Should I Protect You? Understanding Developers' Behavior to Privacy-Preserving APIs

Shubham Jain and Janne Lindqvist
Rutgers University
Email: {shubhamj, janne}@winlab.rutgers.edu

Abstract—**There have been many proposals and developments to improve smartphone users' location privacy with respect to mobile applications. These include user-centric application permission models and disclosures. However, little attention has been paid to how application developers could build privacy-preserving apps. In this paper, we present a laboratory study (N=25) to understand developers' behavior to enhanced APIs, which are a privacy-preserving modification of the existing Android Location API. In contrast to the existing methods, the studied API facilitates acquiring coarse location information without accessing the geocoordinates. Our results indicate that by offering a redesigned API, programmers can be nudged into making choices with their programming that help to preserve privacy of their users.**

I. Introduction

The evolution of smartphones has also led to an increasing popularity of location-based apps. In addition to navigation and maps apps, several popular applications also use location awareness to help people find their way around. The apps use users' location to supply information ranging from the stores and restaurants nearby, to the transit system and to the weather conditions in their region. The emergence of location based services has also brought to light its most criticized facet, the loss of privacy. With the freedom that developers can exercise when implementing location-based services, their responsibility has also grown. Not only do millions of users rely on the features of their applications, they also entrust them with their personal information. A breach in this trust can hamper the long term growth of context aware ubiquitous systems. The Location Privacy Protection Act of 2011 [1] states that a special report by the Department of Justice revealed that based on 2006 data, approximately 26,000 people were victims of GPS stalking annually. However, existing platforms offer little support for developers to build privacy-preserving apps.

Smartphones obtain users' locations by acquiring geographical coordinates represented by the latitude and longitude. However, different applications require the use of different levels of location information. The existing localization techniques provide access to a user's precise location, by means of geographical coordinates. While this may be useful for the functioning of some apps, it threatens the users' privacy through others. Although a navigation application may need your precise location in terms of latitude and longitude, an application providing eCoupons to nearby stores can work efficiently even with less precise location information, such as street address or city name.

To the best of our knowledge, researchers have not yet studied how developers would behave if presented with privacy-preserving APIs as an alternative with baseline localization APIs. To address this problem, we decided to conduct laboratory experiments with an existing smartphone platform (Android) localization API and a modified privacy-preserving version of it. Our aim was to understand how developers deal with privacy issues when empowered with the choice of using privacy-safe options. In this work, for practical reasons, we focus on the redesign of the Android location APIs and developers' behavior using them. This redesign is influenced by privacy-by-design principles [2], such as minimizing the use of sensitive information and purpose-binding of information collection. They enable developers to gather location information of coarser granularity, thus avoiding any unnecessary violations of users' privacy. They also work on the same lines as the existing APIs, with changes that present the developer with a multitude of options to safeguard the user's precise location. To understand if developers can be nudged into making privacy respecting choices, we conducted a controlled laboratory study (N=25). The participants of the study were offered a documentation consisting of the baseline Android and our proposed API and asked to code for a set of programming exercises. Our aim was to understand how developers make their choices when presented with more options than the current location API system. The study also gives us an insight into the basic usability of the redesigned API.

The contributions of this work are as follows:

- Privacy-preserving redesign of the Android location APIs.
- A randomized controlled study for qualitative analysis of developers' behavior to using privacy-preserving APIs.
- Results which demonstrate that developers can be nudged into programming choices that help to preserve privacy of their users.
- The results also show that participants find the redesigned API easier to use.

USEC '14, 23 February 2014, San Diego, CA, USA
Copyright 2014 Internet Society, ISBN 1-891562-37-1
http://dx.doi.org/10.14722/usec.2014.23045

II. Related Work

Researchers have studied the principles and practices of good API design [3]. While a good API design should be easy to learn and use, it should also be hard to misuse and powerful enough to satisfy requirements. Clarke [4] talks about the usability study of an API. Such a study involves providing participants with a documentation and asking them to perform a set of tasks. Our study is similar to this usability study, where we use it for understanding developers. Robillard et al. [5] discuss the obstacles in learning APIs. They perform a study to learn these obstacles in terms of API documentation, code examples and matching API with scenario. This is similar to our procedure of a carefully documented API set and the impact of example code and similar APIs.

There has recently been considerable work on smartphone security and privacy. Becher et al. [6] give an overview of mobile phone security history and developments, Anderson et al. have studied different application markets and installation mechanisms [7], and several authors have written position papers about application markets (e.g. [8]). Considerable effort has been spent on understanding e.g. Android security and permissions [9], and hardening Android security model [10]. Recently, dynamic taint analysis has been implemented on Android platform [11]. Approaches such as AppFence [12], MockDroid [13], and TISSA [14] block data leakage to network by faking the capabilities of the phone, so that potentially sensitive data cannot be retrieved. However, none of these works focus on application developers and how they could build privacy-preserving applications.

Several authors have discussed privacy design principles for interactive and ubiquitous systems. These include e.g. Langheinrich [2], Lederer et al. [15], and Spiekermann and Cranor [16]. On the lines of aiding developers, Felt et al. [17] propose a set of guidelines for developers to determine the appropriate permission-granting mechanism. One of the main principles in data collection and processing is that it uses only the amount of information that is needed.

Bravo-Lillo et al. [18] and Wash [19] have worked on understanding people's mental models related to security. More directly related to our work, Hong et al. [20] implemented a framework to help application developers to build privacy-sensitive applications such as enhanced instant messaging clients. However, that framework's focus is on enforcing user-set restrictions on sending data to a server, not on nudging developers to make better applications.

Amini et al. [21] continued this line of work by implementing an approach for pre-fetching and caching data of location-based applications, however, they did not present an evaluation of any APIs. Tam et al. [22] studied different designs for Facebook disclosures during installation time, and Howell et al. [12] proposed a new model for sensor access: "Show Widget, Allow After Input and Delay", in which access to sensors such as cameras would be allowed only when user has actively given input to the system (e.g. pressing keys) and after waiting period. There is also recent work on using crowdsourcing for understanding users' mental models from a privacy perspective by Lin et al. [23].

III. API Design

Android allows building location sensing capabilities in applications through the Location Services API framework. The classes in *android.location* package allow an application to access these location services. The *LocationManager* system service is the main component of this framework [24]. It provides the APIs to determine a device's location.

A. Current Android Location API

Developers can indicate that they want to receive periodic updates for a user's geographical location from the *LocationManager* by calling the method *requestLocationUpdates()*. This method accepts the following parameters in order - a type of location provider, the minimum time interval between notifications, minimum change in distance between notifications and a *LocationListener*. The *LocationListener* in the application must implement callback methods. The *onLocationChanged()* method from the *LocationListener* implementation is called by the *LocationManager* when the user's location has changed and a *Location* object is supplied to it. The *Location* class in Android is a data class that represents a geographic location [24]. All locations generated by the *LocationManager* are guaranteed to have a valid triple: the latitude, the longitude and a timestamp.

Geographical coordinates are the only location attributes an application obtains directly from the Android location services. A developer interested in obtaining some other location information, such as city name or the postal code, must use reverse geocoding. *Reverse geocoding* is the process of transforming a geographical coordinate (latitude, longitude) into an address.The *Geocoder* class handles reverse geocoding [24]. The method *getFromLocation()* in the *Geocoder* class returns an array of *Addresses* that are known to describe the area immediately surrounding the given latitude and longitude. This method takes the following parameters in order - latitude, longitude, maxResults. It returns a list of *Address* objects. An object of the *Address* class represents a set of Strings describing a location. This class provides the methods to obtain various location components, such as city, postal code etc.

Location based Android applications require Coarse or Fine location permissions. The ACCESS_COARSE_LOCATION permission allows an application to use WiFi/cell-network based localization instead of GPS (ACCESS_FINE_LOCATION), and the exact location will be obfuscated to a coarse level of accuracy. When compared to the proposed API, ACCESS_COARSE_LOCATION localization is more accurate than e.g. zipcode or city level localization.

B. Privacy Problems with Location API

More and more developers are getting easy access to a large amount of potentially sensitive information. Several applications are known to misuse this information [11]. Protecting a user's location privacy needs us to be able to distinguish between necessary and sufficient access to such information. Since we cannot control all access to it, we could offer developers the choice to use only what is sufficient for the effective operation of the application. A coarser granularity location information, such as postal code, might serve the purpose for a subset of existing applications. The current APIs contribute to undermining all suggested privacy protecting methods, by compelling developers to obtain users precise geographical locations before they can acquire any other location component such as the street, city or postal code by reverse geocoding. To use an application that requires a coarser granularity of location information, users must compromise their exact location if they wish to use those apps. As discussed by Zang et al. [25], people can be identified using their location traces. Many applications that do not need fine-grained location attributes are also allowed access to them, owing to the lack of a privacy-preserving system in place.

C. Our Proposed APIs

To prevent unnecessary exposure of users' location data, we propose to redesign the existing APIs to form a privacy-preserving system. Privacy preserving architectures and APIs have been discussed earlier [21], [26], [27], but none of them have investigated how developers would actually use the frameworks. The redesigned API aims to encourage a privacy-centric approach by developers and minimize the transmission and use of sensitive user location information. In contrast to the existing methods, the proposed API will facilitate acquiring coarse location information without accessing the geocoordinates.

The proposed methods are a carefully designed privacy-preserving modification of the existing APIs. We propose diverse methods to request updates for coarser location components in contrast to the *requestLocationUpdates()* method, that updates only for geo location. These could request for street, city, postal code by methods such as *requestCityUpdates(), requestStreetUpdates(), and requestPostalCodeUpdates()* respectively. In this case, the *LocationListener* is notified only when there are updates in coarse granularity location information, such as street, city or postal code. A modified *LocationManager* class is shown in Table I.

Public Methods	
void	*requestLocationUpdates()* Register for updates in geographical coordinates of a location
void	*requestStreetUpdates()*
void	*requestBuildingUpdates()*
void	*requestCityUpdates()*
void	*requestPostalCodeUpdates()*
void	*requestCountyUpdates()*
void	*requestStateUpdates()*
void	*requestCountryUpdates()*

TABLE I: Modified Class LocationManager: This table shows a list of methods used to register for location updates. They have been modified as part of our proposed API, so that updates for coarse granularity location information can also be requested. All methods take the following parameters in order- (String provider, long minTime, float minDistance, *LocationListener* listener). The parameters and description is not shown here for all methods due to space constraints.

Correspondingly, the *LocationListener* will listen for changes in street or city by using *onStreetChanged()* or *onCityChanged()* respectively. Such APIs neither listen nor update for changes in location that are finer than the specified location component. Table II provides a list of the public methods modified for the class *LocationListener*.

Public Methods	
abstract void	*onLocationChanged()* Called when the geo coordinates have changed
abstract void	*onStreetChanged()*
abstract void	*onBuildingChanged()*
abstract void	*onCityChanged()*
abstract void	*onPostalCodeChanged()*
abstract void	*onCountyChanged()*
abstract void	*onStateChanged()*
abstract void	*onCountryChanged()*

TABLE II: Modified Class LocationListener: This table shows a list of the callback methods called every time the user's location has changed. These methods have been modified as part of our proposed API, so that they can be called when user's coarse granularity location changes. All methods take the following parameter- (*Location* location). The parameter and description is not shown here for all methods due to space constraints.

Once the specific onChanged method is called, the related location component information is extracted from the *Location* object. The modified APIs provide get methods for several location components. These vary in granularity of information. Thus, an app concerned with finding the weather in users' location, can simply acquire the name of the users' city or the postal code using *getCity()* or *getPostalCode()*. In fact, other apps such as those that help locate the nearest store, also largely use postal code. This is done without intercepting the exact geographical location to obtain the required city name. A list of the new get methods that are part of the proposed API is in Table III.

Listing 1 provides example code for getting the weather forecast for a user's location. This example depicts how the redesigned API can be used in a privacy preserving manner without having to handle the geographical coordinates of a user's location. This code finds out a user's location and sends it to the server. Weather applications cannot provide weather information at a finer granularity than the city or postal code, and hence have no need for a user's precise location. Weather is only one of the many applications that can function even without the exact geographical location. These applications are a subset of the large pool of smartphone applications.

Figure 1 compares the existing Location API with an example implementation of the proposed API. The API can be implemented either by Google as part of the Android API library or by a Caché implementation at the user's handset. When a third-party application server requests for a user's location, the existing API lets the server access the geographical coordinates of the user. The proposed API, on the other hand, can respond with an appropriate coarse location information, by retrieving it from a local cache. Like all location-based apps reach out to the GPS/network localization, they could fetch different granularity for a location from the cache. The cache gets location information from the GPS and network entities, and higher granularity location components from e.g. Google server. The implementation details of the caching is

```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to postal code updates
LocationListener locationListener = new LocationListener() {
    public void onPostalCodeChanged(Location location) {
      // Called when a new postal code is found by the network location provider.
                String zipCode = location.getPostalCode();
                getMyWeather(zipCode);
    }
    public void onStatusChanged(String provider, int status, Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };

// Register the listener with the Location Manager to receive postal code updates
locationManager.requestPostalCodeUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
```

Listing 1: Example source code using the proposed privacy-preserving APIs. This example is a mock code for an application that provides weather related information. It depicts how the proposed APIs can be used to request coarse location information like the postal code and get updates for the same.

Public Methods	
double	*getLatitude()* Get the latitude
double	*getLongitude()* Get the longitude
String	*getStreet()* Get the street name
String	*getBuilding()* Get the building number
String	*getCity()* Get the city
String	*getPostalCode()* Get the postal code
String	*getCounty()* Get the county
String	*getState()* Get the state
String	*getCountry()* Get the country

TABLE III: Modified Class Location: This table shows a list of the *get* methods for the Location class. These methods have been modified as part of our proposed API, to obtain coarse granularity location information.

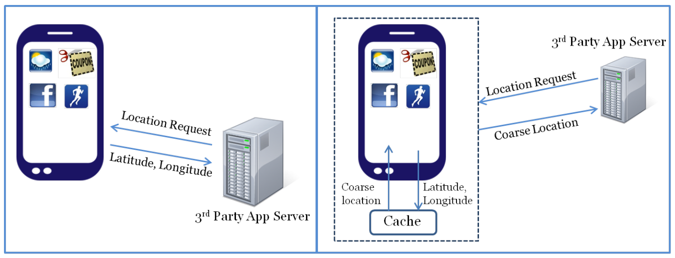
beyond the scope of this paper, for example implementation we refer to e.g. Caché [21].

IV. Method

In this section, we present our study design for understanding how developers would use privacy-preserving APIs.

A. Participants

We recruited 25 participants for our study. Six out of these were females and 19 were males. Our participants ranged from 18-27 years old (μ = 21, σ = 2.59). Five of them were graduate students and 20 were undergraduate students. We asked the participants about their prior experience with Java, Eclipse and Android. Since the knowledge of Java was a prerequisite



(a) Existing Location API (b) Proposed Location API

Fig. 1: A comparison of the existing API with the redesigned API. The existing API will return the geographical coordinates of the user irrespective of the location component needed by the application, while the proposed API can retrieve the required coarse location component from a cache and provide it to the application.

for participating in the study, all participants knew Java. The participant with the maximum experience (TC5) had 6-7 years of experience programming in Java, while the participant with the minimum experience (TB2) started programming in Java only two months before the study. None of our participants had extensive experience programming in Android. TC3 had the most experience programming on Android and had been doing it for a year. Only TD1 had programmed a location-based service earlier. All participants were familiar with the Eclipse IDE. Treatment Group C had the highest average for the knowledge of Java, Eclipse and Android. On an average, participants from Treatment Group C had 3.4 years of Java experience, 2.35 years of familiarity of Eclipse and about 5 months of Android acquaintance. Treatment Group A had only an average of 1.35 years of Java and 1.15 years of Eclipse practice.

B. Study Design

The study was based on a between-subjects design, where every participant was subjected to a single treatment only. There were three types of treatments, one control group and two treatment groups. Participants were randomly assigned to each group.

C. Procedure

We screened people by means of an entry survey and based on their Android familiarity, we divided our participants into two major categories- the Android developers (Group A) and the Non-Android developers (Group NA). Group A included participants with programming experience on the Android framework. Group NA were those participants that were versed in the Java programming language alone and had never coded for the Android platform. The purpose of dividing the participants in these two groups was to account for any bias that an Android developer may have towards existing API and the comfort of knowing them. The NA group, in contrast, accounts for a group of unbiased developers. This group should ideally be inclined to making fair choices in terms of ease and ethics, irrespective of any old habits. Table IV summarizes the study groups.

Category	Group	Participant Tag
Non-Android Group	Control Group	C
(Group NA)	Treatment Group A	TA
	Treatment Group B	TB
Android Group	Treatment Group C	TC
(Group A)	Treatment Group D	TD

TABLE IV: Summary of all the study groups. Participants in the Non-Android group were not familiar with the Android programming platform, while those in the Android group had some programming experience.

1) The Non-Android Group: Group NA was subdivided into three groups. The assignment of participants in these three subgroups was randomized to eliminate selection bias. This ensures that the differences in outcome observed at the end of the study would be due to treatment and not due to differences between groups that existed before the start of the study. A randomized experiment is often referred to as the gold standard for treatment outcome research. These three subgroups comprised one Control Group and two Treatment Groups.

The control group is a comparison group. It provides data about counterfactual inference, i.e. what happens when there is no treatment. The participation was anonymous, and hence for the purpose of the study the participants were referred to by code names. All participants in the control group had code names starting with uppercase letter C, followed by a number. They were provided with a useful subset of the Android Location API documentation from the Android Developers site [24]. This set of documentation serves as the baseline. We refer to the treatment groups as treatment group A (TA) and treatment group B (TB). Treatment group A participants were assigned code TA followed by a number, while those from treatment group B were assigned code TB followed by a number. These two groups were offered a documentation which had baseline Location API, and the proposed API. Both the treatment groups were provided the same information, but in opposite orders. This was the only difference with the treatments. The differing treatment accounts for any bias caused by the order in which information was provided to developers. For TA, the redesigned API were described first in the document, followed by the standard Android location API. For TB, the standard Android API was followed by the redesigned location API.

2) The Android Group: Since the members of Group A were already aware of basic Android programming practices and APIs, we divided them only into two treatment groups. We call these groups treatment group C (TC) and treatment group D (TD). TC was provided the same documentation as TA and TD was provided the same documentation as TB. Participants from treatment group C and treatment group D were assigned codes TC and TD, both followed by a number.

3) The Documentation: Provided below is a detailed description of the document that each group was given:

- Android Location baseline API: This documentation was a subset of the Android location API documentation from the Android developers site [24]. It had a collection of necessary information to implement a small piece of code that acquires a user's location. It included a short introduction to the *android.location* package and information on the three main location related classes and relevant public methods. An example code from the official documentation was presented [24]. There was also a section on reverse geocoding, with the associated classes and public methods.
- Android Location API for treatment groups A, B, C and D: This documentation included everything that the baseline documentation comprised, and our proposed API. We portrayed these to be a part of the usual location-based classes. For TA and TC, the order of the presented information was- the package introduction, the proposed API and reverse geocoding. For TB and TD, the order of presented information was- the package introduction, reverse geocoding and the proposed API.

4) Programming Tasks: Participants were provided a documentation based on their group, and a list of three programming tasks in randomized order. This ensured that all participants did not program the tasks in the same order. Each task described an application context and asked the participants to write code to extract the user's location for that context. The tasks did not inform the participant about the granularity of the information to be acquired. This was left for the participants to figure out and code for what they thought was the best approach. We came up with a list of about a dozen tasks and after careful consideration we came down to three. While all three of them could be performed using the existing API, two of them could be coded for without acquiring the person's geocoordinates and by using the redesigned API. We wanted to see how participants approach these problems using the documentation provided.

The weather app: Most weather applications accept either the city name or the postal code to supply the weather forecast for the user's location. They cannot provide weather information at a location finer than that and hence the task can be perfectly accomplished using the proposed API, without getting hands on the exact geocoordinates of the user. We asked the participants to write a code snippet to extract the user's location for a weather app.

The running app: We asked the participants to write that part of the application code that accesses and stores a runner's location. For this application, we need the geographical coordinates of the runner. This app can be developed using just the existing API.

The address app: In this app, a user's phone is set to silent mode when he reaches office and set to ringer mode once he reaches home. The participants were asked to code for the part that obtains the user's location to compare it against the saved home and office addresses to take further action. This app could ideally be done using any of the APIs, and hence this task was chosen.

Participants were asked to write their code on blank sheets of paper. We adopted this technique to ensure that people put in more thought when writing the code. Since we were not asking for an entire application or a ready-to-be-compiled code, writing code on paper implied lesser hassle on part of the participant. Moreover, our purpose was to observe their understanding of the API and not to test their programming skills. We also asked each participant to code at least one task on a code structure that we provided them with after programming for it on blank sheets.

D. Data Analysis

When the participants finished the exercise, we asked them questions related to their codes, what they thought about the documentation and the tasks that were compiled for them. We also inquired if they were aware of location privacy issues and if they used that knowledge while coding for the tasks. They were asked questions specific to the method calls they made in their codes, and why they made those choices. We asked participants what they thought was the purpose of the study, at the end of the study. None of the participants could guess the actual purpose. Their codes were analyzed based on logic. As long as there was a logical flow of statements and method calls, with comprehensible use of programming syntax and API, we considered the code for API analysis. We inquired about the difficulties they had, if any, while using the APIs that they chose.

V. Results

This section presents the results of the study. We begin by evaluating the programs submitted by the participants and an analysis of the API used. It is seen that the apps where the redesigned API fit well, like the weather app and the address app, participants were naturally disposed to using them, even without being asked to consider privacy. The redesigned API structure nudges developers to make privacy preserving choices, without trying too hard. We also examine participants' response to the provided code skeleton, which brings to light that the basic programming intuition and logic is unchanged by the level of comfort with the syntax of the programming language.

All participants wrote programs for the list of problems provided to them. In the following subsections, we address these and the other aspects of the participants' program code.

A. Evaluation of Completed Tasks

Participants were provided a randomized list of tasks. Nine out of 25 participants completed all the tasks during their session, while two did not code at all. The others completed one or two tasks. As discussed earlier, the coding tasks were the weather app, the running app and the address app. Out of 25, 13 participants completed the weather app, 20 completed the running app and 19 completed the address app. These numbers account for the variability in the tasks attempted and the APIs used.

1) The Non-Android Group:

a) Control Group: The control group constituted of six members. They were provided with the baseline API. This included the *Geocoder* and *Address* classes. Only two (C1, C4) out of six participants used reverse geocoding. While C2 and C7 only copied the example code, C3 did not understand the provided documentation at all. C8 attempted to use the *Address* class without the *Geocoder* class.

b) Treatment Groups: As discussed earlier, Group NA was divided into two treatment groups, TA and TB. The code names for the participants from these groups also had the relevant prefix.

TA had five participants. They were provided the set of documentation that had the proposed API before the *Geocoder* and *Address* classes. *None of the participants used the reverse geocoding. Four (TA2, TA3, TA4, TA5) out of five participants used some redesigned API.* The last participant, TA6, copied the example code provided in the documentation. He did not utilize either the proposed API or reverse geocoding, but used only the raw coordinates for each task. Reverse Geocoding is the process of obtaining raw location coordinates and then transforming them into an address.

"I tried to make it the postal code or city because that is usually what people want. They don't usually want latitude and longitude" - TA2, on using the getPostalCode(), requestPostalCodeUpdates() and onPostalCodeChanged() for the weather task.

"Coordinates are better because they are numbers and the easiest to find out" - TA6.

A list of treatment group A participants, who used the redesigned API, the APIs they used along with the tasks they used them for, is provided in Table V.

Treatment Group A		
Participant	API used	Task

TA2	requestPostalCodeUpdates()
	onPostalCodeChanged()
	getPostalCode()
		Weather

TA3	getBuilding()	Addresses
TA4	onBuildingChanged()	Addresses
TA5	getCity()	Weather

TABLE V: List of treatment group A participants who used the proposed APIs. Against each participant, is a list of the API that they used and the task they used it for. Four out of five participants in TA made use of the proposed APIs to accomplish the coding tasks provided to them. One participant copied the example code and did not use the proposed API or reverse geocoding. This group was provided the documentation with proposed API followed by the reverse geocoding classes.

TB had six participants. They were provided the documentation that had the *Geocoder* and *Address* classes described before the proposed API. *Three (TB1, TB6, TB7) out of six participants used some redesigned API.* One (TB4) participant did not understand the documentation and did not attempt coding at all. TB2 copied the example code as is, while TB5 was the only participant who used reverse geocoding for completing the task.

"You get them [geocoordinates] from location manager. Then you have to use this part - geocoding. I tried to do that for this one but I didn't really know how to" - TB2.

TB2 tried to go beyond the example code to put the obtained coordinates to use, but found it difficult to do so. He mentioned that he did not go further to look at the redesigned API.

Treatment Group B		
Participant	API used	Task
TB1	requestStreetUpdates()	Addresses

TB6	requestStreetUpdates()
	getStreet()
		Addresses

TB7	requestCityUpdates()
	onCityChanged()
		Weather

TABLE VI: List of treatment group B participants who used the proposed APIs. Against each participant, is a list of the API that they used and the task they used it for. Three out of six participants in TB made use of the proposed APIs to accomplish the coding tasks provided to them. One participant copied the example code, and one more did not understand the documentation. Only one participant used reverse geocoding. This group was provided the documentation with the reverse geocoding classes followed by the proposed API.

Table VI provides a list of the APIs from the proposed set that TB1, TB6 and TB7 used and the corresponding tasks.

2) The Android Group: The Android group also had two treatment groups, treatment group C (TC) and treatment group D (TD).

TC comprised five participants who were offered the documentation with the redesigned API listed first. *All five participants made use of the redesigned API to accomplish the coding problems.*

"the current order of information is well organized and its better this way" - TC5.

Despite his feedback, TC5 said that the order would not have made a difference to his coding approach. Table VII enlists all the participants, the APIs they used, and the corresponding tasks.

TD consisted of three participants, who were provided the documentation with the *reverse geocoding* explained before the redesigned API. *Two (TD1, TD2) out of three members chose to use the redesigned API over reverse geocoding.* One participant, TD4, used the *Geocoder* class.

"I may have chosen this [Geocoder class] because it was first. I was reading through and I saw this and I was like, oh that will work" - TD4.

Table VIII provides a participant-wise summary, the API used, and the corresponding tasks. The fact that none of the participants used the proposed API for the running app shows that not all applications can use coarse location information, but those where they can be used, participants attempted them.

Treatment Group C		
Participant	API used	Task

TC1	requestPostalCodeUpdates()
	onPostalCodeChanged()
	getPostalCode()
		Weather

TC2	getStreet()
	getPostalCode()
		Addresses

TC3	getCity()
	getStreet()
		Weather

TC4	requestPostalCodeUpdates()
	getPostalCode()
	requestStreetUpdates()
	getStreet()
		Weather
		
		Addresses

TC5	requestStreetUpdates()
	onStreetChanged()
	getStreet()
	onCityChanged()
		Addresses
		
		Weather

TABLE VII: List of treatment group C participants who used the proposed APIs. Against each participant, is a list of the API that they used and the respective tasks. All five participants in TC made use of the proposed APIs to accomplish the coding tasks provided to them. This group was provided the documentation with proposed API followed by the reverse geocoding classes.

Treatment Group D		
Participant	API used	Task

TD1	getStreet()
	getCity()
		Addresses
		Weather

TD2	requestStreetUpdates()
	onStreetChanged()
	getStreet()
		Addresses

TABLE VIII: List of treatment group D participants who used the proposed APIs. Against each participant, is a list of the API that they used and the tasks they used them for. Two out of three participants in TD made use of the proposed APIs to accomplish the coding tasks provided to them. One participant used reverse geocoding. This group was provided the documentation with the reverse geocoding classes followed by the proposed API.

3) Evaluation: Table IX gives a summary of the treatment groups and the statistics in terms of API usage. It provides concise information on how many participants tried to use the redesigned API or the existing API for fulfillment of the coding assignment. A few that did not understand the APIs or just copied the example code supplied with the documentation, are also accounted for.

Programmers who had Android experience had a better understanding of the documentation and API (Table IX). We can say this with certainty because no participants from the Android group (Group A) copied the example code without attempting to understand the provided APIs. All participants from TC preferred the redesigned API over geocoding, while all but one from TD opted for the redesigned API. TD4, who used geocoding, articulated that this might have been due to the order in which the information was provided to him. The Non-Android programmers on the other hand, had a harder time understanding the APIs and the context for their usage. Three participants from the treatments groups could not understand the document and hence ended up copying the example code.

Group	Participants	Used proposed API	Used reverse geocoding	Copied example	Did not understand documentation
CG	6	-	3	2	1
TA	5	4	0	1	0
TB	6	3	1	1	1
TC	5	5	0	0	0
TD	3	2	1	0	0

TABLE IX: A treatment group-wise list of the participants. It states the number of participants in each treatment group. It offers a comparison of how many participants from each used the proposed APIs as opposed to those that used the baseline reverse geocoding method. There is also those who either copied the example code verbatim, or did not understand the documentation at all.

This also brings to light that developers follow example codes very closely. Hence, if presented with appropriate privacy-preserving examples, they can be nudged into practicing them.

Six participants preferred the redesigned API over Reverse Geocoding because they found it easier, as stated by TB2 in an earlier section and TB5.

"Geocoder was the most confusing part" - TB5.

TC4 thought that using the suggested API gave him more freedom apart from being straightforward and doable. TB7 also remarked that the suggested API complied well with the application requirement, and hence he used them. This reflects the fact that the redesigned API are easy to use. Although few participants actually considered privacy and others used it because it was straight-forward and easy, it leads to the practice of accessing only the required location information.

Table IX also indicates that, since no participants from TA and TC used geocoding, the order in which information is presented to a developer affects the way they perceive the solution for a given problem. TA and TC were presented documents with the reverse geocoding classes later, and no participants opted for it. TA6 copied the example code from the documentation.

"..finding the coordinates was the easiest thing to do" - TA6.

B. Response to the Code Structure

As discussed earlier, we provided each participant with a code structure to complete at least one of the tasks. Since most of our study group had not programmed extensively on Android, we wanted to simulate an environment for them as that of experienced developers. For this, we provided them a code structure to serve as the outline for their programs. It imported the relevant package, defined a listener and acquired a reference to the system *LocationManager*. We wanted our participants to focus on implementing the required functionality rather than worrying about the syntax and code organization, like experienced developers. This motivated them into thinking about the other important matters, such as what APIs to use and their implications. We wanted to study the difference between coding on plain paper versus coding on the outline sketch. Most participants stated that completing an incomplete code was far less worrisome than starting on a blank paper. However, there were no significant changes with their code except for two participants. The Table X summarizes how a code skeleton affected programmers.

Only TC2 and TA5 out of 20 participants changed their programs significantly when using the code skeleton, while 15 participants did not change their code at all. The remaining three did not understand the documentation, irrespective of the coding skeleton.

Group	Participants	Offered code skeleton	No change in code	Significant change in code
CG	6	5	4	0
TA	5	4	3	1
TB	6	4	2	0
TC	5	4	3	1
TD	3	3	3	0

TABLE X: This table lists the number of participants from each group who were provided the incomplete code structure. It also lists the number of participants who changed their code when completing the structure, and those who did not change their codes.

"With the blank piece of paper I had to consider how to start the code and since this skeleton already has the beginning layout I would only have to focus on the problem itself rather than worrying about the outside information, like syntax" - TA6 (no change in code)

"I concentrated on implementing the functionality. Typically environments like Eclipse also provide some skeleton." - TC5 (no change in code)

"...different because I use LocationManager and LocationListener well " - TC2 (change in code)

This also reflects that even though programmers were much comfortable coding on the skeleton, they did put considerate thought to the API usage even before having the skeleton.

C. Location Privacy Concerns

The participants were not asked to consider privacy while coding for the tasks. We wanted to see if the plethora of options for obtaining the location information would automatically nudge a new developer to consider privacy. After the programming task was done, we asked each participant if they were aware of location privacy issues surrounding mobile applications. 13 out of the 25 participants were not aware of any such issues. Six participants acknowledged being aware of it but did not give it a thought while coding for the problems. Six participants had interesting insights about location privacy. Most of them opted for the redesigned API without privacy being an explicit requirement.

"I know about them. It flashed my mind for a second, like do you want to track every single detail? But then I just

continued doing what I was doing " - TA3 (used the redesigned API).

TC1 kept privacy in mind while coding the tasks. Which is why he preferred using the network provider over the GPS provider, whenever he could.

"That's why I tried to avoid GPS when possible because lots of people are sensitive to giving fine location data away. And I tried to use the network when possible because even if they're sure they know you're connected to this tower, still towers cover such a vast area and depending on where you are there is such a huge number of people attached to that network they cant identify who you are without more information on that" - TC1 (used the redesigned API).

"Your phone is capable of sending your coordinates at all times to a server. I chose to use postal code as opposed to street address or coordinates because I didn't want to send out too much information" - TC4, discussing his code on weather application.

There is also a light indication that with the current Android platform, developers might be motivated to use non-privacy preserving practices, because of the user-centric permission model on Android systems.

"I didn't think about it [location privacy] because I just assume that once they [users] install the application they've already given permission for it." - TC3

VI. Discussion

We performed a controlled randomized study for the analysis of privacy-preserving APIs. We analyzed the effects of these APIs and nudging developers into using them to design privacy friendly applications.

The evaluation of participants' codes and post-study interviews affirms that a programmer can be inclined to making privacy preserving choices, if presented with them. This finding is supported by the fact that 73.6% of the participants in our treatment groups, chose to use the redesigned API instead of using the default reverse geocoding. Our participants who were aware of location privacy issues, and considered them when carrying out the tasks, were seen to be disposed to using the proposed privacy-preserving APIs.

We also have evidence that having no prior knowledge of Android does not affect programmers and does not hinder them from making the desired choices. This is exhibited by the numbers in Table IX. A fair number of participants from treatment groups in the Non Android group (TA and TB) preferred the proposed API.

There is indication that apart from being privacy-preserving, the redesigned API is also straightforward to use. Section V(A) describes how many participants chose to use the revised API because they appeared easy and more suited for the task.

The current Android platform does not offer developers a chance to attempt protecting the user's privacy. We perform this study to learn developers' response to an alternative design. The new API design allows the application to acquire only the location information necessary for the effective functioning of the application. It enables them to access coarse granularity location information easily, without intruding the circle of finer location components.

The results clearly indicate that developers, if given a choice, are inclined to using privacy-conserving programming practices. All four participants from the treatment groups, who expressed concern about user's location privacy, preferred to utilize the redesigned API.

The fact, as evident from Table V to Table VIII, that none of the participants used the proposed API for the running app, proves that not all applications can use coarse location information. Although, it does reflect that apps where they can be used, participants attempted them.

There is also a light indication that with the current Android platform, developers might be motivated to use non-privacy preserving practices, because of the user-centric permission model on Android systems.

"I didn't think about it [location privacy] because I just assume that once they [users] install the application they've already given permission for it " - TC3

TC3's comment reflects what might be a common thought among mobile application developers. It must be noted here though, that the choice exercised by users might only be a compromise if they want to make use of the application's features. Our study also indicates that most novice Android developers ground their codes on the examples provided in official documentation. This asserts that if privacy preserving examples were offered in standard API documentation, developers would be more likely to follow the trend. For novice developers, a good privacy protecting example in the documentation would be enough to steer them into following similar programming practices. It is also worth noting that all participants who admitted to using their awareness of location privacy issues during coding the tasks chose to use the proposed API.

"It [having the code skeleton] didn't make much of a difference because I had the example code right here" - C7

A few points worth noting about the redesigned API and the user study are:

- The proposed APIs are a modified version of the existing Android location API. The current API offers no choice but to obtain the user's geographical coordinates, and thus, we propose enabling the collection of coarse levels of location information, such as city, county etc.
- The redesigned API would be privacy preserving only for a subset of applications that do not require fine grained location.
- Our participant base consisted only of students, and not professional developers. This was to avoid the unrealized bias, that developers have towards existing APIs, because they already know them. The analysis of the redesigned API is based on the common sense and intuitions as a programmer, rather than being proficient at the developing platform.
- The participants were not asked to consider any privacy situations before they tried the tasks. This was

to avoid any priming and conscious efforts on the part of the participant, and to see whether or not the increased choices nudge new developers to consider privacy without someone pressing them to.

A. Limitations

As discussed earlier, our participant base is limited only to students. One may question that these participants might have been inclined to using the easier API than considering preserving privacy. To account for this phenomenon, we provided our participants with a code outline after they had first sketched their initial approach, so they would not have to worry about the syntax and organization of the code. To fill in the outline, participants were inclined to use, not what is easy, but rather what fits the functionality of the application context best. This fact is also backed up by our study.

Our example applications are those that directly use the location provided by the Android location services. Applications that use external databases, that require the location in a certain format for part of the information, are not addressed here.

In the study, we do not account for monetary incentives offered by third-party advertisement networks. These networks base their advertisements on the user's current location without much concern for privacy. McDonald et al. [28] discuss about the balance between user's privacy and developer's revenue. Our study results exemplify that programmers follow example codes very closely. Since we anecdotally know that many developers learning new platform turn to sites such as StackOverflow for programming issues, further work could also involve providing privacy-preserving examples on StackOverflow.

VII. Conclusion

We have presented an extensive qualitative analysis of results from a controlled study on how developers with only Java or some Android experience would use location APIs with privacy-preserving designs. This work presents the first step on how to nudge developers for protecting people's privacy with API designs. Our work has implications for redesign and documentation of location APIs. When offered alongside the baseline API for localization, our treatment groups chose to utilize the proposed privacy-preserving API. We show that when approaching API documentation from a "blank slate", people tend to follow the sample code closely. Thus, if they are presented with privacy-preserving examples in official documentation, developers will be using them instead of less privacy-preserving alternatives. Our results affirm that participants who considered privacy, in contrast to the others who did not, were inclined towards using the proposed API.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant Number 1223977. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] "The location privacy protection act of 2011," http://www.franken.senate.gov/files/documents/121011_LocationPrivacyProtection.pdf.

[2] M. Langheinrich, "Privacy by design - principles of privacy-aware ubiquitous systems," in *Proc. of UbiComp'01*.

[3] J. Bloch, "How to design a good api and why it matters," in *Companion to the OOPSLA '06*, 2006.

[4] S. Clarke, "Measuring API usability," *Dr. Dobb's Journal*, 2004.

[5] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Softw. Engg.*, Dec. 2011.

[6] M. Becher, F. C. Freiling, J. Hoffmann, T. Holz, S. Uellenbeck, and C. Wolf, "Mobile security catching up? revealing the nuts and bolts of the security of mobile devices," in *Proc of SP '11*.

[7] J. Anderson, J. Bonneau, and F. Stajano, "Inglourious installers: Security in the application marketplace," in *Proc. of WEIS 2010*, 2010.

[8] P. Gilbert, B.-G. Chun, L. P. Cox, and J. Jung, "Vision: automated security validation of mobile apps at app markets," in *Proc. of MCS*, 2011.

[9] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," in *Proc. of USENIX Security*, 2011.

[10] M. Ongtang, K. Butler, and P. McDaniel, "Porscha: policy oriented secure content handling in android," in *Proc. of ACSAC*, 2010.

[11] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proc. of OSDI'10*.

[12] J. Howell and S. Schechter, "What you see is what they get: Protecting users from unwanted use of microphones, camera, and other sensors," in *In Proc. of W2SP*, 2010.

[13] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan, "Mockdroid: trading privacy for application functionality on smartphones," in *Proc. of HotMobile*, 2011.

[14] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proc. of the TRUST 2011*, 2011.

[15] S. Lederer, I. Hong, K. Dey, and A. Landay, "Personal privacy through understanding and action: five pitfalls for designers," *Personal Ubiquitous Comput.*, November 2004.

[16] S. Spiekermann and L. Cranor, "Engineering privacy," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, January/February 2009.

[17] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," in *Proc. of HotSec'12*, 2012.

[18] C. Bravo-Lillo, L. F. Cranor, J. Downs, and S. Komanduri, "Bridging the gap in computer security warnings: A mental model approach," *IEEE Security and Privacy*, March 2011.

[19] R. Wash, "Folk models of home computer security," in *Proc. of SOUPS '10*, 2010.

[20] S. Guha, M. Jain, and V. N. Padmanabhan, "Koi: a location-privacy platform for smartphone apps," in *Proc. of NSDI'12*, 2012.

[21] S. Amini, J. Lindqvist, J. I. Hong, J. Lin, E. Toch, and N. Sadeh, "Caché: caching location-enhanced content to improve user privacy," in *Proc. of MobiSys'11*, 2011.

[22] J. Tam, R. W. Reeder, , and S. Schechter, "I'm allowing what? disclosing the authority applications demand of users as a condition of installation," Microsoft Research, MSR-TR-2010-54, May 2010.

[23] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang, "Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing," in *Proc. of UbiComp'12*.

[24] "Android developers guide - location strategies," http://developer.android.com/guide/topics/location/strategies.html.

[25] H. Zang and J. Bolot, "Anonymization of location data does not work: a large-scale measurement study," in *Proc. of MobiCom '11*, 2011.

[26] J. I. Hong and J. A. Landay, "An architecture for privacy-sensitive ubiquitous computing," in *Proc. of MobiSys '04*, 2004.

[27] M. F. Mokbel, C.-Y. Chow, and W. G. Aref, "The new casper: query processing for location services without compromising privacy," in *Proc. of VLDB '06*.

[28] A. M. McDonald and L. F. Cranor, "Americans' attitudes about internet behavioral advertising practices," in *Proc. of WPES '10*, 2010.