The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences

Primal Wijesekera[1,2], Arjun Baokar[2], Lynn Tsai[2], Joel Reardon[2],
Serge Egelman[2], David Wagner[2], and Konstantin Beznosov[1]
[1]University of British Columbia, Vancouver, Canada,
}@ece.ubc.ca
[2]University of California, Berkeley, Berkeley, USA,
{ }@berkeley.edu, { }@cs.berkeley.edu

***Abstract*—Current mobile operating systems regulate application permissions by prompting users on an ask-on-first-use basis. Prior research has shown that this method is ineffective because it fails to account for context: the circumstances under which an application first requests access to data may be vastly different than the circumstances under which it subsequently requests access. We performed a longitudinal 131-person field study to analyze the contextuality behind user privacy decisions to regulate access to sensitive resources. We built a classifier to make privacy decisions on the user's behalf by detecting when context has changed and, when necessary, inferring privacy preferences based on the user's past decisions and behavior. Our goal is to automatically grant appropriate resource requests without further user intervention, deny inappropriate requests, and only prompt the user when the system is uncertain of the user's preferences. We show that our approach can accurately predict users' privacy decisions 95.7% of the time, which is a four-fold reduction in error rate compared to current systems.**

I. Introduction

One of the roles of a mobile application platform is to help users avoid unexpected or unwanted use of their personal data [9]. Mobile platforms currently use permission systems to regulate access to sensitive resources, relying on user prompts to determine whether a third-party application should be granted or denied access to data and resources. One critical caveat in this approach, however, is that mobile platforms seek the consent of the user the first time a given application attempts to access a certain data type and then enforce the user's decision for all subsequent cases, regardless of the circumstances surrounding each access. For example, a user may grant an application access to location data because she is using location-based features, but by doing this, the application can subsequently access location data for behavioral advertising, which may violate the user's preferences.

Earlier versions of Android (5.1 and below) asked users to make privacy decisions during application installation as an all-or-nothing ultimatum (ask-on-install): either all requested permissions are approved or the application is not installed. Previous research showed that few people read the requested permissions at install-time and even fewer correctly understood them [14]. Furthermore, install-time permissions do not present users with the context in which those permission will be exercised, which may cause users to make suboptimal decisions not aligned with their actual preferences. Asking users to make permission decisions at runtime, at the moment when the permission will actually be used by the application, provides more context (i.e., what they were doing at the time that data was requested) [12]. However, due to the high frequency of permission requests, it is not feasible to prompt the user every time data is accessed [33].

In iOS and Android M, the user is now prompted at runtime the first time an application attempts to access one of a set of "dangerous" permission types (e.g., location, contacts, etc.). This "ask-on-first-use" (AOFU) model is an improvement over ask-on-install (AOI). Prompting users the first time an application uses one of the designated permissions gives users a better sense of context: their knowledge of what they were doing when the application first tried to access the data should help them determine whether the request is appropriate. However, Wijesekera et al. showed that AOFU fails to meet user expectations over half the time, because it does not account for the varying contexts of future requests [33].

The notion of *contextual integrity* suggests that many permission models fail to protect user privacy because they fail to account for the context surrounding data flows [27]. That is, privacy violations occur when sensitive resources are used in ways that defy users' expectations. We posit that more effective permission models must focus on whether resource accesses are likely to defy users' expectations in a given context—not simply whether the application was authorized to receive data the first time it asked for it. Thus, the challenge for system designers is to correctly infer when the context surrounding a data request has changed, and whether the new context is likely to be deemed "appropriate" or "inappropriate" for the given user. Dynamically regulating data access based on the context requires more user involvement to understand users' contextual preferences. If users are asked to make privacy decisions too frequently, or under circumstances that are seen as low-risk, they may become habituated to future, more serious, privacy decisions. On the other hand, if users are asked to make too few privacy decisions, they may find that the system has acted against their wishes. Thus, research is needed to determine *when* and under *what* circumstances to present users with runtime prompts.

To this end, we collected real-world Android usage data in order to explore whether we could infer users' future privacy decisions based on their past privacy decisions, contextual circumstances surrounding applications' data requests, and users' behavioral traits. We conducted a field study where

131 participants used Android phones that were instrumented to gather data over an average of 32 days per participant. Also, their phones periodically prompted them to make privacy decisions when applications used sensitive permissions, and we logged their decisions. Overall, participants wanted to block 60% of these requests. We found that AOFU yields 84% accuracy, i.e., its policy agree with participants' responses 84% of the time. AOI achieves only 25% accuracy.

We then designed new techniques that use machine learning to automatically predict how users would respond to prompts, so that we can avoid prompting them in most cases. Our classifier uses the user's past decisions in related situations to predict their response to a particular permission prompt. The classifier outputs a prediction and a confidence score; if the classifier is sufficiently confident, we use its prediction, otherwise we prompt the user for their decision. We also incorporate information about the user's behavior and other security and privacy settings: e.g., whether they have a PIN screen lock activated, how often they visit HTTPS websites, and so on. We show that our scheme achieves 95.7% accuracy (a $4\times$ reduction in error rate, compared to AOFU) without too many prompts.

The specific contributions of our work are the following:

- We conducted the first known large-scale study on the effectiveness of ask-on-first-use permissions.
- We show that a significant portion of the studied participants make contextual decisions on permissions using the foreground application and the visibility of the permission-requesting application.
- We show how a machine-learned model can incorporate environmental context and better predict users' privacy decisions.
- To our knowledge, we are the first to use passively observed traits to infer future privacy decisions.

II. Related Work

There is a large body of work demonstrating that install-time prompts fail because users do not understand or pay attention to them [16], [20], [32]. When using install-time prompts, users often do not understand which permission types correspond to which sensitive resources and are surprised by the ability of background applications to collect information [14], [19], [31]. Applications also transmit a large amount of location or other sensitive data to third parties without user consent [9]. When possible risks associated with these requests are revealed to users, their concerns range from annoyance to wanting to seek retribution [13].

To mitigate some of these problems, systems have been developed to track information flows across the Android system [9], [15], [21] or introduce finer-grained permission control into Android [1], [18], [29], but many of these solutions increase user involvement significantly, which can lead to habituation. Additionally, many of these proposals are useful only to the most-motivated or technically savvy users. For example, many such systems require users to configure complicated control panels, which many are unlikely to do [35]. Other approaches involve static analysis in order to better understand how applications *could* request information [3], [7], [11], but these say little about how applications *actually* use information. Dynamic analysis improves upon this by allowing users to see how often this information is requested in real time [9], [30], [33], but substantial work is likely needed to present that information to average users in a meaningful way. Solutions that require runtime prompts (or other user interruptions) need to also minimize user intervention, in order to prevent habituation.

Other researchers have developed recommendation systems to recommend applications based on users' privacy preferences [36]. Systems have also been developed to predict what users would share on mobile social networks [6], which suggests that future systems could potentially infer what information users would be willing to share with third-party applications. By requiring users to self-report privacy preferences, clustering algorithms have been used to define user privacy profiles even in the face of diverse preferences [28]. However, researchers have found that the order in which information is requested has an impact on prediction accuracy [34], which could mean that such systems are only likely to be accurate when they examine actual user behavior over time (rather than relying on one-time self-reports).

Liu et al. clustered users by privacy preferences and used ML techniques to predict whether to allow or deny an application's request for sensitive user data [23]. However, their dataset was collected from a set of highly privacy-conscious individuals—those choosing to install a permission-control mechanism. Furthermore, the researchers removed "conflicting" user decisions, in which a user chose to deny a permission for an application, and then later chose to allow it. However, these conflicting decisions happen nearly 50% of the time in the real world [33], and accurately reflect the nuances of user privacy preferences; they are not experimental mistakes, and therefore models need to account for them. In fact, previous work found that users commonly reassess privacy preferences after usage [2]. Liu et al. also expect users to make 10% of permission decisions manually, which, based on field study results from Wijesekera et al., would result in being prompted every three minutes [33]. This is obviously impractical. Our goal is to design a system that can automatically make decisions on behalf of users, that accurately models their preferences, while also not over-burdening them with repeated requests.

Nissenbaum's theory of contextual integrity suggests that permission models should focus on information flows that are likely to defy user expectations [27]. There are three main components involved in deciding the appropriateness of a flow [5]: the context in which the resource request is made, the role played by the agent requesting the resource (i.e., the role played by the application under the current context), and the type of resource being accessed. Neither previous nor currently deployed permission models take all three factors into account. This model could be used to improve permission models by automatically granting access to data when the system determines that it is appropriate, denying access when it is inappropriate, and prompting the user only when a decision cannot be made automatically.

Wijesekera et al. performed a field study [33] to operationalize the notion of "context," so that an operating system can differentiate between appropriate and inappropriate data requests by a single application for a single data type. They

Permission Type	Activity
ACCESS_WIFI_STATE	View nearby SSIDs
NFC	Communicate via NFC
READ_HISTORY_BOOKMARKS	Read users' browser history
ACCESS_FINE_LOCATION	Read GPS location
ACCESS_COARSE_LOCATION	Read network-inferred location (i.e., cell tower and/or WiFi)
LOCATION_HARDWARE	Directly access GPS data
READ_CALL_LOG	Read call history
ADD_VOICEMAIL	Read call history
READ_SMS	Read sent/received/draft SMS
SEND_SMS	Send SMS
*INTERNET	Access Internet when roaming
*WRITE_SYNC_SETTINGS	Change application sync settings when roaming

TABLE I. FELT ET AL. PROPOSED GRANTING A SELECT SET OF 12 PERMISSIONS AT RUNTIME SO THAT USERS HAVE CONTEXTUAL INFORMATION TO INFER WHY THE DATA MIGHT BE NEEDED [12]. OUR INSTRUMENTATION OMITS THE LAST TWO PERMISSION TYPES (INTERNET & WRITE_SYNC_SETTINGS) AND RECORDS INFORMATION ABOUT THE OTHER 10.

found that users' decisions to allow a permission request were significantly correlated with that application's visibility: in this case, the contexts are using or *not* using the requesting application. They posit visibility of the application could be a strong contextual cue that influences users' responses to permission prompts. They also observed that privacy decisions were highly nuanced, and therefore a one-size-fits-all model is unlikely to be sufficient; a given information flow may be deemed appropriate by one user and inappropriate by another user. They recommended applying machine learning in order to infer individual users' privacy preferences.

To achieve this, research is needed to determine what factors affect user privacy decisions and how to use those factors to make privacy decisions on the user's behalf. While we cannot automatically capture everything involved in Nissenbaum's notion of context, we can try for the next-best thing: we can try to detect when context has likely changed, by seeing whether the circumstances surrounding a data request are similar to previous requests or not.

III. METHODOLOGY

We collected data from 131 participants to understand what factors help infer whether a permission request is likely to be deemed appropriate by the user.

Previous work by Felt et al. made the argument that certain permissions are appropriate for runtime prompts, because they protect sensitive resources—and therefore require user intervention—and because viewing the prompt at runtime imparts additional contextual information about why an application might need the permission [12]. We collected information about 10 of the 12 permissions they suggest are best-suited for runtime prompts; we omitted INTERNET and WRITE_SYNC_SETTINGS, since we did not expect any participant to be roaming while using our instrumentation, and focused on the remaining 10 permission types (Table I). While there are many other sensitive permissions beyond this set, Felt et al. concluded that the others are best handled by other mechanisms (e.g., install-time prompts, OS-drawn widgets).

We used the Experience Sampling Method (ESM) to collect ground truth data about users' privacy preferences [17]. ESM involves repeatedly questioning participants *in situ* about a recently observed event; in this case, we probabilistically asked them about an application's recent access to data on their phone, and whether they would have permitted it, if they had been given the choice. We treated participants' responses to these ESM probes as our main dependent variable (Figure 1).

We also instrumented participants' smartphones to obtain data about their privacy-related behaviors and the frequency with which applications accessed protected resources. The instrumentation required a set of modifications to the Android operating system and flashing a custom Android version onto participants' devices. To facilitate such experiments, the University of Buffalo offers academic researchers access to the PhoneLab panel [26], which consists of more than 200 participants (affiliated with the university). All of these participants had LG Nexus 5 phones running Android 5.1.1 and the phones were periodically updated over-the-air (OTA) with custom modifications to the Android operating system. Participants can decide when to install the OTA update, which marks their entry into new experiments. During our experiment period, different participants installed the OTA update with our instrumentation at different times, thus we neither have data on all PhoneLab participants, nor for the entire period. Our OTA update was available to participants for a period of six weeks, between February 2016 and March 2016. At the end of the study period, we emailed participants a link to an exit survey to collect demographic information. Our study was approved by the relevant institutional review board (IRB).

A. Instrumentation

The goal of our instrumentation was to collect as much runtime and behavioral data as could be observed from the Android platform, with minimal impact on performance. We collected three categories of data: behavioral information, runtime information, and user decisions. We made no modifications to any third-party application code.

Table II contains the complete list of behavioral and runtime events our instrumentation recorded. The behavioral data fell under several categories, all chosen based on several hypotheses that we had about the types of behaviors that might correlate with privacy preferences: web browsing behavior, screen locking behavior, third party application usage behavior, audio preferences, call habits, camera usage patterns (selfie vs. non-selfie), and behavior related to security settings. For example, we hypothesized that someone who manually locks their device screen (as opposed to letting it time out) might be more privacy-conscious than someone who takes many speakerphone calls or selfies.

We also collected runtime information about the context of each permission request, including the visibility of the requesting application at the time of request (i.e., whether it was running in the foreground or not) and what the user was doing when the request was made (i.e., the name of the foreground application). The visibility of an application reflects the extent to which the user was likely aware that

Type	Event Recorded
Behavioral Instrumentation	Changing developer options
	Opening/Closing security settings
	Changing security settings
	Enabling/Disabling NFC
	Changing location mode
	Opening/Closing location settings
	Changing screen-lock type
	Use of two factor authentication
	Log initial settings information
	User locks the screen
	Screen times out
	App locks the screen
	Audio mode changed
	Enabling/Disabling speakerphone
	Connecting/Disconnecting headphones
	Muting the phone
	Taking an audio call
	Taking a picture (selfie vs. non-selfie)
	Visiting a link in chrome
	Responding to a notification
	Unlocking the phone
Runtime Information	An application changing the visibility
	Platform switches to a new activity
Permission Requests	An app requests a sensitive permission
	ESM prompt for a selected permission

TABLE II. INSTRUMENTED EVENTS

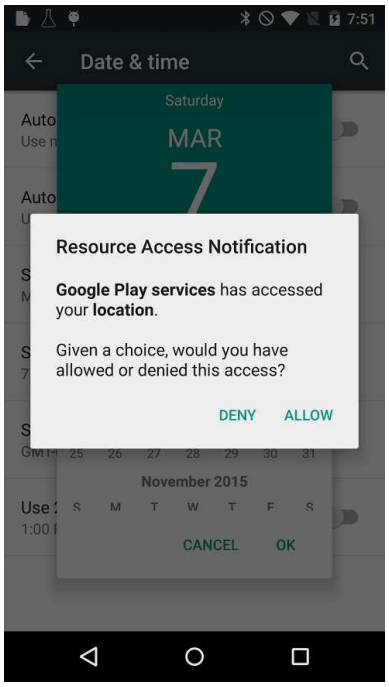

Fig. 1. A screenshot of an ESM prompt.

the application was running; if the application was in the foreground, the user had cues that the application was running, but if it was in the background, then the user was likely not aware that the application was running and therefore might find the permission request unexpected. We also collected information about which Android `Activity` was active in the application; depending on the design of the application, this might tell us only that the user was browsing with Firefox or might provide fine-grained information such as differentiating between reading a news feed vs. searching for a user's profile on Facebook. We monitored processes' memory priority levels to determine the visibility of all active processes.

We recorded every time that an application used one of the 10 permissions mentioned earlier. We also recorded the exact Android API invoked by a third-party application to determine precisely what information was requested.

Finally, once each day we randomly selected one of these permission requests and prompted the user about them (Figure 1). We used weighted reservoir sampling to select a permission request to prompt about. We weight permissions based on their frequency of occurrence seen by the instrumentation; the most-frequent permission request has a higher probability of being shown to participants using ESM. We prompted participants a maximum of three times for each unique combination of requesting application, permission, and visibility of the requesting application (i.e., background vs. foreground). We tuned the wording of the prompt to make it clear that the request had just occurred and their response would not affect the system (a deny response would not actually deny data). These responses serve as the ground truth for all the analysis mentioned in the remainder of the paper.

The intuition behind using a weighted-reservoir sampling is to focus more on the frequently occurring permission requests over rare ones. Common permission requests contribute most to user habituation due their high frequency. Thus, it is more important to learn about user privacy decisions on high frequent permission requests over the rare ones, which might not risk user habituation or annoyance.

B. Exit Survey

At the end of our data collection period, PhoneLab staff emailed participants a link to our online exit survey, which they were incentivized to complete with a raffle for two $100 Amazon gift cards. The survey gathered demographic information and qualitative information on their privacy preferences. Of the 203 participants in our experiment, 53 fully completed the survey, and another 14 partially completed it. Of the 53 participants to fully complete the survey, 21 were male, 31 were female, and 1 undisclosed. Participants ranged from 20 to 72 years of age (μ = 40.83, σ= 14.32). Participants identified themselves as 39.3% staff, 32.1% students, 19.6% faculty, and 9% other. Only 21% of the survey respondents had an academic qualification in STEM, which suggests that the sample is unlikely to be biased towards tech-savvy users.

C. Summary

We collected data from February 5 to March 17, 2016. PhoneLab allows any participant to opt-out of an experiment at any time. Thus, of the 203 participants who installed our custom Android build, there were 131 who used it for more than 20 days. During the study period, we collected 176M events across all participants (31K events per participant/day). Our dataset consists of 1,686 unique applications and 13K

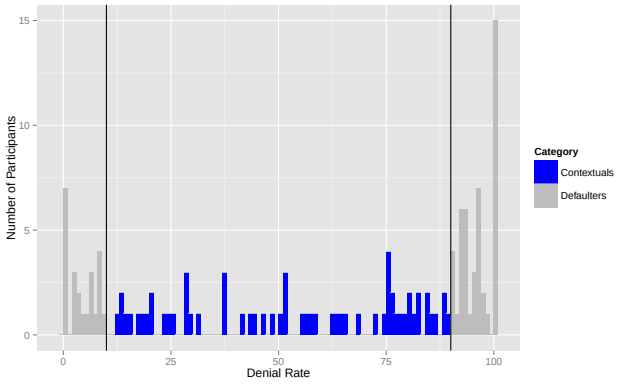

Fig. 2. Histogram of users based on their denial rate. *Defaulters* tended to allow or deny almost all requests without regard for contextual cues, whereas *Contextuals* considered the visibility of the requesting application.

Policy	Contextuals	Defaulters	Overall	Prompts
AOI	44.11%	6.00%	25.00%	0.00
AOFU-AP	**64.49%**	**93.33%**	**84.61%**	**12.34**
AOFU-APV	64.28%	92.85%	83.33%	15.79
AOFU-A$_F$PV	66.67%	98.95%	84.61%	16.91
AOFU-VP	58.65%	94.44%	78.04%	6.43
AOFU-VA	63.39%	93.75%	84.21%	12.24
AOFU-A	64.27%	93.54%	83.33%	9.06
AOFU-P	57.95%	95.45%	82.14%	3.84
AOFU-V	52.27%	95.34%	81.48%	2.00

TABLE III. THE ACCURACY AND NUMBER OF DIFFERENT POSSIBLE ASK-ON-FIRST-USE COMBINATIONS. A: APPLICATION REQUESTING THE PERMISSION, P: PERMISSION TYPE REQUESTED, V: VISIBILITY OF THE APPLICATION REQUESTING THE PERMISSION, A$_F$: APPLICATION RUNNING IN THE FOREGROUND WHEN THE REQUEST IS MADE. AOFU-AP IS THE POLICY USED IN ANDROID MARSHMALLOW I.E., ASKING (PROMPTING) THE USER FOR EACH UNIQUE APPLICATION, PERMISSION COMBINATION. THE TABLE ALSO DIFFERENTIATES POLICY NUMBERS BASED ON THE SUBPOPULATION OF *Contextuals*, *Defaulters*, AND ACROSS ALL USERS.

unique activities. Participants also responded to 4,636 prompts during the study period. We logged 96M sensitive permission requests, which translates to roughly one sensitive permission request every 6 seconds per participant. For the remainder of the paper, we only consider the data from the 131 participants who used the system for at least 20 days, which corresponds to 4,224 ESM prompts.

Of the 4,224 prompts, 55.3% were in response to ACCESS _WIFI_STATE, when trying to access Wifi SSID information that could be used to infer the location of the smartphone; 21.0%, 17.3%, 5.08%, 0.78%, and 0.54% were from accessing location directly, reading SMS, sending SMS, reading call logs, and accessing browser history, respectively. A total of 137 unique applications triggered prompts during the study period. Of the 4,224 prompts, participants wanted to deny 60.01% of them, and 57.65% of the prompts were shown when the requesting application was running in the foreground or the user had visual cues that the application was running (e.g., notifications). A Wilcoxon signed rank test with continuity correction revealed a statistically significant difference in participants' desire to allow or deny a permission request based on the visibility of the requesting application ($p < 0.0152$, $r = 0.221$), which corroborates previous findings [33].

IV. TYPES OF USERS

We hypothesized that there may be different types of users based on their behaviors. While our study size was too small to effectively apply clustering techniques to generate classes of users, we were able to find a meaningful distinction using the denial rate (i.e., the percentage of prompts to which users wanted to deny access). We aggregated users by their denial rate in 10% increments. We discovered that visibility was a significant predictor of user decisions for users with a denial rate of 10–90%, but not for users with a denial rate of 0–10% or 90–100%. We call the former group *Contextuals*, as they care about the surrounding context (i.e., they make nuanced decisions), and the latter group *Defaulters*, because, as we now show, they tend to either allow application permissions or deny them and did not vary their decision-making based on circumstances.

Based on the prompt responses, *Defaulters* accounted for 53% of 131 participants and *Contextuals* accounted for 47%. A Wilcoxon signed-rank test with continuity correction revealed a statistically significant difference in *Contextuals'* responses based on requesting application visibility ($p < 0.013$, $r = 0.312$), while for *Defaulters* there was no statistically significant difference ($p = 0.227$). That is, *Contextuals* used visibility as a contextual cue, when deciding whether or not a given permission request should be permitted, whereas *Defaulters* did not vary their decisions based on this cue, and instead consistently chose one option for the duration of the experiment. Figure 2 shows the distribution of users based on their denial rate. Vertical lines indicate the borders between *Contextuals* (light gray) and *Defaulters* (dark gray). Observe that *Defaulters* appear at both ends of the denial-rate spectrum, while *Contextuals* fully occupy the space between them.

Different permission models affect users differently based on their privacy preferences; performance numbers averaged across a user population could be misleading since different sub-populations might react differently to the same permission model. In the remainder of the paper, we use our *Contextuals–Defaulters* categorization to measure how current and proposed new models affect these two sub-populations, issues unique to these sub-populations, and ways to address these issues.

V. ASK-ON-FIRST-USE PERMISSIONS

Ask-on-first-use (AOFU) is the current Android permission model, which was first adopted in Android 6.0 (Marshmallow). AOFU works by prompting the user whenever an application requests a *dangerous* permission for the first time; the user's response to this prompt is thereafter applied whenever the same application requests the same permission. As of August 2016, only 15.2% of Android users have Android Marshmallow [8], and of those, those who have upgraded from a previous version only see runtime permission prompts for freshly-installed applications.

For the remaining 95.4% of users, the system policy is ask-on-install (AOI), which automatically allows all runtime permission requests. During the study period, all of our participants had AOI running as the default permission model. Be-

cause all runtime permission requests are allowed in AOI, any of our ESM prompts that the user wanted to deny correspond to mispredictions under the AOI model (i.e., the AOI model granted access to the data against users' actual preferences). Table III shows the expected median accuracy for AOI, as well as several other possible variants that we discuss in this section. The low median accuracy for *Defaulters* was due to the significant number of people who simply denied most of the prompts. The prompt count is zero for AOI because it does not prompt the user during runtime; users are only shown permission prompts at installation.

More users will have AOFU in the future, as they upgrade to Android 6.0 and beyond. To the best of our knowledge, no prior work has looked into the effectiveness of AOFU systematically; this section presents analysis of AOFU based on prompt responses collected from participants and creates a baseline against which to measure our system's improvement. We simulate how AOFU performs through our ESM prompt responses. Because AOFU is deterministic, each user's response to the first prompt for each *application:permission* combination tells us how the AOFU model would respond for subsequent requests by that same combination. For participants who responded to more than one prompt for each combination, we can quantify how often AOFU would have been correct for subsequent requests. Similarly, we also measure the accuracy for other possible policies that the platform could use to decide whether to prompt the user. For example, the status quo is for the platform to prompt the user for each new *application:permission* combination, but how would accuracy (and the number of prompts shown) change if the policy were to prompt on all new combinations of *application:permission:visibility*?

Table III shows the expected median accuracy[1] for each policy based on participants' responses. For each policy, *A* represents the application requesting the permission, *P* represents the requested permission, *V* represents the visibility of the requesting application, and A_F represents the application running in the foreground when a sensitive permission request was made. For instance, AOFU-AP is the policy where the user will be prompted for each new instance of an *application:permission* combination, which is the Android 6.0 model. The last column shows the number of runtime prompts a participant would see under each policy over the duration of the study, if that policy were to be implemented. Both AOFU-AP and AOFU-A_FPV show about a $4.9\times$ reduction in error rate compared to AOI; AOFU-A_FPV would require more prompts over AOFU-AP, though yields a similar overall accuracy rate.[2] Moving forward, we focus our analysis only on AOFU-AP.

Instances where the user wants to deny a permission and the policy instead allows it (false positives) are *privacy violations*, because they expose more information to the application than the user desires. Instances where the user wants to allow a permission, but the policy denies it (false negatives) are *functionality losses*. This is because the application is likely to lose some functionality that the user desired when it is incorrectly denied a permission. Privacy violations and functionality losses were approximately evenly split between the two categories for AOFU-AP: median privacy violations and median functionality losses were 6.6% and 5.0%, respectively.

The AOFU policy works well for *Defaulters*, because—by definition—they tend to be consistent after their initial responses for each combination, which increases the accuracy of AOFU. In contrast, the decisions of *Contextuals* vary due to other factors beyond just the application requesting the permission and the requested permission type. Hence, the accuracy of AOFU for *Contextuals* is significantly lower than the accuracy for *Defaulters*. This distinction shows that learning privacy preferences for a significant portion of users requires a deeper understanding of other factors affecting their decisions, such as behavioral tendencies and contextual cues. As Table III suggests, superficially adding more contextual variables (such as visibility of the requesting application) does not necessarily help to increase the accuracy of the AOFU policy.

Our estimated accuracy numbers for AOFU may be inflated because AOFU in deployment does not filter out permission requests that do not reveal any sensitive information. For example, an application can request the ACCESS_FINE_LOCATION permission to check whether the phone has a specific location provider, which does not leak sensitive information. Our AOFU simulation uses the invoked function to determine if sensitive data was actually accessed, and only prompts in those cases (in the interest of limiting the number of ESM prompts participants viewed during the study). Currently deployed AOFU in Marshmallow does not make this distinction. For example, Android users will see a permission request prompt when the application examines the list of location providers, and if the permission is granted, the user will not subsequently see prompts when location data is actually captured. Previous work showed that 79% of first-time permission requests do not reveal any sensitive information [33], and nearly 33.9% of applications that request these sensitive permission types do not access sensitive data at all. The majority of AOFU prompts in Marshmallow are therefore effectively false positives, which incorrectly serve as the basis for future decisions. Given this, the average accuracy for AOFU is likely less than the numbers presented in Table III. We therefore consider our estimates of AOFU to be upper bounds.

VI. Learning Privacy Preferences

Table III shows that a significant portion of users (the 47% classified as *Contextuals*) make privacy decisions that depend on factors other than the application requesting the permission, the permission requested, and the visibility of the requesting application. To make decisions on behalf of the user, we must understand what other factors affect their privacy decisions. We built a machine learning model trained and tested on our labeled dataset of 4,224 prompts collected from 131 users over the period of 42 days. This approach is equivalent to training a model based on runtime prompts from hundreds of users and using it to predict those users' future decisions.

We focus the scope of this work by making the following assumptions. We assume that the platform, i.e., the Android OS, is trusted to manage and enforce permissions for applications. We assume that applications must go through the platform's permission system to gain access to protected resources.

[1] The presented numbers—except for average prompt count, which was normally distributed—are median values, because the distributions were skewed.

[2] While AOFU-A_FPV has greater *median* accuracy when examining *Defaulters* and *Contextuals* separately, because the distributions are skewed, the median overall accuracy is identical to AOFU-AP when combining the groups.

Feature Group	Feature	Type
Behavioral Features (B)	Number of times a website is loaded to the Chrome browser.	Numerical
	Out of all visited websites, the proportion of HTTPS-secured websites.	Numerical
	The number of downloads through Chrome.	Numerical
	Proportion of websites requested location through Chrome.	Numerical
	Number of times PIN/Password was used to unlock the screen.	Numerical
	Amount of time spent unlocking the screen.	Numerical
	Proportion of times screen was timed out instead of pressing the lock button.	Numerical
	Frequency of audio calls.	Numerical
	Amount of time spent on audio calls.	Numerical
	Proportion of time spent on silent mode.	Numerical
Runtime Features (R1)	Application visibility (True/False)	Categorical
	Permission type	Categorical
	User ID	Categorical
	Time of day of permission request	Numerical
Aggregated Features	Average denial rate for (A1) application:permission:visibility	Numerical
	Average denial rate for (A2) $application_F$:permission:visibility	Numerical

TABLE IV. THE COMPLETE LIST OF FEATURES USED IN THE ML MODEL EVALUATION. ALL THE NUMERICAL VALUES UNDER BEHAVIORAL GROUP ARE NORMALIZED PER DAY. WE USE ONE-HOT ENCODING FOR CATEGORICAL VARIABLES. WE NORMALIZED NUMERICAL VARIABLES BY MAKING EACH ONE A Z-SCORE RELATIVE TO ITS OWN AVERAGE.

We assume that we are in a non-adversarial machine-learning setting wherein the adversary does not attempt to circumvent the machine-learned classifier by exploiting knowledge of its decision-making process—though we do present a discussion of this problem and potential solutions in Section IX.

A. Feature Selection

Using the behavioral, contextual, and aggregate features shown in Table II, we constructed 16K candidate features, formed by combinations of specific applications and actions. Then, we selected 20 features by measuring Gini importance through random forests [24], significance testing for correlations, and singular value decomposition (SVD). SVD was particularly helpful to address the sparsity and high dimensionality issues caused by features generated based on application and activity usage. Table IV lists the 20 features used in the rest of this work.

The behavioral features (*B*) that proved predictive relate to browsing habits, audio/call traits, and locking behavior. All behavioral features were normalized per day/user and were scaled in the actual model. Features relating to browsing habits included the number of websites visited, the proportion of HTTPS-secured links visited, the number of downloads, and proportion of sites visited that requested location access. Features relating to locking behavior included whether users employed a passcode/PIN/pattern, the frequency of screen unlocking, the proportion of times they allowed the screen to timeout instead of pressing the lock button, and the average amount of time spent unlocking the screen. Features under the audio and call category were the frequency of audio calls, the amount of time they spend on audio calls, and the proportion of time they spent on silent mode.

Our runtime features (*R1*/*R2*) include the requesting application's visibility, the permission requested, and the time of day a permission request occurred. Initially, we included the user ID to account for user-to-user variance, but as we discuss below, we subsequently removed this feature. Surprisingly, the name of the application requesting the permission did not come out as a predictive feature. Other features based on the requesting application, such as application popularity, similarly failed to be predictive.

Feature Set	Contextuals	Defaulters	Overall
B	67.48%	96.00%	83.21%
R1	69.30%	95.80%	83.71%
R2 + B	69.48%	95.92%	83.93%
R2 + A1	86.41%	96.91%	91.87%
R2 + A2	89.02%	98.08%	93.89%
R2 + A1 + A2	92.45%	98.34%	95.73%

TABLE V. THE ACCURACY OF THE MACHINE LEARNING MODEL FOR DIFFERENT FEATURE GROUPS ACROSS DIFFERENT USER GROUPS.

Different users may have different ways of perceiving privacy threats posed by the same permission request. To account for this, the learning algorithm should be able to determine how each user treats permission requests in order to accurately predict their future decisions. To quantify the difference between users in how they perceive the threat posed by the same set of permission requests, we introduced a set of *aggregate features* that could be measured at runtime and that might partly capture users' privacy stance. We compute the average denial rate for each unique combination of *application:permission:visibility* (*A1*) and of *permission:*$application_F$[3]*:visibility* (*A2*). These aggregate features indicate how the user responded to previous prompts associated with that combination. As expected, after we introduced the aggregate features, the relative importance of the user ID variable diminished and so we removed it (i.e., users no longer needed to be uniquely identified). We define *R2* as *R1* without the user ID.

B. Inference Based on Behavior

One of our main hypotheses is that passively observing users' behaviors could help infer their future privacy decisions. To this end, we instrumented Android to collect a wide array of behavioral data, listed in Table II. We categorize our behavioral instrumentation into interaction with Android privacy/security settings, locking behavior, audio settings and call habits, web browsing habits, and application usage habits. After the feature selection process (§VI-A), we found that only locking behavior, audio habits, and web browsing habits correlated with privacy behaviors.

We trained an SVM model with an RBF kernel on only the behavioral and runtime features listed in Table IV, excluding user ID. The 5-fold cross validation accuracy (with random splitting) was 83% across all users. This first setup assumes we have prior knowledge of previous privacy decisions to a certain extent from each user before inferring their future privacy decisions, so it is primarily relevant after the user has been using their phone for a while. However, the biggest advantage

[3]The application running in the foreground when the permission is requested by another application.

of using behavioral data is that it can be observed passively without any active user involvement.

To measure the extent to which we can infer user privacy decisions with *absolutely no user involvement* (and without any prior data on a user), we utilized leave-one-out cross validation. In this second setup, when a new user starts using a smartphone, we assume there is a ML model which is already trained with behavioral data and privacy decisions collected from a selected set of other users. We then measured the efficacy of such a model to predict the privacy decisions of a new user, purely based on passively observed behavior, without prompting that new user at all. This is an even stricter lower bound on user involvement, which essentially mandates that a user has to make no effort to indicate privacy preferences, something that no system currently does.

We performed leave-one-out cross validation for each of our 131 participants, meaning we predicted a single user's privacy decisions using a model trained using the data from the other 130 users' privacy decisions and behavioral data. The only input for each test user was the passively observed behavioral data and runtime data surrounding each request. The model yielded a median accuracy of 75%, which is a 3X improvement over AOI. Furthermore, AOI requires users to make active decisions during the installation of an application, which our second model does not require.

Examining only behavioral data with leave-one-group-out cross validation yielded a median accuracy of 55% for *Contextuals*, while for *Defaulters* it was 93.01%. Although, prediction using solely behavioral data fell short of AOFU-AP for *Contextuals*, it yielded a similar median accuracy for *Defaulters*; AOFU-AP required 12 prompts to reach this level of accuracy, whereas our model would not have resulted in any prompts. This relative success presents the significant observation that behavioral features, observed passively without user involvement, are useful in learning user privacy preferences. This provides the potential to open entirely new avenues of user learning and reduce the risk of habituation.

C. Inference Based on Context

Our SVM model with a RBF kernel produced the best accuracy. The results in the remainder of the section are trained and tested with five-fold cross validation with random splitting for a SVM model with a RBF kernel using the *ksvm* library in R. In all instances, the training set was bootstrapped with an equal number of allow and deny data points to avoid training a biased model. For each feature group, all hyperparameters were tuned through grid search to achieve highest accuracy. All the numerical values under the behavioral group are normalized per day. We use one-hot encoding for categorical variables. We normalized numerical variables by making each one a z-score relative to its own average. Table V shows how the accuracy changes with different sets of feature groups. As a minor note, the addition of the mentioned behavioral features to runtime features performed only marginally better; this could be due to the fact that those two groups do not complement each other in predictions. In this setup, we assume that there is a single model across all the users of Android.

By incorporating user involvement in the form of prompts, we can use our aggregate features to dramatically increase the accuracy for *Contextuals*, slightly less so for *Defaulters*. The aggregate features primarily capture how consistent users are for particular combinations (i.e., *application:permission:visibility*, *application:permission*, *application$_F$:permission:visibility*), which greatly affects accuracy for *Contextuals*. *Defaulters* have high accuracy with just runtime features (*R1*), as they are likely to stick with a default allow or deny policy regardless of the context surrounding a permission. Thus, even without any aggregate features (which do not impart any new information about this type of user), the model can predict privacy preferences of *Defaulters* with a high degree of accuracy. On the other hand, *Contextuals* are more likely to vary their decision for a given permission request. However, as the accuracy numbers in Table V suggest, their variance in decisions is correlated with some contextual cues that they observed. The high predictive power of aggregate features indicates that they may be capturing the contextual cues used by *Contextuals* to make decisions.

Of the aggregate features, *A2* caused the highest accuracy gain. The fact that $application_F$*:permission:visibility* is highly predictive indicates that user responses for this combination are more consistent than other combinations. The high consistency could relate to the notion that the foreground application (application$_F$) is also a strong contextual cue people use to make their privacy decisions (i.e., even when this is not the same application that is requesting the data); the only previously studied contextual cue was the visibility of the application requesting the sensitive data [33]. We offer a hypothesis for why foreground application could be significant: the sensitivity of the foreground application (i.e., high-sensitivity applications like banking, low-sensitivity applications like games) might impact how users perceive threats posed by requests. Irrespective of the application requesting the data, users may be likely to deny the request because of the elevated sense of risk. We discuss this further in §IX.

The model trained on feature sets *R2*, *A1* and *A2* had the best accuracy (and fewest privacy violations). For the remainder of the paper, we will refer to this model unless otherwise noted. We now compare AOFU-AP (the status quo as of Android 6.0, presented in Table III) and our model (Table V). Across all users, our model reduces the error rate from 15.38% to 4.27%, which is nearly a four-fold improvement. While both approaches perform relatively well for *Defaulters*, the ML model has a 4.72% lower error rate. For *Contextuals*, the ML model's improvements are much more dramatic, increasing accuracy from 64.49% to 92.45%. This gain is largely due to the contextual cues that the model takes into account (i.e., aggregate features). This shows that users do make contextual decisions rather than just basing their decision on application and permission, contrary to what AOFU assumes. That is, the aggregate features capture a notion of context, and these users' decisions are consistent across these notions of context.

Mispredictions (errors) in the ML model were approximately evenly split between privacy violations and functionality losses (54% and 46%). Deciding which error type is more acceptable is subjective and depends on factors like the usability issues surrounding functionality losses and gravity of privacy violations. However, the (approximately) even split between the two error types shows that the ML is not biased towards one particular decision (denying vs. allowing a request).

Furthermore, the area under the ROC curve (AUC), a metric used to measure the fairness of a classifier, is also significantly better in the ML model (0.956 as opposed to 0.796 for AOFU). This indicates that the ML model is equally good at predicting when to both allow and deny a permission request, while the AOFU tends to lean more towards one decision. In particular, with the AOFU policy, users would experience privacy violations for 10.01% of decisions, compared to just 2.32% with the ML model. Privacy violations tend to be more costly to the user than functionality loss, as denied data can always be granted at a later time, but disclosed data usually cannot be taken back.

While increasing the number of prompts improves classifier accuracy, it plateaus after reaching its maximum accuracy, at a point we call the *steady state*. For some users, the classifier might not be able to infer their privacy preference effectively, regardless of the number of prompts. As a metric to measure the effectiveness of the ML model, we measure the confidence of the model in the decisions it makes, based on prediction class probabilities.[4] In cases where the confidence of the model is below a certain threshold, the system should use a runtime prompt to ask the user to make an explicit decision. Thus, we looked into the prevalence of low-confidence predictions among the current predictions. With a 95% confidence interval, on average across five folds, low-confidence predictions accounted for less than 10% of all predictions. The remaining high-confidence predictions (90% of all predictions) had an average accuracy of 99.2%, whereas predictions with low confidence were only predicted with an average accuracy of 72%. §VII-B goes into this aspect in detail and estimates the rate at which users will see prompts in steady state.

The caveat in our ML model is that AOFU-AP only resulted in 12 prompts on average per user during the study, while our model averaged 32. The increased prompting stems from multiple prompts for the same combination of *application:permission:visibility*, whereas in AOFU, prompts are shown only once for each *application:permission* combination. During the study period, users on average saw 2.28 prompts per unique combination. While multiple prompts per combination help the ML model to predict future decisions more accurately, it risks habituation, which may eventually reduce the reliability of the labeled data. The next section presents an in-depth analysis on possible ways to reduce the number of prompts needed to train the ML model.

VII. Simulation

To better understand how to reduce prompting, while maintaining model accuracy over the status quo, we first examine how prompts affect model accuracy. This section presents an analysis of how the ML model's accuracy changes as prompts increase. Since a fully trained model requires twice as many prompts as AOFU, it is necessary to understand how the ML model behaves with fewer prompts. Once the model reaches adequate training, we can use model decision confidence to analyze how the ML model performs for different users and examine the tradeoff between user involvement and accuracy. We also utilize the model's confidence on decisions to present a strategy that can further reduce model error through selective permission prompting.

[4]To calculate the class probabilities, we used the KSVM library in R. It employs a technique proposed by Platt et al. [22] to produce a numerical value for each class's probability.

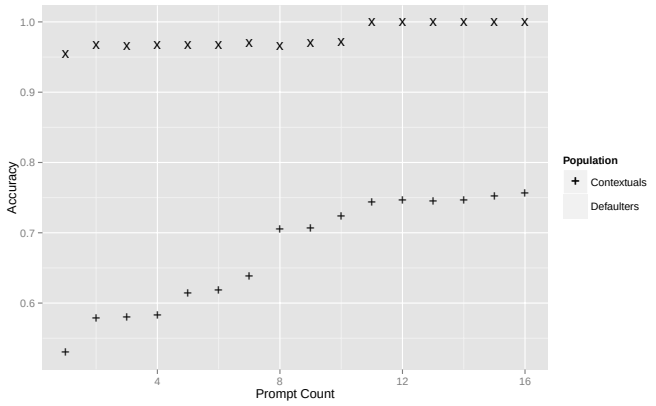

Fig. 3. How the median accuracy varies with the number of seen prompts

A. Bootstrapping

The *bootstrapping* phase occurs when the ML model is presented with a new user about whom the model has no prior information. In this section, we analyze how the accuracy improves as we prompt the user. Since the model presented in §VI is a single model trained with data from all users, the ML model can still predict a new user's privacy decisions by leveraging the data collected on other users' preferences.

We measured the accuracy of the ML model as if it had to predict each user's prompt responses using a model trained using other users' data. Formally, this is called leave-one-out cross-validation, where we remove all the prompt responses from a single user. The training set contains all the prompt responses from 130 users and the test set is the prompt responses collected from the single remaining user. The model had a median accuracy of 66.6% (56.2% for *Contextuals*, 86.4% for *Defaulters*). Although this approach does not prompt new users, it falls short of AOFU. This no-prompt model behaves close to random guessing for *Contextuals* and significantly better for *Defaulters*. Furthermore, Wijesekera et al. found that individuals' privacy preferences varied a lot from each other [33], suggesting that utilizing other users' decisions to predict decisions for a new user has limited effectiveness, especially for *Contextuals*; some level of prompting is necessary.

There are a few interesting avenues to explore when determining the optimal way to prompt the user in the learning phase. One option would be to follow the same weighted-reservoir sampling algorithm mentioned in §III-A. The algorithm is weighted by the frequency of each *application:permission:visibility* combination. The most frequent combination will have the highest probability of creating a permission prompt and after the given combination reaches a maximum of three prompts, the algorithm will no longer consider that combination for prompting, giving the second most frequent combination the new highest probability. Due to frequency-weighting and multiple prompts per combination, the weighted-reservoir sampling approach requires more

prompts to cover a broader set of combinations. However, AOFU prompts only once per combination without frequency-weighting. This may be a useful strategy initially for a new user since it allows the platform to learn about the users' privacy preferences for a wide array of combinations with minimal user interaction.

To simulate such an approach, we extend the aforementioned no-prompt model (leave-one-out validation). In the no-prompt model, there was no overlap of users in the train and test set. In the new approach, the training set includes the data from other users as well as the new user's responses to the first occurrence of each unique combination of *application:permission:visibility*. The first occurrence of each unique combination simulates the AOFU policy. That is, this model is bootstrapped using data from other users and then adopts the AOFU policy to further learn the current user's preferences. The experiment was conducted using the same set of features mentioned in §VI-A (*R2 + A1 + A2* and an SVM with a RBF kernel).

Figure 3 shows how accuracy changes with the varying number of AOFU prompts for *Contextuals* and *Defaulters*. For each of the 131 users, we ran the experiment varying the AOFU prompts from 1 to 16. We chose this upper bound because, on average, a participant saw 16 different unique *application:permission:visibility* combinations during the study period. During the study period, if the AOFU policy was in place with *application:permission:visibility*, a user would have seen a minimum of 16 prompts, because AOFU cannot predict a response to a combination it has not seen. Thus, AOFU needs to prompt at least 16 times before it can even make a prediction for all decisions. On the other hand, our hybrid approach does not have to prompt 16 times to predict privacy decisions across all the different combinations; this is because the model is already trained using other users' data. Hence, the hybrid approach can reach similar to or greater accuracy than AOFU with fewer prompts.

We trained a single model for all users, and analyze its performance for *Defaulters* and *Contextuals* separately, finding that it improves accuracy while reducing user involvement in both cases, compared to the status quo. We first examine how our model performs for *Defaulters*, 53% of our sample. Figure 3 shows that our model trained with AOFU permission-prompt responses outperforms AOFU with as few as 2 prompts. After 2 permission prompts, the model's accuracy steadies at the 96.6% mark (before it reaches close to 100% after 11 prompts), handily exceeding AOFU's 93.33%. This is a 83.3% reduction in permission prompts compared to AOFU-AP (the status quo). Even with such a significant reduction in user involvement, the new approach cuts the prediction error rate in half.

Contextuals needed more prompts to outperform the AOFU policy; the hybrid approach matches AOFU-AP with just 7 prompts, a 42% reduction in prompts. With 11 permission prompts, one less than needed for AOFU-AP, the new approach had a 16% accuracy gain over AOFU-AP. The number of prompts needed to reach this level of accuracy in the new approach is 31.25% less than what is needed for AOFU-APV. We also observed that as the number of prompts increased, the AUC of our predictions also similarly increased.

Our new hybrid approach of using AOFU-style permission prompts in the bootstrapping phase to train our model can achieve much higher accuracy than AOFU, with significantly fewer prompts. *Contextuals* have a higher need for user involvement than *Defaulters*, primarily because it is easy to learn about *Defaulters*, as they are more likely to be consistent with early decisions. On the other hand, *Contextuals* vary their decision based on different contextual cues and require more user involvement for the model to learn the cues for each user. Thus, it is important to find a way to differentiate between *Defaulters* and *Contextuals* early in the bootstrapping phase to determine which users require fewer prompts. The analysis of our hybrid approach addresses the concern of a high number of permission prompts initially for an ML approach. Over time, accuracy can always be improved with more permission prompts.

B. Decision Confidence

In the previous section, we looked into how we can optimize the learning phase by merging AOFU and the ML model to reach higher accuracy with minimal user prompts. However, for a small set of users, more permission prompts will not increase accuracy, regardless of user involvement in the bootstrapping phase. This could be due to the fact that a portion of users in our dataset are making random decisions, or that the features that our ML model takes into account are not predictive of those users' decision processes. While we do not have the data to support either explanation, we examine how we can measure whether the ML model will perform well for a particular user and quantify how often it does not. We present a method to identify difficult users and reduce permission prompting for those users.

While running the experiment in §VII-A, we also measured how confident the ML model was for each decision it made. To measure the ML model's confidence, we record the probability for each decision; since it is a binary classification (deny or allow), the closer the probability is to 0.5, the less confident it is. We then chose a *class probability threshold* above which a decision would be considered a high-confidence decision. In our analysis, we choose a class probability threshold of 0.6, since this value resulted in >99% accuracy for our fully-trained model (≈25 prompts per user) for high-confidence decisions, but this is a tunable threshold. Thus, in the remainder of our analysis, decisions that the ML model made with a probability of >0.60 were labeled as high-confidence decisions, while those made with a probability of <0.60 were labeled as low-confidence decisions.

Since the most accurate version of AOFU uses 12 prompts, we also evaluate the confidence of our model after 12 AOFU-style prompts. This setup is identical to the bootstrapping approach; the model we evaluate here is trained on responses from other users and the first 12 prompts chosen by AOFU. With this scheme, we found that 24 users (18.32% of 131 users) had at least one decision predicted with low confidence. The remaining 81.68% of users had all privacy decisions predicted with high confidence. Among those users whose decisions were predicted with low confidence, the proportion of low-confidence decisions on average accounted for 12.45% (median = 8.69%) out of all their predicted decisions. With a sensitive permission request once every 15 seconds [33], prompting even for 12.45% of predictions is not practical.

Users who had low-confidence predictions had a median accuracy of 70.29%, compared to 93.33% accuracy for the four-fifths of users with only high-confidence predictions. Out of the 24 users who had low-confidence predictions, there was only one *Defaulter*. This further supports the observation in Figure 3 that *Defaulters* require a shorter learning period.

In a real-world scenario, after the platform (ML model) prompts the user for the first 12 AOFU prompts, the platform can measure the confidence of predicting unlabeled data (sensitive permission requests for which the platform did not prompt the user). If the proportion of low-confidence predictions is below some low threshold, the ML model can be deemed to have successfully learned user privacy preferences and the platform should keep on using the regular permission-prompting strategy. Otherwise, the platform may choose to limit prompts (i.e., two per unique *application:permission:visibility* combination). It should also be noted that rather than having a fixed number of prompts (e.g., 12) to measure the low-confidence proportion, the platform can keep track of the low-confidence proportion as it prompts the users according to any heuristic (i.e., unique combinations). If the proportion does not decrease with the number of prompts, we can infer that the ML model is not learning user preferences effectively or the user is making random decisions, indicating that limiting prompts and accepting lower efficacy could be a better option for that specific user to avoid excessive prompting. However, depending on which group the user is in (*Contextual* or *Defaulter*), the point at which the platform could make the decision to continue or limit prompting could change. In general, the platform should be able to reach this deciding point relatively quickly for *Defaulters*.

Among the participants with no low-confidence predictions, we had a median error rate of 6.65% (using the new hybrid approach after just 12 AOFU prompts); for the same set of users AOFU reached a median error rate of 12.00%. However, using AOFU, a user in that set would have needed an average of 15.11 prompts to reach that accuracy. Using the ML model, a user would need just 6.23 prompts on average (*Defaulters* require far fewer prompts, dropping the average); the model only requires 41.23% of the prompts that AOFU requires. Even with significantly fewer prompts in the learning phase, the ML model achieves a 45.42% reduction in error rate as compared to AOFU.

While our model may not perform well for all users, it does seem to work quite well for the majority of users (81.68% of our sample). We provide a way of quickly identifying users for whom our system does not perform well, and propose limiting prompts to avoid excessive user burden for those users, at the cost of reduced efficacy. In the worst case, we could simply employ the AOFU model for users our system does not work well for, resulting in a multifaceted approach that is at least as good as the status quo for all users.

C. Online Model

Our proposed system relies on training models on a trusted server, sending it to client phones (i.e., as a weight vector), and having phones make classifications. By utilizing an online learning model, we can train models incrementally as users respond to prompts over time. There are two key advantages to this: (i) this model adapts to changing user preferences over time; (ii) training models on multiple users' data allows more labeled data points for training.

Our scheme requires two components: a feature extraction and storage mechanism on the phone (a small extension to our existing instrumentation) and a machine learning pipeline on a trusted server. The phone sends feature vectors to the server every few prompts, and the server responds with a weight vector representing the newly trained classifier. To bootstrap the process, the server's models can be initialized with a model trained on a few hundred users, such as our single model across all users. Since each user contributes data points over time, the online model adapts to changing privacy preferences even if they conflict with previous data. When using this scheme, each model takes less than 10 KiB to store. With our current model, each feature and weight vector are at most 3 KiB each, resulting in at most 6 KiB of data transfer per day.

To evaluate the accuracy of our online model, we trained a classifier using stochastic gradient descent (SGD) with five-fold cross validation on our 4,224-point data set. This served as the bootstrapping phase. We then simulated receiving the remaining data one-at-a-time in timestamp order. Any features that changed with time (e.g., running averages for aggregate features, event counts) were computed with each incoming data point, creating a snapshot of features as the phone would see it. We then tested accuracy on the chronologically last 20% of our dataset. Our SGD classifier had 93.8% accuracy (AUC=0.929). We attribute the drop in accuracy (compared to our offline model) to the fact that running averages take multiple data points to reach steady-state, causing some earlier predictions to be incorrect.

A natural concern with a trusted server is compromise. To address this concern, we do not send any personally-identifiable data to the server. Furthermore, features sent to the server have been *scaled*; they are reported in standard deviations from the mean, not in raw values.

VIII. Contextual Integrity

Contextual integrity is a conceptual framework that helps explain why most permission models fail to protect user privacy—they often do not take the context surrounding privacy decisions into account. In addressing this issue, we propose an ML model that infers when context has changed. That is, if the system knows that a user is comfortable sharing data with a particular application under one set of circumstances, it should not bother her with a permission request when the same application requests access to the same data under similar circumstances in the future. However, it should behave differently when those circumstances have changed. We believe that this is an important first step towards operationalizing the notion of *contextual integrity*. In this section, we explain the observations that we made in §VI-C within the context of the contextual integrity framework proposed in [5].

Contextual integrity provides a conceptual framework to better understand how users make privacy decisions; we use Barth et al.'s formalized model [5] as a framework in which to view the Android permission models. Barth et al. model parties as communicating agents (P) knowing information represented as attributes (T). A knowledge state κ is defined as a subset of

$P \times P \times T$. We use $\kappa = (p, q, t)$ to mean that agent p knows attribute t of agent q. Agents play roles (R) in contexts (C).

For example, an agent can be a game application, and have the role of a game provider in an entertainment context. Knowledge transfer happens when information is communicated between agents; all communications can be represented through a series of traces $(\kappa, (p, r), a)$, which is a combination of a knowledge state κ, a role state (p, r), and a communication action a (information sent). The role an agent plays in a given context helps determine whether an information flow is acceptable for a user. Communications can only occur when they follow the norms of context; the relationship between the agent sending the information and the role of the agent ((p, r)) receiving it must follow these norms, too.

With the Android permission model, the same framework can be applied. Both the user and the third-party application are communicating agents, and the information to be transferred is the sensitive data requested by the application. When a third-party application requests permission to access a guarded resource (e.g., location information), knowledge of the guarded resource is transferred from the one agent (i.e., the user/platform) to another agent (i.e., the third-party application). The extent to which a user expects a given request depends not on the agent (the application requesting the permission), but on the role that agent is playing in that context. This explains why the application as a feature itself (i.e., application name) was not predictive in our models: this feature does not represent the role when determining whether it is unexpected. While it is hard, from the platform, to determine the exact role an application is playing, the visibility of the application hints at its role. For instance, when the user is using Google Maps to navigate, it is playing a different role from when Google Maps is running in the background without the user's knowledge. We believe that this is the reason why the visibility of the requesting application is significant: it helps the user to infer the role played by the application requesting the permission.

The user expects applications in certain roles to access resources depending on the context in which the request is made. We believe that the foreground application sets this context. Thus a combination of the role and the context decides whether an information flow is expected to occur or not. Automatically inferring the exact context of a request (e.g., how data will be used, whether it will be shared with any other parties, etc.) is likely an intractable problem. However, for our purposes, it is possible that we need to only infer when context has *changed*, or rather, when data is being requested in a context that is no longer acceptable to the user. Based on our data, we believe that features based on foreground application and visibility are our most useful.

We now combine all of this into a concrete example within the contextual integrity framework: If a user is using Google Maps to reach a destination, the application can play the role of a navigator in a geolocation context, whereby the user feels comfortable sharing her location. In contrast, if the same application requests location while running as a service invisible to the user, the user may not want to give this service the same information. Background applications play the role of "passive listeners" in most contexts; this role as perceived by the user may be why background applications are likelier to violate privacy expectations and consequently be denied information by users.

AOFU primarily focuses on controlling access through rules for *application:permission* combinations. Thus, AOFU neglects the role played by the application (visibility) and relies purely on the agent (the application) and the information subject (permission type). This explains why AOFU is wrong in nearly one-fifth of cases. Based on Table III, both AOFU-VA (possibly identifying the role played by the application) and AOFU-A_FPV (possibly identifying the current context because of the current foreground application-A_F) have higher accuracy than the other AOFU combinations. However, as the framework of contextual integrity suggests, the permission model has to take both the role and the current context into account before making an accurate decision. AOFU only makes it possible to consider a single aspect, a limitation that does not apply to our model.

While the data presented in this work suggest the importance of capturing context to protect user privacy efficiently, more work is needed along these lines to fully understand how people use context to make decisions and what defines all factors that compose context in the Android permission model. Nevertheless, we believe we contribute a significant ground work toward future operationalization of contextual integrity.

IX. Discussion

The primary goal of this research was to improve the accuracy of the Android permission system so that it more correctly aligns with user privacy preferences. We began with four hypotheses: (i) that the currently deployed AOFU policy frequently violates user privacy; (ii) that the contextual information it ignores is useful; (iii) that a ML-based classifier can account for this contextual information and thus improve on the status quo; and (iv) passively observable behavioral traits can be used to infer privacy preferences.

To test these hypotheses, we performed the first large-scale study on the effectiveness of AOFU permission systems in the wild, which showed that hypotheses (i) and (ii) hold. We further built an ML classifier that took user permission decisions along with observations of user behaviors and the context surrounding those decisions to show that (iii) and (iv) hold. Our results show that existing systems have significant room for improvement, and other permission-granting systems may benefit from applying our results.

A. Limitations of Permission Models

Our field study confirms that users care about their privacy and are wary of permission requests that violate their expectations. Our experiments show that 95% of participants chose to block at least one permission request; in fact, the average denial rate was 60%—a staggering amount given Android's earlier AOI model permits all permission requests once an application is installed. This denial rate implies that AOI correctly regulates permission requests only two in five times.

While AOFU improves over the AOI model, it still violates user privacy one in five times as users deviate from their initial response to a permission request about 16% of the time. This amount is significant because of the high frequency of sensitive

permission requests: a 16% error rate translates to thousands of privacy violations for a single user *per day*. It further shows that AOFU's correctness assumption—that users make binary decisions based on the *application:permission* combination alone—is incorrect. Users take a richer space of information into account when making decisions about permission requests.

B. Our ML-Based Model

Our results show that ML techniques are effective at learning from both the user's previous decisions and the current environmental context in order to predict whether to grant permissions on the user's behalf. In fact, our techniques achieve better results than the methods currently deployed on millions of phones worldwide—while imposing significantly less user burden.

Our work incorporates elements of the environmental context into a machine-learning model. This better approximates user decisions by finding factors relevant for users that are not encapsulated by the AOFU model. In fact, our ML model reduces the errors made by the AOFU model by 75%. Our ML model's 96% accuracy is a substantial improvement over AOFU's 84% and AOI's 25%; the latter two of which comprise the status quo in the Android ecosystem.

Our research show that many users make neither random nor fixed decisions: the environmental context plays a significant role in user decision-making. Automatically detecting the precise context surrounding a request for sensitive data is an incredibly difficult problem (e.g., inferring *how* data will be used), and is potentially intractable. However, to better support user privacy, that problem does not need to be solved; instead, we show that systems can be improved by using environmental data to infer when context has *changed*. We found that the most predictive factors in the environmental context were whether the application requesting the permission is visible, and if not, what foreground application actually was visible. These are both strong contextual cues used by users, insofar as they allowed us to better predict changes in context. Our results show that ML techniques have great potential in improving user privacy, by allowing us to infer when context has changed, and therefore when users would want data requests to be brought to their attention.

C. Reducing the User Burden

Our work is also novel in using passively observable data to infer privacy decisions: we show that we can predict a user's preferences without *any* permission prompts. Our model trained solely on behavioral traits yields a three-fold improvement over AOI; for *Defaulters*—who account for 53% of our sample—it was as accurate as AOFU-AP. These results demonstrate that we can match the status quo without *any* active user involvement (i.e., the need for obtrusive prompts). These results imply that learning privacy preferences may be done entirely passively, which, to our knowledge, has not yet been attempted in this domain. Our behavioral feature set provides a promising new direction to guide research in creating permission models that minimize user burden.

The ML model trained with contextual data and past decisions also significantly reduced the user burden while achieving higher accuracy than AOFU. The model yielded a 45% reduction in prediction errors while reducing user involvement by 59%. The significance of this observation is that by reducing the risk of habituation, it increases reliability when user input is needed.

D. User- and Permission-Tailored Models

Our ML-based model incorporates data from all users into a single predictive model. It may be the case, however, that a collection of models tailored to particular types of users outperforms our general-purpose model—provided that the correct model is used for the particular user and permission. To determine if this is true, we clustered users into groups based first on their behavioral features, and then their denial rate, to see if we could build superior cluster-tailored ML models. Having data for only 131 users, however, resulted in clusters too small to carry out an effective analysis. We note that we also created a separate model for each sensitive permission type, using data only for that permission. Our experiments determined, however, that these models were no better (and often worse) than our general model. It is possible that such tailored models may be more useful when our system is implemented at scale.

E. Attacking the ML Model

Attacking the ML model to get access to users' data without prompting is a legitimate concern [4]. There are multiple ways an adversary can influence the proposed permission model: (i) imposing an adversarial ML environment [25]; (ii) polluting the training set to bias the model to accept permissions; and (iii) manipulating input features in order to get access without user notification. We assume in this work that the platform is not compromised; a compromised platform will degrade any permission model's ability to protect resources.

A thorough analysis on this topic is outside of our scope. Despite that, we looked at the possibility of manipulating features to get access to resources without user consent. None of the behavioral features used in the model can be influenced, since that would require compromising the platform. An adversary can control the runtime features for a given permission request by specifically choosing when to request the permission. We generated feature vectors that encompassed every adversary-controlled value and combination from our dataset, and tested them on our model. We did not find any conclusive evidence that the adversary can exploit the ML model by manipulating the input features to get access to resources without user consent.

As this is not a comprehensive analysis on attack vectors, it is possible that there exists a scenario where the adversary is able to access sensitive resources without prompting the user first. Our preliminary analysis suggests that they may be non-trivial, but more work is needed to study and prevent such attacks. In particular, to protect against adversarial ML techniques and formally examining feature brittleness.

F. Experimental Caveat

We repeat a caveat about our experimental data: users were free to deny permissions without any consequences: denying a legitimately-needed permission did not result in loss of functionality. We explicitly informed participants in our study

that their decisions to deny permission requests would have no impact on the actual behavior of their applications. This is important to note because if an application is denied a permission, it may exhibit undefined behavior or lose important functionality. If these consequences are imposed on users, they may decide that the functionality is more important than their privacy decision. Similarly, the loss of functionality may demonstrate the necessity of allowing certain permissions that are otherwise unclear.

If we actually denied permissions, users' decisions may skew towards a decreased denial rate. The denial rates in our experiments therefore represent the preferences of users and their *expectations* of reasonable application behavior—not the result of choosing between application functionality and privacy preferences. It is possible, for instance, that those categorized as *Defaulters* are an artifact of our experiment, as denying all permissions had no consequences. Yet, limiting our analysis to *Contextuals* does not limit our claims.

We leave as future work the replication of this experiment with consequences for denied application permissions. Note that the instrumentation of the Android platform to seamlessly provide this is non-trivial because many applications are not programmed to correctly handle denied permissions. This is despite modern Android already empowering users to deny permissions on their first use. In fact, researchers have noted that many applications crash when permissions are denied [10]. Consequently, we must develop a mock environment where permissions appear—to the application—to be allowed, but in reality only spurious or artificial data is provided. Such an experiment should provide the most accurate user permission data ever collected, and we expect that a significant portion of the default-deny contingent would become more contextual with their observed behaviors.

G. Types of Users

We presented a categorization of users based on the significance that the application's visibility played towards their individual privacy decisions. We believe that in an actual permission denial setting, the distribution will be different from what was observed in our study. Our categorization's significance, however, motivates a deeper analysis on understanding the factors that divide *Contextuals* and *Defaulters*. We believe that visibility is an important factor in this division but there may be others that are more significant. More work needs to be done to explore how *Contextuals* make decisions and which behaviors correlate with their decisions.

H. User Interface Panel

Any model that predicts user decisions has the risk of making incorrect predictions. Making predictions on a user's behalf, however, is necessary because permissions are requested by applications with too high a frequency for manual processing. Thus, platforms need to make these predictions and should strive to be as accurate as possible. While we do not expect any system to be able to obtain perfect accuracy, we do expect that our 96% accuracy can be improved upon.

One plausible way of improving the accuracy of the permission model is to empower the user to review and make changes on how the ML model makes decisions through a user feedback panel. A major benefit is that users would be able to go back and review the decisions made by the ML model. It would also allow users to adjust these decisions according to their preferences, thereby correcting errors. This gives users recourse to correct undesirable decision. The UI panel could also be used to reduce the usability issues and functionality loss stemming from permission denial. The panel should help the user figure out which rule incurred the functionality loss and change it accordingly. A user may also use this to adjust their privacy preferences as they evolve over time.

I. Conclusions

We have shown a number of important results. Users care about their privacy: they deny a significant number of requests to access sensitive data. Existing permission models for Android phones still result in significant privacy violations. User may allow permissions some times, while denying them at others, which means that there are more factors that go into the decision-making process than simply the application name and the permission type. We collected real-world data from 131 users and found that application visibility and the current foreground application were important factors in user decisions. We used the data we collected to build a machine-learning model to make automatic permission decisions. One of our models matched the errors made by AOFU without any user prompting, and another of our models reduced the number of errors by 75% with the same amount of prompting.

References

[1] H. M. Almohri, D. D. Yao, and D. Kafura, "Droidbarrier: Know what is executing on your android," in *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy*, ser. CODASPY '14. New York, NY, USA: ACM, 2014, pp. 257–264. [Online]. Available: http://doi.acm.org/10.1145/2557547.2557571

[2] 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*. ACM, 2015, pp. 787–796.

[3] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: Analyzing the android permission specification," in *Proc. of the 2012 ACM Conf. on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 217–228. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382222

[4] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.

[5] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum, "Privacy and contextual integrity: Framework and applications," in *Proc. of the 2006 IEEE Symposium on Security and Privacy*, ser. SP '06. Washington, DC, USA: IEEE Computer Society, 2006. [Online]. Available: http://dx.doi.org/10.1109/SP.2006.32

[6] I. Bilogrevic, K. Huguenin, B. Agir, M. Jadliwala, and J.-P. Hubaux, "Adaptive information-sharing for privacy-aware mobile social networks," in *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '13. New York, NY, USA: ACM, 2013, pp. 657–666. [Online]. Available: http://doi.acm.org/10.1145/2493432.2493510

[7] E. Bodden, "Easily instrumenting android applications for security purposes," in *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, ser. CCS '13. NY, NY, USA: ACM, 2013, pp. 1499–1502. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516759

[8] G. Developer, "Distribution of android versions," http://developer.android.com/about/dashboards/index.html, accessed: August 16, 2016.

[9] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924971

[10] Z. Fang, W. Han, D. Li, Z. Guo, D. Guo, X. S. Wang, Z. Qian, and H. Chen, "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)*. Xi'an, China: ACM, 2016.

[11] 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.*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 627–638. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046779

[12] 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: USENIX Association, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2372387.2372394

[13] 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*, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 33–44. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381943

[14] 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*, ser. SOUPS '12. New York, NY, USA: ACM, 2012. [Online]. Available: http://doi.acm.org/10.1145/2335356.2335360

[15] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "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*, ser. TRUST'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 291–307. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-30921-2_17

[16] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 1025–1035. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568276

[17] S. E. Hormuth, "The sampling of experiences in situ," *Journal of personality*, vol. 54, no. 1, pp. 262–293, 1986.

[18] 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.*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 639–652. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046780

[19] J. Jung, S. Han, and D. Wetherall, "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*, ser. SPSM '12. New York, NY, USA: ACM, 2012, pp. 45–50. [Online]. Available: http://doi.acm.org/10.1145/2381934.2381944

[20] 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.*, ser. FC'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 68–79. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34638-5_6

[21] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "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*, ser. SOAP '14, New York, NY, USA, 2014. [Online]. Available: http://doi.acm.org/10.1145/2614628.2614633

[22] H.-T. Lin, C.-J. Lin, and R. C. Weng, "A note on platt's probabilistic outputs for support vector machines," *Machine learning*, vol. 68, no. 3, pp. 267–276, 2007.

[23] 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*, ser. WWW '14. New York, NY, USA: ACM, 2014, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/2566486.2568035

[24] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013. [Online]. Available: http://papers.nips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

[25] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM, 2005, pp. 641–647.

[26] A. Nandugudi, A. Maiti, T. Ki, F. Bulut, M. Demirbas, T. Kosar, C. Qiao, S. Y. Ko, and G. Challen, "Phonelab: A large programmable smartphone testbed," in *Proceedings of First International Workshop on Sensing and Big Data Mining*. ACM, 2013, pp. 1–6.

[27] H. Nissenbaum, "Privacy as contextual integrity," *Washington Law Review*, vol. 79, p. 119, February 2004.

[28] 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.

[29] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino, "Identidroid: Android can finally wear its anonymous suit," *Trans. Data Privacy*, vol. 7, no. 1, pp. 27–50, Apr. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2612163.2612165

[30] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: Having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2480362.2480701

[31] 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.

[32] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 31–40. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420956

[33] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov, "Android permissions remystified: A field study on contextual integrity," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 499–514. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wijesekera

[34] H. Wu, B. P. Knijnenburg, and A. Kobsa, "Improving the prediction of users' disclosure behavior by making them disclose more predictably?" in *Symposium on Usable Privacy and Security (SOUPS)*, 2014.

[35] K.-P. Yee, "Guidelines and strategies for secure interaction design," *Security and Usability: Designing Secure Systems That People Can Use*, vol. 247, 2005.

[36] H. Zhu, H. Xiong, Y. Ge, and E. Chen, "Mobile app recommendations with security and privacy awareness," in *Proc. of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2014. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623705