinds of annotations developers have to add, it would be possible to generate a summary of the app's behaviors, such as what websites or cloud services it accesses and what data it sends. This kind of summary could augment the existing privacy interfaces, help users make decisions about whether to install an app (e.g. see PrivacyGrade.org), and help developers fulfill new requirements such as providing privacy labels for iOS apps.

Stepping back, we believe that a compelling vision for privacy is to have developers annotate their code just once, which can then help those developers with a number of privacy-related tasks (so that they will want to include accurate annotations), as well as make apps easier to audit by users and third parties. A challenge here is defining a unified and complete enough set of annotations that can cover all of these use cases.

## 6 CONCLUSION

In this paper, we presented `Honeysuckle`, a programming tool that helps Android developers build in-app privacy notices using a combination of privacy annotations, an IDE plug-in, and a build system plug-in. We conducted a within-subjects study with 12 Android developers to evaluate `Honeysuckle`, and found that developers were able to easily use our annotation approach over a conventional library-based approach to generate in-app privacy notices. Our annotation approach also had much lower cognitive load and was generally preferred by developers.

More broadly, our work suggests two compelling paths forward for future research. The first is additional tool-based support for developers with respect to privacy, for example to alleviate the burden of manually tracking data uses in an app, adding additional features, and supporting other privacy needs not examined in this paper (e.g. internal auditing of apps). The second is other uses of annotations to support developers. Annotations can act as hints about intended behavior of apps, and we speculate that they may also be useful for other kinds of complex development tasks of cross-cutting features distributed across many points in the source code, such as user interface generation, security, energy efficiency, and more.

## ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1801472 This material is also based on research sponsored by Air Force Research Laboratory under agreement number FA8750-15-2-0281. 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 Air Force Research Laboratory or the U.S. Government.

The authors would like to thank Tiffany Yu, Rituparna Roychoudhury, Sijing Sun, Shan Wang for their help with the privacy notice design at the early stage. The authors would like to thank Haoyu Wang, Brad A. Myers, Lorrie Faith Cranor, and Jackie Junrui Yang for providing feedback on this work. The authors would also like to thank the anonymous reviewers for their constructive feedback.

## REFERENCES

[1] 2020. Chapter 20. California Consumer Privacy Act Regulations (Feb 2020 revision). https://web.archive.org/web/20200619170238/https://www.oag.ca.gov/sites/all/files/agweb/pdfs/privacy/ccpa-text-of-mod-redline-020720.pdf. (Accessed on 09/13/2020).

[2] 2021. GitHub - permissions-dispatcher/PermissionsDispatcher: A declarative and comprehensive API to handle Android runtime permissions. https://web.archive.org/web/20201205195815/https://github.com/permissions-dispatcher/PermissionsDispatcher. (Accessed on 02/13/2021).

[3] 2021. Privacy - Features - Apple. https://web.archive.org/web/20210126192113/https://www.apple.com/privacy/features/. (Accessed on 01/29/2021).

[4] 2021. Privacy in Android 11 | Android Developers. https://web.archive.org/web/20210129195758/https://developer.android.com/about/versions/11/privacy. (Accessed on 01/29/2021).

[5] 2021. Requesting Consent from European Users | Android | Google Developers. https://web.archive.org/web/20201109005634/https://developers.google.com/admob/android/eu-consent. (Accessed on 02/13/2021).

[6] Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services - MobiSys '13*. ACM Press. https://doi.org/10.1145/2462456.2464460

[7] Hazim Almuhimedi, Florian Schaub, Norman Sadeh, Idris Adjerid, Alessandro Acquisti, Joshua Gluck, Lorrie Faith Cranor, and Yuvraj Agarwal. 2015. Your Location has been Shared 5,398 Times!: A Field Study on Mobile App Privacy Nudging. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/2702123.2702210

[8] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. *ACM SIGPLAN Notices* 49, 6 (jun 2014), 259–269. https://doi.org/10.1145/2666356.2594299

[9] Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. PScout: analyzing the Android permission specification. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press. https://doi.org/10.1145/2382196.2382222

[10] Rebecca Balebako and Lorrie Cranor. 2014. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security & Privacy* 12, 4 (jul 2014), 55–58. https://doi.org/10.1109/msp.2014.70

[11] Rebecca Balebako, Jaeyeon Jung, Wei Lu, Lorrie Faith Cranor, and Carolyn Nguyen. 2013. "Little brothers watching you": raising awareness of data leaks on smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security - SOUPS '13*. ACM Press. https://doi.org/10.1145/2501604.2501616

[12] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason Hong, and Lorrie Faith Cranor. 2014. The Privacy and Security Behaviors of Smartphone App Developers. In *Proceedings 2014 Workshop on Usable Security*. Internet Society. https://doi.org/10.14722/usec.2014.23006

[13] Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. 2017. Exploring decision making with Android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. 195–210.

[14] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (jan 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[15] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. 2017. Does this App Really Need My Location?: Context-Aware Privacy Management for Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (sep 2017), 1–22. https://doi.org/10.1145/3132029

[16] Federal Trade Commission. 2013. Mobile privacy disclosures: Building trust through transparency: a federal trade commission staff report. https://www.ftc.gov/sites/default/files/documents/reports/mobile-privacy-disclosures-building-trust-through-transparency-federal-trade-commission-staff-report/130201mobileprivacyreport.pdf. (Accessed on 08/02/2021).

[17] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* 13, 3 (sep 1989), 319. https://doi.org/10.2307/249008

[18] Android Developer Document. 2020. App permissions best practices | Android Developers. https://developer.android.com/training/permissions/usage-notes. (Accessed on 09/17/2020).

[19] Michael D. Ernst, René Just, Suzanne Millstein, Werner Dietl, Stuart Pernsteiner, Franziska Roesner, Karl Koscher, Paulo Barros Barros, Ravi Bhoraskar, Seungyeop Han, Paul Vines, and Edward X. Wu. 2014. Collaborative Verification of Information Flow for a High-Assurance App Store. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. https://doi.org/10.1145/2660267.2660343

[20] Benjamin Fabian, Tatiana Ermakova, and Tino Lentz. 2017. Large-scale readability analysis of privacy policies. In *Proceedings of the International Conference on Web Intelligence*. ACM. https://doi.org/10.1145/3106426.3106427

[21] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. 2012. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Trust and Trustworthy Computing*. Springer Berlin Heidelberg, 291–307. https://doi.org/10.1007/978-3-642-30921-2_17

[22] Joshua Gluck, Florian Schaub, Amy Friedman, Hana Habib, Norman Sadeh, Lorrie Faith Cranor, and Yuvraj Agarwal. 2016. How short is too short? implications of length and framing on the effectiveness of privacy notices. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. 321–340.

[23] Irit Hadar, Tomer Hasson, Oshrat Ayalon, Eran Toch, Michael Birnhack, Sofia Sherman, and Arod Balissa. 2017. Privacy by designers: software developers' privacy mindset. *Empirical Software Engineering* 23, 1 (apr 2017), 259–289. https://doi.org/10.1007/s10664-017-9517-1

[24] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. 2014. Using personal examples to improve risk communication for security & privacy decisions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/2556288.2556978

[25] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Advances in Psychology*. Elsevier, 139–183. https://doi.org/10.1016/s0166-4115(08)62386-9

[26] Jianjun Huang, Zhichun Li, Xusheng Xiao, Zhenyu Wu, Kangjie Lu, Xiangyu Zhang, and Guofei Jiang. 2015. SUPOR: Precise and Scalable Sensitive User Input Detection for Android Apps. In *24th USENIX Security Symposium (USENIX Security 15)*. 977–992.

[27] Farzaneh Karegar, John Sören Pettersson, and Simone Fischer-Hübner. 2020. The Dilemma of User Engagement in Privacy Notices: Effects of Interaction Modes and Habituation on User Attention. *ACM Transactions on Privacy and Security* 23, 1 (feb 2020), 1–38. https://doi.org/10.1145/3372296

[28] Abhishek Kumar, Tristan Braud, Young D. Kwon, and Pan Hui. 2020. Aquilis: Using Contextual Integrity for Privacy Protection on Mobile Devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (dec 2020), 1–28. https://doi.org/10.1145/3432205

[29] Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. 2018. Coconut: An IDE Plugin for Developing Privacy-Friendly Apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (dec 2018), 1–35. https://doi.org/10.1145/3287056

[30] Tianshi Li, Elizabeth Louie, Laura Dabbish, and Jason I. Hong. 2021. How Developers Talk About Personal Data and What It Means for User Privacy: A Case Study of a Developer Forum on Reddit. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW3 (jan 2021), 1–28. https://doi.org/10.1145/3432919

[31] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2017. PrivacyStreams: Enabling Transparency in Personal Data Processing for Mobile Apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (sep 2017), 1–26. https://doi.org/10.1145/3130941

[32] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I Hong. 2014. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*. 199–212.

[33] Jialiu Lin, Norman Sadeh, Shahriyar Amini, Janne Lindqvist, Jason I. Hong, and Joy Zhang. 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*. ACM Press. https://doi.org/10.1145/2370216.2370290

[34] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. 2016. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. 27–41.

[35] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A. Myers. 2019. Unakite: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. ACM. https://doi.org/10.1145/3332165.3347908

[36] Xueqing Liu, Yue Leng, Wei Yang, Wenyu Wang, Chengxiang Zhai, and Tao Xie. 2018. A Large-Scale Empirical Study on Android Runtime-Permission Rationale Messages. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. https://doi.org/10.1109/vlhcc.2018.8506574

[37] Kangjie Lu, Zhichun Li, Vasileios P. Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. 2015. Checking More and Alerting Less: Detecting Privacy Leakages via Enhanced Data-flow Analysis and Peer Voting. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2015.23287

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

[39] Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. 2011. Past, Present, and Future of Model-Based User Interface Development. *icom* 10, 3 (nov 2011), 2–11. https://doi.org/10.1524/icom.2011.0026

[40] Tamir Mendel and Eran Toch. 2019. My Mom was Getting this Popup: Understanding Motivations and Processes in Helping Older Relatives with Mobile Security and Privacy. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 4 (dec 2019), 1–20. https://doi.org/10.1145/3369821

[41] Andreas Mueller, Peter Forbrig, and Clemens Cap. 2001. Model-Based User Interface Design Using Markup Concepts. In *Interactive Systems: Design, Specification, and Verification*. Springer Berlin Heidelberg, 16–27. https://doi.org/10.1007/3-540-45522-1_2

[42] Midas Nouwens, Ilaria Liccardi, Michael Veale, David Karger, and Lalana Kagal. 2020. Dark Patterns after the GDPR: Scraping Consent Pop-ups and Demonstrating their Influence. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3313831.3376321

[43] Ashwini Rao, Florian Schaub, Norman Sadeh, Alessandro Acquisti, and Ruogu Kang. 2016. Expecting the unexpected: Understanding mismatched privacy expectations online. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. 77–96.

[44] Siegfried Rasthofer, Steven Arzt, and Eric Bodden. 2014. A Machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. In *Proceedings 2014 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2014.23039

[45] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. 2016. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. ACM. https://doi.org/10.1145/2906388.2906392

[46] Florian Schaub, Rebecca Balebako, and Lorrie Faith Cranor. 2017. Designing Effective Privacy Notices and Controls. *IEEE Internet Computing* (2017), 1–1. https://doi.org/10.1109/mic.2017.265102930

[47] Florian Schaub, Rebecca Balebako, Adam L. Durity, and Lorrie Faith Cranor. [n.d.]. A Design Space for Effective Privacy Notices. In *The Cambridge Handbook of Consumer Privacy*. Cambridge University Press, 365–393. https://doi.org/10.1017/9781316831960.021

[48] Swapneel Sheth, Gail Kaiser, and Walid Maalej. 2014. Us and them: a study of privacy requirements across north america, asia, and europe. In *Proceedings of the 36th International Conference on Software Engineering*. ACM. https://doi.org/10.1145/2568225.2568244

[49] Yihang Song and Urs Hengartner. 2015. PrivacyGuard: A VPN-based Platform to Detect Information Leakage on Android Devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. https://doi.org/10.1145/2808117.2808120

[50] Gaurav Srivastava, Kunal Bhuwalka, Swarup Kumar Sahoo, Saksham Chitkara, Kevin Ku, Matt Fredrikson, Jason Hong, and Yuvraj Agarwal. 2017. PrivacyProxy: Leveraging Crowdsourcing and In Situ Traffic Analysis to Detect and Mitigate Information Leakage. *arXiv preprint arXiv:1708.06384* (2017).

[51] Ryan Stevens, Clint Gibler, Jon Crussell, Jeremy Erickson, and Hao Chen. 2012. Investigating user privacy in android ad libraries. In *Workshop on Mobile Security Technologies (MoST)*, Vol. 10. Citeseer.

[52] Noi Sukaviriya, Srdjan Kovacevic, James D. Foley, Brad A. Myers, Dan R. Olsen, and Matthias Schneider-Hufschmidt. 1994. Model-based user interfaces: what are they and why should we care?. In *Proceedings of the 7th annual ACM symposium on User interface software and technology - UIST '94*. ACM Press. https://doi.org/10.1145/192426.192479

[53] Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negrón-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. 2014. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/2556288.2557400

[54] Zhiyuan Wan, Lingfeng Bao, Debin Gao, Eran Toch, Xin Xia, Tamir Mendel, and David Lo. 2019. AppMoD: Helping Older Adults Manage Mobile Security with Online Social Help. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 4 (dec 2019), 1–22. https://doi.org/10.1145/3369819

[55] Xiaolei Wang, Andrea Continella, Yuexiang Yang, Yongzhong He, and Sencun Zhu. 2019. LeakDoctor: Toward Automatically Diagnosing Privacy Leaks in Mobile Applications. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 3, 1 (mar 2019), 1–25. https://doi.org/10.1145/3314415

[56] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. 2018. Contextualizing Privacy Decisions for Better Prediction (and Protection). In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM. https://doi.org/10.1145/3173574.3173842

[57] Zhongxing Yu, Chenggang Bai, Lionel Seinturier, and Martin Monperrus. 2021. Characterizing the Usage, Evolution and Impact of Java Annotations in Practice. *IEEE Transactions on Software Engineering* 47, 5 (may 2021), 969–986. https://doi.org/10.1109/tse.2019.2910516

## A HONEYSUCKLE SENSITIVE DATA TYPE COVERAGE

Table 2. Honeysuckle sensitive data type coverage

| Permission Requirement | Data Type |
|---|---|
| Protected by dangerous permissions | Calendar<br>Call Logs<br>Camera<br>Contacts<br>Location<br>Microphone<br>Body Sensor<br>Sms<br>User Account |
| Partially protected by dangerous permissions | Unique Identifier<br>User File |
| Protected by other permissions | Notification<br>Accessibility (UI events)<br>Motion Sensor |
| Not protected by permissions | User Input<br>Clipboard<br>Running Apps<br>System Logs |