TurtleGuard: Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai
Primal Wijesekera
Joel Reardon
Irwin Reyes
Jung-Wei Chen
Nathan Good
Serge Egelman
David Wagner

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-44
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-44.html

May 10, 2017

Copyright © 2017, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to give thanks to Primal Wijesekera for the help, kindness, and support he has shown me throughout this project. I would also like to thank David Wagner, Doug Tygar, and Serge Egelman for their words of wisdom and everything they have taught me. Lastly, I would like to thank Alec Guertin and my hamster, Santa, for moral support.

TurtleGuard:

Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai
University of California, Berkeley
lynntsai@berkeley.edu

Primal Wijesekera
University of British Columbia
primal@cece.ubc.ca

Joel Reardon
University of California, Berkeley
joel.reardon@berkeley.edu

Irwin Reyes
University of California, Berkeley
ioreyes@berkeley.edu

Jung-Wei Chen
Good Research
jennifer@goodresearch.com

Nathan Good
Good Research
nathan@goodresearch.com

Serge Egelman
University of California, Berkeley
egelman@berkeley.edu

David Wagner
University of California, Berkeley
daw@berkeley.edu

ABSTRACT

Current mobile platforms provide privacy management interfaces to regulate how applications access sensitive data. Prior research has shown how these interfaces are insufficient from a usability standpoint: they do not allow users to make contextual decisions (i.e., different decisions for a given application based on what the user was actually doing with that application). Prior work has demonstrated that classifiers can be built to automatically make privacy decisions that are more in line with users' preferences. However, if certain privacy decisions are automatically made—without immediate user consent—feedback mechanisms are needed to allow users to both audit those decisions and correct errors. In this paper, we describe our user-centered approach to designing such an interface. In addition to implementing this interface in Android, we created an interactive HTML5 simulation that we used to perform two large-scale user studies. Our final 580-person validation study showed that as compared to the default Android settings interface, users of our new interface were significantly more likely to understand and control the circumstances under which applications could access sensitive data.

1. INTRODUCTION

Smartphones store a great deal of personal information, such as the user's contacts, location, and call history. Mobile operating systems use *permission systems* to control access to this data and prevent potentially malicious third-party applications ("apps") from obtaining sensitive user data. These permission systems inform and empower users to make appropriate decisions about which apps have access to which pieces of personal information.

The popular open-source Android mobile platform has used two general approaches to give users control over permissions. Initially, permissions were presented as an install-time ultimatum, or ask-on-install (AOI): at installation, an application would present the full list of sensitive resources it wished to access. Users must either grant access to all requested permissions or, if any are deemed unacceptable, abort the installation entirely. More recently, an *ask-on-first-use* (AOFU) permission system has replaced these install-time disclosures. Under AOFU, the user is prompted when an application requests access to a sensitive resource for the first time. The user's response to this permission request carries forward to all future requests by that *application* for that *permission*.

The AOFU approach, however, fails to consider that the user's preferences, as expressed in the first-use prompt, may not be appropriate under all future *contexts*. For instance, a user might feel comfortable with an application requesting location data to deliver desirable location-based functionality. The same user, however, might find it unacceptable for the same application to persistently access location data while in the background for advertising or tracking purposes. Prior research has analyzed the effectiveness of AOI as a predictive model by comparing it to user responses to runtime prompts. These studies found that although AOFU is an improvement over install-time permissions, AOFU still frequently fails to match user preferences [38]. When making privacy decisions, users consider a richer space of factors than just simply the requesting application and the resources it wishes to access [26]. Android currently does not provide a practical mechanism for users to audit and adjust contextually-incorrect decisions made by the platform: it lacks a contextually-aware permission manager.

The *contextual integrity* framework suggests that many permission models fail to protect user privacy because of their inability to account for the context surrounding data flows [26].

Copyright is held by the author/owner. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee.
Symposium on Usable Privacy and Security (SOUPS) 2017, July 12–14, 2017, Santa Clara, California.

That is, privacy violations occur when a data flow (e.g., a resource access request) defies the user's expectation. In recent work [37, 38], researchers found that a significant portion of participants made contextual privacy decisions. In theory, asking the user to make a decision for every request is optimal, as the user will be able to account for the surrounding context and can then make decisions on a case-by-case basis. In practice, however, this results in less-usable privacy controls, as the frequency of these requests will likely overwhelm the user [37]. Thus, automating these decisions is likely to yield a balance between respecting users' privacy preferences and not overburdening them with many decisions. When automated decisions are made, however, it is imperative for users to have feedback mechanisms so that prior decisions can be reviewed and errors can be corrected, thereby leading to fewer errors in the future.

Machine learning techniques are capable of capturing the context surrounding each resource request and predicting users' contextual privacy preferences [38, 20]. Recent work has also shown that machine learning techniques can reduce privacy violations by inferring users' privacy profiles [21, 22]. All of these methods, however, still produce erroneous inferences at times. These errors necessitate continuous user feedback to retrain the model and adapt to user expectations. Prior work has emphasized the importance of providing user interfaces as feedback to the classifier: users need a way to correct classifier results in order to improve the accuracy of future decisions, and account for users' changes in preferences over time [38].

To this end, we designed a novel permission manager, TurtleGuard, which allows access to sensitive information at some times and protects the information with a hard shell at others. It helps users to make contextual privacy decisions about the apps that they use by providing a feedback loop for them to audit and modify how automated decisions are made. We allow users to (i) vary their decisions based on the surrounding context, and (ii) have a better understanding of how third-party applications access resources in the real world and under varying contexts. We conducted an initial 400-person experiment to evaluate our initial design. Based on this data, we iterated on our design. We then conducted a 580-person validation study to demonstrate the effectiveness of our design. Both experiments had four tasks: three tasks that involved using the system to locate information about current application permissions, and one task that involved modifying settings. We observed that participants who used TurtleGuard were significantly more likely to apply contextual privacy preferences than the control group. We believe these results are a critical contribution towards empowering mobile users to make their own contextual choices efficiently on mobile phone platforms. Our contributions are as follows:

- We present the first contextually-aware permission manager for third-party applications in Android.
- We show that when using our new interface compared to the existing Android interface, participants were significantly more likely to both understand and control when applications had foreground versus background access to sensitive data.
- Similarly, we observed that our interface requires no learning curve: having never used TurtleGuard before, participants were successful at accomplishing information retrieval tasks that were previously possible in the existing Android interface.

2. RELATED WORK

The Android OS has operated under two permission models: ask-on-install permissions, and ask-on-first-use (AOFU) permissions. Versions of Android before version 6.0 (Marshmallow) implemented ask-on-install permissions. Under this model, applications request that the user grant all permissions to the application at install time. The user must consent to all requested permissions in order to complete installation. Otherwise, if the user wishes to deny any permission, the only option available is to abort the installation entirely. Research has shown that few users read install time permissions, and fewer yet understand their meaning [13, 19].

Versions of Android from 6.0 (Marshmallow) onward use the AOFU permission model instead. Under AOFU, applications prompt users for sensitive permissions at run time; these include access to geolocation data, contact lists, and photos, among others. Prompts appear when the application attempts to access protected resources for the first time. This has the advantage of giving users contextual clues about why an application requires a protected resource: users can consider what they are doing when the prompt appears to help determine whether to approve the request. Although AOFU offers an improvement over the install-time model in this regard, first-use prompts insufficiently capture user privacy preferences after an initial decision is made [37]. That is, the AOFU model does not consider scenarios where an application requests access to data under varying contexts. For instance, a user may grant an application access to location data after she is prompted while attempting to use location-based features. That same user, however, will not receive any more prompts when that same application subsequently accesses location data to perform tracking—a use of location data that the user might find objectionable.

Research on permission models has found that users are often unaware how apps access protected resources and how access may be regulated [13, 9, 12, 35, 33]. Almuhimedi et al. studied AppOps, a permission manager introduced in Android 4.3 but removed in Version 4.4.2 [1]. AppOps allowed users to review and modify application permissions post-install, as well as set default permissions for newly installed applications to follow. They examined the use of AppOps with privacy nudges that were designed to increase user awareness of privacy risks and facilitate the use of AppOps. They concluded that Android users benefit from the use of a permission manager, and that privacy nudges are an effective method of increasing user awareness [1].

Although AppOps was removed from Android, the 6.0 "Marshmallow" release reintroduced permission management. Android 6.0 included an updated interface that allows the user to view all of the permissions that a particular app has been granted, as well as all of the apps that have been granted a particular permission (Figure 1). Unfortunately, these controls are buried deep within the stock Settings app, and therefore it is unlikely that many users know about them. For instance, viewing a particular app's permissions requires navigating four levels of sub-panels, whereas viewing all the apps that have requested a particular permission requires navigating five levels. By comparison, TurtleGuard is one

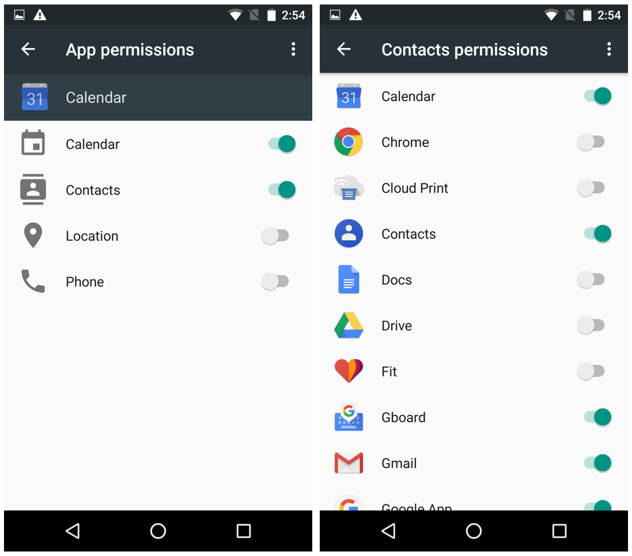


Figure 1: After navigating through four and five levels of sub-panels within the default Android Settings app, respectively, users can limit an individual app's access to specific permissions (left) or limit the apps that can access a particular permission (right).

click from the main Settings panel and explicitly presents the relationships between applications, permissions, and controls.

Beyond AppOps, other third-party privacy and permission management tools exist. Permission Master [24], DonkeyGuard [7], XPrivacy [6], and LineageOS's[1] Privacy Guard [25] are examples of such third-party permission management software. These utilities require additional privileges and techniques to install because Android provides no official mechanisms for third-party programs to modify the permission system. For instance, Privacy Guard is built into the LineageOS custom ROM [25], while the others use the Xposed Framework [31], which itself requires an unlocked bootloader and a custom recovery partition. Such restrictions are necessary to prevent malicious software from interfering with the existing permission system.

Third-party permission managers offer users a variety of features to fine-tune access to sensitive resources on their devices. XPrivacy has the option to pass fake data to applications that have been denied access to protected resources [2]. Hornyack et al.'s AppFence similarly allows users to deny permissions to applications by providing fake data [17]. Providing fake data is more desirable than simply failing to provide any data at all, as the latter may cause functionality loss or application failures.

These managers follow an Identity Based Access Control model (IBAC), where individual permissions can be set for each app [27]. Although this model allows users to specify fine-grained permission preferences, this may be ineffective in practice for two reasons. First, users may be overwhelmed by the number of settings available to them, some of which are only tangentially relevant to privacy. XPrivacy and Permission Master show controls for resources whose direct effects on user privacy are unclear, such as keeping a device awake. Our dashboard improves usability by showing only controls for resources deemed "dangerous" in the Android platform [16] and others that warrant run-time prompts [11]. Second, none of the existing permission managers display the context in which protected resources were accessed. XPrivacy, Donkey Guard, and LineageOS's Privacy Guard provide timestamps for resource accesses, but the user does not receive important information about the app's state, such as whether it was actively being used when it requested access to sensitive data. Permission Master offers no historical information at all. TurtleGuard addresses this problem by listing recently allowed and denied permission access requests, along with the state and visibility of the requesting application at the time the permission was requested.

Apple's iOS mobile platform offers visibility-sensitive location privacy settings: "Never" and "Always" (the two settings analogous to Android's permission on/off toggles), and a "While using the app" option, which only permits an application to access geolocation data while the application is active on the screen. We use the same options for TurtleGuard. Our design is novel in both the extent of these settings and in who controls them. Apple's iOS platform allows developers to control which of the three options are available to users to select [3]. Application developers have faced criticism for removing the "While using the app" option, forcing users to choose between reduced functionality and granting the application unrestricted access to sensitive location data [28]. Our design, by contrast, gives users all three of these options for all *sensitive* permissions. Table 1 lists the permissions we consider sensitive. Furthermore, developers cannot restrict user choice with these settings, as TurtleGuard is implemented in the operating system.

Wijesekera et al. show that although AOFU improves on install-time permissions, AOFU is insufficient because it does not account for the context of the requests [38]. They examined this by instrumenting the Android platform to log all instances of apps accessing sensitive resources. In addition to this instrumentation, the platform randomly prompted users about the appropriateness of various permission requests as those requests occurred. Participant response to these prompts was treated as the dependent variable for a training set. Their study showed that 95% of participants would have chosen to block at least one access request had the system notified them. On average, participants would have preferred to block 60% of permission requests. Indeed, other work suggests that contextual cues are key in determining whether privacy violations may have occurred [26, 5].

A natural extension of AOFU is "ask on *every* use": rather than extrapolating the user's first-time preference to all future accesses to a given resource, each access would require user input instead. Such a model would conceivably allow users to specify their contextual preferences more accurately; users know exactly which apps attempted to gain access to which resources under which circumstances. This approach, however, is unusable in practice. Research has shown that applications request access to permission-protected resources

[1]LineageOS is a recent fork of CyanogenMod after the latter's discontinuation.

Permission	Explanation
CALL_PHONE PROCESS_OUTGOING_CALLS READ_PHONE READ_CALL_LOG ADD_VOICEMAIL WRITE_CALL_LOG	Make and process calls as well as read information about call status, network information and previously made phone calls
CAMERA	Access camera devices
GET_ACCOUNTS	Access to list of accounts
READ_CALENDAR WRITE_CALENDAR	Read and write events to the user's calendar
READ_CONTACTS WRITE_CONTACTS	Read and write to user's contacts
READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE	Read and write files to the user's external storage
RECORD_AUDIO	Record audio
ACCESS_COARSE_LOCATION ACCESS_FINE_LOCATION ACCESS_WIFI_STATE	Read location information in various ways including network SSID-based location
READ_SMS SEND_SMS RECEIVE_SMS	Read SMS messages from the device (including drafts) as well as send and receive new ones SMS

Table 1: Sensitive permissions managed by TurtleGuard. Permissions grouped by a single explanation form the families used in our system to reduce the number of managed permission.

with regular frequency: on an average smartphone, roughly once every 15 seconds [37]. Such a high frequency not only risks user habituation, it is beyond practical limits to seek user consent on every access.

Recent research on permission models has turned towards using machine learning (ML) [38, 21, 22, 20]. One advantage is its ability to capture the surrounding context to predict user preferences; the approach has shown significantly lower error rates over the *status-quo*, i.e., AOFU. Wijesekera et.al [38] showed that ML also reduces user involvement and thereby avoids user habituation. They emphasize, however, the importance of having a user interface that functions as a feedback-loop to the classifier. Users can use the interface to audit the decisions made by the classifier and correct any decisions that do not match their preferences. Such a mechanism not only ensures the machine-learning classifier improves its accuracy over time, it also keeps users aware of decisions that are being made on their behalves and informs them of how third-party apps are accessing sensitive resources under various circumstances.

We present two core components necessary for usability under such contextual privacy models: we present users with key contextual information when deciding access to sensitive resources, and we provide a method for users to regulate and correct decisions automatically made on their behalf by the system. We designed TurtleGuard to be easily understood by users of any technical background.

3. DESIGN OBJECTIVES

In this section, we describe the design philosophy behind TurtleGuard. TurtleGuard's primary function is to inform users about the decisions that have been automatically made for them, while allowing them to easily correct errors (thereby improving the accuracy of future decisions). These errors can be either false positives—an app is denied a permission that it actually needs to function—or false negatives—an app is granted access to data against the user's preferences.

Thompson et al. showed how attribution mechanisms can help users better understand smartphone application resource accesses [36]. They found that users expect this information to be found in the device's *Settings* app. In our initial experiment, we evaluated TurtleGuard as a standalone app, though for this reason we ultimately moved it within the Android *Settings* panel prior to our validation experiment.

3.1 Incorporating Context

In prior work, researchers observed that only 22% of participants understood that applications continue to run when not visible and have the same access to sensitive user data that they do when they are visible [36]. This means that the majority of users incorrectly believe that applications either stop running when in the background or lose the ability to access sensitive data altogether. Wijesekera et al. corroborated this observation in a recent field study of users' privacy expectations: users are more likely to deem permission requests from background applications as being inappropriate or unexpected, and indicate a desire to regulate applications' access to sensitive data based on whether or not those applications are in use [37].

In the default permission manager, users cannot vary their decisions based on the visibility of the requesting application. Our goal in this work is to empower the user to make contextual decisions, to let users vary their decisions based on context (i.e., what they were doing on the device), and to let these contextual decisions be applied to future requests.

option	meaning
always	The permission is always granted to the requesting application, regardless of whether the application is running in the foreground or background.
when in use	The permission is granted to the requesting application only when there are cues that the application is running, and denied to the requested application when the application is running invisibly in the background.
never	The permission is never granted to the requesting application.

Table 2: The three possible permission settings under TurtleGuard. The *when in use* option accounts for the visibility of the requesting app, which is a strong contextual cue.

Moving one step beyond the all-or-nothing approach, our new design gives the user a third option: allowing applications to access protected data only *when in use* (Table 2 and Figure 2). When the *when in use* mode is selected, the platform only allows an application to access a resource if the application is running in such a way that it is *conspicuous* to the user of the device. We consider the following behaviors conspicuous: (i) the application is running in the foreground (i.e., the user is actively using it), (ii) the application has a notification on the screen, (iii) the application is in the background but is producing audio while the device is unmuted. If these conditions do not hold, then the application is considered *not in use* and access to the resource is denied.

In the default permission model, only the application requesting the resource and resource type are taken into account. Prior work has shown that users do make contextual decisions and disregarding contextual information can cause the AOFU model to fail [26, 37, 38]. The *when in use* option allows the system to take the visibility—which provides a strong contextual cue—into account when making decisions.

3.2 Auditing Automatic Decisions

Although Android currently provides an interface to list the applications that recently accessed location data, similar information is unavailable for other protected resources. Additionally, the existing Android interface does not differentiate between actions that applications take when *in use* and when *not in use*. TurtleGuard's main design objective is therefore to communicate the types of sensitive data that have been accessed by applications and under what circumstances.

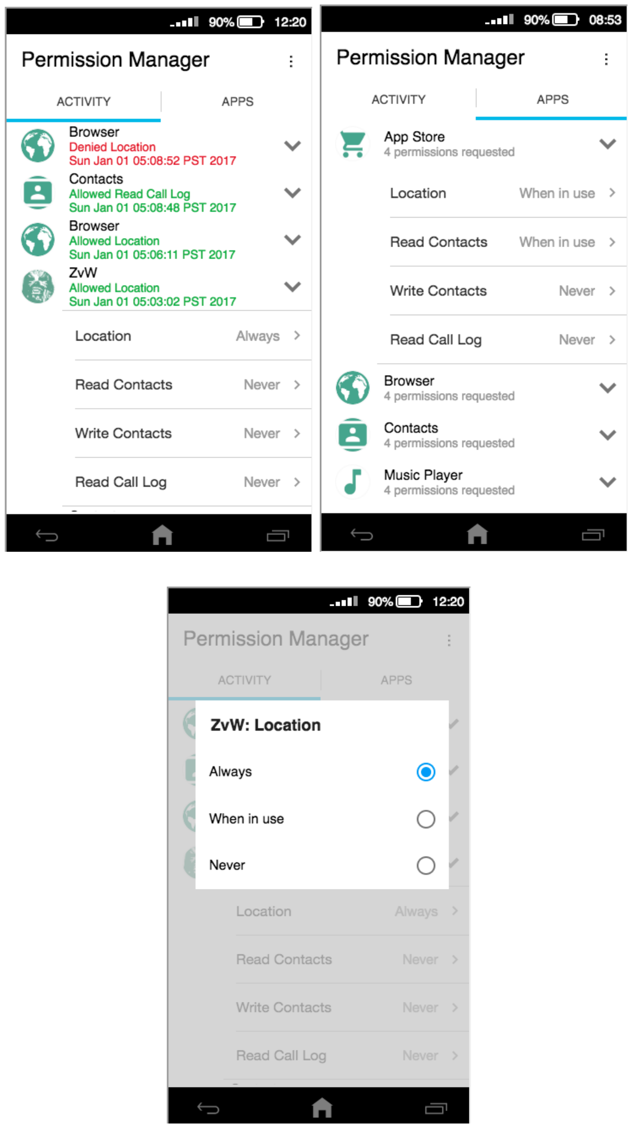
Our initial design of TurtleGuard can be seen in Figure 2. The first tab (labeled "activity") shows all of the recently allowed or denied permission requests, including when those requests occurred and whether the application was in use at the time. Only particularly sensitive permissions (i.e., a list of permissions based on those deemed "dangerous" by the Android platform and those that should have run-time prompts [11]) are displayed in this activity log to avoid overwhelming the user. Table 1 provides our list of permissions.

TurtleGuard presents this information as a running timeline—a log sorted chronologically. A second tab lists all of the apps installed on the phone in alphabetical order, allowing the user to examine what decisions have been made for all permissions requested by a particular app. The user can expand a log entry to change future behavior, if the platform's decision to allow or deny a permission did not align with the user's preferences. When the user uses this interface to change a setting, the classifier is retrained based on the updated information.

3.3 Permission Families

Android uses over 100 permissions and a given resource can have more than one related permission. Felt et al. found that not all the permission types warrant a runtime prompt—it depends on the nature of resource and the severity of potential privacy threat [10]. Consequently, TurtleGuard only manages a subset of permissions (Table 1 in the Appendix) based on those deemed sensitive by prior work and by the latest Android platform. In the original version of TurtleGuard, we had listed the original names of permissions, ungrouped. One of the changes we made as we iterated on our design after our pilot experiment was to implement permission families.

We further simplify permissions by grouping related permissions into a single sensitive resource, so as not to overwhelm users with too many redundant decisions. For example, READ_CONTACTS and WRITE_CONTACTS are grouped into a single CONTACTS permission, or *family*. This means that within TurtleGuard users only see the human-readable


Permission Manager
ACTIVITY APPS

Browser
Denied Location
Sun Jan 01 05:08:52 PST 2017
Contacts
Allowed Read Call Log
Sun Jan 01 05:08:48 PST 2017
Browser
Allowed Location
Sun Jan 01 05:06:11 PST 2017
ZvW
Allowed Location
Sun Jan 01 05:03:02 PST 2017
Location Always
Read Contacts Never
Write Contacts Never
Read Call Log Never

Permission Manager
ACTIVITY APPS

App Store
4 permissions requested
Location When in use
Read Contacts When in use
Write Contacts Never
Read Call Log Never
Browser
4 permissions requested
Contacts
4 permissions requested
Music Player
4 permissions requested

Permission Manager
ACTIVITY APPS

ZvW: Location
[✓ Always]
When in use
Never

Location Always
Read Contacts Never
Write Contacts Never
Read Call Log Never

Figure 2: The pilot design of TurtleGuard listed recent app activity (top left), a list of installed apps and their associated permissions (top right). Permissions can be *always* granted, granted only *when in use*, or *never* granted (bottom).

resource type and not the underlying permissions that are managed by the family. Any changes that a user makes about granting a resource therefore affects all permissions in the same family. For example, there is no longer a distinction between coarse and fine location data; both are either allowed or denied by a location settings change made using the TurtleGuard interface [14].

4. METHODOLOGY

We conducted two online experiments to evaluate the effectiveness of TurtleGuard at providing users with information and control over app permissions, as compared to Android's default permission manager (as of versions 6.0 through 7.1). The first experiment was designed to examine our initial design decisions, as described in the previous section. Based on the results of that experiment, we made changes to our design, and then validated those changes through a second experiment. In both experiments, we asked participants to perform four different tasks using an interactive Android simulation. These tasks involved either retrieving information about an application's prior access to sensitive resources or preventing access in the future (i.e., modifying settings).

In both experiments, we randomly assigned participants to either the *control* or *experimental* conditions. We presented *control* participants with the default permission manager, which is accessible using the Settings app. We presented *experimental* participants with our novel permission manager, TurtleGuard. During our pilot experiment, TurtleGuard was accessible through an icon on the home screen labeled "Privacy Manager," though we added it as a sub-panel to the Settings app prior to the validation experiment (Figure 10 in the Appendix). The questions and tasks for participants were identical for the two conditions and both experiments.

4.1 Tasks

We presented participants with four tasks to complete using an interactive Android simulation: three tasks to retrieve information about permission settings, and one task to modify permission settings. Some of these tasks required participants to find information about a specific app's abilities. In order to avoid biases from participants' prior experiences and knowledge of specific real-world apps, these questions instead focused on a fictitious app, *ZvW*. While we randomized the order of the tasks, we ensured that Task 3 always came before Task 4 (i.e., we never asked them to prevent background location data collection prior to asking them whether background location data was possible). After each task, we asked participants to rate the difficulty of the task using a 5-point Likert scale ("very easy" to "very difficult"). Finally, upon completing all tasks, we asked them several demographic questions and then compensated them $2. We now describe the four tasks in detail.

Task 1: What were the two most recent applications that accessed this device's location?

In this task, we asked participants to use the Android simulation and identify the two applications that most-recently accessed location data. Participants used two open-ended fields to answer this question. In the *control* condition, this task was correctly accomplished by navigating to the "location" screen from within the Settings application (Figure 3). This screen presents information about applications that recently requested location data.

In the *experimental* condition, this task was accomplished by simply studying the "activity" screen, which was displayed immediately upon opening TurtleGuard (Figure 2). Given that this task was already supported by the default permission manager, we wanted to verify that TurtleGuard performed at least as well.

Task 2: Currently, which of the following data types can be accessed by the ZvW application?

In the *control* condition, this was was accomplished by performing the four steps to access the screen in Figure 5 (right): selecting the "Apps" panel within the Settings app (Figure 3, left), selecting the ZvW application, and then selecting the "Permissions." This screen depicted a list of permissions available to the application based on what the application declares as its required permissions; the user is able to fine-tune this by selectively disabling certain permissions using the sliders on this screen. We wanted participants to identify the permissions that were enabled, rather than all of those that *could* be enabled in the future.

In the *experimental* condition, participants could accomplish this task by selecting the "Apps" tab from within TurtleGuard and then expanding the ZvW application to view its requested permissions (Figure 2, top right). In both conditions, the correct answer to the question was that "location" is the only data type that can be accessed by the ZvW application. Again, given that this task was already supported by the default permission manager, we wanted to verify that TurtleGuard performed at least as well.

Task 3: Is the ZvW application able to access location data when it is not being actively used?

We designed this task to determine whether TurtleGuard was effective at communicating to participants in the *experimental* condition the difference between foreground and background data access. Similarly, we wanted to examine whether participants in the *control* condition understood that once granted a permission, an application may access data while not in use. Based on the settings of the simulations, the correct answer across both conditions was "yes."

Participants in the *control* group must navigate to Settings, then the "Apps" panel, and view the list of permissions corresponding to the ZvW application, similar to Task 2. Location is turned on, and so participants must be able to understand that this means that the permission is granted even when it is not actively being used. Participants in the *experimental* condition can use TurtleGuard's "Apps" tab to view the requested permissions for the ZvW application. This shows that the location permission is listed as "always" (Figure 2, top right) and that "when in use" is an unselected option (Figure 2, bottom).

Task 4: Using the simulation, prevent ZvW from being able to access your location when you aren't actively using ZvW (i.e., it can still access location data when it is being used). Please describe the steps you took to accomplish this below, or explain whether you believe this is even possible.

As a follow-up to the third task, the fourth task involved participants explaining the steps that they went through in order to limit background location access, or to explain that it is not possible.

Those in the *experimental* condition could locate and change this permission setting either through the activity timeline or by locating ZvW from the "Apps" tab (Figure 2). We marked answers correct that specifically mentioned changing the setting to "when in use."

Those in the *control* condition could not prevent this access. We marked responses correct if they indicated that this task was impossible to complete. Two coders independently reviewed the responses to this task (Cohen's $\kappa = 0.903$). The objective of this task was to see how successful TurtleGuard is at allowing participants to vary settings based on application use (a strong contextual cue) and to examine whether participants knew that this was not possible when using the default permission manager.

4.2 UI Instrumentation

We built interactive HTML5 simulations of the UI designs described in the previous section, developed using *proto.io*. We instrumented the simulations to log all user interactions (e.g., panels visited, buttons clicked, etc.). This data allowed us to analyze how participants navigated the UI to accomplish their tasks.

4.3 Qualitative Data

In addition to analyzing the participants' responses to the four tasks, their perceived difficulty of each of the tasks, and their demographic information, we also collected responses to two open-ended questions:

Thinking about the tasks that you performed in this survey, have you ever wanted to find similar information about the apps running on your smartphone?
We coded participants' responses as a binary value. Responses indicating sentiments such as "yes" and "I always wanted that" were coded as true. Clear negative answers and weak affirmative answers such as "sometimes" and "maybe" were coded as false. The purpose of this question is to see how prevalent seeking information is in the real world.

Thinking about the simulation that you just used, what could be done to make it easier to find information about how apps access sensitive information?
We coded participants' responses in multiple ways. First, as binary values indicating contentment with the presented design. Responses that affirmed that the user would change nothing about the presented design was coded as true. Any complaint or proposed suggestion was coded as false, as well as responses with uncertainty, confusion, or ambivalence (e.g., "I don't know"). We further coded their responses for those that had specific suggestions, using tags for the different themes of their suggestions.

Each response was coded by two experienced coders working independently, who then compared responses and recorded the number of conflicts. The coders discussed and reconciled the differences using the stricter interpretation given the tasks. This produced the final coding of the data, which is used in our analysis.

5. PILOT EXPERIMENT

Using the methodology outlined in the previous section, we recruited 400 participants from Amazon's Mechanical Turk for a pilot experiment. We discarded 8 incomplete sets of responses, leaving us with 392 participants. Our sample was biased towards male respondents (65% of 392), however, a chi-square test indicated no significant differences between genders with regard to successfully completing each task. Disclosed ages ranged from 19 to 69, with an average age of 33. In the remainder of this section, we describe our results for each task, and then describe several changes we made to TurtleGuard's interface as a result of this initial experiment. In all of our tasks we also asked participants to evaluate perceived difficulty using a 5-point Likert scale.

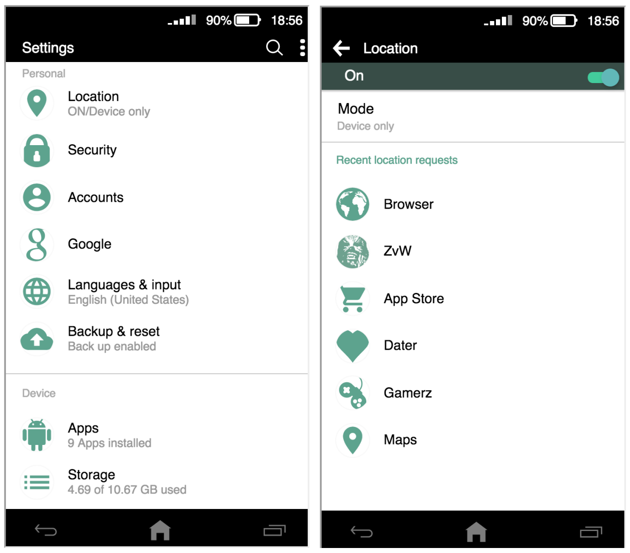


Figure 3: In Task 1, participants in the *control* condition could identify the most recent applications that requested location data from within the Settings application. This was also a valid method for Task 1 in the *experimental* condition for the validation study.

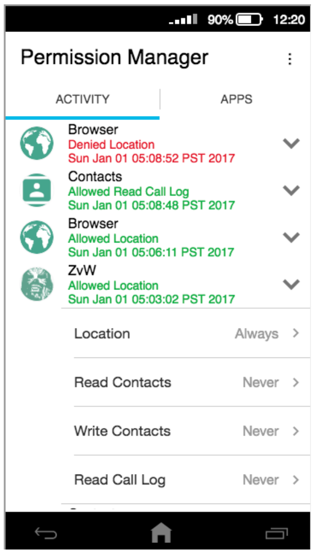
5.1 Task 1: Recent Location Access

In the *control* condition, 84% of participants (167 out of 198) correctly completed this task, whereas only 68% (132 out of 194) completed it correctly in the *experimental* condition. This difference was statistically significant ($\chi^2 = 14.391$, $p < 0.0005$), though with a small-to-medium effect size ($\phi = 0.192$). In both cases, answers were marked correct if they mentioned both the Browser and ZvW applications (Table 3). We examined the incorrect responses from the *experimental* group. Of the 49 participants in this condition who tried but failed, 13 never opened TurtleGuard, and over 73% (36 of 49) entered "Browser" and "Contacts", which were the first two applications listed (Figure 4) in the activity tab of the Permission Manager. The activity tab showed recent resource accesses in a chronological order—"Browser" has been denied a *location* request and "Contact" has successfully accessed *call logs*.

Participants were unable to comprehend that the activity log was not all about location. This confusion might also stem from their familiarity with the location access panel in stock Android, where only location access requests are presented in chronological order. We hypothesize that this confusion is addressable by redesigning the activity log to better distinguish between data types and allowed versus denied permission requests. One possible solution is to create separate tabs for allowed and denied requests, as well as to group similar data types together (rather than presenting all activity in chronological order).

5.2 Task 2: Finding Granted Permissions

In the second task, we asked participants to list all of the data types that the ZvW application *currently* had access to. We observed that 140 participants in the *control* condition (70.7% of 198) and 116 participants in the *experi-*



Permission Manager

ACTIVITY APPS

Browser
Denied Location
Sun Jan 01 05:08:52 PST 2017

Contacts
Allowed Read Call Log
Sun Jan 01 05:08:48 PST 2017

Browser
Allowed Location
Sun Jan 01 05:06:11 PST 2017

ZvW
Allowed Location
Sun Jan 01 05:03:02 PST 2017

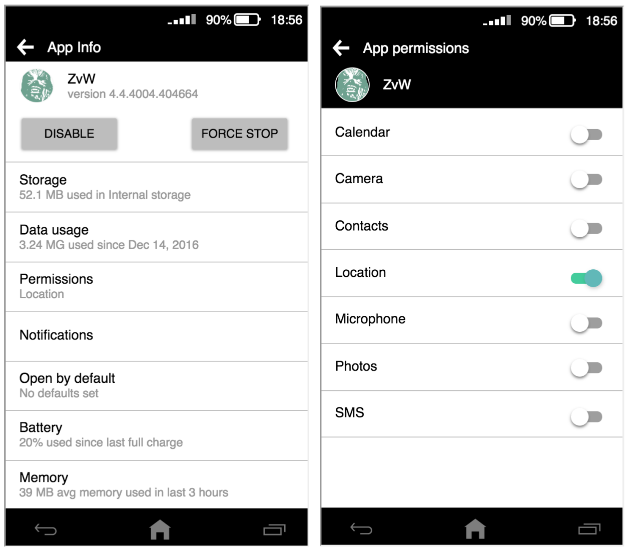
Location	Always
Read Contacts	Never
Write Contacts	Never
Read Call Log	Never

Figure 4: Browser and contacts listed first; a likely reason for incorrect answers in Task 1.

Condition	Correct	Incorrect
Task 1		
control	167 (84%)	31 (16%)
experimental	132 (68%)	62 (32%)
Task 2		
control	140 (71%)	58 (29%)
experimental	116 (60%)	78 (40%)
Task 3		
control	86 (43%)	112 (57%)
experimental	153 (79%)	41 (21%)
Task 4		
control	47 (24%)	151 (76%)
experimental	144 (75%)	49 (25%)

Table 3: Participants in each condition who performed each task correctly during the pilot experiment.

mental condition (59.8% of 194) performed this task correctly. After correcting for multiple testing, this difference was not statistically significant ($\chi^2 = 5.151$, $p < 0.023$). However, despite the lack of statistical significance, we were surprised that not more people in the *experimental* condition answered correctly. Upon investigating further, we noticed several confounding factors that might have made this task more difficult for people in this condition. First, while the *control* condition displays the currently allowed permissions as grayed-out text on the "App Info" page (Figure 5), the *experimental* condition lists all *requested* permissions—which is a superset of the allowed permissions (top-right of Figure 2). Second, we noticed that due to an experimental design error, the permissions requested by the ZvW app in the *experimental* condition included several that are not included in the options presented to participants (e.g., "Write Contacts" and "Read Call Log"). These were were also different from the permissions seen by the control group. This may have made this task confusing for these participants.



App Info

ZvW
version 4.4.4004.404664

DISABLE FORCE STOP

Storage
52.1 MB used in Internal storage

Data usage
3.24 MG used since Dec 14, 2016

Permissions
Location

Notifications

Open by default
No defaults set

Battery
20% used since last full charge

Memory
39 MB avg memory used in last 3 hours

App permissions

ZvW

Calendar
Camera
Contacts
Location
Microphone
Photos
SMS

Figure 5: In Task 2, participants in the *control* condition could identify the permissions granted to the ZvW application by selecting the "Apps" panel from within the Settings application, and then selecting the application, followed by the "Permissions" panel.

5.3 Task 3: Finding Background Activity

In the third task, we asked participants whether the ZvW application had the ability to access location data while not actively being used. We observed that 86 participants in the *control* condition (43% of 198) correctly answered this question, as compared to 153 participants in the *experimental* condition (78% of 194). This difference was statistically significant ($\chi^2 = 51.695$, $p < 0.0005$) with a medium effect size ($\phi = 0.363$). Thus, the new dashboard interface successfully differentiated between foreground and background permission usage.

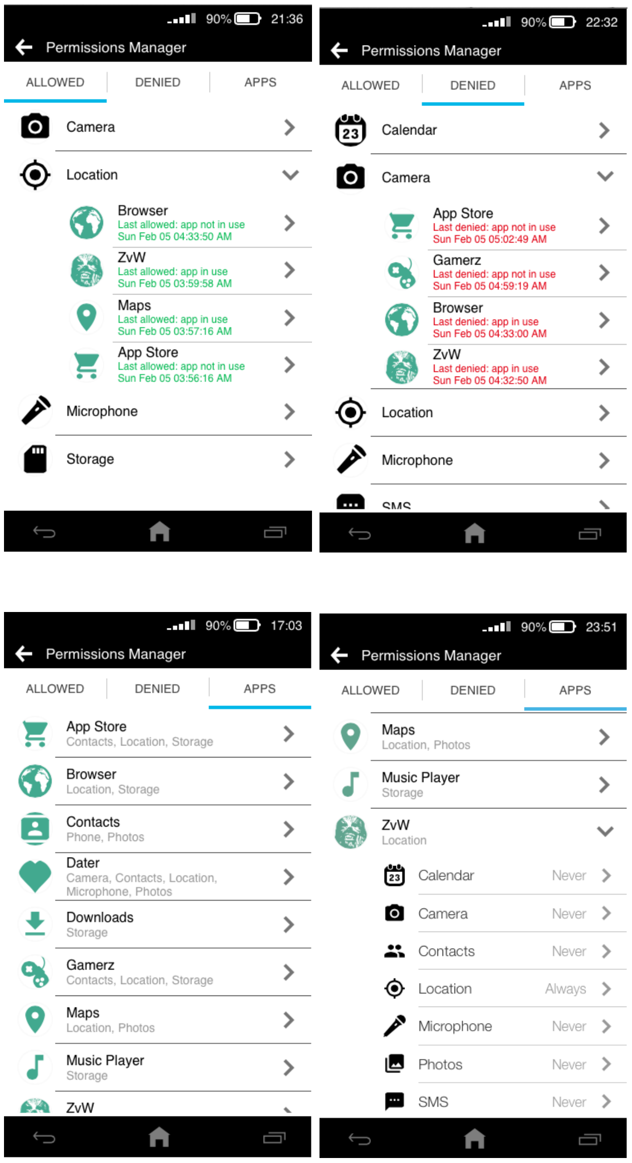
5.4 Task 4: Limiting Background Activity

We observed that only 47 participants in the *control* condition (23% of 198) correctly stated that this task was impossible. In the *experimental* condition, 144 (74% of 193)[2] clearly articulated the steps that they would go through using the privacy dashboard to change location access from "always" to "when in use." This difference was statistically significant ($\chi^2 = 101.234$, $p < 0.0005$) with a large effect size ($\phi = 0.509$).

5.5 Design Changes

Based on the results of our first two tasks, in which participants in the *control* condition were more likely to correctly locate information about recent app activities and the permissions that apps had requested, we made several design changes to the TurtleGuard interface. First, we split the activity timeline into two separate tabs: recently allowed permission requests, and recently denied permission requests. Second, rather than showing all activity in chronological order, the activity timeline is now categorized by resource

[2]One person could not load the `iframe` containing the simulation during this task.


90% 21:36
Permissions Manager
ALLOWED DENIED APPS
Camera
Location
Browser
Last allowed: app not in use
Sun Feb 05 04:33:50 AM
ZvW
Last allowed: app in use
Sun Feb 05 03:59:58 AM
Maps
Last allowed: app in use
Sun Feb 05 03:57:16 AM
App Store
Last allowed: app not in use
Sun Feb 05 03:56:16 AM
Microphone
Storage

90% 22:32
Permissions Manager
ALLOWED DENIED APPS
Calendar
Camera
App Store
Last denied: app not in use
Sun Feb 05 05:02:49 AM
Gamerz
Last denied: app not in use
Sun Feb 05 04:59:19 AM
Browser
Last denied: app in use
Sun Feb 05 04:33:00 AM
ZvW
Last denied: app in use
Sun Feb 05 04:32:50 AM
Location
Microphone
SMS

90% 17:03
Permissions Manager
ALLOWED DENIED APPS
App Store
Contacts, Location, Storage
Browser
Location, Storage
Contacts
Phone, Photos
Dater
Camera, Contacts, Location, Microphone, Photos
Downloads
Storage
Gamerz
Contacts, Location, Storage
Maps
Location, Photos
Music Player
Storage
ZvW

90% 23:51
Permissions Manager
ALLOWED DENIED APPS
Maps
Location, Photos
Music Player
Storage
ZvW
Location
Calendar Never
Camera Never
Contacts Never
Location Always
Microphone Never
Photos Never
SMS Never

Figure 6: TurtleGuard separates recently allowed (top left) and denied (top right) permissions. The "Apps" tab lists the allowed permissions of all apps (bottom left). Expanding an app allows the user to make changes (bottom right).

type, with the events for each resource type sorted chronologically). These changes can be seen in the top of Figure 6.

In addition to these changes, we also modified the apps tab to show grayed-out allowed permissions for each app, similar to the App Info panel in the default permission manager. Due to the error we noted in the *experimental* condition in Task 2, we made sure that all app permissions were the same in both conditions.

Finally, we integrated TurtleGuard into the stock Settings app, so that it appears as a panel labeled "Permissions Manager" (Figure 10 in the Appendix). For consistency, when participants in the *experimental* condition select the "Permissions" sub-panel from within the "App Info" panel (Figure 5, left), they are now redirected to TurtleGuard's "Apps" panel, pre-opened to the app in question (Figure 6, bottom right).

Condition	Correct	Incorrect
Task 1		
control	237 (82.6%)	50 (17.4%)
experimental	241 (82.5%)	52 (17.5%)
Task 2		
control	232 (77.1%)	55 (22.9%)
experimental	226 (80.8%)	67 (19.2%)
Task 3		
control	108 (37.6%)	179 (62.4%)
experimental	230 (78.5%)	63 (21.5%)
Task 4		
control	79 (27.5%)	208 (72.5%)
experimental	224 (76.5%)	69 (23.5%)

Table 4: Participants in each condition who performed each task correctly during the validation experiment.

6. VALIDATION EXPERIMENT

Following our pilot experiment and subsequent design changes, we performed a validation experiment. In the remainder of this section, we discuss our results (Table 4).

6.1 Participants

Because of several known biases in Mechanical Turk's demographics [29, 32, 23], we decided to compare a sample of 298 Mechanical Turk participants to a sample of 300 Prolific Academic participants. Peer et al. recently performed several studies on various crowdsourcing platforms and concluded that the latter yields more diverse participants [30]. We limited both groups to participants based in the U.S., over 18, owning an Android phone, and having a 95% approval rating on their respective platform. After removing 18 incomplete responses, we were left with a combined sample of 580 participants. We analyzed the results from the two groups, and discovered that the high-level findings (i.e., task performance) did not observably differ. For the remainder of our study, we therefore discuss the combined results. Our sample was biased towards male respondents (63% of 580), however, a chi-square test indicated no significant differences between genders with regard to successfully completing each task. Disclosed ages ranged from 19 to 74, with an average age of 33. Participants performed the same tasks as those in the pilot experiment.

6.2 Task 1: Recent Location Access

Recall that in this task, we asked participants to identify the two most recent applications that accessed location data. For the *experimental* condition, in addition to using the same method as the *control* (navigating to the "Location" sub-panel of the Settings app), participants could navigate to the "Allowed" tab within TurtleGuard, and then examine the "Location" permission to see the two most recent accesses (top left of Figure 6). In the *control* condition, 237 participants (82.6% of 287) correctly completed this task, whereas 241 (82.5% of 293) completed it correctly in the *experimen-*

tal condition. A Wilcoxon Rank-Sum test revealed that this difference was not statistically significant ($p < 0.918$).

Examining our instrumentation, we observed that most of the participants in both conditions used the default method of accomplishing this task (i.e., accessing the Location subpanel): 80.1% of those who answered correctly in the *experimental* condition and 92.8% of those in the *control* condition. (There were fifteen participants in the *control* condition who answered correctly despite not accessing the panel—likely by random guessing, and two who answered correctly by exhaustively searching the "App Info" panels of installed apps, to see which had been granted the location permission; 48 participants in the *experimental* condition used TurtleGuard to yield the correct answer.)

A total of 102 participants incorrectly answered the question in Task 1. Of the incorrect responses, five participants failed to properly navigate the emulator and wrote that the emulator was broken or the buttons did not work; 9 participants simply left it blank or wrote that they did not know. From the other 88 participants who provided responses, 38 (43%) listed "App Store" as one of their selections, making it the most-frequent error.

More specifically, 18 of the participants listed their answer as "App Store" and "Browser". We believe that this is because both the stock Android Apps Manager and TurtleGuard's "Apps" tab (Figure 6, bottom) sorts the entries alphabetically, and by looking at the permissions available to both these apps, the participant would see that both have location access. Nevertheless, they are not the most *recent* apps to access location data.

Overall, these results suggest that the changes we made after our pilot experiment resulted in marked improvements. We further investigated this by examining participants' perceived ease-of-use, as measured using the 5-point Likert scale ("very easy (1)" to "very difficult (5)"). In the *experimental* condition, 84 participants accessed TurtleGuard to complete this task (regardless of whether or not they answered correctly). We compared these 84 responses with the 463 responses from participants who only used the default Settings panel (i.e., 195 in the *experimental* group and 268 in the *control* group). The median responses from both groups was "easy" (2), however there was a statistically significant difference between the groups (Wilcoxon Rank-Sum test: $Z = -3.9605$, $p < 0.0005$), with a small effect size ($r = 0.17$)—participants who used TurtleGuard found it more difficult compared to the default Settings panel. This difference appears to be due to those who performed the task incorrectly: the median response for TurtleGuard users who answered incorrectly was "difficult (4)," whereas it was "neutral (3)" for other participants. This may actually be a good thing: participants who confidently answered incorrectly are at greater risk due to over confidence, whereas those who had difficulty may be more likely to seek out more information.

6.3 Task 2: Finding Granted Permissions

In this task, participants had to locate the app's allowed permissions to discover that "location" was the only allowed permission in both the *experimental* and *control* conditions. This could be accomplished by viewing TurtleGuard's Apps tab (bottom of Figure 6) for those in the *experimental* condition, or by viewing an app's App Info panel from within the Settings app (Figure 5), which was available to those in either condition.

In total, 458 participants correctly performed this task (79% of 580). Table 4 displays the breakdown of the results by condition. A chi-square test did not yield statistically significant results between the two conditions in terms of task completion ($\chi^2 = 0.984$, $p < 0.321$).

Of the 226 *experimental* condition participants who performed the task correctly, 127 (56.2%) did so by using TurtleGuard. In total, 145 *experimental* condition participants accessed TurtleGuard, and reported a median task difficulty of "easy (2)." This did not significantly differ from the 375 other participants in both conditions who only examined the default Settings panels to perform the task and also reported a median difficulty of "easy" ($Z = 1.808$, $p < 0.238$); 60 participants never opened Settings (10 of whom answered the question correctly, likely due to random guessing).

6.4 Task 3: Finding Background Activity

To perform this task, participants in the *control* group had to navigate to Settings, then the "Apps" panel, and view the list of permissions corresponding to the ZvW application (Figure 5). However, performing this sequence of steps still did not guarantee they would answer the question correctly: they needed to observe that location data was allowed, as well as understand that this meant that location data could be accessed by the app even when it is not actively being used. Participants in the experimental condition answered this question through TurtleGuard, which shows that the location permission was listed as "Always" (Figure 6), thereby eliminating the ambiguity.

We observed that 230 *experimental* condition participants answered this question correctly (78.5% of 293), as compared to only 108 *control* participants (37.6% of 287). A chi-square test showed that this difference was significant ($\chi = 97.914$, $p < 0.0005$) with a medium-to-large effect size ($\phi = 0.414$). This observation corroborates Thompson et al.'s findings [36] that users are largely unaware that apps can access sensitive data when not in use. TurtleGuard, however, was more effective at communicating this information to participants. Among the participants in the *experimental* condition, 24.57% took the extra step to click on the location entry (bottom right of Figure 6) to see the other options available (Figure 2): *always*, *when in use*, and *never*.

Our instrumentation showed that 129 participants used TurtleGuard to perform this task, which suggests that 101 (34.5% of *experimental* condition participants) still got it correct either based on prior knowledge—a proportion consistent with Thompson et al.'s findings [36]—or after having used TurtleGuard in preceding tasks. There were 383 participants who completed the task by examining existing areas of the Settings app, whereas 68 participants never bothered to open Settings to complete this task. The median ease of use for those who used TurtleGuard was "easy (2)", while the median ease of use for those who used the default permission manager was "neutral (3)". This difference was statistically significant ($Z = -2.885$, $p < 0.004$) with a small effect size ($r = 0.13$). Participants in the *control* condition also took significantly longer to complete the task: 49.63 seconds versus 26.65 seconds. A Wilcoxon Rank-Sum test found this difference to be statistically significant ($Z = -5.239$,

$p < 0.0005$, $r = 0.22$).

6.5 Task 4: Limiting Background Activity

Recall that Task 4 asked participants to describe the steps to prevent an application from accessing location information while the application was not in use, or to state that it is not possible to prevent it. It is only possible to prevent it using TurtleGuard.

In the *experimental* condition, 224 (76.5% of 293) explicitly stated how they would use TurtleGuard to change the permission to "when in use",[3] whereas only 79 (27.5% of 287) *control* group participants correctly stated that this task was impossible using the default permission manager. This difference was statistically significant ($\chi^2 = 137.14$, $p < 0.0005$) with a large effect size ($\phi = 0.49$).

A majority of the participants (72.5%) in the *control* group incorrectly believed that they could vary their decisions based on the *visibility* of the application. The most common responses involved disabling location data altogether, preventing the app from running, or restricting "background data":

- Settings > Apps > ZvW > Toggle Location Off
- Disable or Force Stop the Application
- Settings > Location > ZvW > Permissions > Toggle Location Off
- Settings > Apps > ZvW > Data Usage > Restrict Background Data
- Settings > Location > Toggle Location Off

A considerable portion (14%) chose to "restrict background data," which does something else entirely: it prevents data surcharges while roaming on foreign networks. This is another example of a disconnect between users' mental models and the true meaning of these configuration options. That said, a small number of participants in the *control* condition correctly stated that they would need to disable the app's location permission, and then re-enable it every time they wanted to use that app, a tedious process that is prone to forgetfulness—we treated this response as correct. Another substantial portion in the control condition (46%) wanted to block the location globally (from the default location panel) or block the location access from ZvW app entirely. While this is an overly restrictive option compared to *when in use*, this the closest option provided in Android—we treated this as an incorrect response.

As expected, participants in the *control* condition rated the difficulty of this task as "neutral (3)", whereas the median Likert score from those in the *experimental* condition was "easy (2)". This difference was statistically significant with a large effect size ($p < 0.0005$, $\phi = 0.49$). The participants in the *control* condition who successfully completed the task (e.g., by acknowledging it was impossible) struggled immensely with it, evaluating it as "difficult (4)".

7. USER PERCEPTIONS

After completing the four tasks, participants answered two open-ended questions about whether they have looked for this type of permission information in the past, and whether they have any suggestions to offer us about the design of the interface they had just used. Two researchers independently coded each question and then resolved conflicts. Cohen's inter-rater reliability score (κ) is provided for the coding with each statistic.

[3]We used a very conservative rubric: 11 participants who described using TurtleGuard, but did not explicitly use the phrase "when in use," were coded as being incorrect.

7.1 Prior Experiences

Our first question asked: *Thinking about the tasks that you performed in this survey, have you ever wanted to find similar information about the apps running on your smartphone?*

Our goal was to determine whether or not participants had previously thought about resource access or configuring contextual privacy preferences, and therefore if having such features would be beneficial. On average, 63.1% of participants stated that they had thought about this (Cohen's $\kappa = 0.792$), and the experimental condition they were in proved to be insignificant. We did, however, observe a positive correlation between performance on the four tasks (i.e., number of tasks performed correctly) and reporting having previously thought about these issues ($p < 0.007511$, $r = 0.155$).

Among the people who chose to be more detailed in their responses, several themes emerged. A large portion mentioned that the reason they have tried these tasks before is that they wanted to be able to exert more control over their installed apps:

- "*I was somewhat familiar with these menus already before starting this task. I like to have control over my app permissions including location and data management.*"
- "*Yes, I've often wanted a little more control over what my apps get to access*"

A minority of users expressed their frustrations on how the current default user interfaces in Android were confusing and did not let them set privacy preferences the way they wanted to:

- "*Yes but usually can't find anything on there either like these. So I gave up trying.*"
- "*Yes. I want to know what they collect, although it gets tedious to try to figure it all out. Sometimes I'd rather just ignore it.*"
- "*Yes. I haven't had enough time to play with various apps and/or permissions to understand their functionality without certain permissions. I feel like I've been forced to assume certain permissions to ensure the app in question works correctly.*"

These comments highlight the fact that many users want to have control over resource usage by applications, and that many feel helpless to do so, given the options offered by current privacy management interfaces. These observations further strengthen the need for a more usable interface that will help people to feel more empowered.

7.2 Suggestions

The second question asked: *Thinking about the simulation that you just used, what could be done to make it easier to*

	Changes	**No Changes**
control	245 (85.4%)	42 (14.6%)
experimental	187 (63.8%)	106 (36.3%)

Table 5: Whether participants believed changes were needed to the interfaces they used during the validation study.

find information about how apps access sensitive information?

This question has two purposes: (i) to gather specific design recommendations from participants who used TurtleGuard; (ii) to get general suggestions from participants who used the default permission manager.

In total, 66.03% of participants (383 of 580) suggested at least some change or improvement (Cohen's $\kappa = 0.896$). Table 5 shows the breakdown of how many participants in each condition prefer a change in the dashboard within their condition. A chi-square test shows a statistically significant association between a participant's condition and whether the participant wanted changes in how privacy is managed ($p < 0.00005, \phi = 0.237$). This suggests the participants in the *experimental* condition are more satisfied with the controls provided by the new design than those in the *control* condition. Our work aims to fill the need users have regarding control over permissions and their personal privacy.

The most common suggestion (32.24% of all participants) was to reduce the number of layers to the actual permission interface (Cohen's $\kappa = 0.603$). Participants complained about number of different interfaces they had to traverse before reaching the actual permission interface. Many participants suggested that they prefer to reach respective the permission control interface directly through the application—either as part of the application or by pressing the app icon. TurtleGuard addresses this frequent concern by providing a path to permission management that involves fewer clicks and centralizes all permission functionalities.

- "*Streamline the interface to require less touches to find the information about permissions and make it explicit as to what type of data would be collected if allowed.*"
- "*Perhaps have an easier way to access the app's settings, such as holding onto an app's icon will bring up its specific settings.*"
- "*Make each app itself have the option to find that information instead of going to the general phone settings.*"
- "*There should be one centralized location, or maybe an app for that. Just to toggle with these very important settings.*"
- "*Most of the tasks are easily found, but I would like a setting that plainly shows whether the app can access certain permissions only while it is being actively used.*

Seven participants thought having a log of recent resource usage by applications would be useful. Some went further, mentioning that the log should also provide contextual cues, such as the visibility of the application at the time it makes the request. This finding provides evidence in support of Liu et al. [21] that recent statistics help users make better decisions. TurtleGuard provides this functionality with a surrounding context, where the new interface shows all the recent resource requests along with (i) the decision that platform took on behalf of the users, (ii) the time that the decision was made, and (iii) the visibility of requesting application.

- "*It would be useful to have a dashboard which shows which apps are accessing what and when. Being able to see a log of the actual data that was accessed would also be useful.*"
- "*A log could be provided as an option in the settings that shows all times an app accessed sensitive information.*"

A few participants (14.6%) also suggested that there should be a tutorial, wizard style guide, or a FAQ to explain how to manage permissions (Cohen's $\kappa = 0.651$). Some wanted the applications to explain *why* they need access certain resources. Some even suggested runtime prompts for every sensitive request access. The highlight of the responses was the suggestion to hold a YouTube Q&A session on resource usage after each release.

- "*As the app is being introduced to the users, they should make a youtube q&a to answer any simple questions like this.*"
- "*Perhaps a wizard-style guide that can walk people through the process.*"

Prior work has already shown that having runtime prompts on every sensitive request is not feasible [37]—we believe that a log of recent resource accesses with surrounding context is the closest practical solution, which TurtleGuard provides.

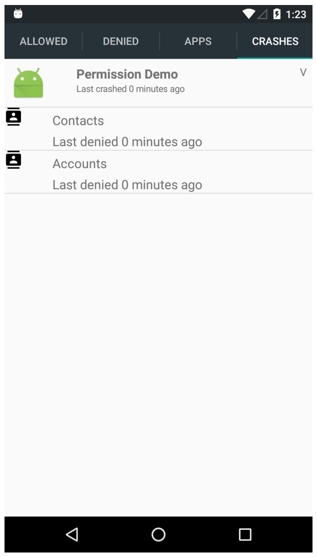
Interestingly, some of these suggestions show a lack of understanding of the Android permissions model. Some participants suggested an integration into the actual application itself, which is impossible until Android forces an API onto the developers and reinforces their policies. Furthermore it would fall into the hands of third party application developers to honestly enforce the permissions requested. Leaving permission enforcement in the hands of third party developers is currently impossible and unreliable. Another user recommended having a notification for every single resource request by every application:

- "*Make a pop-up every time an app is trying to access information.*"
- "*PROVIDE A NOTIFICATION EVERY TIME AN APP IS ACCESSING SUCH DATA.*"

Studies have shown that there would be one sensitive permission request every 6 seconds per participant–and that would only be for *sensitive* permissions [38]. Notifying users of every permission request would be infeasible and likely cause annoyance to the users.

8. FURTHER IMPROVEMENTS

We plan to deploy TurtleGuard as part of a field study. Our goal is to evaluate how TurtleGuard performs in the wild and

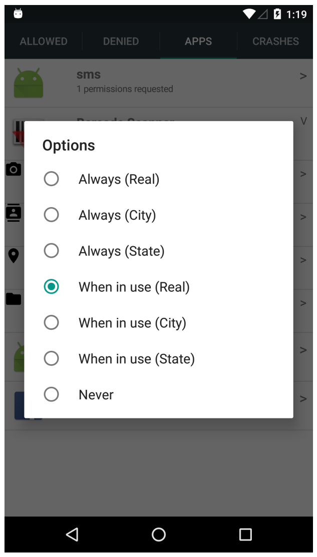


Permission Demo
Last crashed 0 minutes ago

Contacts
Last denied 0 minutes ago

Accounts
Last denied 0 minutes ago

Figure 7: Crash Reporting panel



Options

- Always (Real)
- Always (City)
- Always (State)
- [✓ When in use (Real)]
- When in use (City)
- When in use (State)
- Never

Figure 8: New location granularity options

further improve on the existing model. To give users more fine-grained control and provide them with more detailed application information we made modifications to TurtleGuard that add the following features:

- Crash Reporting
- Location Granularity
- Statistics Reporting

8.1 Crash Reporting

Many third party applications are not equipped for dealing with permission denials. If a user chooses to deny a permission, our goal is to minimize the functionality loss of the application by giving the application fake data. For example, if an application is expecting a location coordinate, we feed them a random or nearby location coordinate, protecting the users' real one.

By feeding third party applications fake data, there is always the possibility that the application will not accept the fake data. This could be either the fake data is not recognized (invalid value), or that apps may even detect fake data and crash when it suspects it. Upon an application's crash, it is imperative that users have a method to fix such a crash. To address this in a future design, we give users an interface through which they can understand and remedy crashes caused by permission denials.

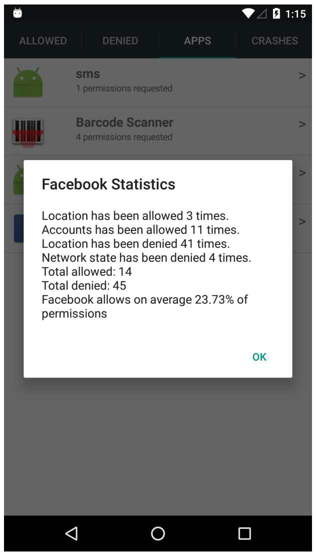
TurtleGuard will feature *crash reporting* (Figure 7). This tab lists the applications that have recently crashed *and* gives users the option to fix it by keeping track of the application's recently denied permissions. Recently denied permissions are the most likely culprit that caused an application to crash, and therefore are the only ones listed. If there were not any denied permissions, we do not list the crash incident under the "Crashes" tab because it cannot be fixed by Turtle Guard and is not a problem caused by feeding the application fake data. If no permission was denied, then no fake data was inputted. Upon clicking on a permission, TurtleGuard attempts to fix the crash. It does so by "allowing" the selected permission for that application. This will cover the scenarios where i) an application considers our fake data invalid, and ii) an application detects our fake data and refuses to accept it by crashing. Upon allowing the permission, the permission will be removed from the list of permissions that may have resulted in the application crash. This ensures that a user will not continuously try to fix a permission that has already been set to allow.

8.2 Location Granularity

In the validation experiment, location was treated in the same way as other permissions. Many mobile privacy studies in the past have focused specifically on the location permission [4, 8, 34]. Location is a sensitive permission as deemed by the users, and therefore should require more attention. To this end, we add additional granularity options to the location permission:

- **Always (Real)**: always allow user's exact location
- **Always (City)**: always allow user's city location
- **Always (State)**: always allow user's state location
- **When in use (Real)**: allow user's exact location when in use
- **When in use (City)**: allow user's city location when in use
- **When in use (Never)**: allow user's state location when in use
- **Never**: never allow user's location in any granularity

These options are reflected in Figure 8. By allowing users to have more control over their location, they are still able to retain application functionality while also protecting their privacy. For example, an application that reports weather does not need your exact location to work: the city level granularity would function just the same. Many applications base the information they present to you based on your general location, rather than exact one. This would help users protect their privacy by still allowing for full functionality of the application.



Facebook Statistics

Location has been allowed 3 times.
Accounts has been allowed 11 times.
Location has been denied 41 times.
Network state has been denied 4 times.
Total allowed: 14
Total denied: 45
Facebook allows on average 23.73% of permissions

Figure 9: Statistics display

8.3 Statistics Reporting

Users are frequently uninformed regarding application resource usage [13, 19]. We aim to inform users regarding application permission usage, including request rates and types of permissions requested. We accomplish this by keeping a record of every time a permission is denied. We do this with various levels of granularity such as just the permission, permission and application, or permission, application, and visibility. This allows users to see the average denial rates, and how many times a particular permission is requested by the application. Not only does this bring awareness to the user, it could also help users in making their decisions regarding whether or not they want to allow a permission for a particular application and under what circumstances.

9. DISCUSSION

Our primary goal is to empower users to make privacy decisions better aligned with their preferences and to keep them informed about how third-party applications exercise granted permissions, and under what circumstances. We performed iterative design on a new permissions management interface, TurtleGuard, which offers users significant improvements in their ability to control permissions when compared to the default permission manager.

Auditing Automated Decision Making Recent research uses machine-learning techniques to automatically predict users' permission preferences [38, 21, 20, 22]. While machine-learning (ML) techniques have been shown to be better at predicting users' preferences [38], they are still prone to errors.

If systems are going to use ML in the future, there must be mechanisms for users to audit the decisions made on their behalves. We believe that the design we present in our study is a critical first step towards achieving that goal. Participants using TurtleGuard are better able to understand and control *when* apps have access to sensitive data than participants using the default permission manager. A substantial portion of participants mentioned the desire to have a log that they could use to see how each application accesses sensitive resources—functionality that is missing in the default permission manager, but is provided by TurtleGuard.

Correcting Mental Models In Task 4, we asked participants to turn off location permissions when the example app, ZvW, was not actively being used, or to explain that this was not possible. We found that 72.5% of the participants in the *control* condition incorrectly believed that this was possible. Analyzing the different paths that participants in the *control* condition took while using the Android simulation, it was evident that the majority of participants did not understand the permission interface's provided functionality (or the limits of that functionality). This mismatch between users' mental models and actual functionality may lead to users into believing that they have denied access to certain requests for sensitive data despite the fact that applications can access the data.

Privacy Nudges Previous work investigated ways to nudge users to configure their privacy settings and make them aware of how applications access their data [21, 15, 18]. While helping motivate users to use TurtleGuard (and other privacy interfaces) is important, it is out of scope for this work. Nevertheless, our survey results shows that 63.1% of participants—independent of condition—previously searched for permission information on their smartphones. This shows that users are keen to understand how applications use their sensitive resources, and interfaces similar to the one we present in this study fill a critical need.

Conclusion Android's existing permission models, ask-on-install and ask-on-first-use, are insufficient at fulfilling users' privacy desires and needs. Neither of the existing models factor context into their decisions, as they are a binary allow-or-deny decision. Users want to protect their sensitive information, but have a hard time understanding when access to data is and is not being allowed. TurtleGuard adds both ease of use and functionality, including the ability to consider application visibility when specifying privacy preferences, which has been shown to be a strong contextual cue. We recruited participants to perform a study of TurtleGuard: we had participants perform permission-related tasks and compared the performance of participants using TurtleGuard with a control group using the default permission manager. Based on our results, we iterated on TurtleGuard's design, and then performed a validation experiment to confirm those changes. Our results show that users are significantly better at performing tasks with TurtleGuard than the default permission manager and offered fewer suggestions for improving it.

10. REFERENCES

[1] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proc. of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 787–796. ACM, 2015.

[2] P. Andriotis and T. Tryfonas. Impact of user data privacy management controls on mobile device investigations. In *IFIP International Conference on Digital Forensics*, pages 89–105. Springer, 2016.

[3] Apple. About privacy and location services for ios 8 and later. `https://support.apple.com/en-us/HT203033`.

Accessed: March 4, 2017.

[4] L. Barkhuus. Privacy in location-based services, concern vs. coolness. In *Workshop on Location System Privacy and Control at MobileHCI '04*, Glasgow, Scotland, 2004.

[5] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proc. of the 2006 IEEE Symposium on Security and Privacy*, SP '06, Washington, DC, USA, 2006. IEEE Computer Society.

[6] M. Bokhorst. Xprivacy. `https://github.com/M66B/XPrivacy`, 2015.

[7] CollegeDev. Donkeyguard. `https://play.google.com/store/apps/details?id=eu.donkeyguard`, 2014.

[8] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge. Location disclosure to social relations: why, when, & what people want to share. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '05, pages 81–90, New York, NY, USA, 2005. ACM.

[9] S. Egelman, A. P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *The 2012 Workshop on the Economics of Information Security (WEIS)*, 2012.

[10] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[11] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner. How to ask for permission. In *Proc. of the 7th USENIX conference on Hot Topics in Security*, Berkeley, CA, USA, 2012. USENIX Association.

[12] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proc. of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile devices*, SPSM '12, pages 33–44, New York, NY, USA, 2012. ACM.

[13] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *Proc. of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. ACM.

[14] H. Fu and J. Lindqvist. General area or approximate location?: How people understand location permissions. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 117–120. ACM, 2014.

[15] H. Fu, Y. Yang, N. Shingte, J. Lindqvist, and M. Gruteser. A field study of run-time location access disclosures on android smartphones. *Proc. USEC*, 14, 2014.

[16] Google. Normal and dangerous permissions. `https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous`.

[17] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

[18] L. Jedrzejczyk, B. A. Price, A. K. Bandara, and B. Nuseibeh. On the impact of real-time feedback on users' behaviour in mobile location-sharing applications. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, page 14. ACM, 2010.

[19] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.*, FC'12, pages 68–79, Berlin, Heidelberg, 2012. Springer-Verlag.

[20] H. Lee and A. Kobsa. Privacy Preference Modeling and Prediction in a Simulated Campuswide IoT Environment. In *IEEE International Conference on Pervasive Computing and Communications*, 2017.

[21] B. Liu, M. S. Andersen, F. Schaub, H. Almuhimedi, S. A. Zhang, N. Sadeh, Y. Agarwal, and A. Acquisti. Follow my recommendations: A personalized assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

[22] B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 201–212, New York, NY, USA, 2014. ACM.

[23] W. Mason and S. Suri. Conducting behavioral research on amazon's mechanical turk. *Behavior Research Methods*, 44(1):1–23, 2012.

[24] D. Mate. Permission master. `https://play.google.com/store/apps/details?id=com.droidmate.permaster`, 2014.

[25] M. McLaughlin. What is lineageos. `https://www.lifewire.com/what-is-cyanogenmod-121679`, 2017.

[26] H. Nissenbaum. Privacy as contextual integrity. *Washington Law Review*, 79:119, February 2004.

[27] A. Oglaza, R. Laborde, A. Benzekri, and F. BarrÃÍre. A recommender-based system for assisting non-technical users in managing android permissions. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 1–9, Aug 2016.

[28] K. Opsahl. Uber should restore user control to location privacy. `https://www.eff.org/deeplinks/2016/12/uber-should-restore-user-control-location-privacy`, 12 2016.

[29] G. Paolacci and J. Chandler. Inside the turk. *Current Directions in Psychological Science*, 23(3):184–188, 2014.

[30] E. Peer, L. Brandimarte, S. Samat, and A. Acquisti. Beyond the turk: Alternative platforms for crowdsourcing behavioral research. *Journal of Experimental Social Psychology*, 70:153–163, May 2016.

[31] X. M. Repository. `http://repo.xposed.info/`, http://repo.xposed.info/.

[32] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson. Who are the crowdworkers?: Shifting demographics in mechanical turk. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 2863–2872, New York, NY, USA, 2010. ACM.

[33] J. L. B. L. N. Sadeh and J. I. Hong. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium on Usable*

Privacy and Security (SOUPS), 2014.

[34] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and capturing people's privacy policies in a mobile social networking application. *Personal and Ubiquitous Computing*, Forthcoming 2008.

[35] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir, and H. Borgthorsson. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proc. of the 32nd Ann. ACM Conf. on Human Factors in Computing Systems*, CHI '14, pages 2347–2356, New York, NY, USA, 2014. ACM.

[36] C. Thompson, M. Johnson, S. Egelman, D. Wagner, and J. King. When it's better to ask forgiveness than get permission: Designing usable audit mechanisms for mobile permissions. In *Proc. of the 2013 Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[37] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. Android permissions remystified: A field study on contextual integrity. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 499–514, Berkeley, CA, USA, 2015. USENIX Association.

[38] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. *arXiv preprint 1703.02090*, 2017.

APPENDIX

Condition	Correct	Incorrect	All
Task 1			
control	2	3	2
experimental	2	4	2
Task 2			
control	2	3	3
experimental	2	3	2
Task 3			
control	2	4	3
experimental	2	3	2
Task 4			
control	4	2	3
experimental	2	2	2

Table 6: Median ease-of-use Likert scores for all tasks, conditions, and correctness. Higher scores indicate more difficulty.

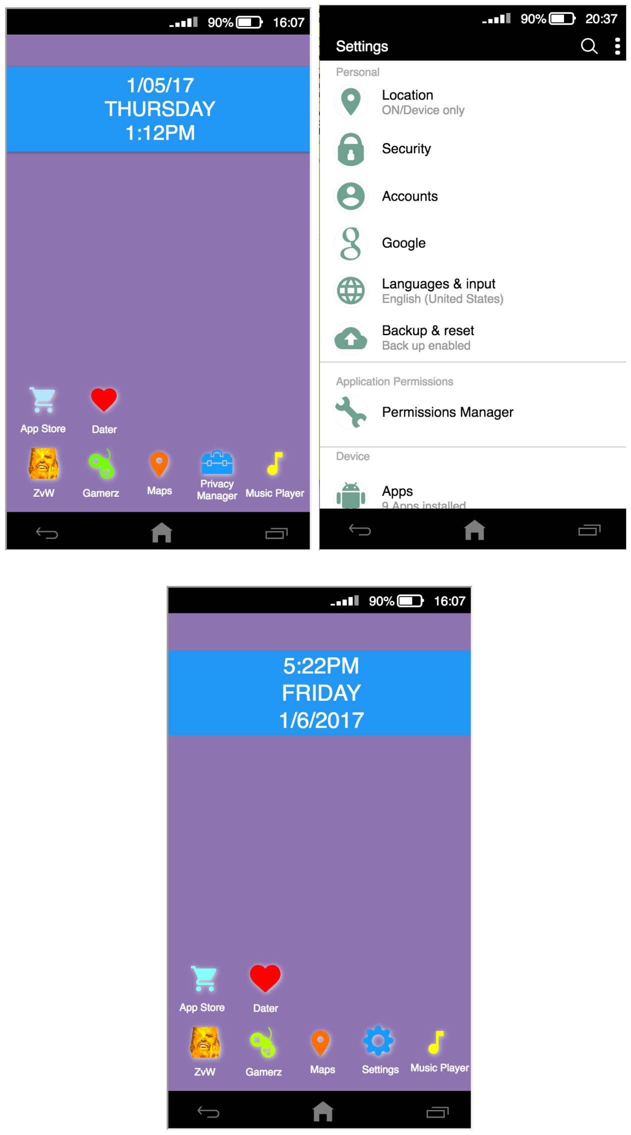


Figure 10: In the pilot experiment, TurtleGuard was launched via the icon labeled “Privacy Manager” (top left), but then added as a sub-panel to the Settings app, labeled “Permissions Manager,” for the validation experiment (top right). In the *control* condition in the pilot experiment and both conditions in the validation experiment, the Settings app was accessible from the home screen (bottom).

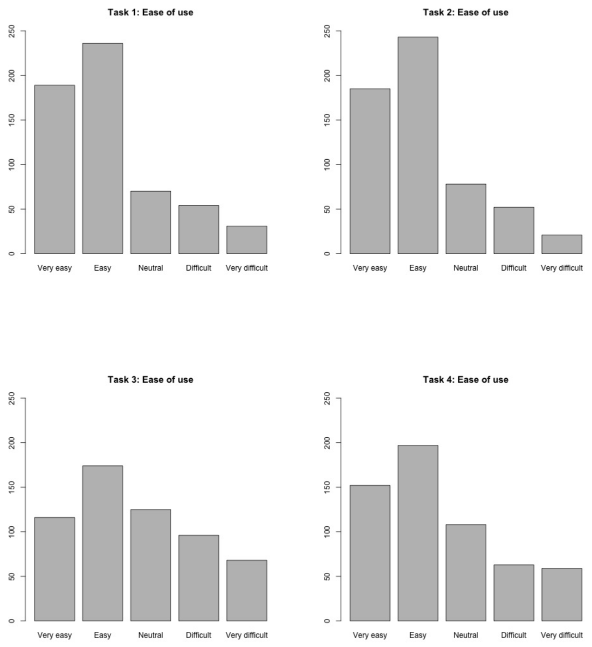


Figure 11: Ease of use histograms for each task

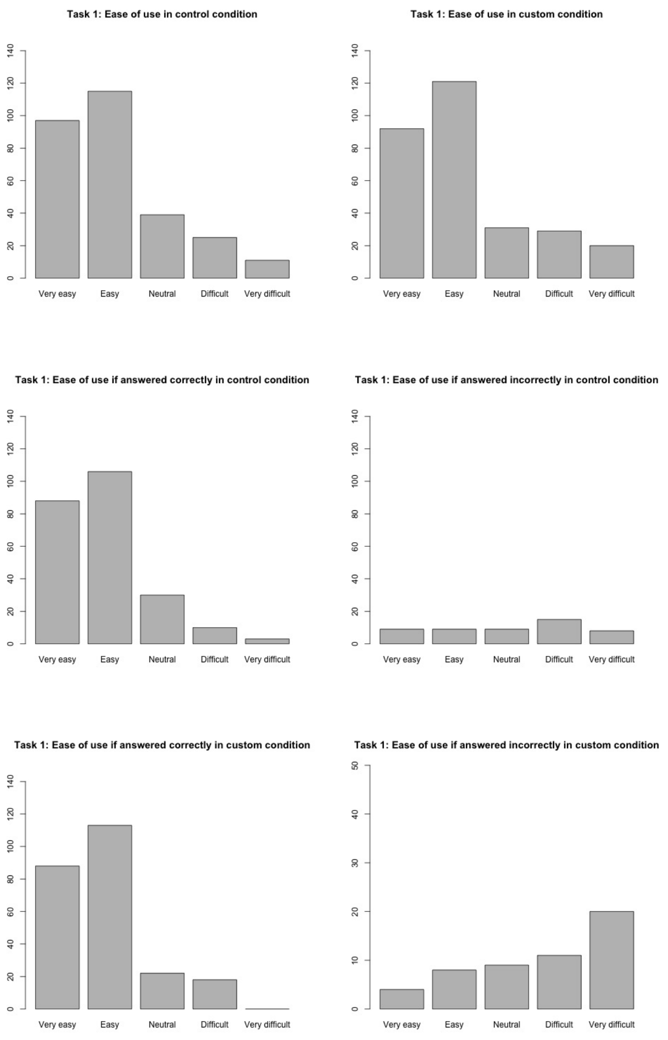


Figure 12: Ease of use histogram for Task 1

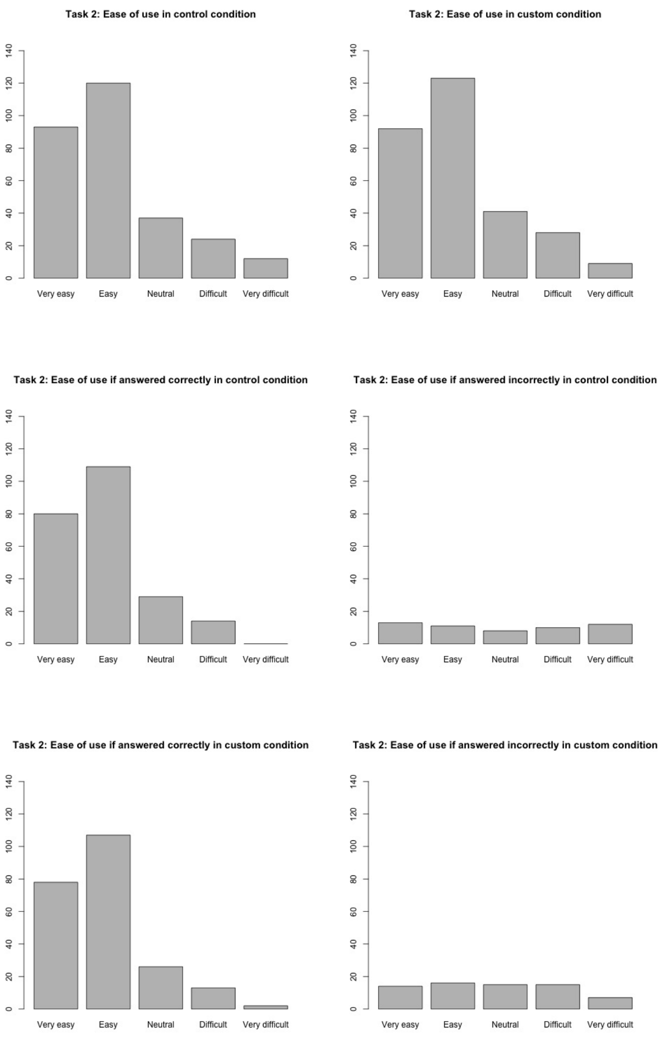


Figure 13: Ease of use histogram for Task 2

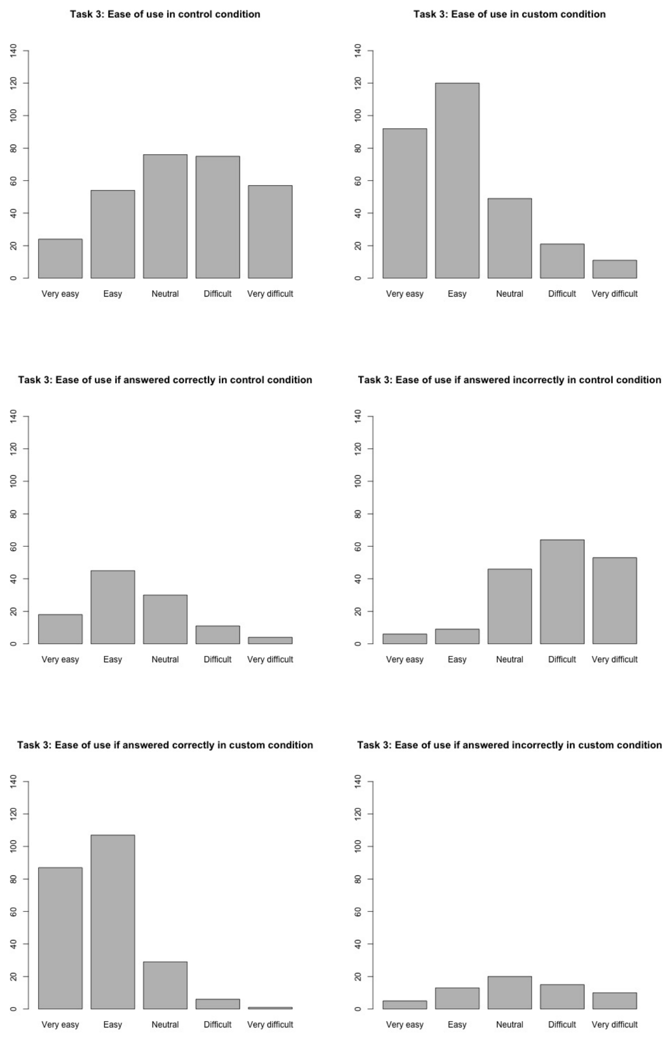


Figure 14: Ease of use histogram for Task 3

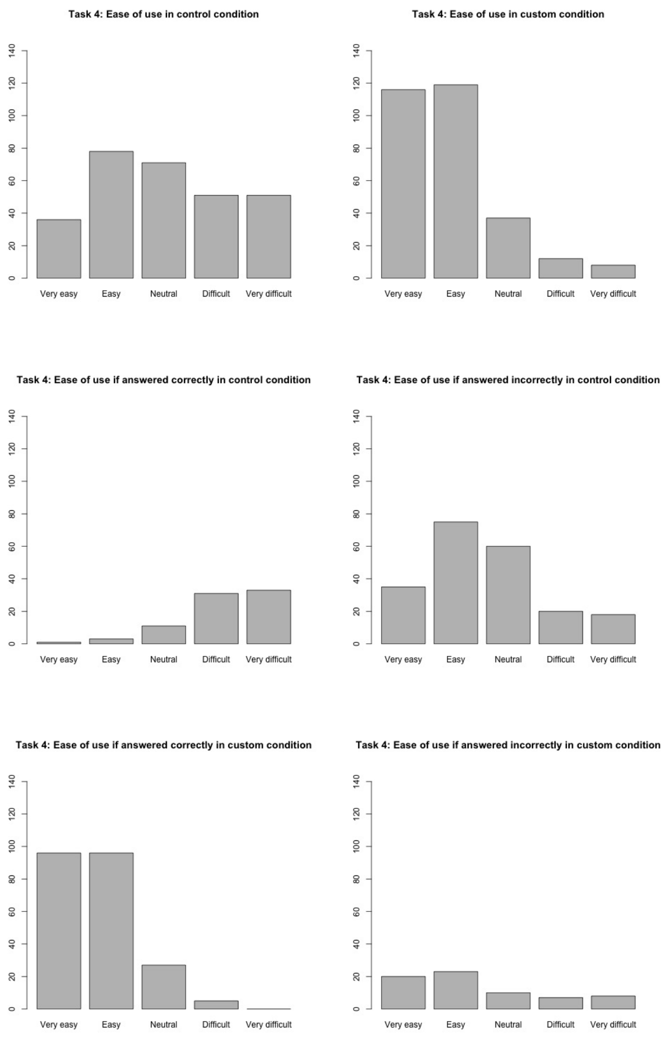


Figure 15: Ease of use histogram for Task 4