Turtle Guard: Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai, ***University of California, Berkeley;*** **Primal Wijesekera,** ***University of British Columbia;*** **Joel Reardon, Irwin Reyes, Serge Egelman, and David Wagner,** ***University of California, Berkeley;*** **Nathan Good and Jung-Wei Chen,** ***Good Research***

https://www.usenix.org/conference/soups2017/technical-sessions/presentation/tsai

This paper is included in the Proceedings of the Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017).

July 12–14, 2017 • Santa Clara, CA, USA

ISBN 978-1-931971-39-3

Open access to the Proceedings of the Thirteenth Symposium on Usable Privacy and Security is sponsored by USENIX.

TurtleGuard: Helping Android Users Apply Contextual Privacy Preferences

Lynn Tsai[1], Primal Wijesekera[2], Joel Reardon[1], Irwin Reyes[3], Jung-Wei Chen[4], Nathan Good[4], Serge Egelman[1,3], and David Wagner[1]

[1]University of California, Berkeley, Berkeley, CA
{lynntsai,jreardon}@berkeley.edu, daw@cs.berkeley.edu

[2]University of British Columbia, Vancouver, BC
primal@ece.ubc.ca

[3]International Computer Science Institute, Berkeley, CA
{ioreyes,egelman}@icsi.berkeley.edu

[4]Good Research, Inc., El Cerrito, CA
{jennifer,nathan}@goodresearch.com

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 account for *context*. In allowing for more contextual decisions, machine-learning techniques have shown great promise for designing systems that automatically make privacy decisions on behalf of the user. However, if such decisions are made automatically, then feedback mechanisms are needed to empower users to both audit those decisions and correct any errors.

In this paper, we describe our user-centered approach towards designing a fully functional privacy feedback interface for the Android platform. We performed two large-scale user studies to research the usability of our design. Our second, 580-person validation study showed that users of our new interface were significantly more likely to both understand and control the selected set of circumstances under which applications could access sensitive data when compared to the default Android privacy settings interface.

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. Part of the purpose of these permission systems is to 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 disclose the full list of sensitive resources it wished to access. Users could either grant access to all requested permissions or abort the installation entirely. Prior research has shown that most users do not pay attention to or do not these prompts when shown at install-time [12].

Recently, an *ask-on-first-use* (AOFU) permission system replaced install-time disclosures on Android. Under AOFU, the user is prompted when an application requests a sensitive permission 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 may change in different contexts. It only learns the user's preferences once under a certain set of contextual circumstances: the first time an application tries to access a particular data type. This system does not account for the fact that subsequent requests may occur under different contextual circumstances and therefore may be deemed less appropriate. 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 access location for the purposes of behavioral advertising, possibly when the application is not even being used.

The *contextual integrity* framework can explain why AOFU is insufficient: it fails to protect user privacy because it does not account for the context surrounding data flows [25]. That is, privacy violations occur when a data flow (e.g., an app's access to a sensitive resource) defies user expectations. In recent work [38, 39], we showed that mobile users *do* make contextual privacy decisions: decisions to allow or deny access are based on what they were doing on their mobile devices at the time that data was requested.

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 unusable privacy controls, as the frequency of these requests could overwhelm the user [38]. Consequently, automating these decisions with machine learning yields a balance between

Symposium on Usable Privacy and Security (SOUPS) 2017, July 12–14, 2017, Santa Clara, California.

accurately implementing users' privacy preferences and not overburdening them with too many decisions [39]. Such automation requires the platform to have feedback mechanisms so that automated decisions can be reviewed and errors can be corrected, thereby yielding fewer future errors.

To this end, we designed a novel permission manager, TurtleGuard, which helps users to vary their privacy preferences based on a few selected contextual circumstances. It also provides information about the apps that they use, by providing a feedback loop for them to audit and modify how automated decisions are made. TurtleGuard allows users to (i) vary their decisions based on the visibility of the requesting application – our previous work showed that the visibility of the requesting application is a critical factor used by users when making mobile app privacy decisions [38], and (ii) have an improved understanding of how third-party applications access resources in the real world and under varying contextual circumstances.

We conducted an initial 400-person experiment to evaluate our preliminary design. Based on our analysis of this data, we then iterated on our design, conducting a 580-person validation study to demonstrate our design's effectiveness. 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 vary their privacy preferences based on surrounding circumstances than the control group. We believe that these results are a critical contribution towards empowering mobile users to make privacy decisions 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 when applications had foreground versus background access to sensitive data and how to correctly control it.
- We show that our proposed interface has a minimal learning curve. Participants, who had never used TurtleGuard before, were as successful at accomplishing information retrieval tasks as those who used the existing Android interface.

2. RELATED WORK

The Android OS has thus far used two different permission models: ask-on-install (AOI) 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 still correctly understand their meaning [12, 18].

Versions of Android from 6.0 (Marshmallow) onward use the AOFU permission model instead. Under AOFU, applications prompt users for sensitive permissions at runtime. These prompts protect access to a set of 24 "dangerous permissions," including geolocation data, contact lists, and SMS. Prompts appear when the application attempts to request 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 a user's privacy preferences [39]. That is, the AOFU model does not consider scenarios where an application requests access to data under varying contexts.

Research on permission models has found that users are often unaware how apps access protected resources and how access may be regulated [12, 8, 11, 36, 34]. Shih et al. showed that users are more likely to disclose privacy information when the purpose is unclear [35]. Prior work has specifically analyzed location data: Benisch et al. show that a vast number of factors (time, day, location) contribute to disclosure preferences [5]; Reilly et al. show that users want minimal interaction with their technology [31]. Additionally, Patil et al. takes into consideration context: they suggest making feedback actionable and allowing for selective control regarding location data [29]. They also show that users have difficulty articulating location access controls, and suggest an interface that includes contextual factors as a potential solution [28]. 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 once installed, as well as set default permissions that newly installed applications must follow. They examined 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, Android 6.0 (*Marshmallow*) reintroduced permission management. It—and subsequent versions as of this writing—include 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 Settings app, and it is therefore unlikely that users are aware of 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 click from the main Settings panel and explicitly presents the relationships between applications, permissions, and controls.

XPrivacy [6], DonkeyGuard [7], Permission Master [23], and LineageOS's[1] Privacy Guard [24] are examples of other 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 [24];

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

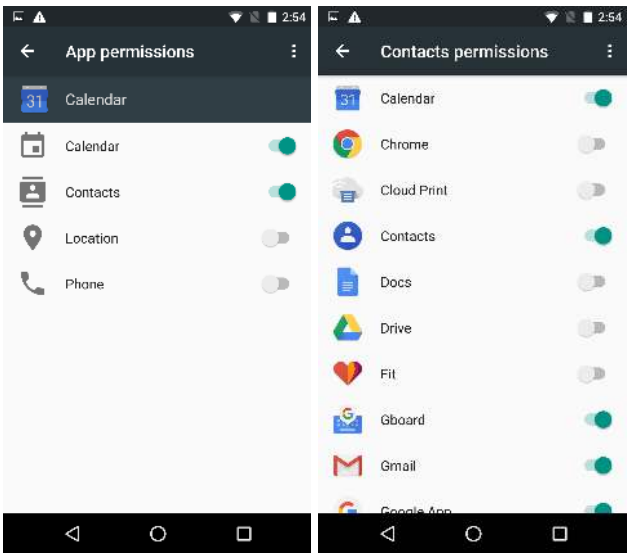


Figure 1: After navigating four and five levels of sub-panels within the Android Settings app, respectively, users can limit a specific app's access to specific permissions (left) or limit the apps that can access a particular permission (right).

others use the Xposed Framework [32], which 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 [16]. 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. 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. This security design failure is known as the *wall of checkboxes* [14]. XPrivacy and Permission Master show controls for resources whose direct effects on user privacy are unclear, such as keeping a device awake. TurtleGuard improves usability by showing only controls for resources deemed "dangerous" in the Android platform [15] and others that previous research has shown are conducive to using run-time prompts [10]. 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 partially 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 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 location data while the application is active on the screen. TurtleGuard uses the same options, but our design is novel in both the extent of these settings and in who controls them. Apple's iOS 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 [26]. Our design, by contrast, gives users all three of these options for all *sensitive* permissions (Table 5, Appendix). 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 [39]. They examined this by instrumenting the Android platform to log all instances of apps accessing sensitive resources. In addition to their instrumentation, the platform randomly prompted users about the appropriateness of various permission requests as those requests occurred. Participant responses to these prompts were 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 detecting privacy violations [25, 4].

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 instead requires user input. Such a model would conceivably allow users to accurately specify their contextual preferences because they know exactly which app attempted to gain access to what resource under which circumstance. This approach, however, is unusable in practice. Research has shown that applications request access to permission-protected resources with great frequency: on an average smartphone, roughly once every 15 seconds [38]. Such a high frequency not only risks habituation, but would render the device inoperable.

Recent research on permission models has turned towards using machine learning (ML) [39, 20, 21, 19]. One advantage is ML's ability to incorporate nuanced contextual data to predict user preferences; the approach has shown significantly lower error rates over the *status quo*, i.e., AOFU. Wijesekera et al. [39] also showed that ML reduces user involvement, thereby minimizing habituation. They emphasize, however, the importance of having a user interface that functions as a feedback-loop to the classifier, since no practical classifier will ever be 100% accurate. 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 that the classifier improves its accuracy over time, it also keeps users aware of decisions that were made on their behalves and informs them of how third-party apps are accessing sensitive resources under various circumstances.

TurtleGuard provides two core components necessary for usability under such contextual privacy models: we provide users with key contextual information when regulating access to sensitive resources, and we provide a method for users to audit and correct the decisions that have been automatically made by the system.

3. DESIGN OBJECTIVES

TurtleGuard's primary function is to inform users about the decisions that have been automatically made on their behalf, 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 [37]. 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 that they have the same access to sensitive user data that they do when actively being used [37]. 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 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 [38].

In the default permission manager, users cannot vary their decisions based on the visibility of the requesting application, or any other contextual factors. Our overarching goal is to empower users to make contextual decisions (i.e., based on what they were doing on the device) and to apply these decisions to future use cases, so that fewer decisions need to be explicitly made overall. As a first step towards allowing users to make contextual decisions, TurtleGuard makes decisions about whether or not to allow or deny access based on whether the requesting application is actively being used. While this is but one contextual factor amongst many, it is likely one of the most important factors [38].

Moving one step beyond the all-or-nothing approach of allowing or denying an application's access to a particular data type, our new design gives the user a third option: allowing applications to access protected data only *when in use* (Table 1 and Figure 2). When the *when in use* option 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 access to the resource is denied.

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 when the application is running invisibly in the background.
never	The permission is never granted to the requesting application.

Table 1: 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.

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. The existing Android interface also 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 (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. 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 automated 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 the resource and the severity of the threat [9]. Consequently, TurtleGuard only manages a subset of permissions (Table 5, Appendix) based on those deemed sensitive by prior work and by the latest Android version. In the first prototype of TurtleGuard, we had listed the original names of the permissions, ungrouped. One of the changes we made as we iterated on our design after our pilot experiment was to implement permission "fami-

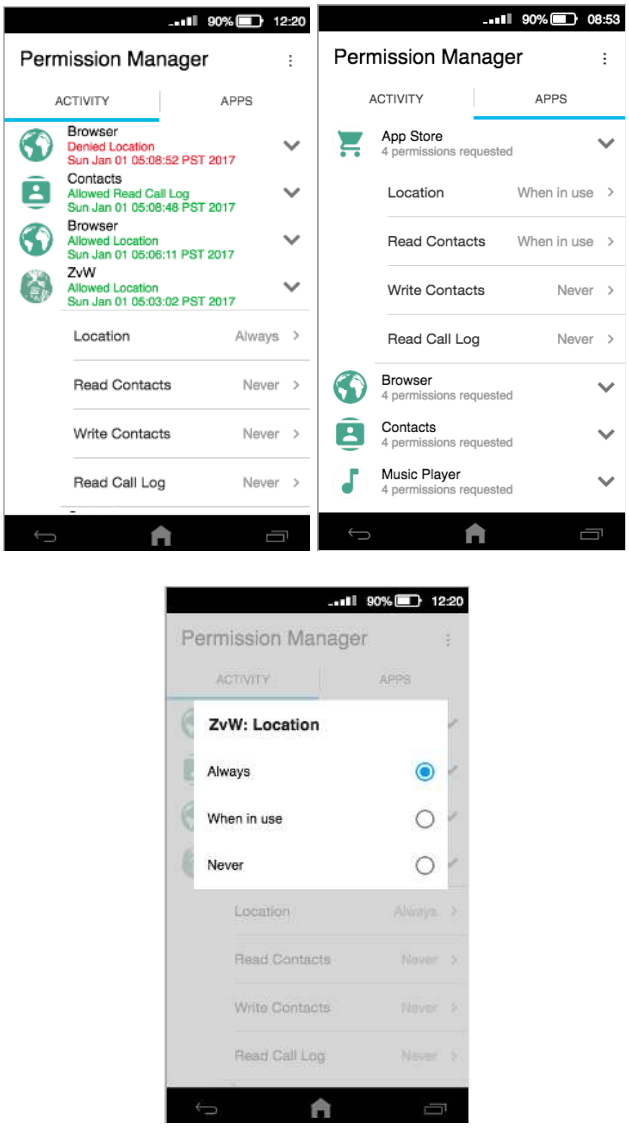


Figure 2: The pilot design of TurtleGuard listed recent app activity (top left), a list of installed apps and their associated permissions (top right). Permissions can be *always* granted, granted only *when in use*, or *never* granted (bottom).

lies." For example, READ_CONTACTS and WRITE_CONTACTS are grouped into a single CONTACTS permission family. This means that within TurtleGuard, users only see the human-readable resource type and not the underlying permissions the family manages. 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.

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). We designed the first experiment to examine our initial prototype, as described in the previous section. Based on the analysis of our first 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). Our study was approved by our IRB (#2013-02-4992).

In both experiments, we randomly assigned participants to either the *control* or *experimental* conditions. We presented *control* participants with an interactive HTML5 simulation of the default permission manager, which is accessible from within the Settings app. We presented *experimental* participants with an interactive HTML5 simulation of 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 6 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 the interactive Android simulations: 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 even 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 accomplished by performing the four steps to access the screen in Figure 4 (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 TurtleGuard's success 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 an interactive HTML5 simulation of the UI designs described in the previous section using *proto.io*. We instrumented the simulations to log all interactions (e.g., panels visited, buttons clicked, etc.). This data allowed us to analyze how participants navigated the UI to perform each task.

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 were coded as true. Any complaints or suggestions were coded as false, as well as responses with uncertainty, confusion, or ambivalence (e.g., "I don't know"). We further coded responses that had specific suggestions, using tags for the different themes.

Each response was coded by two experienced coders working independently, who then compared responses and recorded their coding conflicts. The coders discussed and reconciled the differences using their mutually agreed upon *stricter* interpretation given the nature of 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 changes we made to

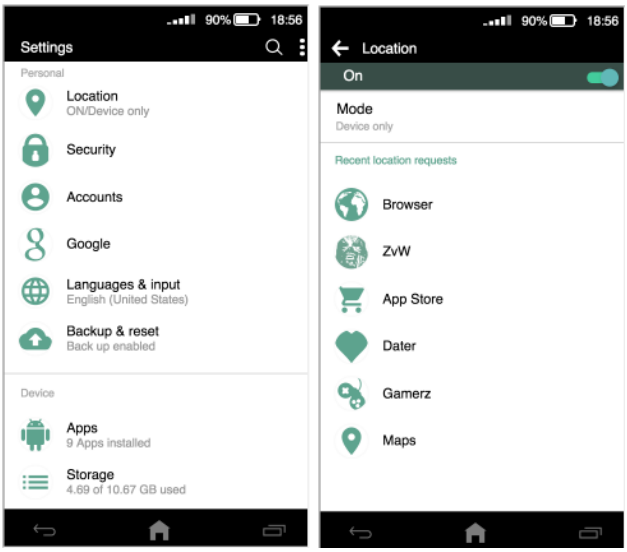


Figure 3: In Task 1, participants in the *control* condition could identify the most recent applications that requested location data from within the Settings application. This was also a valid method for Task 1 in the *experimental* condition for the validation study.

TurtleGuard's interface as a result of this initial experiment. We note that in our simulation, *Settings* can only be reached by tapping on the icon from the home screen. In all of our tasks, we also asked participants to evaluate perceived difficulty using a 5-point Likert scale.

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 2). Of the 49 participants in the *experimental* group 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 in the activity tab of the Permission Manager. The activity tab showed recent resource accesses in a chronological order—"Browser" had been denied a *location* request and "Contact" had successfully accessed *call logs*.

Participants did not seem to understand that the activity log presented entries related to *all* sensitive data types, not just location data. This confusion might also stem from their familiarity with the location access panel in stock Android, in which 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 way of implementing this is to create separate tabs for allowed and denied requests, as well as to group similar data types together (rather than presenting all permission request activity in chronological order).

Condition	Correct	Incorrect
Task 1		
control	167 (84%)	31 (15%)
experimental	132 (68%)	62 (32%)
Task 2		
control	140 (70%)	58 (29%)
experimental	116 (59%)	78 (40%)
Task 3		
control	86 (43%)	112 (56%)
experimental	153 (78%)	41 (21%)
Task 4		
control	47 (23%)	151 (76%)
experimental	144 (75%)	49 (25%)

Table 2: Participants in each condition who performed each task correctly during the pilot experiment.

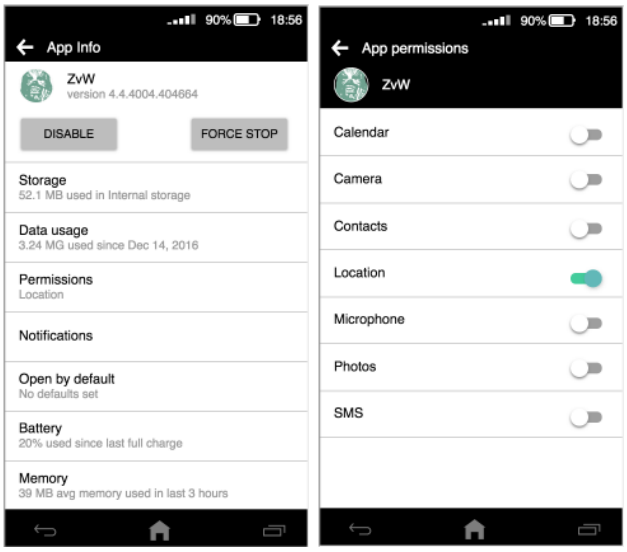


Figure 4: In Task 2, participants in the *control* condition could identify the permissions granted to the ZvW application by selecting the "Apps" panel from within the Settings application, and then selecting the application, followed by the "Permissions" panel.

5.2 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 *experimental* 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 4), 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 were not included in the options presented to participants (e.g., "Write Contacts" and "Read Call Log"). This may have made this task confusing for these participants.

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 type, with the events for each resource type sorted chronologically. These changes can be seen in the top of Figure 5.

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 moved TurtleGuard to be within the Settings app, so that it appears as a panel labeled "Permissions Manager" (Figure 6, Appendix). For consistency, when participants in the *experimental* condition select the "Permissions" sub-panel from within the "App Info" panel (Figure 4, left), they are now redirected to TurtleGuard's "Apps" panel, pre-opened to the app in question (Figure 5, bottom right).

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 3).

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

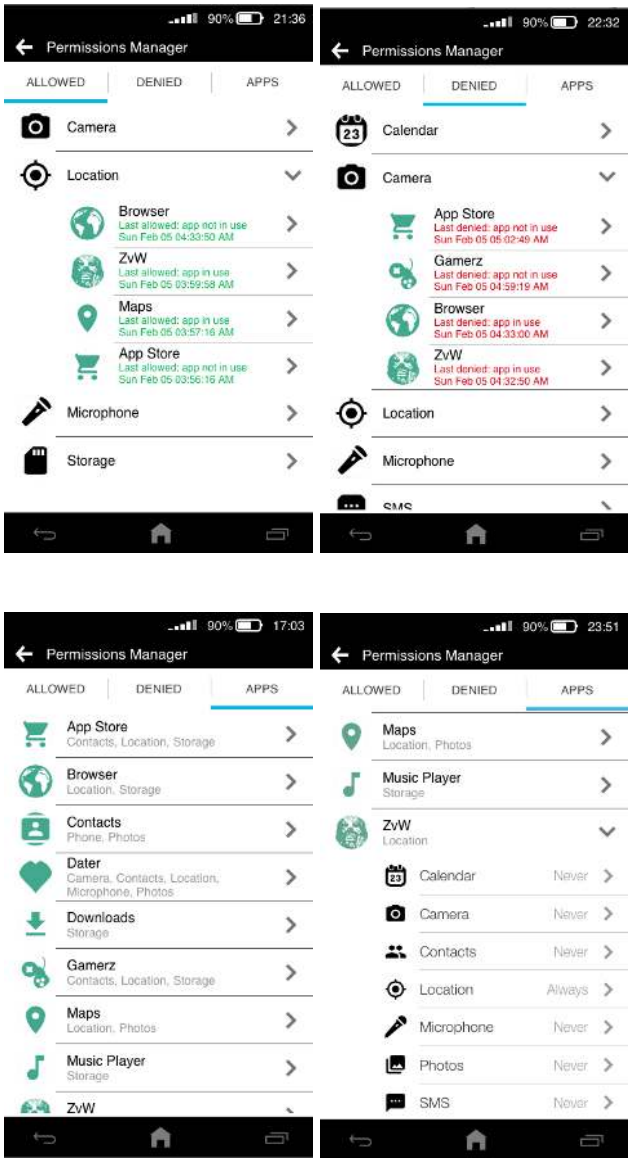


Figure 5: TurtleGuard separates recently allowed (top left) and denied (top right) permissions. The "Apps" tab lists the allowed permissions of all apps (bottom left). Expanding an app allows the user to make changes (bottom right).

6.1 Participants

Because of several known biases in Mechanical Turk's demographics [27, 33, 22], 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.,

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

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 and took on average 9 minutes and 17 seconds to complete the 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 5). 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 *experimental* condition. A chi-square test revealed that this difference was not statistically significant ($\chi^2 = 0.011$, $p < 0.918$).

We observed that most of the participants in both conditions used the default method of accomplishing this task (i.e., accessing the Location sub-panel): 80.1% of those who answered correctly in the *experimental* condition and 92.8% of those in the *control* condition. Fifteen participants in the *control* condition 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 simulation and wrote that it was broken or the buttons did not work; 9 participants did not respond or wrote that they did not know. Of the other 88 participants, 38 (43%) listed "App Store" as one of their selections, making it the most common error.

More specifically, 18 participants listed their answers as both "App Store" and "Browser." We believe that this is because both the stock Android Apps Manager and TurtleGuard's "Apps" tab (Figure 5, bottom) sort the entries alphabetically, and by looking at the permissions available to both of these apps, participants 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 5) for those in the *experimental* condition, or by viewing an app's App Info panel from within the Settings app (Figure 4), which was available to those in either condition.

In total, 458 participants correctly performed this task (79% of 580). Table 3 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 4). 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 5), 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 [37] 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 5) to see the other options available (Figure 2): *always*, *when in use*, and *never*.

We found 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 [37]—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

Task 4 asked participants to describe the steps to prevent an application from accessing location data 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 among the default permission manager 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 is 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. We provide Cohen's inter-rater reliability statistic (κ) for each coding.

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 participants had previously thought about resource access or configuring privacy preferences, and whether having these 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 had tried these tasks before is that they wanted to be able to exert more control over their installed apps:

[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.

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

Table 4: Whether participants believed changes were needed to the interfaces they used during the validation study.

- "*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 participants 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:

- "*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.*"

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

In our second exit survey question, we asked: *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?*

This question had 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% participants (383 of 580) suggested at least some change or improvement (Cohen's $\kappa = 0.896$). Table 4 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 wants changes in the dashboard ($p < 0.00005, \phi = 0.237$). 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 suggestions) 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 would prefer to reach a permission control interface directly through the application—either as part of the application or by pressing the app icon. TurtleGuard addresses this concern by providing a path to permission management that involves fewer clicks and centralizes all permission management functionality.

- "*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.*"

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. [20], that recent statistics help users make better decisions. TurtleGuard provides this functionality by showing 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 the 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 to certain resources. Some even suggested runtime prompts for every sensitive request access. One participant suggested that app developers 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.*"

Prior work has already shown that having runtime prompts on every sensitive request is not feasible [38]—we believe that a log of recent resource accesses with surrounding context is the closest practical solution.

8. 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 user-centered design on a new permission management interface, TurtleGuard, which offers users significant improvements in their ability to control permissions when compared to the default permission manager.

8.1 Auditing Automated Decision Making

Recent research uses machine-learning techniques to automatically predict users' permission preferences [39, 20, 19, 21]. While machine-learning (ML) techniques have been shown to be better at predicting users' preferences [39], 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 were better able to understand and control *when* apps have access to sensitive data than participants using the default permission manager. A substantial proportion 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.

8.2 Correcting Mental Models

In Task 4, we asked participants to disable access to location data 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 limits of the permission interface's provided functionality. This mismatch between users' mental models and actual functionality may lead to users incorrectly believing that they have denied access to certain requests for sensitive data.

8.3 Privacy Nudges

Previous work investigated ways to nudge users to configure their privacy settings and make them aware of how applications access their data [20, 13, 17]. While helping motivate users to use TurtleGuard (and other privacy management interfaces) is important, it is out of scope for this work. Nevertheless, our survey results showed 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.

8.4 Limitations

In our proposed interface, TurtleGuard, we allow users to vary their decisions based on the visibility of the requesting application. We believe this is a significant first step towards enabling users to make contextual privacy decisions. The full extent of the impact of the surrounding context, however, goes beyond the mere visibility of the requesting application. More work is needed to understand different contextual factors and their respective impact on users' privacy decisions. We hope the results of this study will pave the path for future work on implementing *fully* contextually aware permission managers.

Additionally, we acknowledge the limitations in our screening process: participants who selected Android as their mobile device may have varying levels of usage and knowledge regarding the platform. Prior experience may have rendered the default permission manager as being easier to use for some participants in the *control* condition. This suggests that for new Android users, the usability improvements of TurtleGuard may be even greater than what we observed.

We also acknowledge that irregularities in the simulation may have had an impact towards participants' comprehension and completion rates. These confounding factors introduced by the UI, however, would have impacted both conditions equally, because the control condition was simulated using the same infrastructure and development environment. Finally, for users in the control condition, Task 4 may have been deceptively tricky due to its impossibility. Nevertheless, the incorrect answers underscore a very real problem: Android users are not aware that they are unable to deny resources to applications that they are not using.

8.5 Conclusion

Android's existing permission models, ask-on-install (AOI) and ask-on-first-use (AOFU), are insufficient at fulfilling users' privacy desires and needs. Neither of the existing models account for contextual factors in their decisions to allow or deny access to sensitive data. 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. In our study of TurtleGuard, we had participants perform permission-related tasks and compared their performance 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 the validity of our changes. Our results show that users are significantly better at performing permission management tasks with TurtleGuard than the default permission manager.

Acknowledgements

This research was supported by the United States Department of Homeland Security's Science and Technology Directorate under contract FA8750-16-C-0140, the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley, the National Science Foundation under grants CNS-1318680 and CNS-1514457, Intel through the ISTC for Secure Computing, and the AFOSR under MURI award FA9550-12-1-0040. The content of this document does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

9. 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] 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.

[5] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor. Capturing location-privacy preferences: Quantifying accuracy and user-burden tradeoffs. *Personal Ubiquitous Comput.*, 15(7):679–694, Oct. 2011.

[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. 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] N. Good. The Deadly Sins of Security User Interfaces. In M. Jakobsson, editor, *The Death of the Internet*, chapter 7.5, pages 398–415. John Wiley & Sons, 2012.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

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

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

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

[26] 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.

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

[28] S. Patil, Y. Le Gall, A. J. Lee, and A. Kapadia. My privacy policy: Exploring end-user specification of free-form location access rules. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, FC'12, pages 86–97, Berlin, Heidelberg, 2012. Springer-Verlag.

[29] S. Patil, R. Schlegel, A. Kapadia, and A. J. Lee. Reflection or action?: How feedback and control affect location sharing decisions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 101–110, New York, NY, USA, 2014. ACM.

[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] D. Reilly, D. Dearman, V. Ha, I. Smith, and K. Inkpen. "need to know": Examining information need in location discourse. In *Proceedings of the 4th International Conference on Pervasive Computing*, PERVASIVE'06, pages 33–49, Berlin, Heidelberg, 2006. Springer-Verlag.

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

[33] 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.

[34] 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.

[35] F. Shih, I. Liccardi, and D. Weitzner. Privacy tipping points in smartphones privacy preferences. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 807–816, New York, NY, USA, 2015. ACM.

[36] 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.

[37] 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.

[38] 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.

[39] 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

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 5: 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 as discussed in Section 3.

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 in the validation experiment. Higher scores indicate more difficulty.

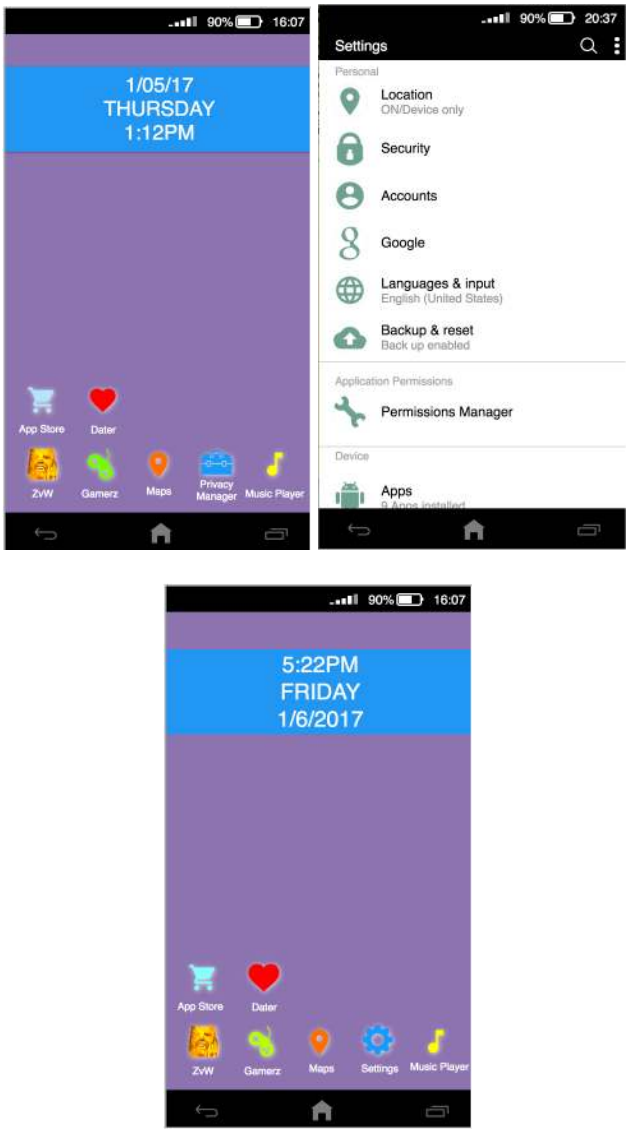


Figure 6: In the pilot experiment, TurtleGuard was launched via the icon labeled "Privacy Manager" (top left), but then added as a sub-panel to the Settings app, labeled "Permissions Manager," for the validation experiment (top right). In the *control* condition in the pilot experiment and both conditions in the validation experiment, the Settings app was accessible from the home screen (bottom).

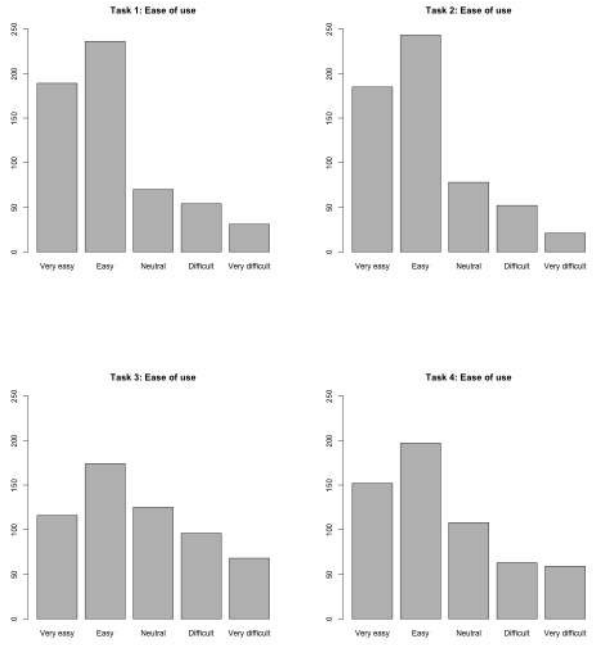


Figure 7: Ease of use histograms for each task (validation experiment)

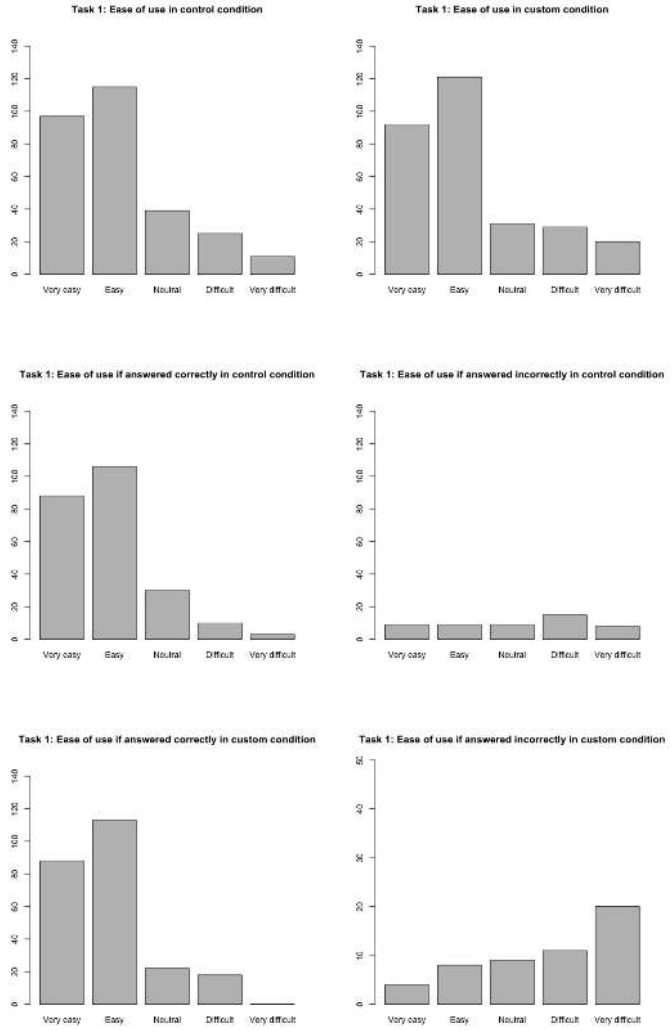
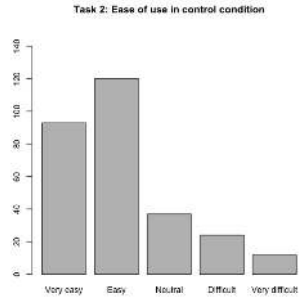
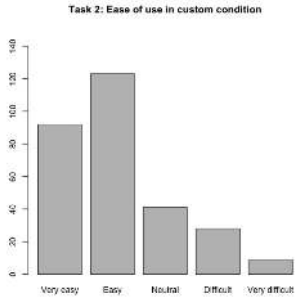
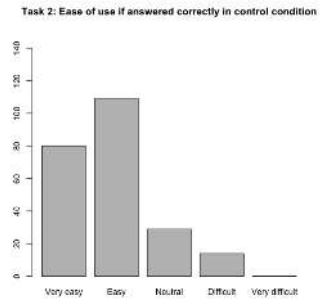
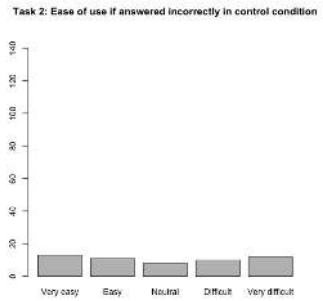


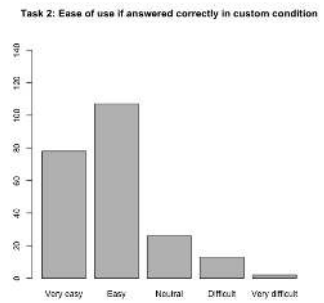
Figure 8: Ease of use histogram for Task 1 (validation experiment)











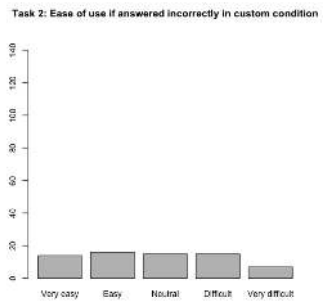
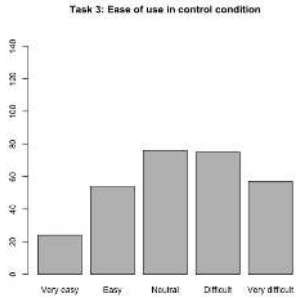
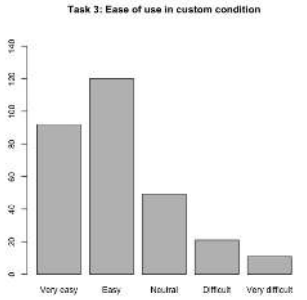
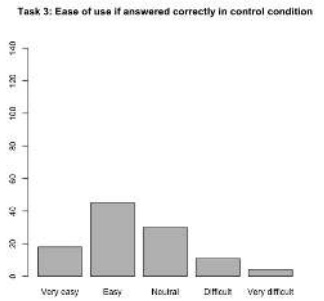
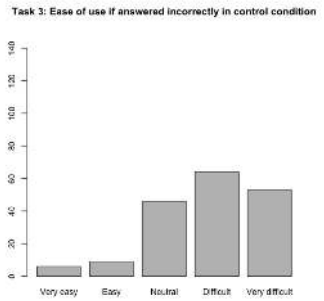


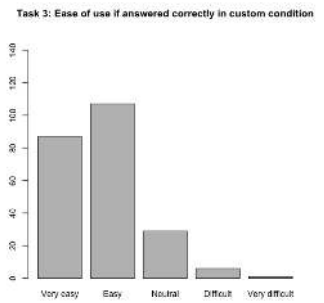
Figure 9: Ease of use histogram for Task 2 (validation experiment)











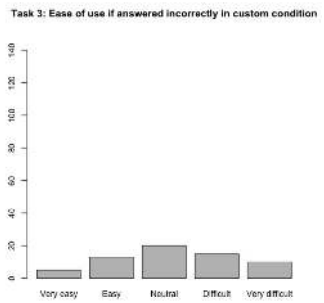


Figure 10: Ease of use histogram for Task 3 (validation experiment)

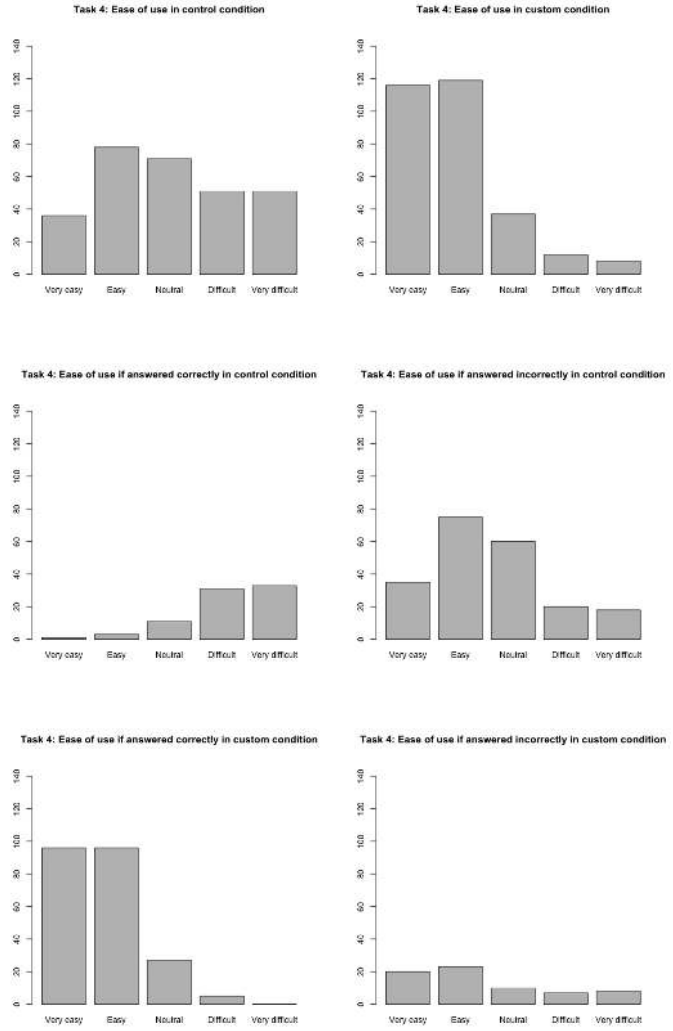


Figure 11: Ease of use histogram for Task 4 (validation experiment)