UC Berkeley
UC Berkeley Previously Published Works

Title

Contextualizing Privacy Decisions for Better Prediction (and Protection)

Permalink

ISBN

Authors

Wijesekera, Primal
Reardon, Joel
Reyes, Irwin
et al.

Publication Date

2018-04-21

DOI

Peer reviewed

Contextualizing Privacy Decisions
for Better Prediction (and Protection)

**Primal Wijesekera[1], Joel Reardon[2], Irwin Reyes[3], Lynn Tsai[4], Jung-Wei Chen[5],
Nathan Good[5], David Wagner[4], Konstantin Beznosov[1], and Serge Egelman[3,4]**
[1]University of British Columbia, Vancouver, BC
[2]University of Calgary, Calgary, AB
[3]International Computer Science Institute, Berkeley, CA
[4]University of California, Berkeley, CA
[5]Good Research, Berkeley, CA
{primal,beznosov}@ece.ubc.ca, joel.reardon@ucalgary.ca, ioreyes@icsi.berkeley.edu,
lynntsai@berkeley.edu, {jennifer,nathan}@goodresearch.com, {daw,egelman}@cs.berkeley.edu

ABSTRACT
Modern mobile operating systems implement an ask-on-first-use policy to regulate applications' access to private user data: the user is prompted to allow or deny access to a sensitive resource the first time an app attempts to use it. Prior research shows that this model may not adequately capture user privacy preferences because subsequent requests may occur under varying contexts. To address this shortcoming, we implemented a novel privacy management system in Android, in which we use contextual signals to build a classifier that predicts user privacy preferences under various scenarios. We performed a 37-person field study to evaluate this new permission model under normal device usage. From our exit interviews and collection of over 5 million data points from participants, we show that this new permission model reduces the error rate by 75% (i.e., fewer privacy violations), while preserving usability. We offer guidelines for how platforms can better support user privacy decision making.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous; K.6.5 Management of Computing and Information Systems: Security and Protection

Author Keywords
Privacy; mobile permissions; access control; user study

INTRODUCTION
Sensitive resources (e.g., address book contacts, location data, etc.) on mobile platforms are protected by "permissions" systems. That is, before a third-party application ("app") can access protected data, it must receive permission from the platform, which in turn is controlled by the end-user. Originally, Android permissions were presented as install-time

ultimatums that were unsuccessful at achieving their goals [13, 8]. More recently, Android started using an *ask-on-first-use* (AOFU) model. In AOFU, the user is explicitly asked to grant a permission via a dialog box when an app first attempts to access the permission-protected resource. This approach gives the user a little more contextual information: e.g., it may be curious that a text messaging app needs to use the microphone, but knowing that the request occurred after the user tried to use a speech-to-text feature clarifies the likely rationale.

By design, AOFU takes the user's decision at one moment and then uses it in perpetuity for all future requests from that app for that permission, unless the user navigates several layers of settings to change it. This has been shown to be error prone: it mispredicts users' preferences, resulting in privacy violations [38, 39]. AOFU fails to account for the contexts in which future requests arise. Users are nuanced and they vary their decisions based on a variety of factors, such as the visibility of the requesting app (i.e., whether it was in use when it requested a permission), what the user was actually doing at the time, as well as a variety of other factors.

We implemented and evaluated the usability of a novel mobile privacy management system that builds heavily on prior theoretical work. To resolve the longstanding challenges of mobile privacy management, Wijesekera et al. [39] proposed applying machine-learning (ML) to dynamically manage app permissions, whereas Tsai et al. [36] proposed a user interface design for that system. Both proposals were motivated by Nissenbaum's theory of Privacy as Contextual Integrity [28], but neither has been heretofore implemented and evaluated on real users *in situ*. We implemented these systems on the Android platform and performed a field study to evaluate their effectiveness at aligning app privacy behaviors with users' expectations. The ML model runs entirely on the device and uses infrequent user prompts to retrain and improve its accuracy over time. When the ML model makes a mistake, the user interface is available to support the user in reviewing and modifying privacy decisions, thereby retraining the classifier to make fewer mistakes in the future.

We performed a 37-person field study to evaluate this permission system, measuring its efficacy and how it interacted with participants and third-party apps. We issued each participant a smartphone running a custom Android OS with our permission system that used a built-in classifier, which participants used as their primary phones for a one-week study period. This produced real-world usage data from 253 unique apps, which corresponded to more than 1,100 permission decisions. Overall, participants denied 24% of permission requests. Our data show that AOFU matched participant privacy preferences only 80% of the time, while the new contextual model matched preferences 95% of the time, reducing the error rate by 75%.

In summary, the contributions of this work are as follows:

- We implemented the first contextually-aware permission system that performs permission denial dynamically, which is an advancement over prior work that only performed offline learning or did not regulate permissions in realtime.
- We show that AOFU's failure to account for context results in privacy violations 20% of the time.
- We show that by applying our contextual model, we are able to reduce privacy violations by up to 75%.

RELATED WORK

Substantial prior work has shown the ineffectiveness of mobile phone permission systems. For ask-on-install (AOI) prompts, earlier studies showed that users frequently did not pay attention to prompts or comprehend them [20, 16, 37, 13]. This lack of understanding hinders users' ability to address potential risks that arise from allowing access to sensitive resources, and may evoke users' anger after they learn about inappropriately accessed data [12]. Another critical flaw is that users are not given contextual cues about how apps might exercise the permissions granted to them. For example, users were surprised to learn that apps can continue to access those resources even when not being actively used [35, 19].

Prior research has used information flow tracking techniques to understand how apps use sensitive data in the wild [9, 14, 21]. While these techniques shed light on how apps access and share sensitive data, none gave users a mechanism to indicate their preferences regarding the access of sensitive data. Other approaches did involve users, but those efforts required high degrees of manual involvement likely beyond the skillset of average users (e.g., [18, 1, 33]).

Nissenbaum's theory of "contextual integrity" suggests that permission models should focus not on sensitive resources, but rather on *information flows*—from source to destination—that are likely to defy the user's expectations [27]. In an attempt to systematize Nissenbaum's theory, Barth et al. [4] extended the theory to smartphones. They suggest that it is important to consider the context in which the resource request is made, the role played by the requesting app under the current context, and the type of resource being accessed. Wijesekera et al. [38] performed a field study to understand how users perceive sensitive resource usage by apps in the wild. They found that users consider the visibility of the requesting app in deciding whether a particular information flow is acceptable.

Machine learning (ML) has recently gained traction as a promising approach to predict user privacy decisions. ML can significantly reduce the user's involvement in decision making and therefore reduce *habituation*—the problem where users see so many notifications that they become desensitized to future requests and thus make poor decisions. Previous work in this direction developed techniques to cluster users [32, 22, 25] and built recommender systems [40]. Liu et al. clustered users by privacy preferences, then subsequently predicted user preferences to apps' future permission requests using the inferred cluster [24]. The authors developed a privacy assistant to recommend privacy settings to users. The biggest drawback in these works, however, is their lack of consideration for the rich signals that context provides, which has been found to be a significant factor in decision making [38, 39].

Access Control Gadgets (ACGs) are a proposed mechanism to more closely associate sensitive resource accesses to particular UI elements [31, 30, 26], so users are more aware when those accesses occur. Examples of this are the "file chooser" and "photo picker" widgets. While such an approach helps users be better aware of resource accesses, it has two main limitations. First, apps are known to legitimately access resources when the user is not actually using the device, and therefore the user cannot receive visual cues. Second, the frequency of permission requests made by smartphone apps makes systematic use of ACGs impractical [38].

After Android's switch to the AOFU permission model, research followed that investigates how users perceive it. Bonné et al. looked into understanding what motivates users to (i) install an app, (ii) allow or deny an AOFU prompt, and (iii) uninstall an app [5]. Other works investigate the consistency of user decisions across app categories under this permission model [3, 2]. Closely related are two works that proposed using contextual cues to better predict and protect user data [29, 39]. In both of these works, contextual cues are used to build a machine-learning-based permission model to increase the accuracy of the system as compared to the default Android permission model. However, Olejnik et al. [29] only focused on a selected set of apps and resources.

In this paper, we build on our prior work [39, 36], in which we proposed using a ML-based permission decider that considers context. In that work, we trained an offline classifier by using participants' responses to runtime prompts. While we demonstrated that the ML approach holds promise, the training data was solely based on participants' stated privacy preferences, without considering the impact that dynamically denying permissions might have on app functionality. That is, if participants deny an unexpected permission request, but then discover that it impacts app functionality, they may wish to reconsider that decision and grant the permission. Thus, the realtime classifier approach requires real-world validation.

IMPLEMENTATION

We implemented a complete ML pipeline that included mechanisms to allow users to review and modify their decisions [36]; ways to mask resource denial from apps to keep them functioning; and a classifier that takes surrounding contextual signals to predict user preferences for each permission request [39].
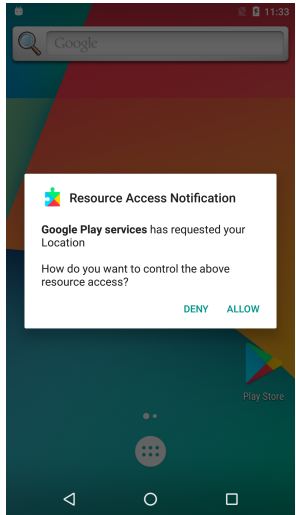
Figure 1: A screenshot of a permission request prompt.

This usability study is a more accurate assessment of how the system behaves in the wild than the previous investigations, which relied on measuring user preferences rather than consequential privacy decisions.

A Local Classifier

Wijesekera et al. implemented an offline model and suggested this could be deployed as a remote web-accessible service in order to shift compute costs from the mobile device to a more powerful dedicated server [39]. We note, however, that this design requires sending privacy-relevant data off of the smartphone, which creates a larger attack surface and increases system costs and complexity. It also creates significant security risks if the server responsible for making decisions is compromised or is trained with spurious data.

To mitigate these security and privacy issues, we implemented and integrated a Support Vector Machine (SVM) classifier into the Android operating system as a system service. We ported the open-source implementation of *libsvm* to Android 6.0.1 (Marshmallow) [6], and built two additional system-level services to interface with the SVM: the *SVMTrainManager*, which trains the model with user-provided privacy preferences through prompts (Figure 1); and the *PermissionService*, which uses the SVM to regulate apps accessing permission-protected resources and prompts the user for cases where the model produces low-confidence predictions. These services are integrated into the core Android operating system, and neither are accessible to third-party apps. On average, model training takes less than five seconds. We instrumented all Android control flows responsible for sharing sensitive permission-protected data types to pass through this new pipeline.

Bootstrapping
We deployed our trainable permission system along with a generic model that was pre-trained with the real-world permission decisions of 131 users, collected from our prior work [39]. This ensured that a new user has an initial model for making privacy decisions. This initial model, however, is inadequate

for accurately predicting any particular individual user's preferences, because it simply has no knowledge of that particular user. Despite that, Wijesekera et al. showed that their model only needs 12 additional user-provided permission decisions before the model attains peak accuracy. Given this, our system requires that the user make 12 decisions early on to train the initial model to that particular user's preferences.

The initial 12 decisions are selected based on weighted reservoir sampling. We weigh the combination of *app:permission:visibility*[1] by the frequency that these are observed; the most-frequent combinations are the likeliest to produce a permission request prompt (Figure 1). The intuition behind this strategy is to focus more on the frequently occurring permission requests over rarer ones. We used these same prompts for validating our classifier during the field study.

Feature Set
The implemented model uses four different contextual cues for making a decision on a resource request: the name of the app requesting the permission, the app in the foreground at the time of the request (if different than the app making the request), the requested permission type (e.g., Location, Camera, Contacts), and the visibility of the app making the request. In a pilot study (discussed later), our system implemented the full feature set described by Wijesekera et al. [39]. This design, however, resulted in a noticeable reduction in device responsiveness as reported by multiple study participants. We subsequently removed the "time of request" feature for the second phase of our study. The removal of the time feature from the ML enabled the platform to cache a greater number of ML decisions, reducing the overhead stemming from running the ML for each different permission request.

We aggregate feature values to calculate two other features, A_1 and A_2, which represent how often users have denied permissions under different circumstances. The aggregate feature A_1 comprises the requesting app, the permission type, and the visibility of requesting app. The aggregate feature A_2 comprises the foreground app, the permission type, and the visibility of the requesting app. When an app requests a permission-protected resource, the ML uses the requested permission type, visibility of the requesting app and the respective values for A_1 and A_2 to produce a decision. During the learning phase (i.e., the first 12 prompts), the system updates A_1 and A_2 based on user responses. The ML uses current feature values to retrieve respective denial rates from a previously-calculated set. For new apps, we use the median of A_1 and A_2 as feature values. Wijesekera et al. [39] showed—using both 5-fold cross validation and leave-one-out validation—that this technique did not lead to an overfitted model.

Sensitive Resources
Previous work by Felt et al. argued that certain permissions should be presented as runtime prompts, as those permissions guard sensitive resources whose use cases typically impart

---

[1]"app" is the app requesting the permission, "permission" is the requested resource type, and "visibility" denotes whether the user is made aware that the app is running on the device.

contextual cues indicating why an app would need that resource [11]. Beginning with Android 6.0 (Marshmallow), the OS designated certain permissions as "dangerous" [15], and now prompts the user to grant the permission when an app tries to use one of them for the first time. The user's response to this prompt then carries forward to all future uses of that resource by the requesting app.

Our experimental permission system uses both Felt's set of recommended permissions for runtime prompts and Android's own "dangerous" ones. We did, however, opt to omit a few permissions from the resulting set that we viewed as irrelevant to most users. The INTERNET and WRITE_SYNC_SETTINGS permissions were discounted, as we did not expect any participant (all recruited locally) to roam internationally during the 1-week study period. We eliminated the NFC permission because previous work demonstrated that few apps operate on NFC tags. Our system ignores the READ_HISTORY_BOOKMARKS permission, as this is no longer supported.

We regulate all attempts by apps to access resources protected by any of the 24 permissions we monitored. We avoid false positives by monitoring both the requested permission and the returned data type.

Permission Denial

Making changes to the permission system carries the risk of app instability, as apps may not expect to have their resource requests denied [10]. If denying permissions results in frequent crashes, then users are likely to become more permissive simply to improve app stability. We therefore designed our implementation with this concern in mind: rather than simply withholding sensitive information in the event of a denied permission, our system supplies apps with well-formed but otherwise non-private "spoofed" data. This enables apps to continue functioning usefully unless access to the permission-protected resource is critical to the app's correct behavior.

For example, if an app requests access to the microphone, but our permission system denies it, the app will still receive a valid audio stream: not an actual signal from the microphone, but that of a pre-recorded generic sound. (In our implementation we used a loop of a whale song.) This design allows apps to operate on valid data while still preserving user privacy.

Permission-protected databases (e.g., contact lists and calendars) require finer-grained regulation under our permission system. For instance, an app may have a legitimate need to access the contact list. Under the stock Android permission system, an app is either able to read *all contacts* or *no contacts*. We improve upon this by adding a notion of *provenance* to each entry: every contact list item contains a field that records the app that created the entry. If our permission system denies an app access to the contact list, the app is still able to write into the contacts database and read back any entries that it previously created. Apps without these database permissions are effectively placed in a sandbox, in which they can still carry out valid operations on their own versions of the data. They neither produce an exception nor obtain all the information in the database. We allow full access to the databases only to apps that are granted the appropriate permission.
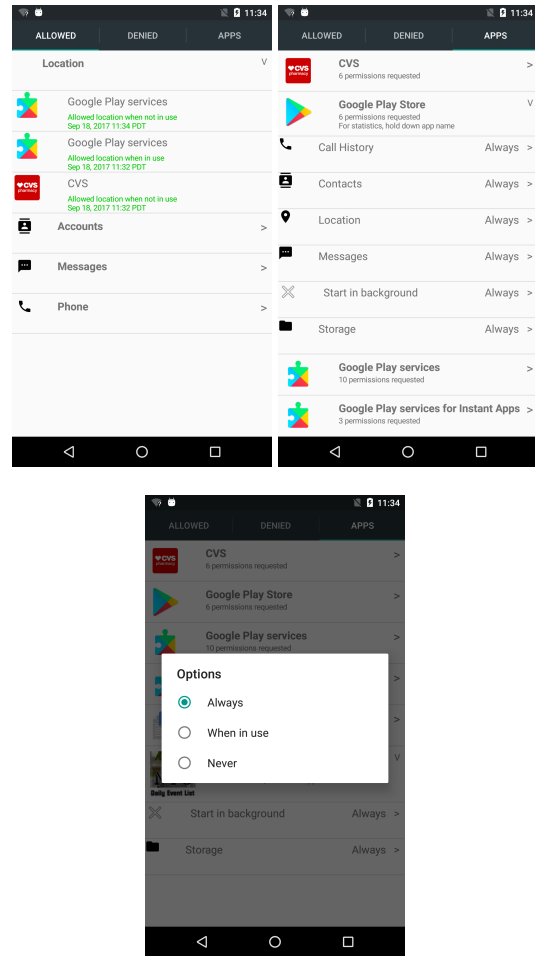

Figure 2: The recently allowed app activity (top left), a list of installed apps and their associated permissions (top right). Permissions can be *always* granted, granted only *when in use*, or *never* granted (bottom).

The system, however, does not spoof any resource that might have collateral privacy issues. The system does not spoof the phone's IMEI, a phone number, or any other device identifiers. This is because any randomized number may be a legitimate ID for another device, thus creating a privacy violation.

Contextually Aware Permission Manager

Because any classifier will make mistakes, it is crucial to provide a mechanism for users to review and alter decisions made on their behalf. Mobile operating systems have configuration panels to manage app permissions, but these fail to provide users key information or options to make informed decisions. We previously proposed a new interface to solve this problem, evaluated it using interactive online mock-ups, and found that the design significantly improved the user experience over the stock configuration panel [36]. As part of the current study, we implemented that interface in Android.

Figure 2 shows our contextual permission manager. We built it to run as a system-space app, similar to Android's *Settings* app. Our permission manager has three main objectives: (i) to

display all recent permission requests and the corresponding "allow" or "deny" decisions from the ML model; (ii) to allow users to review and change app permissions; and (iii) to display all the resources an app can access.

When users set preferences (rules) in the permission manager, before making a ML decision, the platform checks to see if the user has set any rules for the current request; if a match is found, rather than going to the ML, the platform will apply the rule to the permission request accordingly. The system does not use these rules to train the ML model, as it is hard to capture the contextuality behind these changes, so the platform cannot create any of the contextual features to train the ML.

VALIDATION METHODOLOGY

We tested our implementation by performing a field study with 37 participants. Our goals were to understand how third-party apps and end-users react to a more restrictive and selective permission model, as compared to the default AOFU model.

For a period of one week, each participant used a smartphone (Nexus 5X) running a custom version of the Android OS (based on 6.0.1) built with our new permission system. During the study period, all of a participant's sensitive data was protected by the new contextually-aware permission model.

Participant Privacy Preferences

We used the Experience Sampling Method (ESM) to understand how participants want to control certain sensitive resource accesses [17]. ESM involves repeatedly questioning participants *in situ* about a recently observed event. We probabilistically asked them about an app's recent request to access data on their phone, and how they want to control future similar requests (Figure 1). We treated participants' responses to these ESM prompts as our main dependent variable in the training phase and then to validate the accuracy of the decisions that the trained classifier was automatically making.

Each participant during the study period responded to 4 prompts per day, and at most one per hour. The prompting was divided into two phases. The first phase was the *bootstrapping phase*, which we described earlier, to train the classifier. The second phase was the *validation phase*, which was used to measure the accuracy of the ML model. In addition to the validation phase prompts, participants might also have occasional prompts for low-confidence decisions made by the ML; a detailed discussion on low-confidence decisions is provided later. During our study period, only 4 participants ever experienced low-confidence prompts.

Recruitment

We recruited participants in two phases: a pilot in May 2017 and the full study in August 2017. We placed a recruitment ad on Craigslist under "et cetera jobs" and "domestic gigs."[2] The title of the ad was "Smartphone Research Study," and it stated that the study was about how people interact with their smartphones. We made no mention of security or privacy. Interested participants downloaded a screening app from the Google Play store, which asked for demographic information

and collected their smartphone make and model. This step also made sure that all of our participants were Android users. We screened out those who were under 18 or used CDMA providers, because our experimental phones were only GSM-compatible. We collected data on participants' installed apps, so that we could pre-install their free apps prior to them visiting our laboratory. (We only encountered paid apps for a few participants, and those apps were installed once we setup their Google accounts on the test phones.) These apps ranged from games to sensitive financial apps.

We scheduled participants who met the screening requirements to do the initial setup. Overall, 63 people showed up to our laboratory, and of those, 2 were rejected because our screening app did not identify some CDMA carriers. The initial setup took roughly 30 minutes and involved transferring SIM cards, setting up their Google and other accounts, and making sure they had all the apps they used. We compensated each participant with a $35 gift card for showing up.

During the pilot phase, out of 20 people who were given phones, 14 participants had technical issues with the phone preventing them from using it, leaving only 6 participants with usable data. During the main phase, out of 42 people who were given phones, we had the following issues:

- 4 participants misinterpreted our ESM prompts so we filtered out their prompt responses;
- 5 participants suffered from a bug in the code that inhibited the validation phase of the ML;
- 2 participants performed factory resets on the phone before returning it, which deleted our experimental data.

This left 31 participants with usable data from the main phase. We combined the 6 participants with usable data from the first phase with the 31 from the second phase to produce our sample of 37 users, since we did not alter the study between phases (other than fixing technical issues). All our results are drawn from log data and interview responses from those 37 users. Of that sample, 21 were female and 16 were male; ages ranged from 18 to 59 years old ($\mu = 34.25$, $\sigma = 9.92$).

After the initial setup, participants used the experimental phones for one week in lieu of their normal phones. They were allowed to install, use, and uninstall any apps that they wanted. Our logging framework kept track of every protected resource accessed by an app, along with the contextual data surrounding those requests. All the logged events were stored compressed in the local system.

Exit Interview

When participants returned to our laboratory, we first copied the log data from the phones to make sure that they had actually used the phone during the study period. We then administered a semi-structured exit interview, which had four components:

- **New Permission Manager UI**—We asked participants to show us how they would use the UI (Figure 2) to block a given app from accessing background location data, as well as how difficult they found it. We also checked our data to see how they interacted with the UI during the study period, and asked them about the circumstances for those

interactions. The objective of this task was to validate the design objectives of the UI, including whether they use it to resolve issues stemming from resource denial.

- **Permission Prompts**—We asked participants questions about permission prompts they had encountered during the study. We asked why they allowed or denied permission requests and also how they felt about the prompts. We asked them to rate their experience with the prompts across 3 different categories: level of surprise, feelings of control, and to what extent they felt the new system had increased transparency. The objective of this section was to understand the impact of the runtime prompts.
- **Permission Models**—We asked participants questions about their perspectives on the privacy protections in Android. We asked how much they understood the current system. We then explained our new system, and asked how they felt about letting the ML act on their behalf. The objective of this section was to understand how much participants actually understood the new permission model.
- **Privacy Concerns**—Finally, we asked participants how they usually make privacy decisions on their mobile devices, how serious they are about privacy, how much they are willing to pay for privacy, and demographic questions.

Three researchers independently coded 144 responses to the *Permission Prompts* and *Permission Model* questions (the other questions involved either direct observations or reporting participants' responses verbatim without the need for coding). Prior to meeting to achieve consensus, the three coders disagreed on 17 responses, which resulted in an inter-rater agreement of 86.43% (Fleiss' $\kappa = 0.747$).

After the exit survey, we answered any remaining questions, and then assisted them in transferring their SIM cards back into their personal phones. Finally, we compensated each participant with a $100 gift card.

RESULTS
We collected 1,159 prompt responses from 37 participants. A total of 133 unique apps caused prompts for 17 different sensitive permission types. During the study period, 24.23% of all runtime prompts were denied by participants. Most (66%) of these prompts occurred when the requesting app was running visibly. Our instrumentation logged 5.4M sensitive permission requests originating from 253 unique apps for 17 different permission types. On average, a sensitive permission request occurred once every 4 seconds.

In the remainder of the paper, we describe the shortcomings of the existing AOFU permission model, both in accuracy[3] and in aligning with users' expectations; we show how our proposed system has vastly greater accuracy in inferring users' privacy preferences and applying them towards regulating app permissions; and we show that is does this with minimal impact on app functionality. Finally, we present results from the exit interviews regarding participants' perceptions about the training prompts and the privacy management user interface.

---

[3]We report the median accuracy among participants because all distributions were heavily left-skewed. The mean is not representative, and therefore the median better captures the improvements in practice.

Status Quo Problems
In the "ask-on-first-use" (AOFU) model, the user receives prompts to grant individual permissions, but only the first time an app requests them. Requesting these permissions at runtime allows the user to infer the potential reason for the request, based on what they were doing when the request occurred (i.e., context). AOFU's shortcoming, however, is that it naïvely reapplies the user's first-use decision in subsequent scenarios, ignoring different contexts [38].

We previously attempted to measure the accuracy of the AOFU model by collecting users' responses to runtime permission prompts, without actually enforcing them by denying apps access to data [39]. Thus, the accuracy rates reported by that study may not actually be valid, since users may elect to change their permission-granting preferences, if they result in a loss of app functionality. Thus, we evaluated the performance of the AOFU approach (in current use by Android and iOS) by presenting participants with permission prompts that *actually* resulted in the denial of app permissions.

During the study period, each participant responded to combinations of *app:permission* more than once. As AOFU is deterministic, the decision the user makes for a prompt becomes the system's decision in perpetuity. Because our learning phase prompted participants the first time an app requested data—which is the same set of circumstances under which AOFU would have prompted them—we can interpolate from these first prompts to calculate AOFU accuracy. Thus, no control condition is needed, because this interpolation allows us to gather the same data. We use this data to measure how often AOFU matches the user's preference in subsequent requests.

Our data show that the AOFU permission model has a median error rate of 20%: in one-fifth of app requests for permission-protected resources, participants changed their initial response for the same *app:permission* combination. Of our 37 participants, 64% had at least one such discrepancy between the first-use and subsequent preferences. This refutes AOFU's core assumption that few users will deviate from their initial preferences, and corroborates our prior study, in which 79% of 131 participants deviated from their initial responses [39].

Errors under AOFU could be either privacy violations or losses of functionality. Privacy violations occur when the system grants an app access to a protected resource, contrary to user preferences. Loss of functionality occurs when the permission system denies access to a protected resource, which the user would have otherwise permitted. We consider privacy violations to be the more severe type of error, as the user is unable to take back sensitive information once an app has acquired it and transmitted it to a remote server. Loss of functionality, however, is still undesirable because it might incentivize the user to be overly permissive in the future. From our data, we found that 66.67% of AOFU errors were privacy violations; the remaining 33.33% were losses in functionality.

AOFU User Expectations
Errors in permission systems could arise from a variety of reasons. Mismatched user expectations and lack of comprehension are two critical ones, which could hamper any permis-

sion model's utility. User comprehension is critical because users may make suboptimal decisions when they do not fully understand permission prompts, hindering the ability of the permission system to protect sensitive system resources. Users must be able to comprehend the decision they are making and the consequences of their choices. Recent work on AOFU has examined the motives behind users' decisions and how they vary between categories of apps, as well as how people adapt their behavior to the new model [5, 3, 2].

Our participants had an average of 5 years of experience with Android. This indicates that most have experienced both install-time permissions—the permission model prior to Android 6.0—and runtime AOFU permission prompts. The majority of participants said they noticed the shift to AOFU, and they were aware that these prompts are a way to ask the user for consent to share data with an app. A large minority of participants ($\approx$40%), however, had an inadequate understanding of how AOFU works, which could substantially hinder that permission model's effectiveness in protecting user data.

Four of the 37 participants expressed doubts about the rationale behind the prompts; rather than seeing permission prompts as a way for users to regulate access to their sensitive data, these participants viewed these prompts as a mechanism to extract more information from them:

"When I see prompts, I feel like they want to know something about me, not that they want to protect anything." (P21)

A third (31.4%) of our participants were not aware that responding to an AOFU prompt results in a blanket approval (or denial) that carries forward to all the app's future uses of the requested resource. Most participants believed that responses were only valid for a certain amount of time, such as just for that session or just that single request. This misconception significantly hinders AOFU's ability to correctly predict the user's preferences. Again, this observation raises the question of whether users would respond differently if they had a more accurate understanding of how AOFU works:

"[I] didn't know that granting a permission carries forward in the future until otherwise changed. [I] expected permissions to be for just that one use." (P25)

It is clear that granting blanket approval to sensitive resources is not what users expect. On the other hand, had our participants been asked for their input on every permission request, they would have received a prompt every 4 seconds. How, then, can we best protect users' privacy preferences to future scenarios without overwhelming them with prompts?

Classifier Accuracy

During the week-long study period, each participant was subject to two operational phases of the contextual permission system: (a) the initial *learning phase*, where participant responses to prompts were used to re-train the SVM classifier according to each individual's preferences, and (b) the steady-state *validation phase*, where responses to prompts were collected to measure the accuracy of the classifier's decisions.

As previously discussed in our section on bootstrapping, we use weighted reservoir sampling during the learning phase to prioritize prompting for the most commonly observed instances of *app:permission:visibility* combinations. During the validation phase, participants received the same prompts, triggered by random combinations of features. This ensured that we collected validation results both for previously-encountered and new combinations. We placed a maximum limit of 3 prompts per combination in order to further improve prompt diversity and coverage. After presenting participants with prompts, the instrumentation recorded the response and the corresponding decision produced by the classifier. Using participant responses to prompts as ground truth, we measured the classifier's accuracy during the validation phase. From our sample of 37 participants, we had to exclude 6 of them due to a cache coherency bug that was discovered after the pilot, which degraded classifier performance. For the remainder of this section, our results are drawn from the remaining sample of 31, unless otherwise noted.

These 31 participants responded to 640 total prompts in the validation phase. Our contextual permission model produced a median accuracy of 90.24%, compared to 80.00% under AOFU for the same population. The classifier reduced AOFU's error rate by 50%, with the majority of classifier errors consisting of privacy violations (i.e., incorrectly granting access).

Offline Learning

We were curious whether the accuracy of our system could be improved through the use of offline learning, which would require much more computing power. Using participant responses to permission prompts, we analyzed how an offline SVM classifier would perform. We implemented the SVM model using the *KSVM* module in *R*. We performed this analysis on data from all 37 participants, using leave-one-out cross-validation to evaluate how the offline classifier would perform for each participant.

The offline model had a median accuracy of 94.74% across the 37 participants. By comparison, AOFU had a 80% accuracy for the same population. This represents a 75% error reduction in the offline contextual model compared to AOFU. These numbers corroborate prior findings [39]. We stress the significance of this corroboration, because the results hold in the presence of actual resource denial, which was not the case for the prior study. This suggests that users will continue to indicate their true preferences in response to prompts, even when those preferences are enforced, potentially resulting in unanticipated app behavior.

We note the accuracy difference between the SVM classifier we integrated into Android and the *R* model (90.24% vs. 94.74%, respectively). This is due to how the Android SVM implementation performs the bootstrapping. This issue is not inherent to integrating an SVM classifier into Android. An updated implementation has the potential to reach the maximum accuracy observed in the offline model.

We calculated F-score for the two classes: "allow" and "deny." Under the "allow" class, the calculated F-Scores were 0.73 for AOFU, 0.83 for on-device ML, and 0.89 for offline ML.

Under the "deny" class, the F-scores were 0.45, 0.50 and 0.51, respectively. The contextual cues has helped the ML model to outperform AOFU in both predicting when to share the data and when to deny the data.

Decision Confidence

We previously proposed using decision confidence to determine for which *app:permission:visibility* combinations users should be prompted in the validation phase [39]. The rate of decision confidence is also a measure of the extent to which the classifier has correctly learned the user's preferences. We suggested that if this rate does not decrease over time, then AOFU will likely be a better system for those users.

In addition to the prediction, our classifier also produced a class probability, which we used as the measure of decision confidence. The classifier produced a binary result (i.e., allow or deny) with a cutoff point of 0.5. A decision probability close to the cutoff point is a less confident result than one far from it. We used the 95% confidence interval as a threshold for determining low-confidence decisions.

Only four of our field study participants experienced low-confidence classifier decisions that caused a prompt to appear after the bootstrapping period. Each of these participants had just one such low-confidence prompt appear. These prompts retrained the classifier, so the lack of any subsequent low-confidence prompts indicates that the classifier produced high-confidence predictions for the same *app:permission:visibility* combination in future cases.

The lack of additional training prompts also suggests that users are less likely to become habituated to prompting. The four participants who each received one additional prompt saw a total of 13 prompts (including the 12 during the training phase). The remaining 27 participants saw just the 12 training phase prompts. Had our participants been subject to AOFU instead of our contextual permission system, they would have received a median of 15 prompts each, with a quarter of the participants receiving more than 17. Instead, we achieved a 75% error reduction (80.00% vs. 94.74%) and reduced user involvement by 20% (12 prompts vs. 15) through the use of classifier-driven permissions, compared to AOFU.

Impact on App Functionality

Prior work found that many apps do not properly handle cases where they are denied access to protected resources [10]. One core objective of our work was to measure how apps responded to a stricter permission model than AOFU. For example, the system will be unusable if it causes erratic app behavior, through the use of dynamically granted permissions.

In the field study, our platform instrumentation recorded each app crash and its corresponding exception message. This information allowed us to identify the *possible* root cause of the crash and whether it was related to resource denial. We observed 18 different exceptions classes, such as `SecurityException`, `RuntimeException`, and `NullPointerException`. For the remainder of this section, we will only discuss the `SecurityException` class, as it is directly related to resource denials. Nearly all (98.96%) of the recorded `SecurityExceptions` were

observed on the devices of just two participants. Each of the remaining participants encountered, on average, 18 `SecurityExceptions` during the study period (i.e., roughly 3 `SecurityExceptions` per day per participant).

Almost all (99.93%) `SecurityExceptions` were caused when apps attempted to read subscriber information (i.e., the READ_PHONE_STATE permission, used to obtain the phone number). In the event of a READ_PHONE_STATE denial, our implementation did not supply the app with any phone number data. As discussed earlier, we do not supply a randomly-generated phone number to avoid collateral privacy violations.

For other denials, we opted to supply apps with generated data to ensure their continued operation, without actually exposing private user data. During the study period, the classifier denied 10.34% of all permission requests; more than 2,000 denials per participant per day. Our implementation, however, only recorded an average of 3 `SecurityExceptions` per day per participant. This indicates that passing synthetic but well-formed data to apps in lieu of actual private user data does satisfy app functionality expectations to a great extent.

Our results are a positive sign for future permission systems more restrictive than the current AOFU model: permissions can be more restrictive without forcing the user to trade off usability for improved privacy protection, as we will show in the next section. If apps gracefully handle resource denials, then users are free to specify their privacy preferences without risking functionality losses.

User Reactions to Prompts

We measured how much participants were surprised to see the prompts during the course of the study period (on a scale of 1="not surprised" to 5="very surprised"). Participants expressed an average rating of 2.7. Almost half (44%) of the participants indicated that the prompts surprised them, and among them, 70% were surprised at the frequency with which the prompts appeared (up to 4 times per day), though few participants expressed annoyance by that frequency (8%).

We asked participants to rate how much they felt that they were in control of resource usage (on a scale of 1="nothing changed compared to default Android" to 5="very much in control"). On average, our participants rated their experience as 3.44. Almost half (44%) felt that they were in control of the system as a result of the prompts. A small number (14%) still felt helpless, regardless of their responses to the prompts. They felt resigned that apps would always obtain their data.

Finally, we asked how they felt about the transparency provided by the new system (on a scale of 1="nothing changed" to 5="improved system transparency"). On average, participants rated system transparency in the middle (3). Almost half (47%) of them felt that the new system was more transparent. A minority (14%) mentioned wanting to know *why* apps were requesting particular sensitive data types.

From these observations, we believe that our contextual permission system is a positive step toward improving user awareness, enabling users to make more informed privacy decisions.

User Reactions to Controls

Whenever an automated system makes decisions on a user's behalf, there is the inevitable risk that the system will make an incorrect decision. In our case this can cause apps to be over-privileged and risk privacy violations, or be under-privileged and risk app failure or reduced functionality. It is important to empower users so they can easily audit the decisions that were made on their behalf and to amend those decisions that are not aligned with their preferences.

We built a user interface based on our prior work [36], which allowed our participants to view automated permissions decisions made by the classifier, as well as set privacy preferences with respect to context (i.e., the visibility of the requesting app). We included this user interface as part of the operating system, as a panel within the system settings app.

When we on-boarded our participants, we mentioned to them that there was a new "permission manager" available, but to avoid priming them, we made sure not to emphasize it in any particular way. Our instrumented platform logged every time participants interacted with our permission manager to understand how they used it.

Fifteen of the 37 participants (40.5%) opened the permission manager during the study period. Our implementation logged a total of 169 preference changes across these participants. Only 6 out of 37 participants (16.2%) changed the settings to be *more restrictive*. Of the adjustments made towards more restrictiveness, the majority were for the GET_ACCOUNTS permission, which prevents apps from reading the user's stored credentials (e.g., usernames linked to accounts on the device, such as for Google, Twitter, etc.). In contrast, the most-common permission that participants adjusted to be more permissive was READ_CONTACTS. When asked for their motives behind these changes, the majority of participants said that functionality was their main reason for granting more access, and the sensitivity of data for restricting access.

We also asked participants to demonstrate how they would change the settings of a familiar app to only be able to access their location when they are using that app. We based this task off of one of our previous tasks used to evaluate the low-fidelity prototype of the interface [36]. Using the functional interface, all but two of our participants were able to correctly complete this task. Participants rated the average ease of the task as 1.15 (on a scale from 1="very easy" to 5="very hard"). We conclude that participants are able to understand the permission interface after having used it for a week and without special instructions.

The permission manager also enables users to diagnose app crashes that result from resource denial (a feature not present in the low-fidelity prototype). In exit interviews, we examined how participants responded to app crashes in their experiences with the device. The majority of participants reported that their first step was to restart the app that had crashed. If unsuccessful, they would then restart their phone. This informs the design of a future system: if an app crashes as a result of a resource denial, the platform could clearly communicate this to users through a dialog or the notification bar.

DISCUSSION

The core objective of our 37-person field study was to analyze how a contextually-aware, more-restrictive permission model performs in the wild. We examined how participants balanced their privacy preferences with app functionality. This measures the real-world applicability of predicting user privacy decisions with the help of contextual cues surrounding each permission request.

Consequential Denial

Overall, participants denied 24% of all prompted permission requests. This is a 60% reduction in denials compared to our prior results [39], which framed the question using only hypothetical language (i.e., permissions were not actually denied to apps): "given the choice, would you have denied...?" The decreased denial rate we observed is therefore unsurprising given that participants were now potentially making a tradeoff between functionality and privacy, instead of expressing the degree to which privacy is important to them. Our results show that even in the presence of consequential resource denial, contextual cues helped to predict users' privacy decisions and better aligned permission settings with their expectations, as compared to the *status quo*.

Ask on First Use

Our results corroborate our previous findings: AOFU's inability to capture the context surrounding users' decisions is a significant cause of privacy violations [38, 39]. We also found that a significant portion of participants do not have an adequate understanding of how AOFU works, which further limits its utility: 11 participants did not realize that their prompt responses remained in effect in perpetuity, and 4 participants believed that the prompts were in furtherance of privacy-invasive activities. While the actual impact of these inaccurate beliefs is yet to be explored, we believe that these issues need to be addressed in the future, in order to increase Android's ability to support user privacy preferences and protect user data.

Experimental Caveats

Our experiments involved participants using their existing apps on fresh devices, meaning that their existing AOFU preferences were not carried over. This means that were they to use a stock Android device they would be bombarded with AOFU prompts at the beginning of the experience. In contrast, our system, to avoid habituation to our prompts, we rate limited the number of prompts to four prompts per day.

One caveat of this approach is if participants hit this prompting limit then they would not see a prompt for a new *app:permission combination*, though they would see such a prompt later when the rate limit is not in effect. It is possible that our simulated AOFU prompts may have appeared, in some cases, under different circumstances than if they had simply been using the AOFU model. However, our prior work showed that the circumstances surrounding the initial prompt have limited effect on the overall efficacy of the AOFU system [39].

In a real-world deployment, we envision prompting to be sporadic, primarily occurring when a new app is installed. The post-study interviews, however, did not show that the current

rate created user fatigue. In the exit interviews, only 3 of the 37 participants mentioned that the prompts annoyed them. Although 70% of the participants were indeed surprised due to the frequency of the prompts, that surprise did not pose a burden, but rather made them think more about access to sensitive data. Our accuracy rate corroborates our prior work [39], in which participants were prompted only once per day over a longer study period (compared to our study's rate of 4 prompts/day). For example, in our study, AOFU had an accuracy of 80%, while the prior study had an accuracy rate of 84% [39]. Consequently, we do not believe that the relatively shorter study period and higher prompting rate appreciably impacted the results.

SVM Implementation Limitations
While our new permission model reduces the number of mis-predictions compared to AOFU by 50%, our offline analysis shows that it has the potential to reduce mis-predictions by 75%. A further examination revealed that the performance difference is due to the bootstrapping of the training dataset in the implementation. We note that difference is not inherent to running a classifier on Android, and so simply modifying our implementation to use these improvements will allow it to achieve the same performance.

Purpose
While our new permission model outperforms AOFU, it still does not explain to the user *why* an app needs to use a permission. In our exit interviews, we observed that 14% of participants expressed the desire to know *why* apps made a request in the first place. Previous work has shown that app functionality is a key factor in permission decisions [5]. If users were properly informed of the functionality requirement behind a permission request, then they might be better positioned to make decisions that meet their privacy and functionality expectations.

We believe that there are ways to extend contextual permission systems by incorporating the actual purpose of the request. For example, after introducing AOFU permissions, Android started encouraging app developers to provide the reason behind their permission requests so that the user can include that in the decision-making process [7]. Tan et al. [34] showed that similar prompts on iOS actually resulted in users being more permissive about granting permissions to apps. Similarly, prior work has attempted to use static analysis to automatically incorporate inferred purpose [24, 23].

Resource Denial
When deploying more-restrictive permission systems, it is important that apps continue to run without entering into an error state that results from a resource denial. Users should be able to select their privacy preferences with minimal disruption to their experience; apps must not be able to force an ultimatum by simply not functioning if a permission is denied. Indeed, some participants simply allow most permission requests because that ensures their apps run properly.

The platform, therefore, is responsible for ensuring that apps handle resource denials gracefully. To their credit, when Android introduced AOFU, it implemented some permission de-

nials to appear like a lack of available data or the non-existence of hardware, instead of throwing a `SecurityException`. In our implementation, we take the extra step of supplying apps with generic but well-formed data in the event of a denial. We observed that our participants tended to deny more permissions as they progressed through the study period: 20% of permissions were denied in the learning phase, compared to 26% during the validation phase. Those participants also experienced a low rate of app failures due to resource denials. Future research is needed to develop more measures to reduce functionality losses stemming from enforcing stricter privacy preferences. Failing to do so might otherwise compel users to compromise their privacy preferences for the sake of functionality.

Remedying Unexpected Behavior
Regardless of any mitigations to avoid app crashes, it is practical to assume that apps will crash when they fail to receive expected data under certain circumstances. One way to remedy this is to give users tools to adjust the behavior of the permission system, such as being able to be more permissive to certain apps in certain contexts. This approach, however, assumes that (i) users accurately attribute a crash event to a resource denial, which may not always be the case, and (ii) users are able identify which resource denial caused the crash. In our implementation of a new permission manager, we addressed the latter assumption by providing users a timeline of recent decisions made by the new permission system, which can be used to deduce the cause of a crash.

Our exit interviews showed that few participants thought to check the permission manager following an app crash, so clearly more work is needed here. With proposals for more accurate and more restrictive permission models, it is necessary to have usable mechanisms to deal with inevitable crashes due to resource denials. The platform should provide mechanisms either to help the user diagnose and resolve such crashes, or to automatically fix permissions on a temporary basis and give the user an option to make the fix permanent.

Conclusion
This study showed how apps and users respond to a real-world deployment of a novel contextually-aware permission model. The new permission system significantly reduced the error rate from that of the prevailing "ask-on-first-use" model first deployed in Android 6.0. While prior work already demonstrated ways to increase the protection provided by new permission models, we believe our study provides opportunities to further improve performance and address practical limitations in actual implementations.

REFERENCES

1. Hussain M.J. Almohri, Danfeng (Daphne) Yao, and Dennis Kafura. 2014. DroidBarrier: Know What is Executing on Your Android. In *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy (CODASPY '14)*. ACM, New York, NY, USA, 257–264. DOI:`http://dx.doi.org/10.1145/2557547.2557571`

2. Panagiotis Andriotis, Shancang Li, Theodoros Spyridopoulos, and Gianluca Stringhini. 2017. *A Comparative Study of Android Users' Privacy Preferences Under the Runtime Permission Model*. Springer International Publishing, Cham, 604–622. DOI:`http://dx.doi.org/10.1007/978-3-319-58460-7_42`

3. P. Andriotis, M. A. Sasse, and G. Stringhini. 2016. Permissions snapshots: Assessing users' adaptation to the Android runtime permission model. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*. 1–6. DOI:`http://dx.doi.org/10.1109/WIFS.2016.7823922`

4. Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. 2006. Privacy and Contextual Integrity: Framework and Applications. In *Proc. of the 2006 IEEE Symposium on Security and Privacy (SP '06)*. IEEE Computer Society, Washington, DC, USA, 15. DOI:`http://dx.doi.org/10.1109/SP.2006.32`

5. Bram Bonné, Sai Teja Peddinti, Igor Bilogrevic, and Nina Taft. 2017. Exploring decision making with Android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 195–210. `https://www.usenix.org/conference/soups2017/technical-sessions/presentation/bonne`

6. Chih-Chung Chang and Chih-Jen Lin. LIBSVM – A Library for Support Vector Machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. (????). Accessed: September 11, 2017.

7. Google Developer. Requesting Permissions at Run Time. `https://developer.android.com/training/permissions/requesting.html`. (????). Accessed: September 16, 2017.

8. Serge Egelman, Adrienne Porter Felt, and David Wagner. 2012. Choice Architecture and Smartphone Privacy: There's A Price for That. In *The 2012 Workshop on the Economics of Information Security (WEIS)*.

9. William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 1–6. `http://dl.acm.org/citation.cfm?id=1924943.1924971`

10. Zeran Fang, Weili Han, Dong Li, Zeqing Guo, Danhao Guo, Xiaoyang Sean Wang, Zhiyun Qian, and Hao Chen. 2016. revDroid: Code Analysis of the Side Effects after Dynamic Permission Revocation of Android Apps. In *Proceedings of the 11th ACM Asia Conference on Computer and Communications Security (ASIACCS 2016)*. ACM, Xi'an, China.

11. Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. 2012b. How to ask for permission. In *Proc. of the 7th USENIX conference on Hot Topics in Security*. USENIX Association, Berkeley, CA, USA, 1. `http://dl.acm.org/citation.cfm?id=2372387.2372394`

12. Adrienne Porter Felt, Serge Egelman, and David Wagner. 2012a. 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)*. ACM, New York, NY, USA, 33–44. DOI:`http://dx.doi.org/10.1145/2381934.2381943`

13. Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012c. Android permissions: user attention, comprehension, and behavior. In *Proc. of the Eighth Symposium on Usable Privacy and Security (SOUPS '12)*. ACM, New York, NY, USA, Article 3, 14 pages. DOI:`http://dx.doi.org/10.1145/2335356.2335360`

14. Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. 2012. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *Proc. of the 5th Intl. Conf. on Trust and Trustworthy Computing (TRUST'12)*. Springer-Verlag, Berlin, Heidelberg, 291–307. DOI:`http://dx.doi.org/10.1007/978-3-642-30921-2_17`

15. Google. Dangerous Permissions. `https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous`. (????). Accessed: August 17, 2017.

16. Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 1025–1035. DOI:`http://dx.doi.org/10.1145/2568225.2568276`

17. Stefan E Hormuth. 1986. The sampling of experiences in situ. *Journal of personality* 54, 1 (1986), 262–293.

18. Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. 2011. 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)*. ACM, New York, NY, USA, 639–652. DOI:`http://dx.doi.org/10.1145/2046707.2046780`

19. Jaeyeon Jung, Seungyeop Han, and David Wetherall. 2012. Short Paper: Enhancing Mobile Application Permissions with Runtime Feedback and Constraints. In *Proceedings of the Second ACM Workshop on Security*

and Privacy in Smartphones and Mobile Devices (SPSM '12). ACM, New York, NY, USA, 45–50. DOI: `http://dx.doi.org/10.1145/2381934.2381944`

20. Patrick Gage Kelley, Sunny Consolvo, Lorrie Faith Cranor, Jaeyeon Jung, Norman Sadeh, and David Wetherall. 2012. 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)*. Springer-Verlag, Berlin, Heidelberg, 68–79. DOI: `http://dx.doi.org/10.1007/978-3-642-34638-5_6`

21. William Klieber, Lori Flynn, Amar Bhosale, Limin Jia, and Lujo Bauer. 2014. Android Taint Flow Analysis for App Sets. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis (SOAP '14)*. New York, NY, USA, 6. DOI:`http://dx.doi.org/10.1145/2614628.2614633`

22. Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I. Hong. 2014. Modeling Users' Mobile App Privacy Preferences: Restoring Usability in a Sea of Permission Settings. In *Symposium On Usable Privacy and Security (SOUPS 2014)*. USENIX Association, Menlo Park, CA, 199–212.

23. Jialiu Lin, Norman Sadeh, Shahriyar Amini, Janne Lindqvist, Jason I. Hong, and Joy Zhang. 2012. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proc. of the 2012 ACM Conf. on Ubiquitous Computing (UbiComp '12)*. ACM, New York, NY, USA, 501–510. DOI:`http://dx.doi.org/10.1145/2370216.2370290`

24. Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. 2016. Follow My Recommendations: A Personalized Assistant for Mobile App Permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*.

25. Bin Liu, Jialiu Lin, and Norman Sadeh. 2014. 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)*. ACM, New York, NY, USA, 201–212. DOI: `http://dx.doi.org/10.1145/2566486.2568035`

26. Kristopher Micinski, Daniel Votipka, Rock Stevens, Nikolaos Kofinas, Jeffrey S. Foster, and Michelle L. Mazurek. 2017. User Interactions and Permission Use on Android. In *CHI 2017*.

27. Helen Nissenbaum. 2004. Privacy as contextual integrity. *Washington Law Review* 79 (February 2004), 119.

28. Helen Nissenbaum. 2009. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press.

29. Katarzyna Olejnik, Italo Ivan Dacosta Petrocelli, Joana Catarina Soares Machado, Kévin Huguenin, Mohammad Emtiyaz Khan, and Jean-Pierre Hubaux.

2017. SmarPer: Context-Aware and Automatic Runtime-Permissions for Mobile Devices. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP)*. IEEE.

30. Talia Ringer, Dan Grossman, and Franziska Roesner. 2016. AUDACIOUS: User-Driven Access Control with Unmodified Operating Systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 204–216.

31. Franziska Roesner and Tadayoshi Kohno. 2013. Securing embedded user interfaces: Android and beyond. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. 97–112.

32. Jialiu Lin Bin Liu Norman Sadeh and Jason I Hong. 2014. Modeling users' mobile app privacy preferences: Restoring usability in a sea of permission settings. In *Symposium on Usable Privacy and Security (SOUPS)*.

33. Bilal Shebaro, Oyindamola Oluwatimi, Daniele Midi, and Elisa Bertino. 2014. IdentiDroid: Android Can Finally Wear Its Anonymous Suit. *Trans. Data Privacy* 7, 1 (April 2014), 27–50. `http://dl.acm.org/citation.cfm?id=2612163.2612165`

34. Joshua Tan, Khanh Nguyen, Michael Theodorides, Heidi Negron-Arroyo, Christopher Thompson, Serge Egelman, and David Wagner. 2014. The Effect of Developer-Specified Explanations for Permission Requests on Smartphone User Behavior. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*.

35. Christopher Thompson, Maritza Johnson, Serge Egelman, David Wagner, and Jennifer King. 2013. 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)*.

36. Lynn Tsai, Primal Wijesekera, Joel Reardon, Irwin Reyes, Serge Egelman, David Wagner, Nathan Good, and Jung-Wei Chen. 2017. Turtle Guard: Helping Android Users Apply Contextual Privacy Preferences. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*. USENIX Association, Santa Clara, CA, 145–162.

37. Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. 2012. Permission Evolution in the Android Ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12)*. ACM, New York, NY, USA, 31–40. DOI: `http://dx.doi.org/10.1145/2420950.2420956`

38. Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C., 499–514.

39. P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. 2017. The

Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*. 1077–1093. DOI: **http://dx.doi.org/10.1109/SP.2017.51**

40. Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile App Recommendations with Security and Privacy Awareness. In *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 10. DOI: **http://dx.doi.org/10.1145/2623330.2623705**