Coconut: An IDE Plugin for Developing Privacy-Friendly Apps

TIANSHI LI, Carnegie Mellon University, USA
YUVRAJ AGARWAL, Carnegie Mellon University, USA
JASON I. HONG, Carnegie Mellon University, USA

Although app developers are responsible for protecting users' privacy, this task can be very challenging. In this paper, we present Coconut, an Android Studio plugin that helps developers handle privacy requirements by engaging developers to think about privacy during the development process and providing real-time feedback on potential privacy issues. We start by presenting new findings based on a series of semi-structured interviews with Android developers, probing into the difficulties with privacy that developers face when building apps. Based on these findings, we implemented a proof-of-concept prototype of Coconut and evaluated it in a controlled lab study with 18 Android developers (including eight professional developers). Our study results suggest that apps developed with Coconut handled privacy concerns better, and the developers that used Coconut had a better understanding of their code's behavior and wrote a better privacy policy for their app. We also found that requiring developers to do a small amount of annotating work regarding their apps' personal data practices during the development process may result in a significant improvement in app privacy.

CCS Concepts: • **Security and privacy** → *Usability in security and privacy*; • **Human-centered computing** → *Empirical studies in ubiquitous and mobile computing*; • **Software and its engineering** → *Integrated and visual development environments*;

Additional Key Words and Phrases: Android development, privacy, programming environment, human-centered methods

ACM Reference Format:
Tianshi Li, Yuvraj Agarwal, and Jason I. Hong. 2018. Coconut: An IDE Plugin for Developing Privacy-Friendly Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4, Article 178 (December 2018), 35 pages. https://doi.org/10.1145/3287056

1 INTRODUCTION

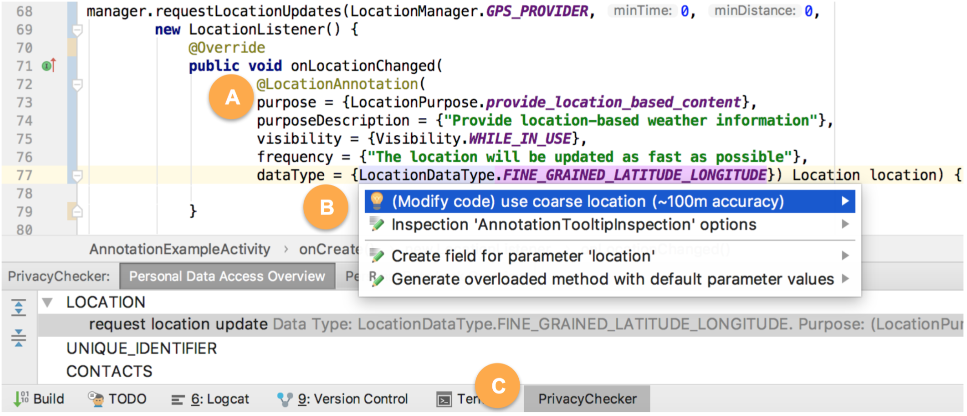
Privacy has become a growing concern with today's smartphone apps, and the need for developers to protect users' privacy has become more urgent with the advent of new privacy regulations, such as the EU General Data Protection Regulation (GDPR). However, developers still face many challenges in handling privacy, which has resulted in many apps using sensitive personal data in a problematic manner [5, 17, 58]. Prior research has identified various causes of this misbehavior. For example, copying-and-pasting can be convenient, but can also introduce security bugs [18]. Unusable documentation and guidelines result in a lack of awareness and understanding of recommended security practices [3]. Poorly designed security and personal data APIs may incur extra overhead to conform to the principles of data privacy [2, 24, 25, 31]. Perhaps most importantly, developers usually treat privacy as a secondary concern [4, 8, 32], and have an incomplete understanding of what needs to be considered regarding data privacy protection [21].

Authors' addresses: Tianshi Li, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA, tianshil@cs.cmu.edu; Yuvraj Agarwal, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA, yuvraj@cs.cmu.edu; Jason I. Hong, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, 15213, USA, jasonh@cs.cmu.edu.

2474-9567/2018/12-ART178 $15.00
https://doi.org/10.1145/3287056

Researchers and practitioners have developed a number of tools to promote secure coding [12, 33, 36, 48, 54]. However, there is currently little support to help developers in creating privacy-friendly apps. Specifically, how to design usable tools to facilitate certain aspects of privacy (e.g. privacy notices, data retention, and data sharing) has received little attention. Prior work suggests that these under-investigated aspects are indeed areas where developers lack the most awareness and understanding [21].

Our overall goal is to create tools to help app developers manage the complex requirements for privacy, especially for those who work independently or in a relatively small organization that cannot afford a privacy team. We first conducted semi-structured interviews with nine Android developers to examine their understanding of privacy and how they deal with privacy issues. Several of our findings echo those from prior work, such as the tendency to de-prioritize privacy-related tasks and being unaware of the personal data automatically shared with some third-party advertising libraries [9]. We also identified additional challenges that have not yet been documented in existing literature. For example, developers may have inaccurate understanding of how their apps handle personal data, due to frequent iterations of the app and lack of documentation in collaboration scenarios.



```
manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTime: 0,  minDistance: 0,
        new LocationListener() {
            @Override
            public void onLocationChanged(
                    @LocationAnnotation(
                    purpose = {LocationPurpose.provide_location_based_content},
                    purposeDescription = {"Provide location-based weather information"},
                    visibility = {Visibility.WHILE_IN_USE},
                    frequency = {"The location will be updated as fast as possible"},
                    dataType = {LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE}) Location location) {

            }

```

LOCATION
request location update Data Type: LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE. Purpose: (LocationPur
UNIQUE_IDENTIFIER
CONTACTS

A
B
C
(Modify code) use coarse location (~100m accuracy)
Inspection 'AnnotationTooltipInspection' options
Create field for parameter 'location'
Generate overloaded method with default parameter values
AnnotationExampleActivity › onCreate
PrivacyChecker:
Personal Data Access Overview
Pe
Build
TODO
6: Logcat
9: Version Control
Ter
PrivacyChecker

Fig. 1. The main features of Coconut. A: LocationAnnotation is a customized Java Annotation with certain fields (e.g. "purpose") that help developers describe how and why the object 'location' obtained from the "requestLocationUpdates" API call is used. B: Coconut can give real-time feedback of potential privacy issues (highlighted in purple) and, in some cases, offer a quick fix. Here, the quick fix makes it easy to change the code to only collect coarse-grained location data. C: Coconut also uses these annotations to generate a summary of personal data practices in the app.

The findings from prior work and our interviews indicate the need for new mechanisms and tools to support privacy when developing apps. Towards this end, we designed and implemented Coconut, an IDE plugin for Android Studio, the most popular IDE for Android development. Figure 1 shows a screenshot of Coconut. Coconut uses heuristics to detect code that handles personal data, and asks the developer to add an annotation that describes how and why the personal data is used. Each annotation is a customized Java annotation of key-value pairs, with keys predefined for the specific type of personal data used (see Figure 1A). While programming, Coconut offers real-time feedback on potential privacy concerns by highlighting the annotation. In some cases, Coconut can offer quick fixes that make it easy to directly adopt the recommended practice (see Figure 1B). Lastly, all annotations are gathered in the "PrivacyChecker" summary panel, to facilitate review (see Figure 1C).

We conducted a between-subjects lab study with 18 Android developers to evaluate the usability and effectiveness of Coconut. The results suggest that Coconut could help developers deal with privacy issues from multiple aspects, such as avoiding violations of privacy principles, gaining more knowledge in the apps' behavior, and providing better privacy notices to end users.

This paper makes the following research contributions:

- We systematically examine developers' understanding of privacy practices through a series of semi-structured interviews. Our results helped us identify common misconceptions and difficulties with privacy.
- We present the design and implementation of Coconut, an Android Studio plugin for privacy, which nudges developers to think about privacy while programming, improves developers' understanding of the app's personal data use, increases developers' knowledge of alternative options to achieve a better trade-off for privacy, and motivates developers to search and learn about privacy.
- We present the results of our lab studies of Coconut. Our results show that developers using Coconut can build more privacy-preserving apps, gain a better understanding of how their app handles personal data, and write better privacy notices for users. We also observed other privacy challenges, such as developers inadvertently introducing privacy issues while tweaking their code for some functionality to work.

2 RELATED WORK

2.1 Demystifying Android Developers' Misbehaviors Related to Privacy

To provide proper developer support, we need to understand what accounts for common privacy issues in the Android ecosystem. Research on this topic is growing, but is still at a relatively early stage and shows a strong bias towards security-related issues.

One fundamental issue relates to how much developers value privacy and how much they pay attention to it. Prior work has found that developers may make design decisions that trade off privacy for better usability or for features they would like to have [4, 8, 21, 32], which may be in contrast to users' actual preferences for more privacy guarantees [46]. Developers also feel that security is the responsibility of other parties [27, 44, 55], which suggests that developers who work independently or in an organization that cannot afford a specialized security team may pay less attention or have a harder time dealing with security issues.

Several researchers have investigated the gap between widely acknowledged principles for data privacy and how developers actually understand privacy. Although most of this work looks at developers outside of Android, the findings should be generalizable for our needs since the studies involved few platform-specific details. For example, Hadar et al. found that developers hold a partial understanding of privacy, mostly limited to security concerns, and the organizational privacy climate was an important factor playing into their perceptions of privacy [21]. Sheth et al. found that developers preferred data anonymization to using privacy policies to reduce privacy concerns, and there exists a large disparity between developers' and users' belief on the same issue [46]. This body of work suggests many developers have a vague and incomplete understanding of what may raise users' privacy concerns and what might actually compromise their privacy.

Some papers drill down into more technical details. One frequently reported issue was that developers may lack knowledge of potential privacy invasions and corresponding coping strategies. For example, many developers are not aware that some advertising and analytics third-party libraries automatically share users' personal data with service providers [9]. This is likely due to poor readability of the privacy policies of third-party services [8] and insufficient exposure to privacy guidelines [9]. A number of papers studied the limitation of existing programming models and developer support, such as program analyzers detecting security vulnerabilities [47, 52, 53], security APIs / personal data APIs [2, 4, 24, 51], and official documentation / other information sources for security [3, 4, 18]. However, most of this body of work focuses solely on security.

Understanding how developers comprehend privacy guidelines and how these principles influence implementation can be useful for informing the design of better developer tools. However, as noted earlier, current work mostly focuses on security, and offers little insight about other aspects of privacy like data retention, purpose limitation, and privacy notices. Towards this end, we conducted a series of semi-structured interviews with

Android developers about their app development experiences, focusing on personal data practices. Our results both echo findings from prior literature and provide new insights.

2.2 Developer Support for Privacy

Here, we review past work in developer support for privacy/security, focusing mostly on Android. We also discuss some gaps in these tools.

2.2.1 Documentation and Tutorials for Privacy/Security. Developer support for educating developers of essential privacy principles is primarily limited to privacy guidelines [7, 14, 22, 38–42, 49] or official documentation from the platform [15]. Although they are carefully designed to be comprehensive and written in plain language, Balebako et al. [9] found that developers still have a low awareness of them. Acar et al. [3] found that they were harder to use than informal information sources such as QA sites. These sources are also highly distributed, with advice spread out across many guidelines. Lastly, they tend to only offer high-level advice, such as only using sensitive data when necessary, which may not be easy to apply concretely. These issues indicate the need for a better form to educate developers of these privacy principles in practice.

2.2.2 Static Analyzers for Privacy/Security. Several program analyzers have been built to detect security vulnerabilities for Java and Android [11, 34, 35, 43]. However, past work found a lack of adoption of these analyzers [52, 53], suggesting that perceived importance of security and visibility of peer developers using these tools are important factors influencing adoption. Furthermore, these analyzers are mostly used by security experts rather than normal developers [50], echoing past findings that some developers tend to treat security (and likely privacy as well) as the responsibility of specialized teams [27, 44, 55]. There are also information flow analyzers which can detect malicious or unwanted information leaks from an app [6, 17, 20, 30, 37]. However, it is currently unclear how to map these leaks to specific privacy risks, let alone helping developers mitigate those risks.

2.2.3 IDE-Level Support for Privacy/Security. There are some existing IDE plugins for detecting security vulnerabilities in Android [1, 23, 36, 54], which can offer developers real-time feedback on security issues as they are coding, and in some cases even provide developers with a direct solution to fix the problem. There is also some IDE-level support for other privacy-related issues. For example, Android Lint tries to mitigate over-privileged data tracking behaviors by detecting hardware identifier usage, and suggests alternative and more privacy friendly choices like advertising ID or instance ID [1]. However, the amount of IDE-level support to help developers handle other privacy-related issues is much more limited than security issues.

2.2.4 Support for Enforcing Privacy Policy Compliance. Researchers have investigated how to automate compliance checking of privacy policies, much of which relies on manual review. For example, Bing has an internal system[45] that can automatically check code compliance with privacy policies. Researchers have also looked at new programming models for enforcing privacy policies by factoring out specification of security and privacy concerns from the rest of the program [56, 57]. In this line of work, privacy is handled in a top-down approach, with privacy policies first specified and then enforced in the implementation. In contrast, we tackle this problem in a bottom-up way, asking developers to think about and annotate essential information about privacy practices when constructing code. Our evaluation suggests our tool may also help developers make privacy policies.

2.2.5 Support for Creating Privacy Policies. Privacy policies are required by some laws and by some app stores. Developers can use online privacy policy generators (e.g., freeprivacypolicy.com, generateprivacypolicy.com, appprivacy.net) to make privacy policies for their apps. These generators typically step developers through a series of questions, which are often too coarse to capture enough details of the personal data use. Furthermore, in our interviews we found that developers may not have an accurate and up-to-date understanding of their apps' data collection behaviors, suggesting that these tools may not be able to solicit reliable answers. Some

Table 1. Privacy guidelines that we used to inform our interviews and the design of our tool. These guidelines discuss best practices around user-facing notices and control for privacy, data collection, data transmission, and data retention.

Name of the privacy guidelines
California Attorney General Mobile Privacy Guide [22]
Article 29 Opinion on Apps (European Union) [42]
Future of Privacy Forum and the Center for Democracy & Technology (FPF-CDT) Best Practices [38]
GSM Association (GSMA) Mobile Privacy Principles [7]
National Telecommunications and Information Administration (NTIA) Short Form Notice [49]
Federal Trade Commission (FTC) Mobile Privacy Disclosures [14]
Office of the Privacy Commissioner (OPC) Good Privacy Practices (Canada) [40]
Office of the Australian Information Commissioner (OAIC) Mobile Practice Guide (Australia) [39]
Information Commissioner's Office (ICO) Privacy in Mobile Apps (UK) [41]
Android Best Practices for Permissions and Identifiers [15]

programming models [31] are specifically designed to streamline analyzing how personal data is processed, which can be potentially used to help generate privacy policies. Our tool applies a similar idea by reminding the developer to add annotations about how the personal data is used, which could be more accessible to developers and may also promote the adoption of these privacy-friendly programming models.

3 BACKGROUND: PRIVACY PRINCIPLES FOR ANDROID DEVELOPERS

To ensure that our tool has the potential to tackle a comprehensive list of important privacy issues, we refer to four core aspects of privacy concerns to guide the design of the interviews with developers and the design of the tool. The four aspects were summarized from guidelines released by government, industry, and non-profit groups [7, 14, 15, 22, 38–42, 49] (listed in Table 1).

3.1 User-Facing Notice, Consent, and Control for Privacy

Making sure users have a clear understanding of what data is stored, how it is used, and for what purposes is the prerequisite for privacy-preserving personal data use. The regulations and guidelines usually include requirements and recommended practices about providing effective privacy notices to users, acquiring informed consent before using sensitive user data, and giving users proper control over their own data.

3.2 Data Collection

The key principles for data collection are purpose limitation and data minimization, meaning that user data collection behaviors should only be for specified, explicit, and legitimate purposes, and that developers should try to use the minimal amount of data to achieve their goal. Collecting the minimal amount of data is a multi-dimensional problem, including considerations of data granularity [31], access frequency, the visibility of collection (in background or while in use), etc. Developers should also be cautious when handling personally identifiable information (PII), especially when the collection behavior happens implicitly to end users, such as obtaining hardware identifiers for data tracking purposes.

3.3 Data Transmission

There are several potential privacy risks when sensitive data egresses from a user's device. Consequently, many regulations and guidelines require encrypted transmission for such data. Developers should also be careful about data sharing practices when data is sent to third-party services. Sometimes data can be shared automatically when using certain third-party libraries, which accounts for a large portion of data sharing in Android [13].

3.4 Data Retention

The privacy guidelines suggest that developers should minimize how long sensitive data is retained to achieve their goals. Developers should consider defining a data retention period and deleting unused, old data. When using third-party libraries, developers should pay attention to whether there is data collected from users that may be retained on the library's server and how the data retention policy is defined. Developers should not keep non-anonymized data indefinitely and should provide users with choices to remove their data by implicitly deleting their app/account or by supporting explicit data deletion requests.

4 SEMI-STRUCTURED INTERVIEWS: UNDERSTANDING DEVELOPERS' PRIVACY CHALLENGES

To inform the design of our tool, we first conducted a formative study with nine Android developers to understand the following: how they manage privacy in practice when developing apps, their general understanding of privacy, their mental models for some well-acknowledged data protection principles, and specific steps and practices used for addressing privacy issues. Balebako et al. also conducted interview studies with mobile app developers to study their privacy and security behaviors [9], which examined developers' perceptions and behaviors regarding privacy using high-level questions. In contrast, we inquired specifically about how apps that our participants had recently developed used personal data, and mapped out these personal data practices to fundamental privacy principles (summarized in Section 3) to identify misconceptions and incorrect behaviors for privacy.

4.1 Participants

We recruited nine Android developers via email, drawing on two sources. We first searched for developers who had published apps on the Google Play store. To cover both developers who use personal data intentionally for core functionality, or unintentionally via third-party libraries, we selected apps from the following categories: social, productivity, weather, finance, and games. The first four categories usually involve the use of personal data for core functionality, while the last category requires it less so. We focused on apps with a large user base by setting a minimum threshold of 1000 installations. We acquired email addresses of participant candidates from the contact information on Google Play, and then directly emailed about 10,000 developers. 34 people signed up for the study, with 10 located in the US. Finally, five people successfully scheduled an interview session with us. The other four interviewees were recruited using an email list of computer science students at our university. Two had published apps on Google Play, while the remaining two had over 3 years of experience of Android development. We interviewed local participants in-person and remote participants via the internet. All participants were located in the US. The study was approved by our university's IRB.

Table 2 shows the demographics of our participants. Our participants had diverse backgrounds, including independent developers creating apps as a hobby, full-time Android developers in companies with several hundred to several hundred million customers, researchers developing apps for their projects, and hackathon participants. We also had some participants who had previously received formal training in privacy.

4.2 Methodology

The interview consisted of two parts. The first part asked general questions about their app development experience, privacy training background, and perceptions about privacy. The second part focused on recent apps (up to 3 apps) they had developed. Specifically, the experimenters asked whether certain categories of personal data were acquired from these apps, and, if so, how. They were also asked what the rationale was for doing this.

All participants first signed a consent form and were compensated with an Amazon giftcard for $20 for each hour of the interview. Each interview was audio recorded and took around 1 hour to finish.

After briefly introducing our study goals and logistics, we asked developers about their Android development background in general, e.g. “When did you start developing Android apps?”. We also asked questions specifically about their understanding and training background in data privacy, e.g., “What is your understanding of protecting users’ privacy?”, which helped us understand their attitudes towards privacy before being primed with any privacy concepts and guidelines. To understand their knowledge about privacy protection, we asked whether they knew about some important privacy guidelines, such as the FTC guidelines on mobile privacy disclosures [14], and whether they have received any formal training in privacy. The complete questions are listed in the interview scripts in Appendix A.

In the second part of the interview, we focused on their three most recently developed apps. We asked participants about the functionality of the app and how it was developed. We were interested in whether the app was developed independently or by a team, and how the team members collaborated. We further drilled down into details about the personal data practices of these apps. We let the developers recall what personal data was used in the app and how it was used. To help with the recall process, we showed them three lists of different types of privacy-sensitive resources: PII data that usually needs to be directly solicited from users (e.g. credit card information), unique identifiers that can be acquired programmatically (e.g. MAC address), and personal data protected by the permission system (e.g. geographic location). The experimenter walked through every type of personal data presented on these lists and asked the developer whether their apps used it. If the answer was yes, the developer was then asked questions about the four core concepts discussed in Section 3, including “Did you have a privacy policy for the app and how did you create the privacy policy?” (3.1), “What’s the purpose for using the data?” (3.2), “Did you send them out of the phone?”, “Did you use any advertising or analytics third party library such as AdMob, Flurry etc.?” (3.3), and “Did you store the data?” (3.4). The complete question list can be found in the interview scripts in Appendix A.

We tried out the publicly available apps during the study to verify whether the actual behavior matched the developer’s description. For example, a developer might mention that they created an in-app notice which explained the data usage, but the notice did not show up when using the app. Once we identified such a mismatch, we would further ask them about the reasons. For developers discovered on Google Play, we installed their apps before the study. For other developers, we did so during the interview. We examined the list of permissions requested by the app, which is displayed in the corresponding app description page on Google Play. If developers asked for a permission but did not mention the corresponding data use in the interview, we would remind them of it. Similarly, we also searched for whether there was a privacy policy linked in the app website or the app description page on Google Play during the study, and checked whether their answer matched our observation. The participants corrected themselves later when they recalled more details about the app, or gave an inconsistent description about the same data practice. At this point, we would further ask them about the actual case and why they gave the inaccurate description.

4.3 Data Analysis

One researcher transcribed and coded the data following a bottom-up, open coding approach, and held periodic discussions with the rest of the research team. We were interested in two high-level topics: developers’ attitudes towards privacy and the challenges developers faced in handling privacy. For the first question, we drew on responses to the open-ended question, “What is your understanding of protecting users’ privacy?”, which was asked up front before introducing any privacy principles to the developer. For the second topic, we analyzed the discussion of their recently developed apps. While reviewing the transcripts, we identified challenges based on

Table 2. Background information of interviewees regarding Android development (Since: the year our participant started Android development; Google Play: whether the developer had published apps on Google Play Store; Purposes: the purposes to develop the apps; Privacy training: whether the developer had received any formal training in privacy and what kind of training; Team size: the number of Android developers in the team; ≥ SDK 23: whether the developer had developed apps for versions ≥ SDK 23 (Android M), for which they need to handle requests to some dangerous permissions at runtime)

ID	Since	Google Play	Purposes	Privacy training	Team size	≥ SDK 23
P1	2012	Y	hobby, full-time	CITI Training for research	1 (hobby), 2	Y
P2	2012	Y	hobby, full-time	N	1 (hobby), 3	Y
P3	2013	Y	hobby, full-time	N	1 (hobby), 4	Y
P4	2012	Y	hobby, full-time	Master's degree in information security	1 (hobby), 3	N
P5	2013	Y	full-time	N	15	Y
P6	2011	N	research	Training in handling user data for research and job	3	N
P7	2011	N	hobby, hackathon	Attended seminars on research in usable privacy	1 (hobby), 4	N
P8	2008	Y	full-time	Worked specifically on privacy 4 years ago	1	Y
P9	2008	Y	hobby	N	1	N

two criteria. The first are difficulties that the developer explicitly mentioned when handling privacy. The second are factors that could lead to violations of privacy guidelines but not explicitly mentioned by the developer.

We extracted quotes from the interview transcripts and looked for those that could be considered examples of the two topics above. For developers' attitudes toward privacy, we identified common themes from the quotes. For analyzing the challenges, we first grouped the quotes based on the four concepts discussed in Section 3, then combined them to generate the final privacy challenges.

4.4 Interview Results and Implications on the Design of Developer Tools for Privacy

In this section, we present what we learned from the interview about developers' attitudes and the challenges they face regarding privacy. Our findings shed light on how to design developer tools to promote privacy best practices, and also lay a foundation for the design of Coconut. We conclude each finding with a brief discussion on how to apply it to the design of a tool that tackles the corresponding issue.

4.4.1 Privacy Attitudes: Our Developers Do Care About Privacy, Though They May Only Hold a Partial Understanding of Privacy. We identified three types of attitudes towards privacy from our interview. The first type is developers who explicitly or implicitly expressed the idea that developers are responsible for carefully handling users' sensitive data. For example, P6 treated privacy as *"definitely something I consider important."*; P8 mentioned that *"My apps are very privacy sensitive, so I try not to keep a lot of information ... From my perspective, I just want to build the application that helps the user."* However, each of them only considered partial aspects of privacy, such as collecting data only when users have fully consent to it (P1), preventing using identifiable information (P1), minimizing data usage (P2, P6, P7, P8), or encrypting or obfuscating data before sending it out of the phone (P4, P8). Besides, most participants have low awareness of privacy concerns related to data sharing and data retention.

The second type is developers who care about privacy but tend to only rely on external advice such as dedicated teams on security/privacy or privacy requirements of the app store. This is similar to what has been identified in [27, 44, 55]. P5 for example worked at a large company with a dedicated security team and hundred millions of customers. He said, *"According to me, this whole logic was to abstract it (privacy), ... I mean that, there is a different security team in the company, so it's their job to do this."* Interestingly, he also mentioned that the security team's late notice of changing HTTP to HTTPS caused some extra overhead, which could be avoided if privacy/security were taken into consideration in advance: *"... what we did was we implemented HTTPS, but the app got slower. And all the product managers told us that you could not release this because the app was slower compared to previous versions, so we had to make different optimizations in other parts of the application in order to counter that."*

The third type is developers who have a passive attitude towards privacy and lack motivation in protecting privacy. For example, P3 commented that he was motivated to explicitly inform users about how the personal data is used only when forced to do so: *"If you can make tool with as little information about the user explicitly asked, the better, because most people don't like to give out a lot of information, especially if it's not a trusted developer."*

The lack of consideration for privacy and the incomplete understanding of different aspects of privacy suggest that developers may need to be better educated about privacy. It also suggests that developer tools for privacy should help developers handle privacy in a systematic way, which provides a broad coverage of different aspects of privacy and the corresponding privacy issues.

4.4.2 Inaccurate Understanding of App Behaviors Creates Hurdles to Making Appropriate Privacy Notices. Knowing what data one's app uses and how it is used is a prerequisite for providing users with accurate and helpful privacy notices. However, we observed that 5 participants' responses about their data practices were inconsistent with the actual case. Three potential causes emerged. First, developers did not have sufficient knowledge in how some system and third-party APIs worked, especially the underlying data collection behaviors of third-party libraries. Second, developers were unable to keep up with the implementation details of the latest app versions, due to fast iterations. For example, developers might collect more data because of a new feature, or remove some data collection behaviors because the feature did not work out or they did not need the data anymore. The third reason is team dynamics. The members of a development team vary, and new members need to be onboarded about existing data practices. Similarly, when someone leaves the team, it may be a problem if what that person knows is not well documented. As such, a developer tool can mitigate this issue by explicitly informing developers of what data is obtained from API calls, automatically tracking data practices across multiple versions, and presenting the data practices as an alternative to manually maintained documentation.

4.4.3 Lacking Knowledge of Feasible, Less Privacy Invasive, Alternatives. Managing privacy often involves tradeoffs with app functionality, usability, and performance. However, while sometimes there are less privacy-invasive alternatives, such as using a randomized ID rather than a MAC address, they are not used due to lack of developer awareness. For example, 7 participants stored data with a unique identifier or other PII such as email address or user name. Three used hardware identifiers such as Android ID, IMEI, or Bluetooth MAC address. Using hardware identifiers for tracking purposes is privacy invasive if the data is leaked since it is persistent and can be linked to a specific user/device. In fact, we found that developers did not really need to use these hardware IDs, and were oblivious to their privacy ramifications and unaware of better alternatives.

As such, another useful feature for a developer tool is to proactively remind them of privacy-preserving alternatives that can achieve similar goals.

4.4.4 Privacy Is Treated as a Secondary Task. Our results are consistent with prior work which demonstrated that developers tended to consider privacy as less critical than other factors such as usability [4, 8, 32]. For example, P3 told us that his app would pop up its own alert dialog explaining how it would use the permission to obtain data and for what purposes. However, when we tested the app, it did not show such an alert dialog along with the permission request. Later he explained that another feature for a fast release was given priority ahead of privacy considerations, delaying the in-app privacy notice implementation.

To address this issue, a developer tool should be able to keep track of what tasks for privacy have not been completed yet, and support privacy task management and reminder at the appropriate timing.

4.4.5 Developers May Lack Motivation for Privacy When There Are Few Constraints in What They Can Do.

Developers are susceptible to ignoring privacy issues when the design of the programming model lacks sufficient constraints. Android's 'identity' permission lets developers programmatically get information related to users' identity, such as account name and email address. However, since multiple data items are controlled by one permission, developers can also use the information that they do not need once users grant the permission. Participants P3 and P6 said that they collected and stored account name and email address information when they only needed the latter. Besides, a developer might request a permission for one purpose but use it for other purposes because they feel these features are hard to justify, such as for a potential future project (P5). As another example, some sensitive data is not protected by permissions in some versions of Android, such as IP address and some device IDs like Android ID, so developers may be less motivated to constrain the use of such data or explain it to the user.

It is important to prevent apps from being over-privileged in terms of privacy when the system-level constraints are too coarse or even non-existent. One possible direction that has been explored in recent work is to design better programming models [31]. However, there are often multiple APIs to obtain the same user data and developers can choose to use other APIs regardless. We believe that adding suitable privacy support to the developer tools themselves can help with this issue. For example, a developer tool can detect the use of APIs that access personal data, and remind developers to only use data needed for legitimate purposes.

5 DESIGN OF COCONUT: AN IDE PLUGIN FOR PRIVACY

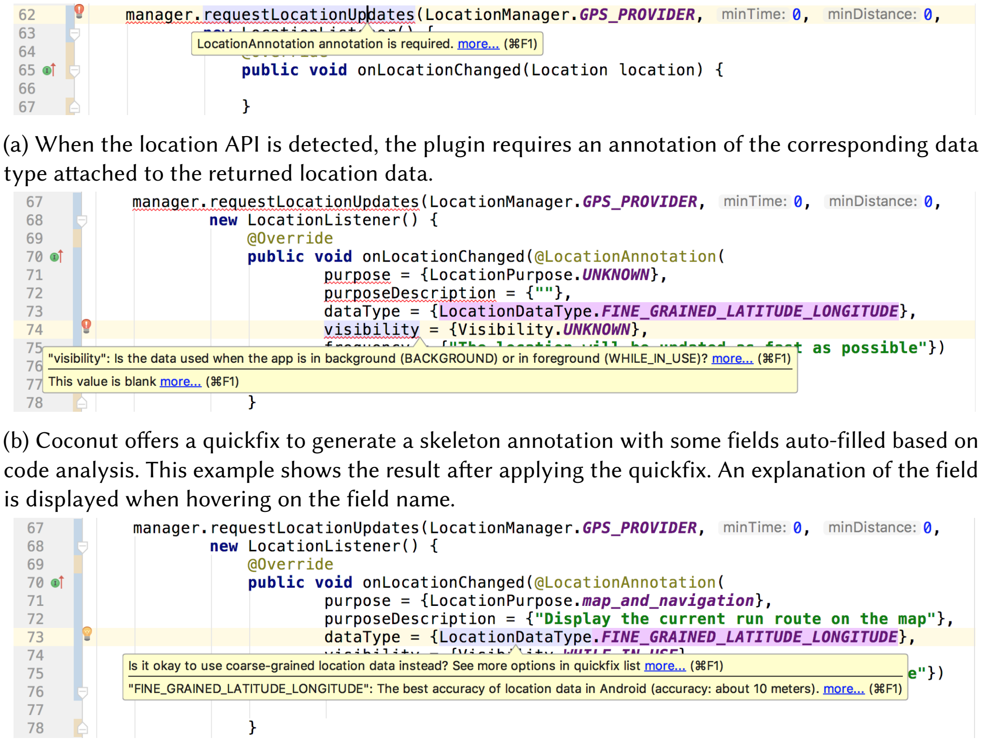
The goal of Coconut is to get developers to think about privacy and make it a natural part of their app development process. We decided to build our tool as an IDE plugin, thereby enhancing the Android Studio development environment already familiar to, and used by, all developers. We first present a hypothetical use case of our tool, and then describe the design of the plugin along with our design rationale.

5.1 Coconut Use Case

Here, we present a fictional scenario to help illustrate how Coconut can help developers write privacy-preserving apps. Ann is developing a run tracking app, and a core feature of this app is to render the current route on the screen in real time, and then upload the route to the remote server database, where it is backed up.

When collecting location data for rendering the route on the map, a '@LocationAnnotation' is needed to describe how and why the location data is used (see Figure 2). Ann decides to use the '`requestLocationUpdates`' among the series of native '`LocationManager`' APIs. After writing the API call in the Android Studio IDE, Coconut uses simple heuristics to detect this API, and requests the developer add a '`@LocationAnnotation`' annotation to the returned location object. (Figure 2a). Ann uses a quickfix to automatically generate an annotation skeleton with several fields (i.e. '`dataType`', '`frequency`') automatically filled in based on the parameters in the API call (i.e. '`GPS_PROVIDER`', min time and distance interval for location updates), while other fields that can not be inferred from the code need to be filled manually. Ann then inspects each of them and fills them out one by one. When she is not sure about what she is supposed to put in a particular field (e.g. '`visibility`'), she just hovers on the field name to read a further explanation in the tooltips (Figure 2b).

Filling out the annotation gets Ann to think through this use of location data. She first examines the collection of predefined purposes and picks '`LocationPurpose.map_and_navigation`' for her case. She then elaborates a bit more about this purpose in the field '`purposeDescription`'. During this process, she is reminded to check that she has a legitimate purpose to collect location data. When it comes to '`visibility`', she considers for a while and decides that the data is only needed when the user is using the app, so she opts for '`Visibility.WHILE_IN_USE`'.



```
        manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTime: 0,  minDistance: 0,
                    public void onLocationChanged(Location location) {

                    }
```

LocationAnnotation annotation is required. more... (⌘F1)

(a) When the location API is detected, the plugin requires an annotation of the corresponding data type attached to the returned location data.

```
        manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTime: 0,  minDistance: 0,
                new LocationListener() {
                    @Override
                    public void onLocationChanged(@LocationAnnotation(
                            purpose = {LocationPurpose.UNKNOWN},
                            purposeDescription = {""},
                            dataType = {LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE},
                            visibility = {Visibility.UNKNOWN},
                                                                           as possible"})
                    }
```

"visibility": Is the data used when the app is in background (BACKGROUND) or in foreground (WHILE_IN_USE)? more... (⌘F1)

This value is blank more... (⌘F1)

(b) Coconut offers a quickfix to generate a skeleton annotation with some fields auto-filled based on code analysis. This example shows the result after applying the quickfix. An explanation of the field is displayed when hovering on the field name.

```
        manager.requestLocationUpdates(LocationManager.GPS_PROVIDER,  minTime: 0,  minDistance: 0,
                new LocationListener() {
                    @Override
                    public void onLocationChanged(@LocationAnnotation(
                            purpose = {LocationPurpose.map_and_navigation},
                            purposeDescription = {"Display the current run route on the map"},
                            dataType = {LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE},
                                                                              e"})
                    }
```

Is it okay to use coarse-grained location data instead? See more options in quickfix list more... (⌘F1)

"FINE_GRAINED_LATITUDE_LONGITUDE": The best accuracy of location data in Android (accuracy: about 10 meters). more... (⌘F1)

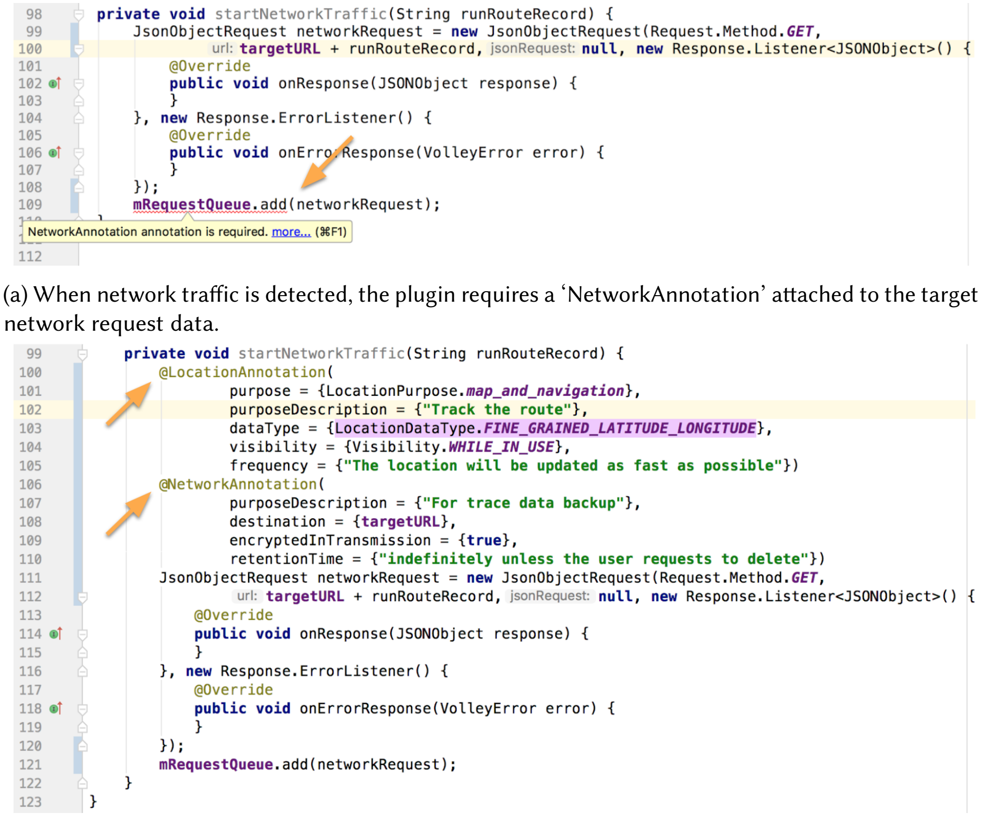
(c) Potential privacy issues are defined as a warning and highlighted in purple. Where possible, quickfixes are provided so that developers can apply another option with one click if they think it is a better fit for their purposes.

Fig. 2. Building a run tracking and mapping app with Coconut: When collecting location data for rendering the route on the map, a '@LocationAnnotation' annotation is needed to describe how and why the location data is used.

After resolving all "errors" marked with a squiggly red underline, she attends to the warning at the value 'LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE' highlighted in purple (Figure 2c). Before reading this tooltip, she has a rough idea that she can collect fine-grained location because the 'ACCESS_FINE_LOCATION' permission is requested, but she does not know the exact accuracy of "fine-grained location", or what other accuracy is available in Android. After exploring the quickfixes for this warning, she learns more about the alternatives and finally decides to stick with the original choice because it best fits her purpose.

When sending the routes to the remote server, a '@NetworkAnnotation' annotation is needed to describe how the data is transmitted from the phone, along with other annotations (the '@LocationAnnotation') that describe what personal data may leave the phone and for what purposes (see Figure 3). Ann starts to construct a network request to send the routes aggregated from the location data to the remote server. Similar to the location API in the previous example, Coconut detects the network connection, and requires developers to add a '@NetworkAnnotation' that describes how data is transmitted out of the phone (Figure 3a). In addition, Coconut also requires Ann to annotate what data may leave the phone at this point. Since this request contains location gathered previously, Ann just reuses the prior annotation. The final result is presented in Figure 3b.

When writing the privacy policy for this app, Ann reviews personal data practices using the PrivacyChecker window (see Figure 4). One month later, Ann finishes the app and plans to release it on the app store. The app



```
private void startNetworkTraffic(String runRouteRecord) {
    JsonObjectRequest networkRequest = new JsonObjectRequest(Request.Method.GET,
            url: targetURL + runRouteRecord, jsonRequest: null, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
        }
    });
    mRequestQueue.add(networkRequest);
}
```

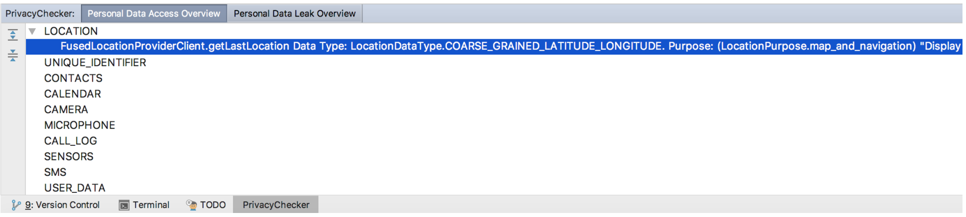
NetworkAnnotation annotation is required. more... (⌘F1)

(a) When network traffic is detected, the plugin requires a ‘NetworkAnnotation’ attached to the target network request data.

```
    private void startNetworkTraffic(String runRouteRecord) {
        @LocationAnnotation(
                purpose = {LocationPurpose.map_and_navigation},
                purposeDescription = {"Track the route"},
                dataType = {LocationDataType.FINE_GRAINED_LATITUDE_LONGITUDE},
                visibility = {Visibility.WHILE_IN_USE},
                frequency = {"The location will be updated as fast as possible"})
        @NetworkAnnotation(
                purposeDescription = {"For trace data backup"},
                destination = {targetURL},
                encryptedInTransmission = {true},
                retentionTime = {"indefinitely unless the user requests to delete"})
        JsonObjectRequest networkRequest = new JsonObjectRequest(Request.Method.GET,
                url: targetURL + runRouteRecord, jsonRequest: null, new Response.Listener<JSONObject>() {
            @Override
            public void onResponse(JSONObject response) {
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
            }
        });
        mRequestQueue.add(networkRequest);
    }
}
```

(b) After completing all required annotations. NetworkAnnotation describes how the data is transmitted, other personal data annotation describes what data will leave the phone at this point.

Fig. 3. Building a run tracking and mapping app with Coconut: When sending the routes to the remote server, a ‘@NetworkAnnotation’ annotation is needed to describe how the data is transmitted out of the phone, along with other annotations (e.g. the ‘@LocationAnnotation’) that describe what data may leave the phone at this point and what the purposes are.



```
▼ LOCATION
      FusedLocationProviderClient.getLastLocation Data Type: LocationDataType.COARSE_GRAINED_LATITUDE_LONGITUDE. Purpose: (LocationPurpose.map_and_navigation) "Display
  UNIQUE_IDENTIFIER
  CONTACTS
  CALENDAR
  CAMERA
  MICROPHONE
  CALL_LOG
  SENSORS
  SMS
  USER_DATA
```

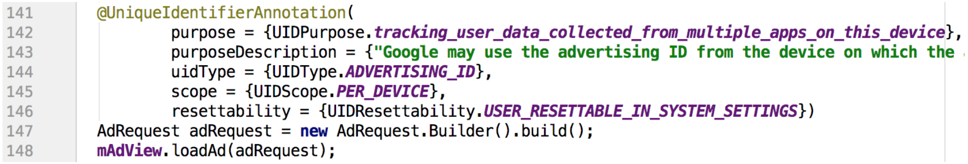
Fig. 4. The developer can examine the global personal data practices using the PrivacyChecker window. Personal data practices are categorized by data type and whether it leaves the internal runtime environment. All cells that display data use instances are clickable and can help the developer navigate to the corresponding code snippet when they click on it.

store requires her to provide a privacy policy for her app. She starts looking at other apps’ privacy policies and tries to adapt them to her app’s practices. During this process, she uses the PrivacyChecker tool window, which is a feature of Coconut that gathers all data practices from the annotations. By referring to the annotations, Ann can precisely describe how personal data is used for the current version of the app, including how the location data is collected for tracking running routes and how data is sent over the network for backup purposes.

5.2 Designing Coconut to Promote Privacy-Preserving Personal Data Use

Above, we described how an Android developer can use Coconut across the entire development process to gain a better understanding of app behavior, learn about potential privacy issues and better choices, and manage data practices. Next, we detail the design of the main features and explain the underlying design rationale.

5.2.1 Coconut Requires the Developer to Annotate Personal Data Practices in a Pre-Defined Format. Privacy annotation is a key feature of Coconut (e.g. the '`@LocationAnnotation`' and '`@NetworkAnnotation`' in Figure 3b). To implement this concept, we use Java Annotations[1], a form of syntactic metadata in Java. Coconut has two major kinds of annotations, namely 'source annotations' like '`@LocationAnnotation`', and 'sink annotations' like '`@NetworkAnnotation`'. Using these two types of annotation, developers can track how personal data flows within an app. More specifically, the source annotation specifies where the personal data is acquired, and the sink annotation specifies where it leaves the app. Since developers are likely to have an incomplete understanding of privacy as discussed in Section 4.4.1, we draw on previous research in data privacy modeling [26, 28] to design the fields for each type of annotation, so that they cover important aspects of data privacy that are supposed to be described in the privacy policy. Coconut runs code inspection continuously in the background. If a relevant API call is detected and the corresponding annotation is missing, it is treated as a "missing-annotation error", and the API name will be marked with the same indicator of compile error in Android Studio (i.e. a red squiggly underline) to inform developers that this is a required task.



```
@UniqueIdentifierAnnotation(
        purpose = {UIDPurpose.tracking_user_data_collected_from_multiple_apps_on_this_device},
        purposeDescription = {"Google may use the advertising ID from the device on which the
        uidType = {UIDType.ADVERTISING_ID},
        scope = {UIDScope.PER_DEVICE},
        resettability = {UIDResettability.USER_RESETTABLE_IN_SYSTEM_SETTINGS})
AdRequest adRequest = new AdRequest.Builder().build();
mAdView.loadAd(adRequest);
```

Fig. 5. The entire @UniqueIdentifierAnnotation is automatically generated because Coconut is pre-programmed with the implicit data collection behavior of Google AdMob library.

5.2.2 Coconut Helps Generate Annotations and Checks the Consistency Between Annotation and Code Using Code Analysis. To reduce cognitive load when annotating personal data practices, and to mitigate errors in annotation due to carelessness or misconception of how the API works, Coconut analyzes the code and infers the value of certain fields of an annotation or even the entire annotation. For example, the '`datatype`' field in Figure 2b and the entire '`@UniqueIdentifier`' annotation in Figure 5 are automatically generated. In addition, Coconut also automatically locates the object that contains the personal data which the annotation should be attached to.

Additionally, if there is a discrepancy between the value of these fields in the annotation and the behavior of the code, Coconut will report an "annotation-code inconsistency warning" and highlight the corresponding field name in red. We use a different color than normal warnings in Android Studio and other types of warnings in Coconut, so that developers can establish a mapping in their mental model between the color and the type of issue. This feature helps developers stick to the most updated and accurate understanding of their apps' behavior, which addresses the problem discussed in Section 4.4.2.

[1] https://docs.oracle.com/javase/tutorial/java/annotations/

5.2.3 Coconut Provides Real-Time Feedback on Potential Privacy Issues. Coconut offers a “privacy-concern warning” when any potential violation of privacy principles is detected (e.g. network traffic not encrypted; using hardware identifier when not necessary; collecting user data for purposes that may not be clear to users), or when the developer may not know that there exists less privacy-invasive approaches to achieve similar goals (e.g. changing parameters to get coarse-grained data when high accuracy is not needed; using APIs designed for privacy [31]). The corresponding field value is highlighted in purple to notify the developer.

To address the issue of developers not being able to balance the trade-off between privacy and other factors (discussed in Section 4.4.3), Coconut provides developers with alternative recommendations in the tooltips. If possible, we also provide quickfixes that can directly apply the recommended change in the code. As a result, developers’ awareness of alternatives increases, so that they can make more informed design decisions. The quickfix feature may also lower the threshold for adopting recommended practices.

5.2.4 Coconut Gathers Personal Data Practices in One Place to Facilitate Review. Coconut offers *PrivacyChecker*, a tool window that gathers all the personal data practices described by the annotations. It displays the data used internally and the data that may leave the the internal runtime environment in separate panels, and categorizes them by personal data types (See Figure 4).

The content in the panel is constructed dynamically, so the developer will always have a clear understanding in the personal data practices of a certain version. Developers can also review what privacy issues still exist based on unresolved Coconut “privacy-concern warnings” (related to the problem discussed in Section 4.4.4).

5.3 Implementation

We built a proof-of-concept prototype of Coconut, which consists of two parts: *an Android Studio plugin* and a *privacy annotation support library*. The plugin was developed using the IntelliJ Platform SDK[2]. We use this SDK to monitor API usage by capturing file changes in the IDE in real-time, to inform developers of privacy errors and warnings by registering errors/warnings using code inspection APIs, to generate pre-filled annotation skeleton and fix privacy issues by manipulating the code via the syntax tree, and to present the personal data use in the app in a tool window. The version for our lab study comprises 7770 lines of Java code, most of which is used for establishing the basic framework and handling the user interface.

The current implementation supports analyzing a selected group of system APIs and third-party libraries that use personal data (listed in Appendix D Table 8). The scope was determined with the goal of providing sufficient flexibility in the programming tasks of our lab study (described in Section 6.2). Given the current framework, this list can be easily extended. Recent research has demonstrated the long-tail distribution in the usage of third-party libraries [13], which suggests that it is feasible to cover a wide range of personal data collection behaviors caused by third party libraries. There are always-running threads that monitor the use of these APIs. When a target API is detected, the system will infer the target location of annotations for this API, check if all required annotations are provided, valid, and consistent with the actual code behavior, and also check for other potential privacy issues. The privacy errors and warnings and the automatic quickfixes provided by Coconut are presented in Table 3.

Privacy annotations are customized Java annotations, which need to be predefined in Coconut’s privacy annotation support library. We list the definitions of annotations that our current implementation supports in Appendix D Table 7. The *source* and *sink* annotation annotate the code when the data is acquired and when it leaves the app. The *third-party lib* annotation is used to specify some configurations not defined in the code. For example, AdMob offers developers a web-based management system to specify whether the location-based advertising service is enabled. And we request the developer to also specify the same thing in the annotation so that Coconut can correctly infer the code behavior.

[2] https://www.jetbrains.org/intellij/sdk/docs/welcome.html

Table 3. Privacy issues that can be detected in the current proof-of-concept prototype for the lab study. These issues are detected by combining the code analysis and the annotations completed by the developer. Automatic quickfixes are also provided for developers, which can be applied with one click.

Severity	Privacy issue	Explanation	How to fix
Error	Missing valid annotation	The required annotation is not attached to the target variable, or when the annotation is not completed with valid values	Coconut can generate pre-filled annotation skeletons
Warning	Inconsistent annotation	The values in the annotation are inconsistent with what is speculated from the code. For example, such a warning will be generated if the user specifies PRIVATE_ACCESS in the Storage annotation while Coconut finds the data is actually stored in the public area by analyzing the API parameters.	Coconut can automatically modify the annotation to match the actual code behavior or automatically modify the API call to adapt to the value specified in the annotation
Warning	Potential violation of purpose limitation	The current data collection does not match the best practice for the purpose specified by the developer. For example, such a warning will be generated if the user collects a hardware ID (e.g. MAC address) for tracking user behavior within the app.	Coconut can automatically modify the code to acquire a UID that fits the purpose specified by the developer best
Warning	Implicit data collection that may not be expected by users	The visibility is specified as IN_BACKGROUND in the corresponding source annotation, or the background data collection is initiated by a third-party library whose behavior is already known	The tooltip of the warning reminds the developer to provide explicit explanation of this data collection behavior in the privacy notice for users.

5.4 Pilot Study and Feedback

During the development process of Coconut, we solicited early feedback from six developers to improve usability. We accepted suggestions we felt were feasible for the first prototype of Coconut. For example, we observed that developers tended to quickly test how to use new APIs. However, an error that prevents compilation (e.g. a missing or incomplete annotation) made it harder to just try out an API. Consequently, we updated Coconut to let developers temporarily ignore annotation compile errors. However, missing annotations still appear as errors, and need to be addressed before building the final app, because we still treat these annotations as a requirement.

6 EVALUATION METHODOLOGY

6.1 Participants

To evaluate the usability and effectiveness of Coconut, and to gain a better understanding of how developers handle users' personal data with and without our tool, we conducted a series of in-lab studies with 18 participants from March to May 2018. Our participants were recruited from two sources. The first source was developers on LinkedIn. We used "Android Developer" as a keyword and set the location filter to our local area to make sure developers can come to our lab to participate in the study. We found 30 qualified candidates in total and then contacted them using LinkedIn InMail. The second source was computer science students at our university. We

Table 4. Background information of the lab study participants regarding Android development (yrs of exp: how many years of experience do they have in Android development; active?: whether they were actively working on any Android development project as a software developer; professional?: whether they had worked as a professional Android developer; all apps: how many Android apps they had developed; playstore apps: how many apps were published on the Google Play store; made privacy policy?: whether they had the experience in making privacy policies.)

ID	yrs of exp	active?	professional?	all apps	playstore apps	made privacy policy?
C1	4	✓	✓	>5	3	✓
C2	2.5	✓	✓	3	1	
C3	3	✓	✓	3	3	
C4	4			4	2	
C5	2	✓		>5	3	
C6	4		✓	>5	8	
C7	1			3	0	
C8	1	✓		3	0	
C9	1	✓		3	0	
E1	4	✓	✓	>5	4	
E2	1	✓	✓	2	0	
E3	4	✓		>5	12	✓
E4	3	✓	✓	3	1	
E5	1			1	1	
E6	1	✓		3	0	
E7	4			3	0	
E8	2.5		✓	3	1	
E9	2			3	2	

posted an advertisement in a related Facebook group and email lists, and also put up physical posters for this study in our school. The study was approved by our university's IRB.

Among the 18 participants (14 males, 4 females), 8 self-identified as professional Android developers. All participants were over 18 years old and were familiar with using Android Studio. We used a between-subjects design, with a control group (not using the plugin) and an experimental group (using the plugin). We controlled the number of professional Android developers to be the same in each group. Both groups had an average of 2.5 years of Android development experience. Participants in the experimental group published an average of 2.3 apps on Google Play Store, and the control group 2.2. See Table 4 for more information.

6.2 Study Task Design

Study participants were asked to complete two programming tasks: a warm-up task and a main task. The warm-up task helped participants get familiar with the programming environment, and also served as a reference of their expertise in developing Android apps that handle personal data. In the warm-up task, developers needed to handle a run-time permission request for location, collect the actual location data, and display the current latitude and longitude on the main UI.

When designing the main task, we focused on three key aspects: the feasibility of finishing it within a limited amount of time, the scope of the privacy principles it can cover, and the ecological validity of the use cases. The high-level goal of the main task was to build a weather app. We drew on a widely-used feature in popular weather apps such as The Weather Channel[3] and AccuWeather[4], which is to provide location-based weather information. Multiple privacy principles should be considered here. First, the granularity of location should be subject to the purpose of getting weather information, which does not need to be very precise (purpose limitation). Second, the app needs to send the location data to a third-party weather provider service to get the weather information, which entails data sharing practices for the core functionality of the app and should be clearly conveyed to the users in advance, to conform to the privacy principles about providing privacy notices, especially for data practices that may be implicit to end users.

In addition to the core functionality, we also included some requirements for monetization and analytics. These are common purposes for using personal data, but fewer end users are aware of them, which may lead to privacy concerns. For app monetization, we asked developers to integrate a banner ad using Google's AdMob library, which is the most popular advertising library on the market. Prior research has demonstrated that using these libraries can cause severe privacy concerns because they can collect users' location data in the background when the location permission is granted, which most users and developers are not aware of [9]. For analytics, we asked developers to store the location data locally and assume it would be used for analyzing users' location history. Since location history can disclose sensitive information about a person, it should be stored securely. We also required developers to collect a unique identifier for the user and to store it with the location data. The selection of a unique identifier affects whether the personal data is properly anonymized, and is also subject to the purpose limitation principle. The final result of the main task is presented in Figure 7.

For both the warm-up and main task, developers were free to choose any system APIs and parameters to use for collecting location data, generating/acquiring the unique identifier, and storing the data. We provided skeleton code that handled things not related to the focus of this study, such as updating the UI and initiating the network request to the weather web API[5], to save time and let participants focus on privacy.

6.3 Study Procedure

The study lasted for 1.5 to 2 hours. All participants came to our lab. We provided them with a laptop (Mac OS or Windows based on their preferences) and an Android phone for development and testing. Both Android Studio IDE and our Coconut plugin were installed. This study follows a between-subjects design. Both groups did the warm-up task without the plugin enabled, and only the experimental group had the plugin enabled for the plugin training process and the main task. Participants were compensated with a $75 Amazon gift card.

After introducing the study goals and logistics, both groups started with the warm-up task, which required them to obtain the current latitude-longitude location data and display it on the screen. Both groups had 40 minutes for the warm-up task. The warm-up task was designed to familiarize the participants with the development environment. It is also designed to compare the performance of the two groups when both in the control condition.

Next, the experimenter walked participants in the experimental group through a tutorial of our Coconut plugin, which used the same fictional use case in Section 5.1. The experimenter explained the semantics of the source and sink annotations, explained the meaning of each field of the annotations involved in this example, showed how to expand the field definition tooltip by hovering on the field name, and introduced the three types of feedback from Coconut: "missing-annotation error", "annotation-code inconsistency warning", and "privacy-concern warning". The tutorial was always available to the user during the study. This training process was to familiarize them with programming with Coconut, especially the concept of annotations.

Then both groups were asked to complete the main task of building a weather app. Only the participants in the experimental group had the plugin enabled. Both groups had one hour to work on the main task. We explicitly

[3] https://play.google.com/store/apps/details?id=com.weather.Weather

[4] https://play.google.com/store/apps/details?id=com.accuweather.android

[5] api.openweathermap.org

asked developers from both groups to imagine they were developing an app that would be used by real users and to take privacy into consideration during the development process.

After finishing all programming tasks, we asked participants to fill out an exit survey that had three sections. The first section asked them to write a privacy policy for the weather app they just built. When writing the privacy policy, both groups could refer to the code they just wrote. The experimental group could also use the PrivacyChecker to review their data practices, as documented in their annotations. The second section had some factual questions about the personal data practices of their app. The experimental group had a third section which solicited feedback about the Coconut plugin. The complete survey is provided in Appendix B.

Finally, the experimenter briefly interviewed the participants, asking about what considerations they had for privacy when developing the app, what they thought about the plugin (only for the experimental group), and the rationale behind some of their behaviors, such as ignoring web resources that contained more privacy-preserving options, ignoring particular privacy errors/warnings or not addressing them properly, and not correctly answering the factual questions in the exit survey. Notes were taken during the process.

The screens of the laptop and the Android phone were videotaped for later analysis, and no audio was recorded. Screen recordings were played back to developers during the exit interviews to help prompt their memories. During the study, developers could use any web resource. However, we required them to only use native APIs or Google Play Service APIs when obtaining personal data due to the scope of APIs currently supported by Coconut.

In the training process, we would answer any question they had regarding the plugin. During the programming tasks, we only answered clarification questions on task requirements, what they could do (such as which part of the code they could change), and what resources they could use. If the participant did not successfully finish the warm-up task within the specified time, we would offer hints on what they were doing wrong (for both groups) afterwards. In this way, we could observe more use of Coconut in the experimental group during the main task when ensuring both groups received the same amount of help.

During each session, one experimenter observed the participant and took notes. These notes include the actions the developer took, the search queries they used, and the thoughts they expressed verbally. After the study, the experimenter reviewed screen recordings to count the time spent on each task and subtask and document some coding and information foraging behaviors.

6.4 Privacy Policy Evaluation Methodology

To avoid potential bias, we invited two external judges for evaluating the privacy policies created by participants, as collected from the first section of the survey. One was a Ph.D. student studying usable privacy; the other was a master's student from a privacy engineering program and also had research experience in usable privacy.

We invited the judges to come to our lab for this evaluation session. During the session, all information was presented to the judge in one spreadsheet. Data from the two conditions were aggregated and stripped of any group information. We first introduced to them the expected personal data practices if the entire task was completed, and presented all valid responses of privacy policies with a description of the actual behavior of each app, since not all developers finished all task requirements. We created two different spreadsheets for these judges. The responses and app behavior description in these two spreadsheets were the same, with the order randomly shuffled to minimize any ordering bias.

We asked the judges to first skim through all of the responses and then came up with a set of criteria for judging these blurbs of privacy policies. We relied on their expertise and did not give detailed instructions on how to judge the quality of privacy policies. Judge 1 referred to the Fair Information Practices[6] as the main criteria, and also added two more requirements "providing true information" and "avoiding jargon" for evaluating the truthfulness and readability of privacy policies. Judge 2 did not refer to any specific material. He considered

[6] https://iapp.org/resources/article/fair-information-practices/

requirements including "providing accurate description", "providing useful information", "using plain language", "explaining all data being collected in the app", "specifying the purpose of using personal data", "providing users with ways to control data use", "informing users of data sharing practices". Overall, their standards for good privacy policies almost overlapped. We asked the judges to assign the rating for each response from 1 to 10, with 1 being the worst quality and 10 the best, based on the rubrics they developed. They were also asked to think aloud and explain the reason for each rating. The experimenter took notes in the meanwhile. The two judges conducted the evaluation separately and independently.

6.5 Research Questions

Nudging developers to adopt better privacy practices while programming can be challenging. For example, some developers may habitually ignore and fail to address warnings, and it is unclear whether developers can understand the underlying privacy concerns when filling the annotations. As such, we aimed to examine the overall effectiveness of Coconut using the following research questions:

- RQ1: Can Coconut help developers avoid more privacy violations?
- RQ2: Can Coconut help developers gain a better understanding of their app's personal data practices (RQ2.1) and write better privacy policies (RQ2.2)?
- RQ3: Do developers consider Coconut as useful and usable?

Previous studies on how developers think about and handle privacy mostly used retrospective inquiries. In contrast, our lab study allowed us to closely observe how developers deal with privacy issues while programming. Thus, we added a fourth research question:

- RQ4: What kind of challenges for privacy do developers face while programming and when they have been primed with the notion of privacy in advance of the main task?

7 RESULTS

7.1 Results of the Warm-up and Main Task Completion Are Similar Between Two Conditions

We first present the results about task completion and the time spent on each task. Table 5 presents an overview of how participants in the control and the experimental group performed in the warm-up task and the main task.

For those who completed the warm-up task, the average time for the control and the experimental group was 27 and 24 minutes respectively. For those who completed the main task, the average time for the control and the experimental group was 41 and 47 minutes respectively. The average time spent on annotations (including completing the annotation, reading the tooltips, attending to and resolving issues specified in the annotation) was 10 minutes. The number of people who completed the main task was the same, while developers in the experimental group spent slightly more time than those in the control group, possibly because of the extra cost of annotation and the time spent on thinking about and dealing with privacy considerations. We will further detail how developers perceive this cost in the following section as well as in the discussion section at the end.

The success rate of the warm-up task was similar across the two groups, which suggest the app development expertise was well balanced. We did not expect the success rate of the warm-up task would be low, since we pre-tested these tasks with a convenience sample of graduate students who had experience with Android programming and mobile privacy. Our results suggest that obtaining location data, which includes requesting location permission for the app, selecting the location API to use, and debugging, can be very challenging for average developers. Note that the main task had a higher completion rate, because parts of the main task overlapped with the warm-up task.

Table 5. Overview of lab study results. C1-C9 are the nine participants in the control group, and E1-E9 the experimental group using Coconut. Some developers did not complete all the tasks, denoted '–'. Practices better for privacy are in bold. The meaning of each column are as follows. "warm-up": time spent on warm-up task; "main": time spent on main weather app; "weather location": granularity of location for weather info (fine-grained: about 10m accuracy, coarse-grained: about 100m); "ad location": granularity of location collected by AdMob library; "storage": whether data stored locally is only accessible to this app (private) or also other apps (public); "UID": type of unique identifier in the weather app (GUID is custom globally unique IDs. Google Instance ID and GUID are user-resettable, app-level unique identifiers which are recommended. Android ID and Telephony ID are hardware identifiers in some versions of Android which are more privacy invasive. See more at [16])

ID	warm-up	main	weather location	ad location	storage	UID
C1	22'34"	32'25"	**coarse-grained**	fine-grained	**private**	**Google Instance ID**
C2	19'54"	38'47"	fine-grained	fine-grained	**private**	**GUID**
C3	38'27"	37'30"	fine-grained	fine-grained	public	Android ID
C4	–	53'19"	fine-grained	fine-grained	**private**	**GUID**
C5	–	–	**coarse-grained**	fine-grained	**private**	–
C6	–	–	fine-grained	fine-grained	–	pseudo UID
C7	–	–	fine-grained	fine-grained	**private**	Telephony ID
C8	–	–	fine-grained	–	–	–
C9	–	44'19"	fine-grained	fine-grained	–	**GUID**
E1	15'14"	60'	fine-grained	fine-grained	**private**	**GUID**
E2	–	–	**coarse-grained**	**coarse-grained**	–	–
E3	22'51"	30'34"	**coarse-grained**	fine-grained	**private**	**GUID**
E4	25'27"	51'14"	fine-grained	fine-grained	**private**	**GUID**
E5	–	–	fine-grained	–	–	–
E6	–	–	**coarse-grained**	–	–	–
E7	17'29"	34'29"	**coarse-grained**	**coarse-grained**	**private**	**Google Instance ID**
E8	38'58"	58'44"	**coarse-grained**	**coarse-grained**	**private**	**GUID**
E9	–	–	**coarse-grained**	**coarse-grained**	**private**	–

7.2 Coconut Can Help Developers Write More Privacy-Preserving Code

Overall, the comparison between the two groups shows an improvement for privacy when using Coconut. More developers using Coconut only collected coarse-grained location data for weather information, only shared coarse-grained location data with the AdMob library, used private storage for sensitive personal data, and selected the proper unique identifiers (Table 5).

We also examine why some developers using Coconut stayed with less optimal solutions, such as getting fine-grained location data when the purpose did not require this level of accuracy. We observed that some developers tended to ignore warnings. It could be that our choice of using different colors from normal warnings in Android Studio led to developers not recognizing the warning. Participant E4 also mentioned that he did not always trust the auto-generated annotation, and he would like to first manually verify them several times to develop trust. This issue of trust was discussed further as one of the limitations of our methodology in Section 8.4.

7.3 Coconut Can Help Developers Better Understand an App's Behavior

Table 6 presents the responses to the second section of our post-study survey, which tested developers' comprehension of their apps' behaviors. Some questions were not applicable for developers who did not finish all

Table 6. Percentage of correct answers to factual questions regarding apps' behavior. For the fraction after the percentage, the denominator indicates the number of people that answered this question, and the numerator indicates the number of people who answered it correctly. The last row ('Total') calculates the percentage by directly accumulating the number of total answers and total correct answers in each group.

	Control	Coconut
What is the granularity of the location data that you collected for getting the local weather info?	66.67% (6/9)	88.89% (8/9)
What is the granularity of the location data that you collected for the analytics purpose and were buffered on the phone?	75.00% (6/8)	100.0% (6/6)
How frequently does the app access location data for getting weather information?	77.78% (7/9)	75.00% (6/8)
How frequently does the app store location data for analytics purposes?	75.00% (6/8)	100.0% (6/6)
Are users able to reset the unique identifier that you choose?	100.0% (7/7)	80.00% (4/5)
What is the scope of use of the unique identifier that you chose?	71.43% (5/7)	100.0% (5/5)
Did you store the location data in a way that is only accessible to your app?	50.00% (3/6)	83.33% (5/6)
Did you store the unique identifier in a way that is only accessible to your app?	83.33% (5/6)	80.00% (4/5)
Where will the location data be transmitted to in the current version of the app? (Please list all places that you can think of)	11.11% (1/9)	88.89% (8/9)
Total	66.67% (46/69)	88.14% (52/59)

required tasks, so some of the total counts (the denominator in the fraction) was less than nine (size of each group).

For the overall comparison between these two groups, we only calculated descriptive statistics because of the missing values due to the uncompleted tasks. We accumulated the number of all valid answers and correct answers to calculate a overall correct answer rate. The overall rate of correct answers for the experimental group was 88.14%, higher than the control group (66.67%). The habit to ignore warnings and the insufficient trust in the auto-generated content may explain some incorrect answers of developers using Coconut.

We expected developers who finished the weather feature to answer that the data was transmitted to the server that hosts the weather API, and developers who integrated the banner ad to answer that the data was transmitted to Google's servers. However, most developers in the control group failed to identify these two facts, especially the one for the ad library. The follow-up interviews showed that the auto-generated annotations for this third-party library helped Coconut users learned about this data sharing practice. The interview results also suggested that although developers from both groups would know that the location was sent to a weather website if asked directly about it, it was harder to recall it from memory without the prompt of annotations.

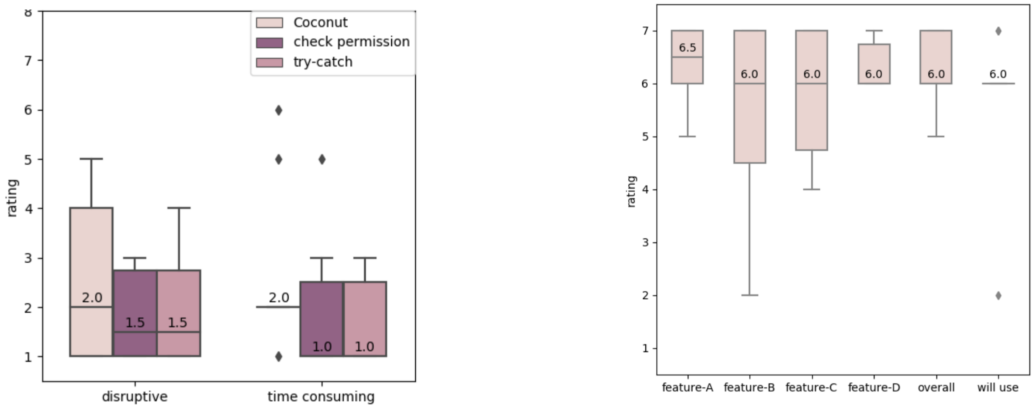
7.4 Coconut Can Help Developers Write Better Privacy Policies

We collected 18 privacy policies written for the weather app, and there was one invalid answer from E2, which just contains one vague phrase "highly recommended". This may be because that person did not know what privacy policy means. In the following result analysis, we discarded this outlier. We calculated an average of the scores from the two judges as the final store for each privacy policy.

The median of the aggregated final scores for the privacy policies of the experimental group and the control group were 5.875 and 2.750 respectively, and the former was significantly higher (Mann-Whitney U Test[7], U=6.0, p=0.002284). This suggests that using Coconut can help developers write better privacy policies.

Overall, the experimental group's privacy policies covered more points in the two external judges' criteria, including "purpose specification", "statement truthfulness", "giving opt-out options", "discussing security protections", "openness of implicit data practices (such as data storing and data egress)", and "readibility". Several aspects exhibited the most salient improvement, including "purpose specification", "statement truthfulness", and "openness of implicit data practices".

7.5 Coconut Is Perceived as Useful and Usable



(a) The results of how disruptive and time-consuming developers perceived about the new requirement of adding annotations (Coconut), and some existing requirements in Android Studio, including adding code to check whether corresponding permissions are granted before invoking personal data API (check permission) and adding try-catch blocks (try-catch) on a 7-point likert scale (1 for not disruptive/time-consuming at all, 7 for very disruptive and time-consuming).

(b) Perceived usefulness of Coconut and key features, as well as how much developers are willing to use Coconut, on a 7-point Likert scale (1=not useful/willing to use at all, 7=very useful/wiling to use). feature-A: generate pre-filled annotation skeleton; feature-B: navigate between the API call and annotation for the API call; feature-C: change annotation to the speculated value (when an inconsistency is detected); feature-D: change code to privacy-preserving options.

Fig. 6. Results of the plugin evaluation part of the survey: most developers found Coconut and features of Coconut very useful, and considered the cost of adding annotations as moderate and comparable to existing requirements. The median values of each group of results are marked in the box.

Since the requirement of adding annotations can incur more cost to the developers, we specifically asked them to evaluate if they perceived annotations as disruptive and time-consuming. We also established a baseline by asking them to rate other kinds of coding tasks, such as adding a permission check before using a protected API and adding try-catch blocks to handle exceptions. The results (presented in Figure 6a) suggest that developers perceived a moderate cost for adding annotations, which was comparable to the cost of these existing requirements. Besides, they may gradually feel more comfortable with it after using it as part of their regular development

[7] We use Mann-Whitney U Test because the ratings do not follow a normal distribution.

process for a while. When asked why they did not consider the annotation to be very disruptive, some participants ascribed that to the quickfix that generates annotation skeletons and the flexibility to defer filling the annotation.

Our survey results also suggest that developers considered many features of Coconut to be very useful, and had great interest in using it for their future projects (See Figure 6b). Participants explicitly mentioned how they benefited from Coconut in the interviews, including thinking more about privacy and recognizing privacy issues that they were previously unaware of because of the annotations (E1, E4, E5, E6, E7), gaining more knowledge of their app's behavior because of the auto-filled values in an annotation (E2, E3, E6, E7, E8), getting to know more options and making better design decisions due to the real-time feedback (E5, E6, E9), and streamlining the process of reviewing the personal data practices using the PrivacyChecker overview panel (E1, E4, E6, E9).

7.6 Challenges to Handling Personal Data Properly Observed While Programming.

Our lab study also provided us with an opportunity to observe how developers deal with privacy concerns while programming. We present some new challenges and how Coconut helped address them.

7.6.1 Challenges for Privacy in the Information Foraging Process. Similar to prior work that studied the implications of information sources on code security [3], we also observed challenges for privacy involved in the information foraging process. Developers used a wide range of information resources during the development process, including official tutorials, API documentation, Q&A sites such as Stack Overflow, tutorials from other websites, and code examples on GitHub. Many of the official tutorials were designed to take privacy into consideration. For example, the code example in "Get the last known location"[8] only requests the '`ACCESS_COARSE_LOCATION`' permission, so developers might not collect location data too fine-grained for their purposes if they just followed the default option. Similarly, the "Best practices for unique identifier" tutorial[9] gives an extensive overview of principles and recommendations for specific use cases to choose the proper unique identifier that protects users' privacy.

However, according to the observed developers' behavior and the exit interview results, developers did not make proper use of these resources to improve privacy, for four reasons.

First, some developers already had some experience with the APIs and a rough implementation plan. As such, they either searched very specific keywords that did not elicit these resources in the search result, or directly skipped them because they wanted to just see a code snippet that could refresh their memory of how to use a certain API (C2, C4, E8).

Second, some developers found or had the expectation that there would be too much information in such documentation, and instead favored Q&A sites that had more concrete examples to show *"how this API works in context, more than just API doc that explains what's the functionality of each API in detail"* (C7).

Third, some official tutorials did not cover all necessary steps, which caused trouble for novice developers. Although 14 participants opened the "Get the last known location" tutorial which recommended developers to use Google Play services location APIs, only 3 of them successfully implemented the required feature with these APIs. Most developers got stuck when setting up the Google Play Services in their project because there were no direct code examples in that tutorial to refer to. As a result, many developers sought other resources for help, and finally switched to using '`LocationManager`' APIs, which were easier to implement, but had fewer granularity options, and required the developers to manage the low-level location provider specification manually.

Fourth, developers did not always not fully grasp the recommendations and the rationale behind them. E3 admitted that although he saw the principle "Avoid using hardware identifiers" in the "Best practices for unique identifiers" guidelines, he did not pay much attention to it because he was lazy. Furthermore, we frequently observed during the coding process (and confirmed in the interview) that developers tended to directly jump

[8] https://developer.android.com/training/location/retrieve-current

[9] https://developer.android.com/training/articles/user-data-ids

to code examples or keywords related to code. For example, the same "Best practices for unique identifiers" document directly caused C6 to search how to acquire Android ID when it actually opposed using Android ID for his case, because he immediately noticed the keyword SSAID (Android ID) without paying attention to the context where the keyword appeared. The results further motivated the need for a tool that could actively check with the developer about their understanding of the behavior of the code and remind them of better options.

7.6.2 The Common Strategy of "Getting the App to Work First" May Get in the Way of Privacy. When interviewed about the rationale behind the decision-making process, some developers referred to a common theme: "doing things just for getting the app to work first" (C7, E2, E4). Additionally, we frequently saw study participants do fast testing of APIs, which can act in conflict with privacy requirements.

The location collection task was the most complicated one for most developers because it involved a run-time permission request and because there were multiple APIs (e.g. location APIs from '`LocationManager`', '`FusedLocationProviderApi`', and '`FusedLocationProviderClient`') and parameters to choose from. Furthermore, sometimes the sensor data may not be available or might update too slowly. When the app did not work as expected, developers with little experience in these APIs tended to try out each solution they found online or simply tweaked some parameters in the API call until it worked. Meanwhile, some decisions that may be less privacy friendly (such as using fine-grained location) were made because the app happened to work after changing the parameter that determined the granularity, although it was not the right solution (C7).

Our plugin can be helpful in mitigating this issue. For example, E8 in the experimental group also faced the problem of not getting the location update fast enough. The right solution is to change the parameter that controls the minimum update time interval. At first, she changed the parameter that controls the granularity of the location data, and happened to get the app to work temporarily. However, after this change, the parameter '`LocationDataType.FINE_GRAINED_LOCATION`' was highlighted in purple, which indicated a potential privacy concern. Then she kept investigating this problem and finally found the correct solution. This example shows that our annotations and real-time code inspection can help developers stay aware of any new privacy concerns in their code. Even if they decide to not deal with the concerns immediately, the warning can work as a reminder so they will not forget about it and handle it in the future.

8 DISCUSSION AND FUTURE WORK

8.1 The Value of Privacy Annotations

Our study results suggest that, although asking developers to do annotations for privacy involves some extra burden, developers do not perceive them as very disruptive or time-consuming, while also appreciating the value it provides. Here we discuss four reasons why these annotations improved privacy.

First, the process of completing the annotation may lead developers to think through the implications of their code on privacy. Although the task switching between normal development work and filling annotations could incur more cost in time and cognitive demands, many developers appreciated it because they think that they would be more willing to make modifications for privacy at the early stage of app development. This also resonates with P5's example in the problem finding interviews, which demonstrated that considering a security/privacy issue at late stages may cause extra cost that could have been avoided. Besides, the predefined fields of annotations could remind them to think about a wide range of privacy concerns. For example, some developers (E5, E6, E7) mentioned in the interview that the annotation helped them select the appropriate granularity of location based on the purpose. Although they may not have heard about the "purpose limitation" principle, they could still unconsciously apply that in their decision-making process by drawing connections between the purpose and the location granularity specified in the annotation.

Second, the privacy annotation may help developers be more aware of how the app uses personal data. This is not only because developers could reflect on their data practices when filling the fields of the annotation and correct misconceptions when seeing the real-time feedback highlighted on the annotation, but also because the annotation was kept as part of the code and summarized in one place. Many of our participants considered the overview panel to be potentially useful for large-scale project development and maintenance.

Third, developers may learn more options via the annotation and make better trade-off between privacy and other factors such as usability. More privacy-preserving alternatives can be presented in the real-time feedback, quickfixes, and the pre-defined values for some fields (such as the `LocationDataType` that provides several different granularities of location data). As a result, developers will make more informed design decisions and make a better trade-off between privacy and other factors.

Lastly, doing annotation may motivate self-taught developers to learn more about privacy. Many Android developers did not receive formal privacy training, and the situation becomes worse when they do not use tutorials that promote best privacy practices or get a wrong idea from it (discussed in Section 7.6.1). Annotation can be helpful because developers may use the keywords from the tooltips or the auto-generated content to search for related guidance. We observed developers searched the new keywords after attending to the annotation and also revisited a material that was opened but not paid enough attention to before.

We want to note that the current design and implementation may be more useful to developers who worked in small teams and have limited knowledge and bandwidth to deal with privacy requirements. The nudge to better privacy practices may not be as effective for developers who were subject to the decisions of other people in a large team, and we would like to probe into this problem in the corporate and collaborative context in the future.

8.2 Design Recommendations for Addressing Annotation Maintenance Issues

Because annotations do not directly affect functional code, it is possible for developers to specify annotations that were not consistent with the actual behavior of the code. In our current implementation, we tried to tackle this problem by inferring some field values using heuristics, continuously checking whether the value specified in the annotation was consistent with the inferred value and providing real-time feedback to developers in the IDE. Our study results suggest that this strategy is effective at assisting developers to maintain valid annotations.

However, we identified some other barriers to maintaining accurate annotations. The first was a lack of trust in automatically filled values in the annotation. Besides letting users gradually develop trust over time, we can also consider adopting some approaches studied in other intelligent systems, such as to provide explanations of how the value was speculated, or even just to provide the origin of the speculation [19]. The second barrier was that developers may understand certain terms differently. For example, developers expressed different interpretations of the '`WHILE_IN_USE`' option for the '`visibility`' field. Some people considered that if an activity was paused but not destroyed, it should still be in use, while other people thought an activity was only in use if the user interface is active. Both stronger automatic analysis techniques that check and enforce a unified semantic and advanced modeling of developers' perception of terms that describe the app status could be helpful.

8.3 Generalizability of Using Annotations to Enhance Privacy in Other Programming Languages

In this paper, we explore and demonstrate the potential of using customized annotation to help developers handle privacy specifically for Android apps written in Java. We also expect this idea to be applicable to other languages, because annotation/attribute is also used in other mainstream programming languages. For example, C# supports adding customizable attributes to various types of programming elements, which was similar to Java's annotations. As a result, privacy annotations can also be used in C# code by implementing support libraries.

8.4 Limitation of the Evaluation Methodology

Acquiring a large sample for lab studies that involve observing programming process has long been recognized as difficult. First, it is often hard to recruit software developers who usually have a much higher income than the study compensation [4]. Second, the length of such studies can be several hours due to the programming task, which can be hard to scale up. In our study, we recruited 18 developers for a between-subject study, which was similar to many prior studies with similar designs [10, 29]. However, the small sample size may still have implications on the confidence to draw conclusions from the study results. Besides, although we have recruited developers with diverse development backgrounds from multiple sources, half of our participants were still college student developers, which may not be representative enough of the entire population of Android developers.

Although we have tried to design tasks that contain realistic use scenarios and explicitly asked both groups to treat it as an app with real users, the lab study method still has its inherent limitations. Because of the constraint on the time to finish the tasks, some of them may start building the app with less careful plans and exhibit more fast prototyping behaviors than usual. Besides, developers in both groups may pay more attention to privacy than usual, because they were primed by the notion of privacy before working on the main task. Regarding the use of Coconut, developers who used this plugin for the study may react more actively and positively towards the plugin's suggestions and warnings. On the other hand, their performance may also be negatively impacted by the unfamiliarity and lack of trust of the tool because of the short-term exposure in the lab session.

Some potential benefits of Coconut need to be examined with a larger and less controlled study. For example, when developers work on a large-scale project, they may find the annotations and the summary panel more useful for maintaining a correct understanding of the apps' behavior; when they are not subject to the study task requirements, the process of doing annotations and the feedback conveyed through the annotations may trigger more radical changes in the design of the app to improve privacy.

8.5 Future Work

In the future, we would like to explore using annotations to improve privacy in two directions. The first is to enhance the current tool. For example, we have considered integrating taint analysis or other more sophisticated static analysis techniques to further reduce the burden on developers. The second is to use annotations to benefit end users and auditors. Specifically, we would like to help developers create privacy policies with annotations and design a better privacy notice interface that can be automatically generated based on these annotations.

9 CONCLUSION

We presented the design, implementation, and evaluation results of Coconut, an Android Studio plugin for building privacy-preserving apps. Study results indicate that our tool can help developers write privacy-preserving code, gain further knowledge of the apps' behavior, and write better privacy notices. We demonstrated the potential of using Java Annotation to improve privacy, which helps developers think about privacy from a broad range of perspectives while programming, organize and maintain information about how personal data is used, make better trade-offs between privacy and other factors such as usability, and provide developers with a starting point to learn more about privacy. Our work also entails some new findings regarding what challenges developers are facing via semi-structured interviews and in-situ observation of the programming process.

A APPENDIX: INTERVIEW SCRIPT

A.1 Android Development and Privacy Training Background

- Since we are going to talk about the apps that you have developed, do you mind if you first briefly introduce your background in Android development?
  - When did you start to learn Android development?
  - When did you start to learn Android development?
  - What Android apps have you developed before?
  - What Android version have you worked on?

- What's your understanding of protecting users' privacy?
- Did you know FTC guidelines on how to make privacy-preserving app?
- Have you received training for privacy?
- Did you have any questions in privacy? Who did you consult when you have questions in privacy?
- Does your app have a privacy policy?

A.2 General Questions About the Three Latest Apps

- What were these apps built for?
- Was it developed by a team or just yourself?
- If it's by a team then:
  - how did you divide your work and collaborate? What's your responsibilities?
  - Is there anyone working on determining what feature in your app and personal data is needed to implement them?
  - Do you design what the feature the app should have and what personal data you might need to collect prior to the development process?
- If it's developed independently then:
  - Do you design what the feature the app should have and what personal data you might need to collect prior to the development process?

A.3 Personal Data Use in the Three Latest Apps

- For PII data needs to be actively provided by the user (A list of different types of PII data is provided):
  - Did any of these apps use this data?
  - What's the purpose for that?
  - Did you send them out of the phone?
  - Did you store them?
  - Do you think your users are clear about what PII are used and how they are used?
- For Unique identifiers (A list of different unique identifiers is provided):
  - Which of the following unique identifiers do you know?
  - Do you know the "Best practices for unique identifier"?
  - Did any of these apps use the following information?
  - Do you know how to reset it?
  - Do you know it will be shared by what apps? What's the purpose for that? How did you choose the unique identifier?
  - Did you send them out of the phone?
  - Did you store them?
  - Do you think your users are clear about what UID are used and how they are used?
- For Personal data protected by permission (A list of data of this kind is provided)
  - Did any of these apps use the following information?
  - What's the purpose for that?
  - Did you send them out of the phone?
  - Did you store them?
  - How frequently did you access this information?
  - In foreground or background?
  - What are the Android versions that these apps are targeting at?

        - ∗ If M(Marshmallow, 6): When do you request for the permission during runtime? Do you tell the user about the purpose of the permission and how to use the data
        - ∗ If not: Do you know the new permission system in M? What do you think of the runtime request for permission approval? What if comparing it with show a list of required permissions before install the app?
    - Do you think your users are clear about what data are used and how they are used?
    - Did you convey the data collection practices to your user?
    - Have you met any technical issues when you implemented the features that required personal data?
- Did you use any advertising or analytics third party library such as admob, flurry etc.?
    - If yes, what libraries did you use? Why did you pick them? Do you know what are the data collection practices of the libraries that you used? How did you know that? Did you find a way to convey this data collection practices and the choices they have to your users? What if that will make you earn less money?
    - If no, why? Have you considered monetizing your app?
    - Has any user contacted you for their concerns with their privacy?

B APPENDIX: LAB STUDY SURVEY QUESTIONNAIRE

B.1 Section 1: Writing a Privacy Policy for the App

- How would you describe the personal data practices of your weather app to your users? Imagine this is a paragraph in the privacy policy of your weather app.

B.2 Section 2: Factual Questions About the App Behavior

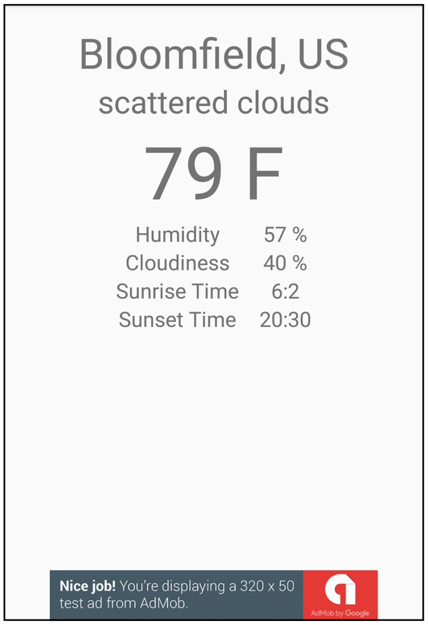
- What is the granularity of the location data that you collected for getting the local weather info? (Select all that apply)
    - Fine-grained (10 meter accuracy under the best situations).
    - Coarse-grained (>100 meter accuracy).
    - Block-level (100 meter accuracy).
    - City-level (10km accuracy).
    - I'm not sure.
- What is the granularity of the location data that you collected for the analytics purpose and were buffered on the phone? (Select all that apply)
    - Fine-grained (10 meter accuracy under the best situations).
    - Coarse-grained (>100 meter accuracy).
    - Block-level (100 meter accuracy).
    - City-level (10km accuracy).
    - I'm not sure.
- How frequently does the app access location data for getting weather information?
- How frequently does the app access location data for analytics purposes?
- Are users able to reset the unique identifier that you choose?
    - Yes, they can reset it.
    - No, they can't reset it.
    - I'm not sure.
- What is the scope of use of the unique identifier that you chose?
    - It's shared by all other apps on the same device.
    - Only my app use this identifier to identify the user.
    - I'm not sure.

- Did you store the location data in a way that is only accessible to your app?
  - Yes.
  - No.
  - I'm not sure.
- Did you store the unique identifier in a way that is only accessible to your app?
  - Yes.
  - No.
  - I'm not sure.
- Where will the location data be transmitted to in the current version of the app? (Please list all places that you can think of)

B.3 Section 3: Feedback on Coconut

(Only for the experimental group; Screenshot examples are omitted; All questions required a subjective rating on a 1 to 7 likert scale, in which 1 means strongly disagree and 7 means strongly agree)

- I felt that having to use annotations was disruptive.
- I felt that adding annotations was time-consuming.
- I felt that this error (checking permission) message was disruptive. Skip this question if you don't know what it means.
- I felt that this error (checking permission) message was time-consuming. Skip this question if you don't know what it means.
- I felt that this error (adding try-catch block) message was disruptive. Skip this question if you don't know what it means.
- I felt that this error (adding try-catch block) message was time-consuming. Skip this question if you don't know what it means.
- I found the "Add LocationAnnotation annotation" type of quickfixes useful. Skip this question if you don't know what this quickfix is.
- I found the "Navigate to the annotation" type of quickfixes useful. Skip this question if you don't know what this quickfix is.
- I found the "Use the speculated value for this field" quickfix useful. Skip this question if you don't know what this quickfix is.
- I found the "Use Google Instance ID instead" type of quickfixes useful. Skip this question if you don't know what this quickfix is.
- I found this IDE plugin useful.
- Please describe in one sentence what you think the purple highlight indicates and what you would do if you see that during programming. Skip this question if you never saw this before.
- Please describe in one sentence what you think the red highlight indicates and what you would do if you see that during programming. Skip this question if you never saw this before.
- Please describe in one sentence what you think the red squiggly underline indicates and what you would do if you see that during programming.
- I would like to use this IDE plugin in the future for my own projects.
- Do you have suggestions for other features for this plugin to help developers with privacy when developing apps?



Bloomfield, US

scattered clouds

79 F

Humidity	57 %
Cloudiness	40 %
Sunrise Time	6:2
Sunset Time	20:30

Nice job! You're displaying a 320 x 50 test ad from AdMob.

Fig. 7. An example of the weather app in the main task.

C APPENDIX: LAB STUDY PROGRAMMING TASK (FIG. 7)

D APPENDIX: IMPLEMENTATION DETAILS (TABLE 7 AND 8)

ACKNOWLEDGMENTS

This material is 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. We would also like to thank the following grants for supporting this work: National Science Foundation grants SaTC-1801472, TWC-1564009 as well as multiple Google Faculty Research Awards on mobile and IoT privacy.

The authors would like to acknowledge Yuanchun Li for his feedback on the early prototype and the study design. In addition, the authors would like to thank Laura Dabbish, Jackie Yang, Haojian Jin, and Michael Liu for their comments and help on the drafts of this paper. The authors also want to thank Lauren Hardwig for helping proofread and edit the paper. Finally, the authors would also like to thank the anonymous reviewers for their constructive feedback.

Table 7. Annotations included in the current proof-of-concept prototype for the lab study: LocationDataType (location representation type such as latitude-longitude, altitude, speed, etc.), Visibility (data accessed when app is in foreground or background), UIDType, UIDScope, UIDResettability, AccessControlOption are pre-defined enum classes defined in the privacy annotation library.

Annotation Type	Annotation Definition
Source Annotation	
Location	• LocationDataType dataType • Visibility visibility • LocationPurpose purpose • String purposeDescription • String frequency
UniqueIdentifier	• UIDType uidType • UIDScope scope • UIDResettability resettability • UIDPurpose purpose • String purposeDescription
UndefinedPersonalDataType	• String dataType • Visibility visibility • String purposeDescription • String frequency
NotPersonalData	N/A
Sink Annotation	
Network	• String retentionTime • String destination • String purposeDescription • boolean encryptedInTransmission
Storage	• String retentionTime • String purposeDescription • AccessControlOption accessControl
Third-party Lib Annotation	
AdmobAnnotation	isLocationDataEnabledInAppSettings

Table 8. APIs being tracked, and the corresponding annotation(s) required for the API in the current proof-of-concept prototype for the lab study. Several package names are not presented in the complete form due to space limit. The complete forms are: android.location.LocationManager, com.google.android.gms.location.FusedLocationProviderClient, com.google.android.gms.location.FusedLocationProviderApi, android.telephony.TelephonyManager, android.bluetooth.BluetoothAdapter.

package containing the API	API name	Required annotation(s)
Data Source API		
LocationManager	getLastKnownLocation, requestLocationUpdates, requestSingleUpdate	Location annotation
FusedLocationProviderClient	getLastLocation, requestLocationUpdates	
FusedLocationProviderApi	getLastLocation, requestLocationUpdates	
android.provider.Settings	Secure.getString (only track when corresponding parameter has the value "android_id")	UniqueIdentifier annotation
java.util.UUID	randomUUID	
TelephonyManager	getDeviceId	
TelephonyManager	getImei	
TelephonyManager	getMeid	
TelephonyManager	getLine1Number	
android.net.wifi.WifiInfo	getMacAddress	
BluetoothAdapter	getAddress	
com.google.android.gms.iid	InstanceID.getId (Google Instance ID)	
com.google.android.gms.ads	identifier.AdvertisingIdClient.Info.getId (Google Advertising ID)	
Data Sink API		
SharedPreferences.Editor	putBoolean, putFloat, putInt, putLong, putString, putStringSet	Storage annotation, corresponding source annotation(s) or NotPersonalData annotation
java.io	OutputStream.write, Writer.write, Writer.append...	
com.android.volley	RequestQueue.add	Network annotation, corresponding source annotation(s) or NotPersonalData annotation
Third-party Lib API		
com.google.android.gms.ads	AdView.loadAd	Admob annotation corresponding source and annotation(s) for data collected by the third party at this point. Annotations for this API are automatically generated.

REFERENCES

[1] 2017. Improve Your Code With Lint. Available at https://developer.android.com/studio/write/lint.html (2017/05/14). (2017).

[2] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp.2017.52

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp.2016.25

[4] Yasemin Acar, Sascha Fahl, and Michelle L. Mazurek. 2016. You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. In *2016 IEEE Cybersecurity Development (SecDev)*. IEEE. https://doi.org/10.1109/secdev.2016.013

[5] Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy. 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

[6] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2013. FlowDroid. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '14*. ACM Press. https://doi.org/10.1145/2594291.2594299

[7] GSM Association. 2012. Mobile Privacy Principles, promoting consumer privacy in the mobile ecosystem. Available at http://www.gsma.com/publicpolicy/wp-content/uploads/2016/10/GSMA-Privacy-Principles.pdf (2017/05/14). (2012).

[8] 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

[9] 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

[10] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. 2010. Example-centric programming. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press. https://doi.org/10.1145/1753326.1753402

[11] W. Cheng, Qin Zhao, Bei Yu, and S. Hiroshige. 2006. TaintTrace: Efficient Flow Tracing with Dynamic Binary Rewriting. In *11th IEEE Symposium on Computers and Communications (ISCC'06)*. IEEE. https://doi.org/10.1109/iscc.2006.158

[12] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. 2011. Analyzing inter-application communication in Android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services - MobiSys '11*. ACM Press. https://doi.org/10.1145/1999995.2000018

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

[14] Federal Trade Commission et al. 2013. Mobile privacy disclosures: Building trust through transparency. *USA: Federal Trade Commission* (2013).

[15] Android Official Documentation. 2017. Best Practices for Permissions and Identifiers. Available at https://developer.android.com/training/best-permissions-ids.html (2017/05/14). (2017).

[16] Android Official Documentation. 2017. Best Practices for Unique Identifiers. Available at https://developer.android.com/training/articles/user-data-ids.html (2017/05/14). (2017).

[17] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2014. TaintDroid. *ACM Transactions on Computer Systems* 32, 2 (jun 2014), 1–29. https://doi.org/10.1145/2619091

[18] Felix Fischer, Konstantin Bottinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. 2017. Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. https://doi.org/10.1109/sp.2017.31

[19] Alyssa Glass, Deborah L. McGuinness, and Michael Wolverton. 2008. Toward establishing trust in adaptive agents. In *Proceedings of the 13th international conference on Intelligent user interfaces - IUI '08*. ACM Press. https://doi.org/10.1145/1378773.1378804

[20] Michael I. Gordon, Deokhwan Kim, Jeff Perkins, Limei Gilham, Nguyen Nguyen, and Martin Rinard. 2015. Information-Flow Analysis of Android Applications in DroidSafe. In *Proceedings 2015 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2015.23089

[21] 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

[22] Kamala D. Harris. 2013. Privacy on the go, recommendations for the mobile ecosystem. Available at https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/privacy_on_the_go.pdf (2017/05/14). (2013).

[23] David Hovemeyer and William Pugh. 2004. Finding bugs is easy. *ACM SIGPLAN Notices* 39, 12 (dec 2004), 92. https://doi.org/10.1145/1052883.1052895

[24] Luigi Lo Iacono and Peter Leo Gorski. 2017. I Do and I Understand. Not Yet True for Security APIs. So Sad. In *Proceedings 2nd European Workshop on Usable Security*. Internet Society. https://doi.org/10.14722/eurousec.2017.23015

[25] Shubham Jain and Janne Lindqvist. 2014. Should I Protect You? Understanding Developers' Behavior to Privacy-Preserving APIs. In *Proceedings 2014 Workshop on Usable Security*. Internet Society. https://doi.org/10.14722/usec.2014.23045

[26] Haojian Jin, Minyi Liu, Kevan Dodhia, Yuanchun Li, Gaurav Srivastava, Matthew Fredrikson, Yuvraj Agarwal, and Jason I. Hong. 2018. "Why are they collecting my data?": Inferring the Purposes of Network Traffic in Mobile Apps. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* (2018).

[27] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE. https://doi.org/10.1109/icse.2013.6606613

[28] G. Karjoth and M. Schunter. [n. d.]. A privacy policy model for enterprises. In *Proceedings 15th IEEE Computer Security Foundations Workshop. CSFW-15*. IEEE Comput. Soc. https://doi.org/10.1109/csfw.2002.1021821

[29] Andrew J. Ko and Brad A. Myers. 2004. Designing the whyline. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. ACM Press. https://doi.org/10.1145/985692.985712

[30] Li Li, Alexandre Bartel, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. 2014. I know what leaked in your pocket: uncovering privacy leaks on Android Apps with Static Taint Analysis. *arXiv preprint arXiv:1404.7431* (2014).

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

[32] Kai-Uwe Loser and Martin Degeling. 2014. Security and Privacy as Hygiene Factors of Developer Behavior in Small and Agile Teams. In *IFIP Advances in Information and Communication Technology*. Springer Berlin Heidelberg, 255–265. https://doi.org/10.1007/978-3-662-44208-1_21

[33] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. 2012. CHEX. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*. ACM Press. https://doi.org/10.1145/2382196.2382223

[34] Stephen McCamant and Michael D Ernst. 2006. Quantitative information-flow tracking for C and related languages. (2006).

[35] James Newsome and Dawn Song. 2005. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. (2005).

[36] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security - CCS '17*. ACM Press. https://doi.org/10.1145/3133956.3133977

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

[38] Future of Privacy Forum and the Center for Democracy & Technology. 2012. Best Practices for Mobile Application Developers. Available at https://www.cdt.org/files/pdfs/Best-Practices-Mobile-App-Developers.pdf (2017/05/14). (2012).

[39] Office of the Australian Information Commissioner. 2014. Mobile privacy: a better practice guide for mobile app developers. Available at https://www.oaic.gov.au/agencies-and-organisations/guides/guide-for-mobile-app-developers (2017/05/14). (2014).

[40] Office of the Privacy Commissioner of Canada. 2012. Seizing Opportunity: Good Privacy Practices for Developing Mobile Apps. Available at https://www.priv.gc.ca/en/privacy-topics/technology-and-privacy/mobile-devices-and-apps/gd_app_201210/ (2017/05/14). (2012).

[41] Information Commissioner's Office. 2013. Privacy in mobile apps, guidance for app developers. Available at https://ico.org.uk/media/for-organisations/documents/1596/privacy-in-mobile-apps-dp-guidance.pdf (2017/05/14). (2013).

[42] Article 29 Data Protection Working Party. 2013. Opinion 02/2013 on apps on smart devices. Available at http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2013/wp202_en.pdf (2017/05/14). (2013).

[43] Feng Qin, Cheng Wang, Zhenmin Li, Ho seop Kim, Yuanyuan Zhou, and Youfeng Wu. 2006. LIFT: A Low-Overhead Practical Information Flow Tracking System for Detecting Security Attacks. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE. https://doi.org/10.1109/micro.2006.29

[44] Caitlin Sadowski, Jeffrey van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE. https://doi.org/10.1109/icse.2015.76

[45] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K. Rajamani, Janice Tsai, and Jeannette M. Wing. 2014. Bootstrapping Privacy Compliance in Big Data Systems. In *2014 IEEE Symposium on Security and Privacy*. IEEE. https://doi.org/10.1109/sp.2014.28

[46] 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 - ICSE 2014*. ACM Press. https://doi.org/10.1145/2568225.2568244

[47] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press. https://doi.org/10.1145/2786805.2786812

[48] Sooel Son, Kathryn S. McKinley, and Vitaly Shmatikov. 2011. RoleCast. *ACM SIGPLAN Notices* 46, 10 (oct 2011), 1069. https://doi.org/10.1145/2076021.2048146

[49] National Telecommunications and Information Administration. 2013. Short Form Notice Code of Conduct to Promote Transparency in Mobile App Practices. Available at https://www.ntia.doc.gov/files/ntia/publications/july_25_code_draft.pdf (2017/05/14). (2013).

[50] Tyler W. Thomas, Madiha Tabassum, Bill Chu, and Heather Lipford. 2018. Security During Application Development. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press. https://doi.org/10.1145/3173574.3173836

[51] Chamila Wijayarathna, Nalin A. G. Arachchilage, and Jill Slay. 2017. A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs. In *Human Aspects of Information Security, Privacy and Trust*. Springer International Publishing, 160–173. https://doi.org/10.1007/978-3-319-58460-7_11

[52] Jim Witschey, Olga Zielinska, Allaire Welk, Emerson Murphy-Hill, Chris Mayhorn, and Thomas Zimmermann. 2015. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. ACM Press. https://doi.org/10.1145/2786805.2786816

[53] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social influences on secure development tool adoption. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing - CSCW '14*. ACM Press. https://doi.org/10.1145/2531602.2531722

[54] Jing Xie, Heather Lipford, and Bei-Tseng Chu. 2012. Evaluating interactive support for secure programming. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM Press. https://doi.org/10.1145/2207676.2208665

[55] Jing Xie, H. R. Lipford, and Bill Chu. 2011. Why do programmers make security errors?. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. https://doi.org/10.1109/vlhcc.2011.6070393

[56] Jean Yang, Travis Hance, Thomas H. Austin, Armando Solar-Lezama, Cormac Flanagan, and Stephen Chong. 2016. Precise, dynamic information flow for database-backed applications. *ACM SIGPLAN Notices* 51, 6 (jun 2016), 631–647. https://doi.org/10.1145/2980983.2908098

[57] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. 2012. A language for automatically enforcing privacy policies. *ACM SIGPLAN Notices* 47, 1 (jan 2012), 85. https://doi.org/10.1145/2103621.2103669

[58] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. 2015. Who knows what about me? A survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science* 30 (2015).

Received May 2018; revised August 2018; accepted October 2018