Dynamically Regulating Mobile Application Permissions

Primal Wijesekera[1], Arjun Baokar[2], Lynn Tsai[2], Joel Reardon[3],
Serge Egelman[2], David Wagner[2], and Konstantin Beznosov[1]
[1]University of British Columbia, Vancouver, Canada,
{primal,beznosov}@ece.ubc.ca
[2]University of California, Berkeley, USA,
{arjunbaokar,lynntsai}@berkeley.edu, {egelman,daw}@cs.berkeley.edu
[3]University of Calgary, Calgary, Canada,
joel.reardon@ucalgary.ca

Abstract

Current smartphone operating systems employ permission systems to regulate how apps access sensitive resources. These systems are not well-aligned with users' privacy expectations: users often have no idea how often and under what circumstances their personal data is accessed. We conducted a 131-person field study to devise ways to systematically reduce this disconnect between expectations and reality. We found that a significant portion of participants make *contextual* privacy decisions: when determining whether access to sensitive data is appropriate, they consider what they are doing on their phones at the time, including whether they are actively using the applications requesting their data. We show that current privacy mechanisms do not do a good job of accounting for these contextual factors, but that by applying machine learning to account for context, we can reduce privacy violations by 80%, while also minimizing user involvement.

***Keywords*—Mobile privacy, Permission systems, Machine learning**

1 Introduction

"Flashlight apps" have become a trope for the entire concept of mobile phone apps abusing users' privacy [1]. It began when someone realized that you can turn on (and leave on) the camera's flash and thus give users a handy little pocket torch wherever they are. This creative use of the camera's flash was not (originally) part of Android's user-facing system, and as a result, a bevy of "free" flashlight apps entered the app marketplace.

It was quickly observed, however, that the majority of these apps requested a number of *permissions* far outside the scope of the functionality provided by the apps, instead using information like geolocation data and address book contacts for tracking and marketing purposes. The permission system is how Android tries to protect sensitive user data from being exploited by app developers. For instance, apps cannot simply read and write to your address book contacts: they must say ahead of time whether they will need access, so the user can decide if this ability is reasonable for the particular app to have. Flashlight apps got a particular reputation for asking for unreasonable permissions and this can still be seen today. The top results when searching for "flashlight" in the Google Play store include apps that demand access to all files on a phone's external storage, the ability to connect to nearby Bluetooth devices, the ability to read the phone's identity (e.g., IMEI and phone number), and to know the user's precise location at all times. A reasonable implementation of a flashlight, however, only needs two API calls—one to turn on and one to turn off the flash—and the graphical veneer of a flashlight.

The Need for Permission Systems The core functionality of the permission system is to regulate access to sensitive data on the phone, such as location information, address book contacts, and text messages [2]. Permission systems for mobile platforms are user-centric, in that the user makes the first decision about granting permission to an app for a particular resource. Current smartphone permission systems will then apply the user's decision to all subsequent cases of the same app accessing the same resource.

Permission systems are designed in such a way so that individual users can regulate which app gets access to which resource. In theory, users should be able to set their own privacy preferences using the permission system. By giving users ultimate control over their data, the platform makes the implicit assumption that the user can always make an informed decision when necessary. In our work, we explore this notion in practice and measure the efficacy of the mechanisms that permission systems use for obtaining user consent.

Give Me Permission! Asking the user to provide permission is not a bad way, in general, of regulating access to sensitive data. Unfortunately, Android's permission system has been plagued by design failures from the start. The original implementation was akin to an "install-time ultimatum." The user was presented with the list of permissions the app required and could only click "next" to continue installation (or use the system buttons to abort the installation altogether). Denying any requested permissions was not even presented as an option!

Research soon showed that users neither correctly understood the permissions that were being requested, nor the reasons why they were needed [3]. Prompting the user during the app's installation did not result in thoughtful and careful consideration of the permissions. Users had no recourse beyond uninstalling the app, nor did they have an easy way of finding similar apps without these inexplicable permission requests.

Can I Have Permission? Recently, Android Marshmallow (version 6.0) addressed the shortcomings of install-time permissions approach. Now, permissions are hidden from the user except for a small number of so-called "dangerous" ones. These are the permissions that protect sensitive data like address book contacts, storage, phone call records, using the camera or microphone, and so on [4]. Permissions deemed not "dangerous" are granted automatically without informing the user.

For "dangerous" permissions, Android now prompts the user at runtime, the first time the application attempts to access the data. So, for example, if a flashlight app tries to access address book contacts, the user is not only notified that this is about to happen (and can make a more informed decisions based on what she happened to be doing with the app at the time), but she is further given the option to *deny* the access altogether. "Denying a permission" in this context means that the app is prevented by the operating system from accessing the address book contacts. In fact, the decision that the user makes at this moment is taken as the rubric for all future permission decision for that `application:permission` combination (i.e., the flashlight app will always be denied access to address book contacts, and the user will never be explicitly asked again). While it is possible for users to change their decision, it is buried deep within multiple levels of system settings options. In effect, the user's decision at this moment dictates whether the app can access the resource at all future times.

Blanket Approval Android's new permission model is called *ask on first use* (AOFU) [5], because the user is only prompted the first time the application tries to access the data. AOFU is an improvement over the ask-on-install (AOI) ultimatum because it actually gives users the chance to deny a permission, while still allowing them to otherwise use the app. By involving the user during the execution of the app, when the data is actually needed, the user has a better chance of grasping the "possible" motivation behind the resource request. A user may find it bizarre that a messaging app wants to access their microphone at install time, but if prompted at the time when they first try to send a voice message to a friend, the rationale behind the permission request becomes clear. This is important, because it means that the user has a better understanding of the features that the permission enables and has more contextual information available when making decisions.

Unfortunately, AOFU is not a perfect solution for managing user privacy. Just because a user approves a permission request at one time does not mean that she would give the app *blanket approval* to access that data type in perpetuity, including when the user is not even using the app. But this is exactly how AOFU functions. In a sense, AOFU acts as a predictor of the user's privacy preferences: it uses prompting to see how the user feels about a particular `application:permission` combination—the first time an application requests a particular permission—and assumes that's how the user will always feel about it. This ignores a huge wealth of contextual information that could go into the user's decision-making process: what are they doing at the time? Are they even using the app that is requesting permission? Are they even aware the app is running? The answers to these questions could act as contextual cues to better inform the user about whether a particular resource request is likely to be appropriate. In our research, we are trying to incorporate this type of information into permission decisions in order to build contextually-aware privacy management systems.

Contextual Awareness The theory of contextual integrity [6] states that privacy violations occur when users' expectations about data flows are defied. The theory posits that expectations are in part governed by the role an actor plays in accessing sensitive data, and not simply who that actor is. For example, patients expect their doctors to talk about their medical conditions while at a clinic, but not at a supermarket; while the actor (the doctor) remains same in both contexts (the clinic and the supermarket), their role changes from doctor to fellow shopper and therefore expectations change.

Current mobile permission systems do not account for contextual information, such as *why* data is being requested by an app or how it will be used [7,8]. One of our key objectives is to infer contextual information so that it can be used to help users make privacy decisions only when necessary; if systems can infer that a request to sensitive data is likely expected by the user (e.g., a mapping application attempting to access GPS data), then it should be automatically granted without user intervention.

Ask on Every Use One might suggest that a way to allow users to apply contextual information is to simply prompt them *every* time an app requests sensitive data. We performed a previous experiment and found that this would mean bombarding the user with prompts roughly every 6 seconds [9]—a non-starter. Thus, the goal is to find a suitable number of prompts to show the user, so that the system can learn her preferences across varying contexts, and then automatically apply those preferences in the future.

Effective Learning The million-dollar question is whether we can reduce user involvement to avoid habituation while simultaneously increasing privacy protection by learning about users' preferences more accurately and with greater granularity. Effective learning entails understanding contextual factors affecting users' privacy decisions and how the impact varies under different contextual circumstances.

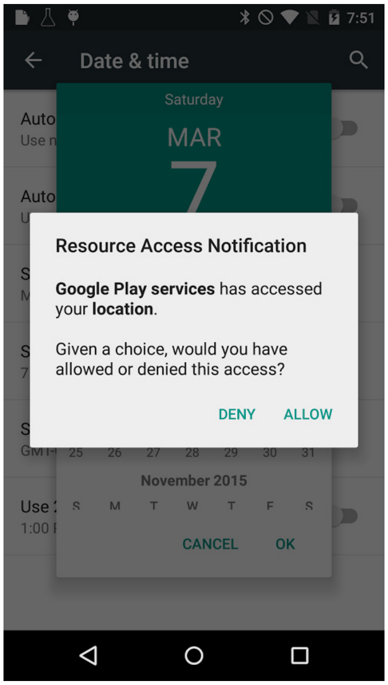
To do this, we use machine learning to make permission decisions on behalf of users. We use infrequent prompting to learn users' preferences for a given request, and then we capture the contextual information surrounding each decision. When we encounter a new permission request (based on contextual information, application, and data requested), our machine learning model generates a decision based on how the user has reacted under similar contextual circumstances in the past. By accounting for surrounding contextual factors, the new proposed model reduces the error rate by 80% as compared to the current AOFU approach.

2 Collecting User Privacy Preferences

The use of machine learning (ML) to tackle this problem means that we must collect users' privacy preferences to train an ML model, and then test this model's accuracy for another non-overlapping set of preferences. This simulates the conditions that our system would be working under in practice: training a model based on past decisions and using it to make decisions for future cases. We used a two-step process to collect ground truth about user privacy preferences to train our model. We first prepared an instrumented version of Android that collected data about permission requests and how users feel about them (as determined by their responses to runtime prompts). We then deployed this instrumented system in a real-world environment to collect ground truth from actual mobile phone users. The main challenge was to collect users' privacy preferences in their natural habitats without priming them, that is, avoiding having them think about security and privacy concerns more than they would normally.

Instrumentation The fact that Android is open source makes it easy to modify to try out new things—this is entirely why our research is focused on Android and not iOS. We took advantage of this in making a new version of the Android platform that recorded lots of data about privacy-related behaviors, details about permission accesses at runtime, and decisions from our users by prompting them about whether they would like to allow or deny recent permission requests.

To examine their privacy-related behaviors, we collected information about things we thought would correlate with their privacy preferences. For example, whether they used a secure lock screen, whether they used the speakerphone or earpiece for calls, how often (or, if ever) they changed or interacted with security



Resource Access Notification

Google Play services has accessed your **location**.

Given a choice, would you have allowed or denied this access?

Figure 1: A screenshot of an ESM prompt.

settings, and whether they actively locked their phone screen or just let it time out. As we will show later on, it turns out that just these passively observable behaviors are predictive of whether users would want to grant applications permissions. In fact, we built an ML model that has comparable accuracy to the status quo using only these passively observable traits. This is the holy grail of usability: applying users' preferences without needing to bother them.

For permission accesses, we focused our study on a handful of permissions that research has shown warrant runtime prompting [10]. These are things like accessing a user's browser history, location, call history, NFC, and stored SMS messages. In fact, this list has substantial overlap with Android 6.0's introduction of "dangerous permissions." We recorded whenever an app accessed a permission-protected resource, as well as contextual cues such as what the user was doing at that moment and whether the app requesting the permission was running visibly to the user (as opposed to in the background). We define visibility as a boolean value depicting whether the user had *any* visible clue that the given app was running, e.g., an app playing a sound or displaying a notification icon were both treated as a visible cue to the user, even if the app wasn't taking up the whole screen at the time of the request.

Finally, once per day we displayed a prompt to the user about a resource access that happened recently; this is known as experience sampling. The Experience Sampling Method (ESM) is used to collect participant preferences and reaction—in this case, their privacy preferences regarding an app's recent permission request—*in situ*. Collecting their preferences *in situ* puts them in the best position to better understand the context surrounding the request and the potential reason why the app requested the resource. The ESM prompts consisted of a message informing the user that a particular app had just accessed a particular permission, and asked if given a choice, would they have allowed or denied the request (Figure 1). We collected the answers to these prompts to use as ground truth in training and validating our ML model.

To promote useful and thoughtful responses, we limited our prompting to one prompt per day (i.e., one per participant-day). In deciding when to prompt or for which permission request, we made sure that all of the most frequent combinations of `application:permission:visibility` were covered: the app that was actually requesting the resource, the resource the app was accessing, and whether the user had any visible cues that the requesting app was running. By prompting users for permissions that are used more frequently, we can tailor our ML model to better serve the most likely requests, and we can use other techniques, such as additional runtime prompting, for rarer ones. To make sure that we have a broader coverage over different combinations, we stopped prompting for a particular `application:permission:visibility` combination once we had collected three responses for the triplet. Prompting multiple times on the same combination allowed us to analyze how users varied their decisions for the same combination under different contextual circumstances, the lack of which is a fundamental shortcoming of AOFU.

Field Study With our modified Android platform, our next step was to put the custom operating system onto the mobile phones of real users to collect meaningful data. We wanted to get a few hundred participants using our system for about a month—collecting runtime data and answering prompts. This would give us data from a large sample of individuals and apps, as well as enough time for them to familiarize themselves with the system and answer prompts thoughtfully.

We deployed a field study with the help of the PhoneLab panel [11], a University of Buffalo project that had a couple hundred participants running modified versions of Android on their primary phones. PhoneLab allows researchers from other universities to push software updates to their participants for the purpose of conducting field experiments.

Our study was approved by the UC Berkeley institutional review board (IRB). PhoneLab notified participants of our software update, giving them a choice as to whether or not to participate (participants could leave the experiment at any time). This meant that different participants installed our update at different times over the six weeks that we had it available and so we neither have data for all PhoneLab participants nor for the entire six weeks. Of the 201 participants who chose to join our experiment, there were 131 participants who used it for more than 20 days. From these 131, we collected responses to 4,224 of our ESM prompts (i.e., whether they would have allowed or denied an app access to a particular resource), as well as data for each of nearly 100 million sensitive permission requests—roughly once every 6 seconds *per participant*.

3 Making Decisions

At the end of the study, we had collected 176 million data points related to privacy decisions, behavioral events, and sensitive permission requests and their surrounding contextual information. We used this data to both measure the efficacy of the existing permission systems (AOFU, AOI), and explore the feasibility of predicting a user's future privacy decisions based on past decisions.

What the Data Tells Us First, our results replicate previous results in the literature. We found that 95% of participants chose to block at least one request and the median denial rate across all users was 75%! Android's original *ask-on-install* (AOI) policy would have allowed *all* of these permission requests that users wanted to deny.

AOFU is an improvement over the AOI model, but it still makes the wrong decision about one in seven times (e.g., allowing access to data that the user wanted to deny). This is of concern, since its 15% error rate translates to thousands of mistakes per day—about one bad decision a minute. Since AOFU uses the first decision a user makes for any `application:permission` pair, the error rate indicates that users do not simply base their decisions on the app name and data type. Rather, the frequency of AOFU's mistakes is evidence that users make contextual decisions; they change their minds based on circumstances and consider more than just the app and the permission combination when deciding about granting permission.

Contextuals and Defaulters Based on their responses to the prompts, we found that not everyone makes contextual decisions: a considerable portion of participants (53%) were more likely to default to either allowing or denying all requests, disregarding the surrounding context of each request; we call these participants "Defaulters." The remaining set of participants we call "Contextuals," because they are likely to make contextual decisions. While we do not have the data to explain what makes an individual one of these two types, we observe that both stand to benefit from a contextually-aware permission model.

AOFU performs significantly better among Defaulters (accuracy of 93.33%) over

Contextuals (accuracy of 64.49%). The significant difference between these groups explains AOFU's inability to learn the preferences of people who are likely to make contextual privacy decisions.

Behavioral Cues One of our most interesting results is that there is potential for a permission system that doesn't bother the user at all. We built a behavior-only model: it uses passively observable traits based on four types of user behaviors: mobile web usage habits, screen locking habits, call usage habits, and runtime data from the permission request. It *passively* observes statistics about these habits, and never prompts the user to make decisions about resource accesses.

The accuracy of behavioral model is below that of AOFU (75.1% compared to 84.6% in AOFU). Nevertheless, the significance of this observation is that there is great potential to use completely passively observable traits in predicting users' privacy decisions and we expect more research to follow in this direction. By completely removing active user involvement, this approach provides two untapped opportunities in the future of learning privacy preferences: it could completely avoid the problem of habitation and systems with limited interfaces (e.g., IoT devices) can still learn and apply user preferences. Given more exploration in this space, it is likely possible to collect more behavioral data to improve the accuracy and to use crowd-sourced data from other users to build accurate models for permission decisions that require no user involvement whatsoever.

Contextual Cues The most significant contributions of our work are in showing how surrounding contextual cues can be used with machine learning to predict future privacy decisions, resulting in greater accuracy than current permission models. Our approach resulted in reducing the error rate by 80% compared to the status quo.

The contextual ML model we developed uses three categories of information to supplement previous user decision data: runtime information for a given permission request, the specific app being used at the time, and whether users had knowledge that the requesting app was running (visibility).

The permission runtime information provides us with an understanding of what activity was occurring on the device when the permission request was made. Some of the metadata included in this is the resource requested and the time of day of the request. This information could hint at behavior patterns that people commonly follow. For example, a user may want to allow location data for navigation applications to help them avoid traffic on their way to work on a weekday morning, but not for fanciful vacation planning on a weekend afternoon.

We wanted to explore whether the foreground app, the app the participant is using when the given permission request occurs (especially if different from the app requesting the data), has an effect on the user's level of privacy consciousness. We hypothesized that the foreground app forces the participant to be either more privacy conscious (e.g., when using a banking app) or to be laxer (e.g, when the user is playing a game). The foreground app may help set the stage for the user's wariness towards sensitive resource requests.

The visibility of the requesting app could also provide an indicator for the user about the functional relevance of the request to the core app functionality. If the user is unaware that the requesting app is even running, then the user is more likely to assume that the resource request is less functionally relevant to the app and deny the request.

Contextual Integrity posits that user privacy expectations are governed by context and by the role played by the requesting entity (the requesting app in this case) in that context. Based on this, we believe that the foreground app helps users to differentiate between different contexts. The visibility of the requesting app is likely to relate to the role played by the requesting app. For example, a GPS navigation app in the foreground plays the role of a navigator, but in the background, the same app now plays the role of a passive information collector. A user may be okay with surrendering their location data to a navigator, but not a passive collector collecting information for a purpose that is unknown to them.

Our analysis showed that the ML model could reach an accuracy of 96.8%, with just 12 prompts during the study period. That is the same number of prompts users would have seen during the study period if they had been using the latest Android version with the AOFU permission model. However, our ML model significantly outperforms AOFU without adding any more burden on the users (i.e., no additional prompts). We also found that even when we reduce the number of prompts by 25%, our model can still achieve 50% reduction in error rate relative to AOFU. Again, the significance of this observation is that we can employ predictive models to reduce user involvement to help avoid habituation.

Denying with Consequence One caveat about this work is that users were free to deny permissions without any consequences (i.e., no data was actually denied as a result of the prompts, and therefore no app functionality suffered). We even told them that the permission request had already been granted and phrased the question hypothetically: "Given a choice, would you have allowed or denied the permission request?" Their responses reflected their *expectations* about privacy in the moment.

If our experimental design resulted in the actual denial of data to apps, then two things could occur that would not have happened in our study setup. First, apps that weren't given access to a resource that they were expecting could have either behaved strangely or crashed altogether. Second, unexpected app behaviors could have caused the user to grant permissions they would normally choose to deny, just to keep apps running as expected.

A likely scenario would be that a portion of users would become more selective in their denial decisions, reducing the number of users who, by default, deny nearly all permission requests. This would likely put the proportion of users that make contextual decisions (*Contextuals*) higher than the 47% we saw in our study sample. Our model provides the most improvements over AOFU for this group of users; we found that our proposed contextually-aware model can significantly reduce the error rate (an error reduction of 44%) for users who are more selective in their decision process. As an added benefit, our model can change and evolve over time as it learns and adapts to users' changing privacy preferences.

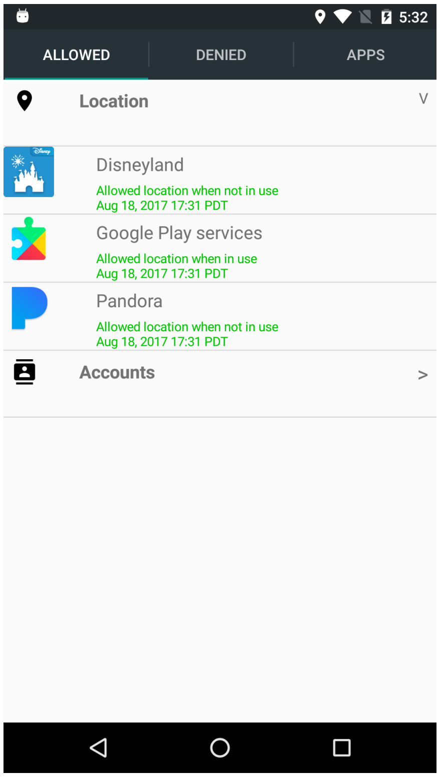
4 Free Will

The platform should not push users to be either more permissive or more restrictive towards apps or the ecosystem. Instead, the ideal of a permission system is to align its decisions with the decisions users would have actually made, balancing app functionality, privacy preferences, and standard tradeoffs involving privacy and free apps. We present steps that we are taking to make sure the new model lets users be as free as they want to be when making privacy decisions.

Obfuscated Data We want to empower users to confidently deny permissions that violate their privacy expectations, while maintaining a highly usable and enjoyable experience. To accomplish this, we've systematically looked at the different permissions and devised ways to help keep apps running smoothly in the face of being denied access to sensitive data. We do this by providing apps fake data in many cases, so that they cannot detect when they have been denied access.

For data being read in from sensors, such as the camera and microphone, we edit the data buffers if our ML model says to deny the permission. Apps then still get the amount of data they expected—only now we feed the app a loop of whale song (which could be substituted for any other audio stream). For events and notifications, we simply suppress the event. Instead of throwing an exception when trying to make phone calls or send texts, we simply pretend like it worked out fine without actually taking the action.

Finally, we consider how to regulate user data such as contacts and calendars. Here, there may be legitimate uses for these system-provided services. Unfortunately, the granularity of the choice the user gets to make is either to allow an app to read *all contacts* or to read no contacts. We improve this by adding a notion of *provenance* to each row stored in the database. Now, the app that created the entry in the database is recorded alongside the data. If we deny



Disneyland
Allowed location when not in use
Aug 18, 2017 17:31 PDT

Google Play services
Allowed location when in use
Aug 18, 2017 17:31 PDT

Pandora
Allowed location when not in use
Aug 18, 2017 17:31 PDT

Figure 2: A screenshot of the new Permission Manager. The list shows recently allowed permission requests.

permission to the database, the queries that the app issues are modified before executing them to restrict them only to the rows where the requesting app was the one who created it. This means that an app can create, read, update, and delete only its own entries in isolation from the full set of data, while other apps that are granted the permission may access the full set.

Dashboard While the proposed new model reduces the error by 80%, the high frequency with which permission requests occur means that an error rate of 3.2% is still substantial. Reaching a 100% rate of compliance with users' preferences is likely impossible, especially since privacy preferences and expectations change over time. However, there is still an open question as to what the theoretical maximum accuracy rate could be with such a system.

We believe that the best way to minimize the gap between user expectations and how the permission system actually makes decisions is to give users a greater degree of control and transparency into sensitive resource usage, so that they can make informed decisions. While this seems like an easy solution, users are already overwhelmed with the multitude of privacy and security decisions they make in day-to-day life. There is an inordinate challenge in presenting a user interface to an ML-based privacy management system so that it doesn't overwhelm or confuse users.

To this end, we developed a new permission manager (Figure 2) which increases the transparency into resource usage by listing all the recently allowed and denied resource requests [12]. Each request also shows the visibility of the requesting app at the time of the request (the contextual factor we observed to be most important in user decision-making). By selecting a particular item in the list, a user can specify how they want the platform to react under future similar circumstances. Users have an extra option to allow the resource only when the app is being used, apart from the standard "allow" or "deny" options. During iterative pilot studies with hundreds of participants, we observed that this interface was much more intuitive than current privacy settings interfaces.

5 Conclusions

When we take contextual factors into account when deciding when apps should be allowed to be granted permissions, we can predict participant's future privacy decisions with a significantly higher rate of accuracy than what is already being used today. The use of machine learning techniques have not only enabled the higher accuracy, but also helped to reduce user involvement, which is important for preventing habituation.

There are three broader impacts of this work. First, by using ML techniques, future systems can not only reduce the number of decisions users must make, but can also increase the likelihood of making privacy settings more aligned with users' desires and expectations. Second, when configuring privacy preferences, future systems should let users make contextual decisions—the exact set of contextual factors that will affect their thought process might differ depending on the nature of the system. Third, the use of passively observable privacy behaviors could be immensely useful in the space of IoT, when managing privacy settings, especially due to devices' limited user interfaces.

6 Acknowledgments

This research was supported by the United States Department of Homeland Security's Science and Technology Directorate under contract FA8750-16-C-0140, the Center for Long-Term Cybersecurity (CLTC) at UC Berkeley, the National Science Foundation under grant CNS-1318680, and Intel through the ISTC for

Secure Computing. The content of this document does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

References

[1] FTC. Android flashlight app developer settles ftc charges it deceived consumers. https://www.ftc.gov/news-events/press-releases/2013/12/android-flashlight-app-developer-settles-ftc-charges-it-deceived. Accessed: August 17, 2017.

[2] Android. Requesting permissions. https://developer.android.com/guide/topics/permissions/requesting.html. Accessed: August 17, 2017.

[3] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: user attention, comprehension, and behavior. In *Proc. of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. ACM.

[4] Google. Dangerous permissions. https://developer.android.com/guide/topics/permissions/requesting.html#normal-dangerous. Accessed: August 17, 2017.

[5] Android Developer. Requesting runtime permissions. https://developer.android.com/training/permissions/requesting.html. Accessed: August 17, 2017.

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

[7] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, 2016.

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

[9] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. Android permissions remystified: A field study on contextual integrity. In *24th USENIX*

Security Symposium (USENIX Security 15), pages 499–514, Washington, D.C., August 2015. USENIX Association.

[10] Adrienne Porter Felt, Serge Egelman, Matthew Finifter, Devdatta Akhawe, and David Wagner. How to ask for permission. In *Proc. of the 7th USENIX conference on Hot Topics in Security*, Berkeley, CA, USA, 2012. USENIX Association.

[11] Anandatirtha Nandugudi, Anudipa Maiti, Taeyeon Ki, Fatih Bulut, Murat Demirbas, Tevfik Kosar, Chunming Qiao, Steven Y Ko, and Geoffrey Challen. Phonelab: A large programmable smartphone testbed. In *Proceedings of First International Workshop on Sensing and Big Data Mining*, pages 1–6. ACM, 2013.

[12] Lynn Tsai, Primal Wijesekera, Joel Reardon, Irwin Reyes, Serge Egelman, David Wagner, Nathan Good, and Jung-Wei Chen. Turtle guard: Helping android users apply contextual privacy preferences. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 145–162, Santa Clara, CA, 2017. USENIX Association.