Contextual Permission Models for Better Privacy Protection

by

Primal Wijesekera

B.Sc in Computer Science, University of Colombo (Sri Lanka), 2008

M.Sc in Computer Science, The University of British Columbia (Canada), 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

June 2018

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Contextual Permission Models for Better Privacy Protection

submitted by **Primal Wijesekera** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Electrical and Computer Engineering**.

Examining Committee:

Konstantin Beznosov, Electrical and Computer Engineering
Co-supervisor

Serge Egelman, Computer Science, UC Berkeley/ICSI
Co-supervisor

Sathish Gopalakrishnan, Electrical and Computer Engineering
Supervisory Committee Member

Philippe Kruchten, Electrical and Computer Engineering
University Examiner

Joanna McGrenere, Computer Science
University Examiner

Additional Supervisory Committee Members:

Karthik Pattabiraman, Electrical and Computer Engineering
Supervisory Committee Member

Abstract

Despite corporate cyber intrusions attracting all the attention, privacy breaches that we, as ordinary users, should be worried about occur every day without any scrutiny. Smartphones, a household item, have inadvertently become a major enabler of privacy breaches. Smartphone platforms use permission systems to regulate access to sensitive resources. These permission systems, however, lack the ability to understand users' privacy expectations leaving a significant gap between how permission models behave and how users would want the platform to protect their sensitive data. This dissertation provides an in-depth analysis of how users make privacy decisions in the context of Smartphones and how platforms can accommodate user's privacy requirements systematically.

We first performed a 36-person field study to quantify how often applications access protected resources when users are not expecting it. We found that when the application requesting the permission is running invisibly to the user, they are more likely to deny applications access to protected resources. At least 80% of our participants would have preferred to prevent at least one permission request.

To explore the feasibility of predicting user's privacy decisions based on their past decisions, we performed a longitudinal 131-person field study. Based on the data, we built a classifier to make privacy decisions on the user's behalf by detecting when the context has changed and inferring privacy preferences based on the user's past decisions. We showed that our approach can accurately predict users' privacy decisions 96.8% of the time, which is an 80% reduction in error rate compared to current systems.

Based on these findings, we developed a custom Android version with a contextually aware permission model. The new model guards resources based on user's past decisions under similar contextual circumstances. We performed a 38-person field study to measure the efficiency and usability of the new permission model. Based on exit interviews and 5M data points, we found that the new system is effective in reducing the potential violations by 75%. Despite being significantly more restrictive over the default permission systems, participants did not find the new model to cause any usability issues in terms of application functionality.

Lay Summary

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. The thesis devises 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.

Preface

This research was the product of a fruitful collaboration between the author of the dissertation and the following people: Konstantin Beznosov (co-advisor) from the University of British Columbia, Arjun Baokar, Serge Egelman (co-advisor), Ashkan Hosseini, Joel Reardon, Lynn Tsai, David Wagner from the University of California, Berkeley and Irwin Reyes from International Computer Science Institute, Berkeley and Jung-Wei Chen, Nathan Good from Good Research, Inc.

It is worth mentioning that the work presented herein consists of research studies that have been published or under review in peer-reviewed international conferences. In particular, the field study presented in Chapter 2, and partly discussed in Chapter 5, are based on the following publication:

- Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. "Android Permissions Remystified: A Field Study on Contextual Integrity." In USENIX Security Symposium, pp. 499-514. 2015.

The prediction model on privacy decisions presented in Chapter 3 and partly discussed in Chapter 5, is based on the following publication:

- Primal Wijesekera, Arjun Baokar, Lynn Tsai, Joel Reardon, Serge Egelman, David Wagner, and Konstantin Beznosov. 2017. The Feasibility of Dynamically Granted Permissions: Aligning Mobile Privacy with User Preferences. In 2017 IEEE Symposium on Security and Privacy (SP). 1077–1093.

The implementation field study presented in Chapter 4 and partly discussed in Chapter 5, is based on following publications:

- Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. "Contextualizing Privacy Decisions for Better Prediction (and Protection)". Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '18), 2018. * Honorable Mention Award.

- 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), pp. 145-162. USENIX Association, 2017.

As the author of the thesis, I have involved in implementing more than 95% of the systems presented in this thesis: the instrumentation framework used in the first study, the instrumentation and behavioral interception in the second study and the implementing ML model and instrumentation in the third study. I have also conducted more than 90% of all the analysis presented in the thesis.

All of the field studies carried out throughout the thesis received institutional review board (IRB) approval prior to the respective study – approved by the UC Berkeley IRB under protocol #2013-02-4992 and by UBC BREB under protocol #H18-00856 for secondary use of data.

Table of Contents

List of Tables

List of Figures

Acknowledgments

First and foremost, I would like to thank the God Almighty for all the blessings I received in my life so far, for giving me the opportunity to work with a wonderful set of people during the course of my PhD.

I am grateful to my research supervisors Konstantin Beznosov (UBC) and Serge Egelman (UC Berkeley/ICSI) for all the necessary guidance and support given throughout the PhD and for everything they are continuing to do for me. I must specially thank Kosta for letting me do my research in UC Berkeley and Serge for taking me in to UC Berkeley. I am also very grateful for the valuable guidance I received from David Wagner in UC Berkeley and for all the fruitful discussions I had regarding my thesis research and in general about research.

Any journey in life is hard without good companionship, let alone the PhD. I have met amazing set of people from all around the world during the course of my PhD. Specially, my close Sri Lankan friends in UBC, friends from LERSSE and also friends/research collaborators in UC Berkeley and ICSI. The list is just too many to name, but I appreciate everything you have done for me and I hope I get a chance one day to return the favor.

Everything in life boils down to the family, I thank my parents, sister, and other close family members who have stood next to me in all of my endeavors specially during the difficult times and it was the encouragement and prayers from them that helped me to be who I am today.

To my parents and sister who are always there for me and never stopped believing in me

Chapter 1

Introduction

The explosive growth of smartphone usage can be attributed to both rapidly increasing hardware capabilities and the diverse set of available mobile applications providing a variety of services to end users. Among the two most popular smartphone platforms (Android and iOS), there are over one million third party mobile applications [1] available to download which have seen more than 100 billion downloads collectively. While providing valuable and useful services, mobile applications have come under intense criticism for it's use of user's personal data in ways not expected by the end users [4, 43, 71, 79, 88, 109].

Mobile platform permission models regulate how applications access certain resources, such as users' personal information or sensor data (e.g., camera, GPS, etc.). 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 (AOI),: either all requested permissions are approved or the application is not installed. Previous work has shown issues with the AOI permission model: a) few people read the requested permissions at install-time and even fewer correctly understood them [49, 65], b) install-time permissions do not present users with the context in which that permission will be exercised [47], which may cause

[1]All the applications which are not part of the platform and can be removed by the user are referred as the third party applications and will be referred as simply applications in the future for brevity.

users to make suboptimal decisions not aligned with their actual preferences. For example, Egelman et al. observed that when an application requests access to location data without providing context, users are just as likely to see this as a signal for desirable location-based features as they are an invasion of privacy [42]. These issues have kept the users in the dark about how these applications exercise their permissions to access sensitive resources [14, 50, 69, 87, 101]. 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) [16, 47, 62].

In iOS and Android M (6.0 and above), 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. The biggest caveat in AOFU is its reuse of the user consent for the first prompt, in all subsequent instances not accounting for the surrounding context. 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. Previous work has already shown that applications access sensitive data for a variety of reasons other than the core app functionality [4, 31, 43, 79]

Nissenbaum's *Privacy as Contextual Integrity* explains why context matters in Privacy. She posits that user privacy expectations are governed by their expectations on potential information flows that could occur in a given context. Thus, privacy violations occur if their expectations are defied. The notion implies that any permission system in place should account for user expectations in the context of the permission request. Despite that, none of the permission systems in Android account for the context and iOS started to support some form of the context (based on the foreground/background) for location requests which has been largely focused on location leaving other sensitive data unprotected.

Barth et al. formalized the notion of Contextual Integrity providing a concrete conceptual framework on privacy as contextual integrity. The work is, largely, followed by a long line of work formalizing the privacy policy and access controls focusing more towards theoretical aspects of privacy [26, 35]. A more closely related work on understanding users expectations found that context does influence their expectations on sensitive data usage [71]. Despite these useful observations, very little had been done to systematize the notion of Contextual Integrity so that platforms can better learn user's privacy expectations and protect their data in real time.

In this dissertation, we[2] operationalize the notion of *Contextual Integrity*. We show how the platforms can use the notion of Contextual Integrity (CI) and better protect sensitive user data aligning with their privacy expectations.

1.1 Problem Statement

Applications in mobile platforms provide a spectrum of functionality to the end users; while doing so, these applications access a variety of data. While end users are aware of the intended functionality provided by these applications, users are mostly unaware of how applications are accessing their data [50, 69, 87, 108, 117]. Users trust these applications with the functionality provided by the applications but the question remains do they trust these applications with their private data.

The core research question of this thesis is to investigate an effective approach to better help users to protect their sensitive, private data from unexpected accesses originated by the third party applications in the smartphone. The thesis presents a new permission model where users are better empowered and informed to make optimal privacy decisions and with a better-aligned protection.

[2]Use of the plural pronoun is customary even in solely authored research work.

1.1.1 Challenges

Privacy is a hard socio-technological problem where substantial technical contribution is needed to understand and involve humans to make sure everyone's expectations are met. Here we present three of the critical challenges found.

Users already have an overwhelming set of choices to make in day to day life in the digital world, yet they do not seem to have the control of making correct privacy decisions in the context of smartphones, these overwhelming set of choices are the source of the security fatigue which leads them to risky behavior compromising their privacy or security [86, 93, 105]. The important question remains as to how can smartphone platforms give more control to the user without putting more burden on the user? When is the correct time to ask the question (of acquiring their preferences for allowing or denying a access request)? What is the right question to ask? In the current technical setup, users will continue to make sub-optimal privacy choices or worse, current systems could further reduce their ability to make better privacy decisions [1, 2]. Thus the platform should reduce the user involvement and should only try to get the attention of the user when it is absolutely necessary because attention is a finite resource which should be used carefully[24].

Finite attention is not the only problem, privacy is usually not the user's main priority [101, 122], but the functionality of the application is. Platforms should never force the user to choose between the usability of the phone and their privacy expectations. Therefore, a finer grained permission model should be able to accommodate access requests to the guarded resources when it is expected by the user so that applications can deliver the expected functionality to the user. If applications start to crash due to unavailability of resources, it will increase user frustration and annoyance and make them go back and change their privacy decisions trading privacy for functionality [6, 32]. Thus, the platform should be able to preserve both user privacy and usability of the smartphone.

Restricted with finite attention and lack of motivation, the involvement of the users should be comprehensible to the end users. Previous work has shown that only a small portion of participants could understand the privacy prompt properly hindering the protection significantly [49, 65]. Previous attempts on fixing privacy controls have already come up with complicated controls to match with the user expectations [29, 33, 81, 98]. These systems fail due their complexity barring an average user from properly configuring the permission controls. Thus, independent of user's complicated privacy expectations, interfaces should be easily comprehensible so that users can make an informed, optimal decision.

1.1.2 Adversary Model

The proposed new permission model will protect the user from applications already installed by the user, from using sensitive resources in unexpected ways violating the privacy of the user. The application can be either a benign or a malicious application. The application, however, does not exploit the Android operating system itself; it operates within the permission privileges it has been granted to request data. The integrity of the underlying operating system (and the kernel) is assumed to be intact at all times for the permission model to function as expected. We do not handle the case where applications are coupled with some exploit that undermines the operating system or has any such malware [5, 46] that subverts the permission system and security monitor altogether.

The model also does not cover all permission protected resources in the mobile phone. A previous work by Felt et al. [47] showed that not all permission controlled resources need runtime attention and only the ones that can result in irreversible harm, highly sensitive data or data that could incur cost need runtime user attention so that users make a runtime decision. The first two studies presented in the thesis are only focusing on the 12 resources that Felt suggested in her prior study; this is explain in detail in Section 2.2.1. In the last study, on top of the above mentioned 12 resources, we also added the newly classified set of resources by Google as *Dangerous* resources [54]. While we believe the other resources needs user attention, they are best served with other user consent mechanisms such as auditing or other mechanisms mentioned in [47].

1.1.3 Research Methodology

We have conducted two one-week long field studies and one 6-week long longitudinal field study to gather real-world user data – all of the numbers we are presenting in this thesis are derived from real-world observations. We have used experience sampling and retrospective experience sampling throughout the studies to collect empirical datapoints [59]. We have also used both quantitative and qualitative analysis in evaluating the real-world datasets. After each study, we have conducted either an online exit survey or in-person interviews to understand behavior more insightfully. All of the quantification of qualitative data (user codes) were done by three independent coders [12]. We have detailed each study's methodology in depth under each different section. All of the field studies carried out throughout the thesis received institutional review board (IRB) approval prior to the respective study – approved by the UC Berkeley IRB under protocol #2013-02-4992 and by UBC BREB under protocol #H18-00856 for secondary use of data.

During the design and the execution phases of the presented studies, we took extra steps to make sure the ecological validity of the studies are well preserved. All of the study participants were randomly recruited through *Craiglist* except for the second study. The participant pool for the second study was all somehow affiliated to University of Buffalo, NY. The results of that study, however, were validated in a follow-up study using a completely random sample set to prove that the results of the second study are not affected by the selected participant pool. All of the statistical tests were properly used while making sure all of the assumed criteria are met for respective tests.

1.2 Research Summary

The thesis is evolved around four main research questions geared towards a) better understanding how third-party application exercise their permissions in the wild, b) understanding how user expectations on privacy vary from one context to another, c) exploring the feasibility of predicting user privacy decisions, and d) finally how both applications and users react to a more user-aligned and a contextually aware permission model. In the remainder of this section, we introduce the three different projects we carried out in the process of this thesis and their objectives.

1.2.1 Resource Usage and User Expectations in the Wild

Permission systems in mobile platforms protect private user data from third-party applications. To design an effective permission system, it is imperative to understand how applications access these protected sensitive resources; the understanding entails knowing how often and under what circumstances these applications are accessing private data. It is also critical to understand how these different circumstances change users' expectations for the resource access. If a permission system is to be effective on acting on behalf of the users, it is important to understand how their expectations are changing under different circumstances so that permission systems can act accordingly.

We instrumented the Android platform (v 4.1.1) to collect data regarding how often and under what circumstances smartphone applications access protected resources regulated by permissions. We performed a 36-person field study to explore how often applications access protected resources when users are not expecting it. Based on our collection of 27M data points and exit interviews with participants, we examine the situations in which users would like the ability to deny applications access to protected resources.

Main Findings

Following are the main findings based on the analysis of the 27M permission requests and based on the exit survey responses from 36 participants.

- During the study period (one week), on average, each participant had one sensitive permission requests every 15 seconds to a set of 12 sensitive resources. This observation makes the premise of using runtime prompts to give a better context impractical.

- During the exit survey, 80% of our participants wanted to block at least one permission request that occurred during the study. Overall, participants wanted to block access to protected resources a third of the time. This suggests that some requests should be granted by runtime consent dialog, rather than Android's previous all-or-nothing install-time approval approach.
- The visibility of the requesting application and the frequency at which requests occur are two important factors which need to be taken into account in designing a future permission system.

1.2.2 The Impact of the Surrounding Context

The observation of impracticality of the runtime prompts calls for finding ways to provide the context to the user when they make privacy decisions and also for approaches to figure out how to best act on behalf of the users. The frequency of sensitive permission requests increases the risk of habituation significantly, thus platforms should be able to act on behalf of the user majority of the time. This requires understanding how users want to act under different contexts so that the platform won't make a mistake that could end up violating the user's privacy.

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 the 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.

Main Findings

Following are the main findings based on the analysis of the 176M data points collected from 131 users over a period of six weeks.

- We show that the new permission model in Android, Ask-on-first-use (AOFU), has a median error rate of 15% causing the platform to make an error once in every minute. An error occurs everytime AOFU makes a decision different from what the user would have done if they had the opportunity.
- We show that a significant portion of the studied participants makes contextual decisions on permissions—the foreground application and the visibility of the permission-requesting application are strong cues participants used to make contextual decisions.
- We show how a machine-learned model can incorporate context and better predict users' privacy decisions. We show that our approach can accurately predict users' privacy decisions 96.8% of the time, which is a four-fold reduction in error rate compared to current systems.
- To the best of our knowledge, we are the first to use passively (data acquired without prompting the user) observed traits to infer future privacy decisions on a case-by-case basis at runtime.

1.2.3 Impact of a Real World Contextual Permission Model

In the aforementioned model, participants were expressing their expectations without any consequences. Given a choice between application functionality and privacy expectations, their privacy decisions could be different. The practicality of the entire premise of having a contextually aware, more restrictive permission model depends on how well users and third-party applications receive the new permission model. A little work has been done on analyzing how third-party applications will behave in the presence of actual resource denial and on understanding how users receive the new permission system[89].

To find answers to these questions, we implemented a novel privacy management system in Android, in which we use contextual signals to build a classifier that predicts user privacy preferences under various scenarios. We performed a 37-person field study to evaluate this new permission model under normal device usage. We show that this new permission model reduces the error rate by 75% (i.e., fewer privacy violations) while preserving usability. We offer guidelines for how platforms can better support user privacy decision-making.

Main Findings

Following are the main findings based on our exit interviews and collection of over 5 million data points from participants,

- Given the choice between functionality and privacy, users are still likely to take a restrictive stance – data shows that the context still matters even in the presence of actual resource denial.
- Desire to stick to their restrictive decisions shows the success of the data spoofing technique in places in the custom Android to make the app believe that they are getting genuine data.
- We show that AOFU not only lacks context, but it also fails to match users' understanding of the method 25% of the time, substantially hampering its ability to protect user data.

1.3 Main Contributions

In this section, we enlist the most important contributions of our work and how we envision each of these contributions will impact the future research in increasing privacy protection in mobile platforms. In summary, the main goal of the work is to understand, design and develop a better user aligned permission model which we believe achieved at the end of the study.

1.3.1 Understanding How Applications Access Data

As a research community, there was very little understanding of how applications actually use the resources which could benefit research on permission systems.

To the best of our knowledge, we are the first to run a field study to understand how applications access sensitive resources in the real world and to report actual frequencies of these accesses under different circumstances – this observation made the proposal on frequent runtime prompts impractical and forced researchers to think ways to engage the user while giving them the context. In the same study, we found a substantial portion of the requests occur when the user doesn't have a clue that the application is running at all which had a statistically significant impact towards their expectations and their decision towards denying such permission requests. More details on this are presented in the Chapter 2.

The following are the most critical impacts of this contribution;

- Any future permission system should account for frequency. This observation makes any calls for increased user intervention in permission models, impractical and obsolete.
- Given the significant portion of the invisible requests, our work suggests that visibility has to be a factor in any future permission systems enabling the users to control data accesses that occur when they are not using the apps.

1.3.2 How Users Perceive the Context

Although, the notion of Contextual Integrity existed for some time and conceptualized frameworks have been proposed, very little has been done to concretely systematize the notion in the context of smartphones. Any attempt to understand how the context is affecting user's decision needs to first identify the factors that help users to grasp the context and how they actually vary their decisions accordingly.

While iOS has already started enabling users to vary their decision based the visibility of the requesting application, we are the first to quantitatively show that the visibility of the requesting application has a statistically significant impact towards their privacy decisions. We also demonstrated the impact of the foreground application [3] in user's privacy decisions. Extending the notion of *Contextual Integrity*, we have demonstrated how visibility and the foreground application fit in the notion of Privacy as Contextual Integrity. More details on this are presented in the Chapter 3.

The following is the most critical impact of this contribution;

- Visibility of the requesting application is a critical factor in the user's decision process. This implies – a) users should be able to vary their decision based on the visibility of the requesting application, and b) platforms should take the visibility into account when acting on behalf of the user.

- Any future permission models should take the visibility of the requesting application and the foreground application at the time of user decision, if they want to reuse the user decision in future subsequent instances.

1.3.3 Ability to Predict Future Contextual Preferences

The previous two key contributions demonstrated that one plausible way to move forward is to understand how users make privacy decisions and predict their future decisions so that the platform can take the decision on behalf of them without involving the user in the process. The observation of two key contextual factors (foreground application at the time of the request and the visibility of the requesting application) showed that those two factors are needed to be in the system if the predictions are to be effective and aligned with the user's preferences. Previous work has already looked into predicting users' privacy decisions using clustering [73, 74]. The project, however, does not take the surrounding context into

[3] Foreground application is the application the user is engaged with at the time of the request

account and did not measure the accuracy of the systems with respect to the adaptation of the new permission model of Android (AOFU) among the test subjects.

We present an accurate predictive model that can predict users' future privacy decisions with a 96.8% accuracy, using their past decisions. The use of context and machine learning not only increased the privacy protection significantly; it has helped to reduce the user involvement avoiding the risk of habituation. More details on this are presented in the Chapter 3.

The following are the most critical impacts of this contribution:

- Successful use of the context to predict future privacy decisions which could be useful for other domains as well.
- Demonstration of the feasibility of machine learning to counter user habituation and increase their privacy protection significantly.

1.3.4 Contextually Aware Permission Model

There is a long line of work suggesting fixes and exposing defects in the previous and current permission systems in Android [5, 60, 74, 81, 84, 100]. None of the previous work, however, was contextually aware or analyzed the impact of the new permission model in terms of user experience and preserving the application functionality.

To the best of our knowledge, we are the first to implement a real-world contextually aware permission model that takes surrounding contextual signals into account before deciding on allowing (or denying) a given permission request. We show that if suitable data spoofing [4] mechanisms are in-place, a restrictive permission model can preserve both privacy of the user and application functionality. More details on this are presented in the Chapter 4.

The following are the most critical impacts of this contribution:

[4]Any mechanism that can feed synthetic data to the requesting application without risking user privacy.

- We demonstrated running a full machine learning pipeline in Android with real-world applications and users. This is the first proof that such a system can be implemented in Android for taking individual privacy decisions *real-time* based on user preferences.

- Contrary to the previous work by Fang et. al. [44], a more restrictive model can preserve usability by handling resource denial gracefully; rather than denying a resource platform can feed synthetic data that has no privacy implications.

1.4 Minor Contributions

Here, we present a few minor contributions that will have a significant impact towards future research along the lines of permission systems and privacy.

- We are, to the best of our knowledge, the first to report the use of behavioral traits to predict future privacy decisions. This is a critically important observation for any system which intends to learn users privacy decisions. All of the user behavioral traits are passively observable – user involvement in the learning phase is zero, which is an important victory against user habituation.

  There has been previous work done on connecting privacy and human behavior [3, 63] in a general setup. We, however, report the first successful attempt to predict individual privacy decisions based on behavioral traits in the context of smartphone usage.

  This observation of using behavioral traits [5] is even more important in the domain of IoT where the user involvement/engagement is minimal if it's not nonexistent. With behavioral traits are in placed, IoT devices can learn about user's privacy preferences with minimal resources available and do a better job at protecting their user's privacy.

[5]Participant's behavioral habits that we hypothesized to be useful in predicting their privacy behavior

Behavioral traits can be used for any permission model to further reduce the user involvement in the learning phase. There is more work needed to fully explore the potential of these passively observable traits.

- We have collected one of the largest (if not *the* largest) real-world data on mobile user behavior (80M), privacy decisions (6K), sensitive data access by third-party applications (130M), and third-party library data usage data points (8M). During the course of the project, we have collected these data points from 300+ real-world users.

  We believe more exploration is needed to understand and fix privacy protection and understanding of the impact of the context. This data could be immensely valuable for future research in privacy, mobile user behavior, measurements, etc. We have already shared a portion of these data points with the research community [84] and hope to continue doing the same in the future.

Chapter 2

Resource Usage

Few people read the Android install-time permission requests and even fewer comprehend them [49]. Another problem is habituation: on average, Android applications present the user with four permission requests during the installation process [45]. While iOS users are likely to see fewer permission requests than Android users, because there are fewer possible permissions and they are only displayed the first time the data is actually requested, it is not clear whether or not users are being prompted about access to data that they actually find concerning, or whether they would approve of subsequent requests [48].

Nissenbaum posited that the reason why most privacy models fail to predict violations is that they fail to consider contextual integrity [82]. That is, privacy violations occur when personal information is used in ways that defy users' expectations. We believe that this notion of "privacy as contextual integrity" can be applied to smartphone permission systems to yield more effective permissions by only prompting users when an application's access to sensitive data is likely to defy expectations. As a first step down this path, we examined how applications are currently accessing this data and then examined whether or not it complied with users' expectations.

We modified Android to log whenever an application accessed a permission-protected resource and then gave these modified smartphones to 36 participants who used them as their primary phones for one week. The purpose of this was to perform dynamic analysis to determine how often various applications are actually accessing protected resources under realistic circumstances. Afterwards, subjects returned the phones to our laboratory and completed exit surveys. We showed them various instances over the past week where applications had accessed certain types of data and asked whether those instances were expected, and whether they would have wanted to deny access. Participants wanted to block a third of the requests. Their decisions were governed primarily by two factors: whether they had privacy concerns surrounding the specific data type and whether they understood why the application needed it.

Key contributions of this chapter are:

- To our knowledge, we performed the first field study to quantify the permission usage by third-party applications under realistic circumstances.
- We show that our participants wanted to block access to protected resources a third of the time. This suggests that some requests should be granted by runtime consent dialogs, rather than Android's previous all-or-nothing install-time approval approach.
- We show that the visibility of the requesting application and the frequency at which requests occur are two important factors which need to be taken into account in designing a runtime consent platform.

2.1 Related Work

While users are required to approve Android application permission requests during installation, most do not pay attention and fewer comprehend these requests [49, 65]. In fact, even developers are not fully knowledgeable about permissions [106], and are given a lot of freedom when posting an application to the Google Play Store [19]. Applications often do not follow the principle of least privilege, intentionally or unintentionally [116]. Other work has suggested improving the Android permission model with better definitions and hierarchical breakdowns [18].

Some researchers have experimented with adding fine-grained access control to the Android model [28]. Providing users with more privacy information and personal examples has been shown to help users in choosing applications with fewer permissions [56, 66].

Previous work has examined the overuse of permissions by applications [45, 55], and attempted to identify malicious applications through their permission requests [99] or through natural language processing of application descriptions [85]. Researchers have also developed static analysis tools to analyze Android permission specifications [11, 23, 45]. Our work complements this static analysis by applying dynamic analysis to permission usage. Other researchers have applied dynamic analysis to native (non-Java) APIs among third-party mobile markets [104]; we apply it to the Java APIs available to developers in the Google Play Store.

Researchers examined user privacy expectations surrounding application permissions, and found that users were often surprised by the abilities of background applications to collect data [64, 108]. Their level of concern varied from annoyance to seeking retribution when presented with possible risks associated with permissions [48]. Some studies employed crowdsourcing to create a privacy model based on user expectations [71].

Researchers have designed systems to track or reduce privacy violations by recommending applications based on users' security concerns [5, 43, 53, 60, 67, 121, 123, 125]. Other tools dynamically block runtime permission requests [100]. Enck et al. found that a considerable number of applications transmitted location or other user data to third parties without requiring user consent [43]. Hornyack et al.'s AppFence system gave users the ability to deny data to applications or substitute fake data [60]. However, this broke application functionality for one-third of the applications tested.

Reducing the number of security decisions a user must make is likely to decrease habituation, and therefore, it is critical to identify *which* security decisions users should be asked to make. Based on this theory, Felt et al. created a decision tree to aid platform designers in determining the most appropriate permission-granting mechanism for a given resource (e.g., access to benign resources should be granted automatically, whereas access to dangerous resources should require approval) [47]. They concluded that the majority of Android permissions can be automatically granted, but 16% (corresponding to the 12 permissions in Table 2.1) should be granted via runtime dialogs.

Nissenbaum's theory of contextual integrity can help us to analyze "the appropriateness of a flow" in the context of permissions granted to Android applications [82]. There is ambiguity in defining when an application actually needs access to user data to run properly. It is quite easy to see why a location-sharing application would need access to GPS data, whereas that same request coming from a game like Angry Birds is less obvious. "Contextual integrity is preserved if information flows according to contextual norms" [82], however, the lack of thorough documentation on the Android permission model makes it easier for programmers to neglect these norms, whether intentionally or accidentally [101]. Deciding on whether an application is violating users' privacy can be quite complicated since "the scope of privacy is wide-ranging" [82]. To that end, we performed dynamic analysis to measure how often (and under what circumstances) applications were accessing protected resources, whether this complied with users' expectations, as well as how often they might be prompted if we adopt Felt et al.'s proposal to require runtime user confirmation before accessing a subset of these resources [47]. Finally, we show how it is possible to develop a classifier to automatically determine whether or not to prompt the user based on varying contextual factors.

2.2 Methodology

Our long-term research goal is to minimize habituation by only confronting users with *necessary* security decisions and avoiding showing them permission requests that are either expected, reversible, or unconcerning. Selecting which permissions to ask about requires understanding how often users would be confronted with each type of request (to assess the risk of habituation) and user reactions to these requests (to assess the benefit to users). In this study, we explored the problem space in two parts: we instrumented Android so that we could collect actual usage data to understand how often access to various protected resources is requested by applications in practice, and then we surveyed our participants to understand the requests that they would not have granted, if given the option. This field study involved 36 participants over the course of one week of normal smartphone usage. In this section, we describe the log data that we collected, our recruitment procedure, and then our exit survey.

2.2.1 Tracking Access to Sensitive Data

In Android, when applications attempt to access protected resources (e.g., personal information, sensor data, etc.) at runtime, the operating system checks to see whether or not the requesting application was previously granted access during installation. We modified the Android platform to add a logging framework so that we could determine every time one of these resources was accessed by an application at runtime. Because our target device was a Samsung Nexus S smartphone, we modified Android 4.1.1 (Jellybean), which was the newest version of Android supported by our hardware.

Data Collection Architecture

Our goal was to collect as much data as possible about each applications' access to protected resources, while minimizing our impact on system performance. Our data collection framework consisted of two main components: a series of "producers" that hooked various Android API calls and a "consumer" embedded in the main Android framework service that wrote the data to a log file and uploaded it to our collection server.

We logged three kinds of permission requests. First, we logged function calls checked by `checkPermission()` in the Android `Context` implementation. Instrumenting the `Context` implementation, instead of the `ActivityManagerService` or `PackageManager`, allowed us to also log the function name invoked by the user-space application. Next, we logged access to the `ContentProvider` class, which verifies the read and write permissions of an application prior to it accessing structured data (e.g., contacts or calendars) [9]. Finally, we tracked permission checks during `Intent` transmission by instrumenting the `ActivityManagerService` and `BroadcastQueue`. `Intents` allow an application to pass messages to another application when an activity is to be performed in that other application (e.g., opening a URL in the web browser) [10].

We created a component called `Producer` that fetches the data from the above instrumented points and sends it back to the `Consumer`, which is responsible for logging everything reported. `Producers` are scattered across the Android Platform, since permission checks occur in multiple places. The `Producer` that logged the most data was in `system_server` and recorded direct function calls to Android's Java API. For a majority of privileged function calls, when a user application invokes the function, it sends the request to `system_server` via `Binder`. `Binder` is the most prominent IPC mechanism implemented to communicate with the Android Platform (whereas `Intents` communicate between applications). For requests that do not make IPC calls to the `system_server`, a `Producer` is placed in the user application context (e.g., in the case of `ContentProviders`).

The `Consumer` class is responsible for logging data produced by each `Producer`. Additionally, the `Consumer` also stores contextual information, which we describe in Section 2.2.1. The `Consumer` syncs data with the filesystem periodically to minimize impact on system performance. All log data is written to the internal storage of the device because the Android kernel is not allowed to write to external storage for security reasons. Although this protects our data from curious or careless users, it also limits our storage capacity. Thus, we compressed the log files once every two hours and upload them to our collection servers whenever the phone had an active Internet connection (the average uploaded and zipped log file was around 108KB and contained 9,000 events).

Due to the high volume of permission checks we encountered and our goal of keeping system performance at acceptable levels, we added rate-limiting logic to the `Consumer`. Specifically, if it has logged permission checks for a particular application/permission combination more than 10,000 times, it examines whether it did so while exceeding an average rate of 1 permission check every 2 seconds. If so, the `Consumer` will only record 10% of all future requests for this application/permission combination. When this rate-limiting is enabled, the `Consumer` tracks these application/permission combinations and updates all the `Producers` so that they start dropping these log entries. Finally, the `Consumer` makes a note of whenever this occurs so that we can extrapolate the true number of permission checks that occurred.

Data Collection

We hooked the permission-checking APIs so that every time the system checked whether an application had been granted a particular permission, we logged the name of the permission, the name of the application, and the API method that resulted in the check. In addition to timestamps, we collected the following contextual data:

- **Visibility**—We categorized whether the requesting application was visible to the user, using four categories: running (a) as a service with no user interaction; (b) as a service, but with user interaction via notifications or sounds; (c) as a foreground process, but in the background due to multitasking; or (d) as a foreground process with direct user interaction.
- **Screen Status**—Whether the screen was on/off.
- **Connectivity**—The phone's WiFi connection state.
- **Location**—The user's last known coordinates. In order to preserve battery life, we collected cached location data, rather than directly querying the GPS.
- **View**—The UI elements in the requesting application that were exposed to the user at the time that a protected resource was accessed. Specifically, since the UI is built from an XML file, we recorded the name of the screen as defined in the DOM.

- **History**—A list of applications with which the user interacted prior to the requesting application.
- **Path**—When access to a `ContentProvider` object was requested, the path to the specific content.

Felt et al. proposed granting most Android permissions without *a priori* user approval and granting 12 permissions (Table 2.1) at runtime so that users have contextual information to infer why the data might be needed [47]. The idea is that, if the user is asked to grant a permission while using an application, she may have some understanding of why the application needs that permission based on what she was doing. We initially wanted to perform experience sampling by probabilistically questioning participants whenever any of these 12 permissions were checked [68]. Our goal was to survey participants about whether access to these resources was expected and whether it should proceed, but we were concerned that this would prime them to the security focus of our experiment, biasing their subsequent behaviors. Instead, we instrumented the phones to probabilistically take screenshots of what participants were doing when these 12 permissions were checked so that we could ask them about it during the exit survey. We used reservoir sampling to minimize storage and performance impacts, while also ensuring that the screenshots covered a broad set of applications and permissions [114].

Figure 4.1 shows a screenshot captured during the study along with its corresponding log entry shown in Table2.2. The user was playing the Solitaire game while Spotify requested a WiFi scan. Since this permission was of interest (Table 2.1), our instrumentation took a screenshot. Since Spotify was not the application the participant was interacting with, its visibility was set to *false*. The history shows that prior to Spotify calling `getScanResults()`, the user had viewed Solitaire, the call screen, the launcher, and the list of MMS conversations.

Permission Type	Activity
WRITE_SYNC_ SETTINGS	Change application sync settings when the user is roaming
ACCESS_WIFI_ STATE	View nearby SSIDs
INTERNET	Access Internet when roaming
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

Table 2.1: The 12 permissions that Felt et al. recommend be granted via runtime dialogs [47]. We randomly took screenshots when these permissions were requested by applications, and we asked about them in our exit survey.

2.2.2 Recruitment

We placed an online recruitment advertisement on Craigslist in October of 2014, under the "et cetera jobs" section.[1] The title of the advertisement was "Research Study on Android Smartphones," and it stated that the study was about how people interact with their smartphones. We made no mention of security or privacy. Those interested in participating were directed to an online consent form. Upon agreeing to the consent form, potential participants were directed to a screening application in the Google Play store. The screening application asked for information about each potential participant's age, gender, smartphone make and model.

[1]Approved by the UC Berkeley IRB under protocol #2013-02-4992

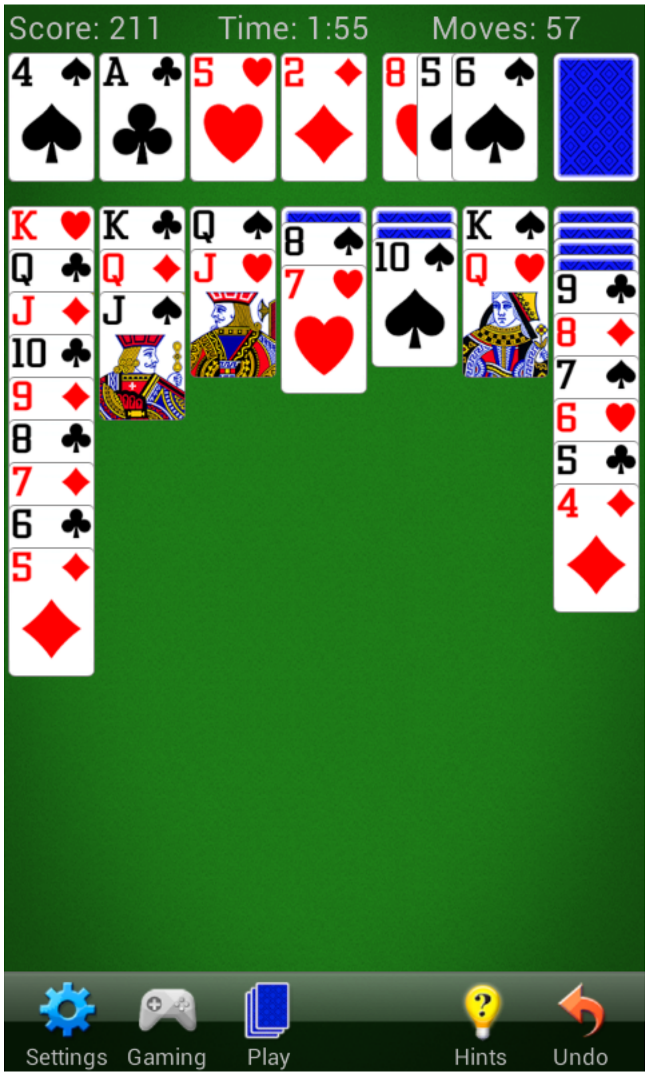


Figure 2.1: Screenshot

It also collected data on their phones' internal memory size and the installed applications. We screened out applicants who were under 18 years of age or used providers other than T-Mobile, since our experimental phones could not attain 3G speeds on other providers. We collected data on participants' installed applications so that we could pre-install free applications prior to them visiting our laboratory. (We copied paid applications from their phones, since we could not

Name	**Log Data**
Type	API_FUNC
Permission	ACCESS_WIFI_STATE
App_Name	com.spotify.music
Timestamp	1412888326273
API Function	getScanResults()
Visibility	FALSE
Screen Status	SCREEN_ON
Connectivity	NOT_CONNECTED
Location	Lat 37.XXX Long -122.XXX
View	com.mobilityware.solitaire/.Solitaire
History	com.android.phone/.InCallScreen com.android.launcher/com.android.-launcher2.Launcher com.android.mms/ConversationList

Table 2.2: Corresponding log entry

download those ahead of time.)

We contacted participants who met our screening requirements to schedule a time to do the initial setup. Overall, 48 people showed up to our laboratory, and of those, 40 qualified (8 were rejected because our screening application did not distinguish some Metro PCS users from T-Mobile users). In the email, we noted that due to the space constraints of our experimental phones, we might not be able to install all the applications on their existing phones, and therefore they needed to make a note of the ones that they planned to use that week. The initial setup took roughly 30 minutes and involved transferring their SIM cards, helping them set up their Google and other accounts, and making sure they had all the applications they needed. We compensated each participant with a $35 gift card for showing up at the setup session. Out of 40 people who were given phones, 2 did not return them, and 2 did not regularly use them during the study period. Of our 36 remaining participants who used the phones regularly, 19 were male and 17 were female; ages ranged from 20 to 63 years old (μ = 32, σ= 11).

After the initial setup session, participants used the experimental phones for one week in lieu of their normal phones. They were allowed to install and uninstall applications, and we instructed them to use these phones as they would their normal phones. Our logging framework kept track of every protected resource accessed by a user-level application along with the previously-mentioned contextual data. Due to storage constraints on the devices, our software uploaded log files to our server every two hours. However, to preserve participants' privacy, screenshots remained on the phones during the course of the week. At the end of the week, each participant returned to our laboratory, completed an exit survey, returned the phone, and then received an additional \$100 gift card (i.e., slightly more than the value of the phone).

2.2.3 Exit Survey

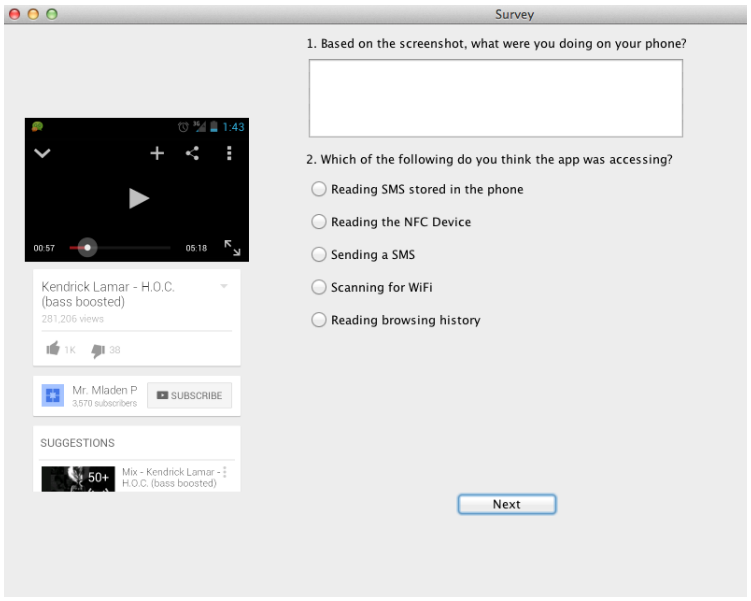
When participants returned to our laboratory, they completed an exit survey. The exit survey software ran on a laptop in a private room so that it could ask questions about what they were doing on their phones during the course of the week without raising privacy concerns. We did not view their screenshots until participants gave us permission. The survey had three components:

- **Screenshots**—Our software displayed a screenshot taken after one of the 12 resources in Table 2.1 was accessed. Next to the screenshot (Figure 2.2), we asked participants what they were doing on the phone when the screenshot was taken (open-ended). We also asked them to indicate which of several actions they believed the application was performing, chosen from a multiple-choice list of permissions presented in plain language (e.g., "reading browser history," "sending a SMS," etc.). After answering these questions, they proceeded to a second page of questions (Figure 2.3). We informed participants at the top of this page of the resource that the application had accessed when the screenshot was taken, and asked them to indicate how much they expected this (5-point Likert scale). Next, we asked, "if you were given the choice, would you have prevented the app from accessing this data," and to explain why or why not. Finally, we asked for permission to view the screenshot. This phase of the exit survey was repeated for 10-15 different screenshots per participant, based on the number of screenshots saved by our reservoir sampling algorithm.

- **Locked Screens**—The second part of our survey involved questions about the same protected resources, though accessed while device screens were off (i.e., participants were not using their phones). Because there were no contextual cues (i.e., screenshots), we outright told participants which applications were accessing which resources and asked them multiple choice questions about whether they wanted to prevent this and the degree to which these behaviors were expected. They answered these questions for up to 10 requests, similarly chosen by our reservoir sampling algorithm to yield a breadth of application/permission combinations.
- **Personal Privacy Preferences**—Finally, in order to correlate survey responses with privacy preferences, participants completed two privacy scales. Because of the numerous reliability problems with the Westin index [119], we computed the average of both Buchanan et al.'s Privacy Concerns Scale (PCS) [27] and Malhotra et al.'s Internet Users' Information Privacy Concerns (IUIPC) scale [77].

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

Three researchers independently coded 423 responses to the open-ended question in the screenshot portion of the survey. The number of responses per participant varied, as they were randomly selected based on the number of screenshots taken: participants who used their phones more heavily had more screenshots, and thus answered more questions. Prior to meeting to achieve consensus, the three coders disagreed on 42 responses, which resulted in an inter-rater agreement of 90%. Taking into account the 9 possible codings for each response, Fleiss' kappa yielded 0.61, indicating substantial agreement.



1. Based on the screenshot, what were you doing on your phone?

2. Which of the following do you think the app was accessing?

- Reading SMS stored in the phone
- Reading the NFC Device
- Sending a SMS
- Scanning for WiFi
- Reading browsing history

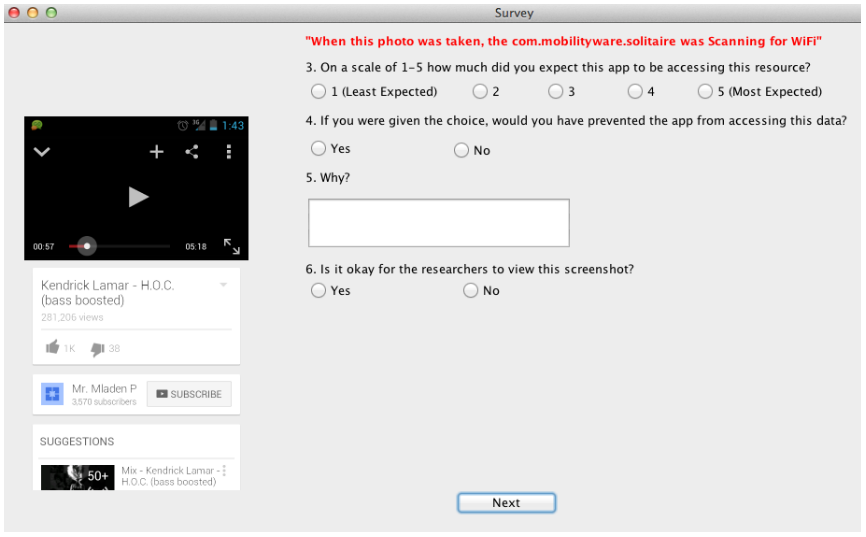
Figure 2.2: On the first screen, participants answered questions to establish awareness of the permission request based on the screenshot.

2.3 Application Behaviors

Over the week-long period, we logged 27M application requests to protected resources governed by Android permissions. This translates to over 100,000 requests per user/day. In this section, we quantify the circumstances under which these resources were accessed. We focus on the rate at which resources were accessed when participants were not actively using those applications (i.e., situations likely to defy users' expectations), access to certain resources with particularly high frequency, and the impact of replacing certain requests with runtime confirmation dialogs (as per Felt et al.'s suggestion [47]).

2.3.1 Invisible Permission Requests

In many cases, it is entirely expected that an application might make frequent requests to resources protected by permissions. For instance, the INTERNET permission is used every time an application needs to open a socket, ACCESS_FINE_LOCATION



"When this photo was taken, the com.mobilityware.solitaire was Scanning for WiFi"

3. On a scale of 1-5 how much did you expect this app to be accessing this resource?

◯ 1 (Least Expected) ◯ 2 ◯ 3 ◯ 4 ◯ 5 (Most Expected)

4. If you were given the choice, would you have prevented the app from accessing this data?

◯ Yes ◯ No

5. Why?

6. Is it okay for the researchers to view this screenshot?

◯ Yes ◯ No

Figure 2.3: On the second screen, they saw the resource accessed, stated whether it was expected, and whether it should have been blocked.

is used every time the user's location is checked by a mapping application, and so on. However, in these cases, one expects users to have certain contextual cues to help them understand that these applications are running and making these requests. Based on our log data, most requests occurred while participants were not actually interacting with those applications, nor did they have any cues to indicate that the applications were even running. When resources are accessed, applications can be in five different states, with regard to their visibility to users:

1. **Visible foreground application (12.04%)**: the user is using the application requesting the resource.
2. **Invisible background application (0.70%)**: due to multitasking, the application is in the background.
3. **Visible background service (12.86%)**: the application is a background service, but the user may be aware of its presence due to other cues (e.g., it is playing music or is present in the notification bar).
4. **Invisible background service (14.40%)**: the application is a background service without visibility.

5. **Screen off (60.00%)**: the application is running, but the phone screen is off because it is not in use.

Combining the 3.3M (12.04% of 27M) requests that were granted when the user was actively using the application (Category 1) with the 3.5M (12.86% of 27M) requests that were granted when the user had other contextual cues to indicate that the application was running (Category 3), we can see that fewer than one quarter of all permission requests (24.90% of 27M) occurred when the user had clear indications that those applications were running. This suggests that during the vast majority of the time, access to protected resources occurs opaquely to users. We focus on these 20.3M "invisible" requests (75.10% of 27M) in the remainder of this subsection.

Harbach et al. found that users' phone screens are off 94% of the time on average [57]. We observed that 60% of permission requests occurred while participants' phone screens were off, which suggests that permission requests occurred less frequently than when participants were using their phones. At the same time, certain applications made more requests when participants were not using their phones: "Brave Frontier Service," "Microsoft Sky Drive," and "Tile game by UMoni." Our study collected data on over 300 applications, and therefore it is possible that with a larger sample size, we would observe other applications engaging in this behavior. All of the aforementioned applications primarily requested ACCESS_WIFI_STATE and INTERNET. While a definitive explanation for this behavior requires examining source code or the call stacks of these applications, we hypothesize that they were continuously updating local data from remote servers. For instance, Sky Drive may have been updating documents, whereas the other two applications may have been checking the status of multiplayer games.

Table 2.3 shows the most frequently requested permissions from applications running invisibly to the user (i.e., Categories 2, 4, and 5); Table 2.4 shows the applications responsible for these requests (Appendix A.1 lists the permissions requested by these applications). We normalized the numbers to show requests per user/day. ACCESS_NETWORK_STATE was most frequently requested, av-

Permission	Requests
ACCESS_NETWORK_STATE	31,206
WAKE_LOCK	23,816
ACCESS_FINE_LOCATION	5,652
GET_ACCOUNTS	3,411
ACCESS_WIFI_STATE	1,826
UPDATE_DEVICE_STATS	1,426
ACCESS_COARSE_LOCATION	1,277
AUTHENTICATE_ACCOUNTS	644
READ_SYNC_SETTINGS	426
INTERNET	416

Table 2.3: The most frequently requested permissions by applications with zero visibility to the user.

Application	Requests
Facebook	36,346
Google Location Reporting	31,747
Facebook Messenger	22,008
Taptu DJ	10,662
Google Maps	5,483
Google Gapps	4,472
Foursquare	3,527
Yahoo Weather	2,659
Devexpert Weather	2,567
Tile Game(Umoni)	2,239

Table 2.4: The applications making the most permission requests while running invisibly to the user.

eraging 31,206 times per user/day—roughly once every 3 seconds. This is due to applications constantly checking for Internet connectivity. However, the 5,562 requests/day to ACCESS_FINE_LOCATION and 1,277 requests/day to ACCESS_COARSE_LOCATION are more concerning, as this could enable detailed tracking of the user's movement throughout the day. Similarly, a user's location can be inferred by using

ACCESS_WIFI_STATE to get data on nearby WiFi SSIDs.

Contextual integrity means ensuring that information flows are appropriate, as determined by the user. Thus, users need the ability to see information flows. Current mobile platforms have done some work to let the user know about location tracking. For instance, recent versions of Android allow users to see which applications have used location data recently. While attribution is a positive step towards contextual integrity, attribution is most beneficial for actions that are reversible, whereas the disclosure of location information is not something that can be undone [47]. We observed that fewer than 1% of location requests were made when the applications were visible to the user or resulted in the displaying of a GPS notification icon. Given that Thompson et al. showed that most users do not understand that applications running in the background may have the same abilities as applications running in the foreground [108], it is likely that in the vast majority of cases, users do not know when their locations are being disclosed.

This low visibility rate is because Android only shows a notification icon when the GPS sensor is accessed, while offering alternative ways of inferring location. In 66.1% of applications' location requests, they directly queried the `TelephonyManager`, which can be used to determine location via cellular tower information. In 33.3% of the cases, applications requested the SSIDs of nearby WiFi networks. In the remaining 0.6% of cases, applications accessed location information using one of three built-in location providers: GPS, network, or passive. Applications accessed the GPS location provider only 6% of the time (which displayed a GPS notification). In the other 94% of the time, 13% queried the network provider (i.e., approximate location based on nearby cellular towers and WiFi SSIDs) and 81% queried the passive location provider. The passive location provider caches prior requests made to either the GPS or network providers. Thus, across all requests for location data, the GPS notification icon appeared 0.04% of the time.

While the alternatives to querying the GPS are less accurate, users are still surprised by their accuracy [51]. This suggests a serious violation of contextual integrity, since users likely have no idea their locations are being requested in the vast majority of cases. Thus, runtime notifications for location tracking need to be improved [52].

Apart from these invisible location requests, we also observed applications reading stored SMS messages (125 times per user/day), reading browser history (5 times per user/day), and accessing the camera (once per user/day). Though the use of these permissions does not necessarily lead to privacy violations, users have no contextual cues to understand that these requests are occurring.

2.3.2 High Frequency Requests

Some permission requests occurred so frequently that a few applications (i.e., Facebook, Facebook Messenger, Google Location Reporting, Google Maps, Farm Heroes Saga) had to be rate limited in our log files (see Section 2.2.1), so that the logs would not fill up users' remaining storage or incur performance overhead. Table 2.5 shows the complete list of application/permission combinations that exceeded the threshold. For instance, the most frequent requests came from Facebook requesting ACCESS_NETWORK_STATE with an average interval of 213.88 ms (i.e., almost 5 times per second).

With the exception of Google's applications, all rate-limited applications made excessive requests for the connectivity state. We hypothesize that once these applications lose connectivity, they continuously poll the system until it is regained. Their use of the `getActiveNetworkInfo()` method results in permission checks and returns `NetworkInfo` objects, which allow them to determine connection state (e.g., connected, disconnected, etc.) and type (e.g., WiFi, Bluetooth, cellular, etc.). Thus, these requests do not appear to be leaking sensitive information *per se*, but their frequency may have adverse effects on performance and battery life. It is possible that using the `ConnectivityManager`'s `NetworkCallback` method may be able to fulfill this need with far fewer permission checks.

Application / Permission	**Peak (ms)**	**Avg. (ms)**
com.facebook.katana	213.88	956.97
ACCESS_NETWORK_STATE		
com.facebook.orca	334.78	1146.05
ACCESS_NETWORK_STATE		
com.google.android.apps.maps	247.89	624.61
ACCESS_NETWORK_STATE		
com.google.process.gapps	315.31	315.31
AUTHENTICATE_ACCOUNTS		
com.google.process.gapps	898.94	1400.20
WAKE_LOCK		
com.google.process.location	176.11	991.46
WAKE_LOCK		
com.google.process.location	1387.26	1387.26
ACCESS_FINE_LOCATION		
com.google.process.location	373.41	1878.88
GET_ACCOUNTS		
com.google.process.location	1901.91	1901.91
ACCESS_WIFI_STATE		
com.king.farmheroessaga	284.02	731.27
ACCESS_NETWORK_STATE		
com.pandora.android	541.37	541.37
ACCESS_NETWORK_STATE		
com.taptu.streams	1746.36	1746.36
ACCESS_NETWORK_STATE		

Table 2.5: The application/permission combinations that needed to be rate limited during the study. The last two columns show the fastest interval recorded and the average of all the intervals recorded before rate-limiting.

2.3.3 Frequency of Data Exposure

Felt et al. posited that while most permissions can be granted automatically in order to not habituate users to relatively benign risks, certain requests should require runtime consent [47]. They advocated using runtime dialogs before the following

Resource	**Visible**		**Invisible**		**Total**	
	Data Exposed	**Requests**	**Data Exposed**	**Requests**	**Data Exposed**	**Requests**
Location	758	2,205	3,881	8,755	4,639	10,960
Read SMS data	378	486	72	125	450	611
Sending SMS	7	7	1	1	8	8
Browser History	12	14	2	5	14	19
Total	1,155	2,712	3,956	8,886	5,111	11,598

Table 2.6: The sensitive permission requests (per user/day) when requesting applications were visible/invisible to users. "Data exposed" reflects the subset of permission-protected requests that resulted in sensitive data being accessed.

actions should proceed:

1. Reading location information (e.g., using conventional location APIs, scanning WiFi SSIDs, etc.).
2. Reading the user's web browser history.
3. Reading saved SMS messages.
4. Sending SMS messages that incur charges, or inappropriately spamming the user's contact list.

These four actions are governed by the 12 Android permissions listed in Table 2.1. Of the 300 applications that we observed during the experiment, 91 (30.3%) performed one of these actions. On average, these permissions were requested 213 times per hour/user—roughly every 20 seconds. However, permission checks occur under a variety of circumstances, only a subset of which expose sensitive resources. As a result, platform developers may decide to only show runtime warnings to users when protected data is read or modified. Thus, we attempted to quantify the frequency with which permission checks actually result in access to sensitive resources for each of these four categories. Table 2.6 shows the number of requests seen per user/day under each of these four categories, separating the instances in which sensitive data was exposed from the total permission requests observed. Unlike Section 2.3.1, we include "visible" permission requests (i.e., those occurring while the user was actively using the application or had other contextual information to indicate it was running). We didn't observe any uses of NFC, READ_CALL_LOG, ADD_VOICEMAIL, accessing WRITE_SYNC_SETTINGS or INTERNET while roaming in our dataset.

Of the location permission checks, a majority were due to requests for location provider information (e.g., `getBestProvider()` returns the best location provider based on application requirements), or checking WiFi state (e.g., `getWifiState()` only reveals whether WiFi is enabled). Only a portion of the requests actually exposed participants' locations (e.g., `getLastKnownLocation()` or `getScanResults()` exposed SSIDs of nearby WiFi networks).

Although a majority of requests for the READ_SMS permission exposed content in the SMS store (e.g., `Query()` reads the contents of the SMS store), a considerable portion simply read information about the SMS store (e.g., `renewMmsConnectivity()` resets an applications' connection to the MMS store). An exception to this is the use of SEND_SMS, which resulted in the transmission of an SMS message every time the permission was requested.

Regarding browser history, both accessing visited URLs (`getAllVisitedUrls()`) and reorganizing bookmark folders (`addFolderToCurrent()`) result in the same permission being checked. However, the latter does not expose specific URLs to the invoking application.

Our analysis of the API calls indicated that on average, only half of all permission checks granted applications access to sensitive data. For instance, across both visible and invisible requests, 5,111 of the 11,598 (44.3%) permission checks involving the 12 permissions in Table 2.1 resulted in the exposure of sensitive data (Table 2.6).

While limiting runtime permission requests to only the cases in which protected resources are exposed will greatly decrease the number of user interruptions, the frequency with which these requests occur is still too great. Prompting the user on the first request is also not appropriate (e.g., à la iOS and Android M), because our data show that in the vast majority of cases, the user has no contextual cues to understand when protected resources are being accessed. Thus, a user may grant a request the first time an application asks, because it is appropriate in that instance, but then she may be surprised to find that the application continues to access that resource in other contexts (e.g., when the application is not actively used). As a result, a more intelligent method is needed to determine when a given permission request is likely to be deemed appropriate by the user.

2.4 User Expectations and Reactions

To identify when users might want to be prompted about permission requests, our exit survey focused on participants' reactions to the 12 permissions in Table 2.1, limiting the number of requests shown to each participant based on our reservoir sampling algorithm, which was designed to ask participants about a diverse set of permission/application combinations. We collected participants' reactions to 673 permission requests ($\approx$19/participant). Of these, 423 included screenshots because participants were actively using their phones when the requests were made, whereas 250 permission requests were performed while device screens were off.[2] Of the former, 243 screenshots were taken while the requesting application was visible (Category 1 and 3 from Section 2.3.1), whereas 180 were taken while the application was invisible (Category 2 and 4 from Section 2.3.1). In this section, we describe the situations in which requests defied users' expectations. We present explanations for why participants wanted to block certain requests, the factors influencing those decisions, and how expectations changed when devices were not in use.

2.4.1 Reasons for Blocking

When viewing screenshots of what they were doing when an application requested a permission, 30 participants (80% of 36) stated that they would have preferred

[2]Our first 11 participants did not answer questions about permission requests occurring while not using their devices, and therefore the data only corresponds to our last 25 participants.

to block at least one request, whereas 6 stated a willingness to allow all requests, regardless of resource type or application. Across the entire study, participants wanted to block 35% of these 423 permission requests. When we asked participants to explain their rationales for these decisions, two main themes emerged: the request did not—in their minds—pertain to application functionality or it involved information they were uncomfortable sharing.

Relevance to Application Functionality

When prompted for the reason behind blocking a permission request, 19 (53% of 36) participants did not believe it was necessary for the application to perform its task. Of the 149 (35% of 423) requests that participants would have preferred to block, 79 (53%) were perceived as being irrelevant to the functionality of the application:

- *"It wasn't doing anything that needed my current location."* (P1)
- *"I don't understand why this app would do anything with SMS."* (P10)

Accordingly, functionality was the most common reason for wanting a permission request to proceed. Out of the 274 permissible requests, 195 (71% of 274) were perceived as necessary for the core functionality of the application, as noted by thirty-one (86% of 36) participants:

- *"Because it's a weather app and it needs to know where you are to give you weather information."*(P13)
- *"I think it needs to read the SMS to keep track of the chat conversation."*(P12)

Beyond being necessary for core functionality, participants wanted 10% (27 of 274) of requests to proceed because they offered convenience; 90% of these requests were for location data, and the majority of those applications were published under the Weather, Social, and Travel & Local categories in the Google Play store:

- *"It selects the closest stop to me so I don't have to scroll through the whole list."* (P0)
- *"This app should read my current location. I'd like for it to, so I won't have to manually enter in my zip code / area."* (P4)

Thus, requests were allowed when they were expected: when participants rated the extent to which each request was expected on a 5-point Likert scale, allowable requests averaged 3.2, whereas blocked requests averaged 2.3 (lower is less expected).

Privacy Concerns

Participants also wanted to deny permission requests that involved data that they considered sensitive, regardless of whether they believed the application actually needed the data to function. Nineteen (53% of 36) participants noted privacy as a concern while blocking a request, and of the 149 requests that participants wanted to block, 49 (32% of 149) requests were blocked for this reason:

- *"SMS messages are quite personal."* (P14)
- *"It is part of a personal conversation."* (P11)
- *"Pictures could be very private and I wouldn't like for anybody to have access."* (P16)

Conversely, 24 participants (66% of 36) wanted requests to proceed simply because they did not believe that the data involved was particularly sensitive; this reasoning accounted for 21% of the 274 allowable requests:

- *"I'm ok with my location being recorded, no concerns."* (P3)
- *"No personal info being shared."* (P29)

2.4.2 Influential Factors

Based on participants' responses to the 423 permission requests involving screenshots (i.e., requests occurring while they were actively using their phones), we quantitatively examined how various factors influenced their desire to block some of these requests.

Effects of Identifying Permissions on Blocking: In the exit survey, we asked participants to guess the permission an application was requesting, based on the screenshot of what they were doing at the time. The real answer was among four other incorrect answers. Of the 149 cases where participants wanted to block permission requests, they were only able to correctly state what permission was being requested 24% of the time; whereas when wanting a request to proceed, they correctly identified the requested permission 44% (120 of 274) of the time. However, Pearson's product-moment test on the average number of blocked requests per user and the average number of correct answers per user[3] did not yield a statistically significant correlation ($r{=}-0.171$, $p{<}0.317$).

Effects of Visibility on Expectations: We were particularly interested in exploring if permission requests originating from foreground applications (i.e., visible to the user) were more expected than ones from background applications. Of the 243 visible permission requests that we asked about in our exit survey, participants correctly identified the requested permission 44% of the time, and their average rating on our expectation scale was 3.4. On the other hand, participants correctly identified the resources accessed by background applications only 29% of the time (52 of 180), and their average rating on our expectation scale was 3.0. A Wilcoxon Signed-Rank test with continuity correction revealed a statistically significant difference in participants' expectations between these two groups ($V{=}441.5$, $p{<}0.001$).

Effects of Visibility on Blocking: Participants wanted to block 71 (29% of 243) permission requests originating from applications running in the foreground, whereas this increased by almost 50% when the applications were in the background invisible to them (43% of 180). We calculated the percentage of denials for each participant, for both visible and invisible requests. A Wilcoxon Signed-Rank test with continuity correction revealed a statistically significant difference

[3]Both measures were normally distributed.

(V=58, p<0.001).

Effects of Privacy Preferences on Blocking: Participants completed the Privacy Concerns Scale (PCS) [27] and the Internet Users' Information Privacy Concerns (IUIPC) scale [77]. A Spearman's rank test yielded no statistically significant correlation between their privacy preferences and their desire to block permission requests ($\rho = 0.156$, p<0.364).

Effects of Expectations on Blocking: We examined whether participants' expectations surrounding requests correlated with their desire to block them. For each participant, we calculated their average Likert scores for their expectations and the percentage of requests that they wanted to block. Pearson's product-moment test showed a statistically significant correlation (r=−0.39, p<0.018). The negative correlation shows that participants were more likely to deny unexpected requests.

2.4.3 User Inactivity and Resource Access

In the second part of the exit survey, participants answered questions about 10 resource requests that occurred when the screen was off (not in use). Overall, they were more likely to expect resource requests to occur when using their devices ($\mu = 3.26$ versus $\mu = 2.66$). They also stated a willingness to block almost half of the permission requests (49.6% of 250) when not in use, compared to a third of the requests that occurred when using their phones (35.2% of 423). However, neither of these differences was statistically significant.

2.5 Feasibility of Runtime Requests

Felt et al. posited that certain sensitive permissions (Table 2.1) should require runtime consent [47], but in Section 2.3.3 we showed that the frequencies with which applications are requesting these permissions make it impractical to prompt the user each time a request occurs. Instead, the major mobile platforms have shifted towards a model of prompting the user the first time an application requests access to certain resources: iOS does this for a selected set of resources, such as location and contacts, and Android M does this for "dangerous" permissions.

How many prompts would users see, if we added runtime prompts for the first use of these 12 permissions? We analyzed a scheme where a runtime prompt is displayed at most once for each unique triplet of (application, permission, application visibility), assuming the screen is on. With a naïve scheme, our study data indicates our participants would have seen an average of 34 runtime prompts (ranging from 13 to 77, σ=11). As a refinement, we propose that the user should be prompted only if sensitive data will be exposed (Section 2.3.3), reducing the average number of prompts to 29.

Of these 29 prompts, 21 (72%) are related to location. Apple iOS already prompts users when an application accesses location for the first time, with no evidence of user habituation or annoyance. Focusing on the remaining prompts, we see that our policy would introduce an average of 8 new prompts per user: about 5 for reading SMS, 1 for sending SMS, and 2 for reading browser history. Our data covers only the first week of use, but as we only prompt on first use of a permission, we expect that the number of prompts would decline greatly in subsequent weeks, suggesting that this policy would likely not introduce significant risk of habituation or annoyance. Thus, our results suggest adding runtime prompts for reading SMS, sending SMS, and reading browser history would be useful given their sensitivity and low frequency.

Our data suggests that taking visibility into account is important. If we ignore visibility and prompted only once for each pair of (application, permission), users would have no way to select a different policy for when the application is visible or not visible. In contrast, "ask-on-first-use" for the triple (application, permission, visibility) gives users the option to vary their decision based on the visibility of the requesting application. We evaluated these two policies by analyzing the exit survey data (limited to situations where the screen was on) for cases where the same user was asked twice in the survey about situations with the same (application, permission) pair or the same (application, permission, visibility) triplet, to see whether the user's first decision to block or not matched their subsequent decisions. For the former policy, we saw only 51.3% agreement; for the latter, agreement increased to 83.5%. This suggests that the (application, permission, visibility) triplet captures many of the contextual factors that users care about, and thus it is reasonable to prompt only once per unique triplet.

A complicating factor is that applications can also run even when the user is not actively using the phone. In addition to the 29 prompts mentioned above, our data indicates applications would have triggered an average of 7 more prompts while the user was not actively using the phone: 6 for location and one for reading SMS. It is not clear how to handle prompts when the user is not available to respond to the prompt: attribution might be helpful, but further research is needed.

2.5.1 Modeling Users' Decisions

We constructed several statistical models to examine whether users' desire to block certain permission requests could be predicted using the contextual data that we collected. If such a relationship exists, a classifier could determine when to deny potentially unexpected permission requests without user intervention. Conversely, the classifier could be used to only prompt the user about questionable data requests. Thus, the response variable in our models is the user's choice of whether to block the given permission request. Our predictive variables consisted of the information that might be available at runtime: permission type (with the restriction that the invoked function exposes data), requesting application, and visibility of that application. We constructed several mixed effects binary logistic regression models to account for both inter-subject and intra-subject effects.

Model Selection

In our mixed effects models, permission types and the visibility of the requesting application were fixed effects, because all possible values for each variable existed in our data set. Visibility had two values: visible (the user is interacting with the application or has other contextual cues to know that it is running) and invisible.

Permission types were categorized based on Table 2.6. The application name and the participant ID were included as random effects, because our survey data did not have an exhaustive list of all possible applications a user could run, and the participant has a non-systematic effect on the data.

Table 2.7 shows two goodness-of-fit metrics: the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). Lower values for AIC and BIC represent better fit. Table 2.7 shows the different parameters included in each model. We found no evidence of interaction effects and therefore did not include them. Visual inspection of residual plots of each model did not reveal obvious deviations from homoscedasticity or normality.

We initially included the phone's screen state as another variable. However, we found that creating two separate models based on the screen state resulted in better fit than using a single model that accounted for screen state as a fixed effect. When the screen was on, the best fit was a model including application visibility and application name, while controlling for subject effects. Here, fit improved once permission type was removed from the model, which shows that the decision to block a permission request was based on contextual factors: users do not categorically deny permission requests based solely on the type of resource being accessed (i.e., they also account for their trust in the application, as well as whether they happened to be actively using it). When the screen was off, however, the effect of permission type was relatively stronger. The strong subject effect in both models indicates that these decisions vary from one user to the next. As a result, any classifier developed to automatically decide whether to block a permission at runtime (or prompt the user) will need to be tailored to that particular user's needs.

Predicting User Reactions

Using these two models, we built two classifiers to make decisions about whether to block any of the sensitive permission requests listed in Table 2.6. We used our exit survey data as ground truth, and used 5-fold cross-validation to evaluate

Predictors	**AIC**	**BIC**	**Screen State**
UserCode	490.60	498.69	Screen On
Application	545.98	554.07	Screen On
Application UserCode	491.86	503.99	Screen On
Permission Application UserCode	494.69	527.05	Screen On
Visibility Application UserCode	**481.65**	**497.83**	**Screen On**
Permission Visibility Application UserCode	484.23	520.64	Screen On
UserCode	245.13	252.25	Screen Off
Application	349.38	356.50	Screen Off
Application UserCode	238.84	249.52	Screen Off
Permission Application UserCode	**235.48**	**263.97**	**Screen Off**

Table 2.7: Goodness-of-fit metrics for various mixed effects logistic regression models on the exit survey data.

model accuracy.

We calculated the receiver operating characteristic (ROC) to capture the trade-off between true-positive and false-positive rate. The quality of the classifier can be quantified with a single value by calculating the area under its ROC curve (AUC) [58]. The closer the AUC gets to 1.0, the better the classifier is. When screens were on, the AUC was 0.7, which is 40% better than the random baseline (0.5). When screens were off, the AUC was 0.8, which is 60% better than a random baseline.

2.6 Discussion

During the study, 80% of our participants deemed at least one permission request as inappropriate. This violates Nissenbaum's notion of "privacy as contextual integrity" because applications were performing actions that defied users' expectations [83]. Felt et al. posited that users may be able to better understand why permission requests are needed if some of these requests are granted via runtime consent dialogs, rather than Android's previous install-time notification approach [47]. By granting permissions at runtime, users will have additional contextual information; based on what they were doing at the time that resources are requested, they may have a better idea of why those resources are being requested.

We make two primary contributions that system designers can use to make more usable permissions systems. We show that the visibility of the requesting application and the frequency at which requests occur are two important factors in designing a runtime consent platform. Also, we show that "prompt-on-first-use" per triplet could be implemented for some sensitive permissions without risking user habituation or annoyance.

Based on the frequency with which runtime permissions are requested (Section 2.3), it is infeasible to prompt users every time. Doing so would overwhelm them and lead to habituation. At the same time, drawing user attention to the situations in which users are likely to be concerned will lead to greater control and awareness. Thus, the challenge is in acquiring their preferences by confronting them minimally and then automatically inferring *when* users are likely to find a permission request unexpected, and only prompting them in these cases. Our data suggests that participants' desires to block particular permissions were heavily influenced by two main factors: their understanding of the relevance of a permission request to the functionality of the requesting application and their individual privacy concerns.

Our models in Section 2.5.1 showed that individual characteristics greatly explain the variance between what different users deem appropriate, in terms of access to protected resources. While responses to privacy scales failed to explain these differences, this was not a surprise, as the disconnect between stated privacy preferences and behaviors is well-documented (e.g., [2]). This means that in order to accurately model user preferences, the system will need to learn what a specific user deems inappropriate over time. Thus, a feedback loop is likely needed: when devices are “new,” users will be required to provide more input surrounding permission requests, and then based on their responses, they will see fewer requests in the future. Our data suggests that prompting once for each unique (application, permission, application visibility) triplet can serve as a practical mechanism in acquiring users’ privacy preferences.

Beyond individual subject characteristics (i.e., personal preferences), participants based their decisions to block certain permission requests on the specific applications making the requests and whether they had contextual cues to indicate that the applications were running (and therefore needed the data to function). Future systems could take these factors into account when deciding whether or not to draw user attention to a particular request. For example, when an application that a user is not actively using requests access to a protected resource, she should be shown a runtime prompt. Our data indicates that, if the user decides to grant a request in this situation, then with probability 0.84 the same decision will hold in future situations where she is actively using that same application, and therefore a subsequent prompt may not be needed. At a minimum, platforms need to treat permission requests from background applications differently than those originating from foreground applications. Similarly, applications running in the background should use passive indicators to communicate when they are accessing particular resources. Platforms can also be designed to make decisions about whether or not access to resources should be granted based on whether contextual cues are present, or at its most basic, whether the device screen is even on.

Finally, we built our models and analyzed our data within the framework of what resources our participants *believed* were necessary for applications to correctly function. Obviously, their perceptions may have been incorrect: if they better understood why a particular resource was necessary, they may have been more permissive. Thus, it is incumbent on developers to adequately communicate why particular resources are needed, as this impacts user notions of contextual integrity. Yet, no mechanisms in Android exist for developers to do this as part of the permission-granting process. For example, one could imagine requiring metadata to be provided that explains how each requested resource will be used, and then automatically integrating this information into permission requests. Tan et al. examined a similar feature on iOS that allows developers to include free-form text in runtime permission dialogs and observed that users were more likely to grant requests that included this text [107]. Thus, we believe that including succinct explanations in these requests would help preserve contextual integrity by promoting greater transparency.

In conclusion, we believe this study was instructive in showing the circumstances in which Android permission requests are made under real-world usage. While prior work has already identified some limitations of deployed mobile permissions systems, we believe our study can benefit system designers by demonstrating several ways in which contextual integrity can be improved, thereby empowering users to make better security decisions.

Chapter 3

Prediction

In iOS and Android M, the platform prompts the user when an application attempts to access one of a set of "dangerous"[1] permission types (e.g., location, contacts, etc.) for the first time. 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.

One critical caveat of this approach 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. Based on the results presented in the previous chapter, however, it is not feasible to prompt the user every time data is accessed, due to the high frequency of permission requests.

In the previous chapter we presented data from a field to operationalize the

[1]Android designated a selected set of 24 resource types as more sensitive over the other resources.[54]

notion of “context,” to allow an operating system to differentiate between appropriate and inappropriate data requests by a single application for a single data type. We show that users’ decisions to allow a permission request significantly correlated with that application’s visibility – that this visibility is a strong contextual cue that influences users’ responses to permission prompts. We also observed that privacy decisions were highly nuanced, demonstrating that a one-size-fits-all model is unlikely to be sufficient; a given information flow may be deemed appropriate by one user but not by another user – we suggest 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 to detect when context has likely changed (insofar as to decide whether a different privacy decision should be made for the same application and data type), by seeing whether the circumstances surrounding a data request are similar to previous requests.

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 agrees with participants’ prompted responses 84% of the time. AOI achieves only 25% accuracy.

We 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, thereby reducing user burden. Our classifier uses the user’s past decisions in similar situations to predict their response to a particular permission request. 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 in other security and privacy situations to make inferences about their preferences: whether they have a screen lock activated, how often they visit HTTPS websites, and so on. We show that our scheme achieves 96.8% accuracy (a 4$\times$ reduction in error rate over AOFU) with significantly less user involvement than the *status quo*.

The specific contributions of this chapter are the following:

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

3.1 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 [55, 65, 116]. 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 [49, 64, 108]. Applications also transmit a large amount of location or other sensitive data to third parties without user consent [43]. When possible risks associated with these requests are revealed to users, their concerns range from being annoyed to wanting to seek retribution [48].

To mitigate some of these problems, systems have been developed to track information flows across the Android system [43, 53, 67] or introduce finer-grained permission control into Android [5, 60, 100], 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 [122]. Other approaches involve static analysis in order to better understand how applications *could* request information [11, 23, 45], 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 [43, 104, 117], but substantial work is likely needed to present that information to average users in a meaningful way. Solutions that require 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 [125], or detect privacy violations and suggest preferences based on crowdsourcing [4, 71], but such approaches often do not take individual user differences into account without significant user intervention. Systems have also been developed to predict what users would share on mobile social networks [22], 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 [72, 97]. However, researchers have found that the order in which information is requested has an impact on prediction accuracy [120], which could mean that such systems are only likely to be accurate when they examine actual user behavior over time (as opposed to 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 [73]. Their dataset, however, was collected from a set of highly privacy-conscious individuals: those who choose 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. These conflicting decisions, however, do not represent noisy data. They occur nearly 50% of the time in the real world as it was shown in the previous chapter [117], and accurately reflect the nuances of user privacy preferences. Models must therefore account for them. In fact, previous work found that users commonly reassess privacy preferences after usage [6]. Liu et al. also expect users to make 10% of permission decisions manually, which, based on our previous field study results, would result in being prompted every three minutes [117]. 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.

Closely related to this work, Liu et al. [74] performed a field study to measure the effectiveness of a Privacy Assistant that offers recommendations to users on privacy settings that they could adopt based on each user's privacy profile—the privacy assistant predicts what the user might want based on the inferred privacy profile and static analysis of the third-party application. While this approach increased user awareness on resource usage, the recommendations are static: they do not consider each application's access to sensitive data on a case-by-case basis. Such a coarse-grained approach goes against our previous work suggesting that users do want to vary their decisions based on contextual circumstances [117]. A blanket approval or denial of a permission to a given application carries a considerable risk of privacy violations or loss of desired functionality. In contrast, our work uses dynamic analysis to infer the appropriateness of each given request by considering the surrounding contextual cues and how the user has behaved in similar situations in the past. As with Liu et al., their dataset was also collected from privacy-conscious and considerably tech-savvy individuals, which may limit the generalization of their results. The field study we conduct in our work uses a more representative sample.

Nissenbaum's theory of contextual integrity suggests that permission models should focus on information flows that are likely to defy user expectations [82]. There are three main components involved in deciding the appropriateness of a flow [20]: the context in which the resource request is made, the role played by the requesting 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, thereby reducing user burden.

Access Control Gadgets (ACGs) were proposed as a mechanism to tie sensitive resource access to certain UI elements [78, 94–96]. Authors posit that such an approach will increase user expectations, as a significant portion of participants expected a UI interaction before a sensitive resource usage, giving users an implicit mechanism to control access and increasing awareness on resource usage. The biggest caveat in this approach is that tying a UI interaction to each sensitive resource access is impossible in practice because resources are accessed at a high frequency [117], and because many legitimate resource accesses occur without user initiation [47].

3.2 Methodology

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

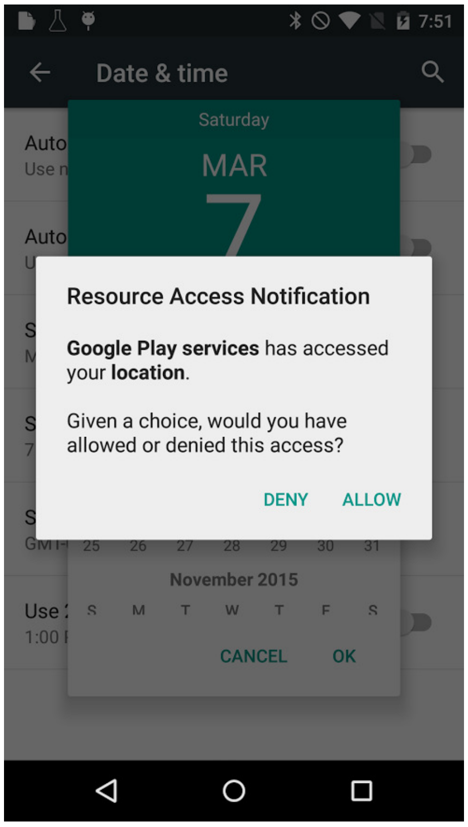
Previous work by Felt et al. made the argument that certain permissions are appropriate for runtime prompts, because they protect sensitive resources and because viewing the prompt at runtime imparts additional contextual information about why an application might need the permission [47]. Similarly, Thompson et al. showed that other permission requests could be replaced with audit mechanisms, because they represent either reversible changes or are sufficiently low

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 3.1: 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 [47]. Our instrumentation omits the last two permission types (INTERNET & WRITE_SYNC_SETTINGS) and records information about the other 10.

risk to not warrant habituating the user to prompts [108]. We collected information about 10 of the 12 permissions Felt et al. suggest are best-suited for runtime prompts. We omitted INTERNET and WRITE_SYNC_SETTINGS, because those permissions only warrant runtime prompts if the user is roaming and we did not expect any participant to be roaming during the study period, and focused on the remaining 10 permission types (Table 3.1). 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, ACGs, etc.).

We used the Experience Sampling Method (ESM) to collect ground truth data about users' privacy preferences [59]. ESM involves repeatedly questioning par-



Resource Access Notification

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

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

Figure 3.1: A screenshot of an ESM prompt.

ticipants *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 given the choice. We treated participants' responses to these ESM probes as our main dependent variable (Figure 4.1).

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 non-affiliated academic researchers access to the PhoneLab panel [80], which consists of more than 200 participants. 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 have neither data on all PhoneLab participants nor data 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 received institutional review board (IRB) approval.[2]

3.2.1 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 performance cost. We collected three categories of data: behavioral information, runtime information, and user decisions. We made no modifications to any third-party application code; our dynamic analysis techniques could be used on any third-party Android application.

Table 3.2 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 habits, screen locking behavior, third-party application usage behavior, audio preferences, call habits, camera usage patterns, and behavior related to security settings. For example, we hypothesized that someone who manually locks their device screen are more privacy-conscious than someone who lets it time out.

We also collected runtime information about the context of each permission

[2]Approved by the UC Berkeley IRB under protocol #2013-02-4992

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 (front- vs. rear-facing)
	Visiting an HTTPS 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 3.2: Instrumented events that form our feature set

request, including the visibility of the requesting application at the time of request, what the user was doing when the request was made (i.e., the name of the foreground application), and the exact Android API function invoked by the application to determine what information was requested. The visibility of an application reflects the extent to which the user was likely aware that 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—some background services can still be visible to the user due to on-screen notification or other cues that could be perceptible. We monitored processes' memory priority levels to determine the visibility of all Android processes. We also collected information about which Android `Activity` was active in the application.[3]

Once per day we *probabilistically* selected one of these permission requests and prompted the user about them at runtime (Figure 4.1). We used weighted reservoir sampling to select a permission request to prompt about. We weight the combination of *application, permission, visibility* based on their frequency of occurrence seen by the instrumentation; the most-frequent combination has a higher probability of being shown to participants using ESM. We prompted participants a maximum of three times for each unique combination. 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 chapter.

The intuition behind using 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 to their high frequency. Thus, it is more important to learn about user privacy decisions on highly frequent permission requests over the rare ones, which might not risk user habituation or annoyance (and the context of rare requests may be less likely to change).

3.2.2 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

[3]An Android `Activity` represents the application screen and UI elements currently exposed to the user.

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 ($\mu = 40.83$, $\sigma = 14.32$). 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.

3.2.3 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 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 findings in the previous chapter [117].

3.3 Types of Users

We hypothesized that there may be different types of users based on how they want to disclose their private information to third parties. It is imperative to identify these different sub-populations since 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.

While our study size was too small to effectively apply clustering techniques to generate classes of users, we did 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 and examined how these different participants considered the surrounding contextual circumstances in their decisions.

We discovered that application visibility was a significant factor 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 seem to care about the surrounding context (i.e., they make nuanced decisions, allowing or denying a permission request based on whether they had contextual cues that indicated that the requesting application was running), and the latter group *Defaulters*, because they seem to simply always allow or always deny requests, regardless of contextual cues. We analyzed how the effects of the visibility of the requesting application varies among different participants to decide the boundaries of the two groups. The reason for us to draw the boundary at the two ends was better explained later when we explain the benefit of the figuring out the defaulter as early as possible for

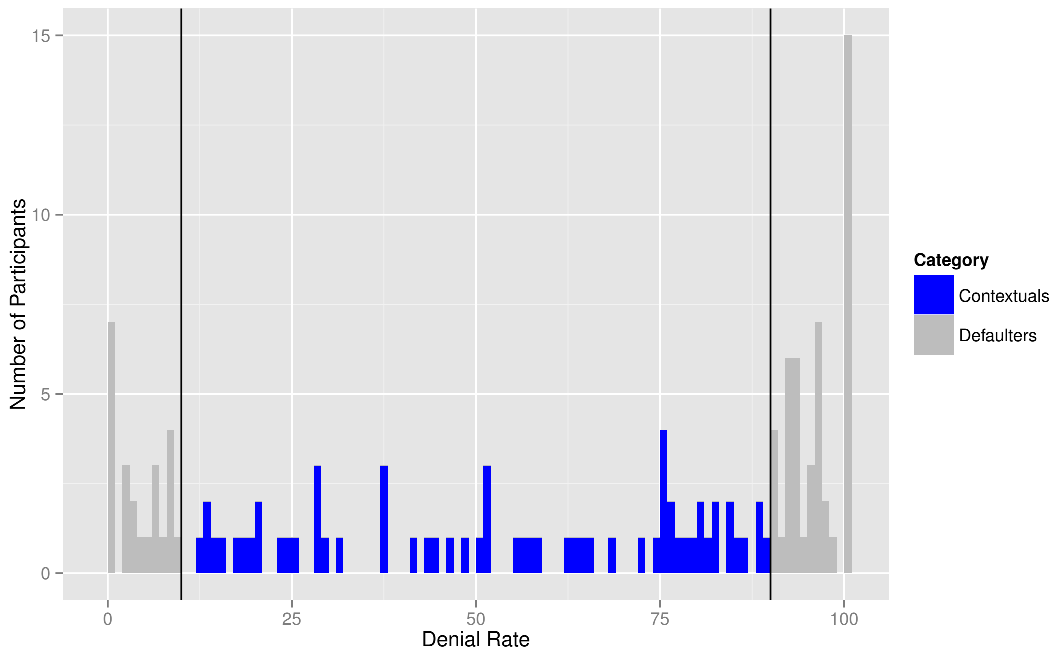


Figure 3.2: Histogram of users based on their denial rate. *Defaulters* tended to allow or deny almost all requests without regard for contextual cues, whereas *Contextuals* considered the visibility of the requesting application.

better performance.

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 the appropriateness of a given permission request, whereas *Defaulters* did not vary their decisions based on this cue. Figure 3.2 shows the distribution of users based on their denial rate. Vertical lines indicate the borders between *Contextuals* and *Defaulters*.

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_FPV	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 3.3: 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.

In the remainder of this chapter, we use our *Contextuals–Defaulters* categorization to measure how current and proposed models affect these two sub-populations, issues unique to these sub-populations, and ways to address these issues.

3.4 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 prompts the user whenever an application requests a *dangerous* permission for the first time [36]; the user's response to this prompt is thereafter applied whenever the same application requests the same permission. As of March 2017, only 34.1% of Android users have Android Marshmallow or a higher version [38], and among these Marshmallow users, those who upgraded from a previous version only see runtime permission prompts for freshly-installed applications.

For the remaining 65.9% 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. Because 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 3.3 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 quantifying 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 3.3 shows the expected median accuracy[4] for each policy based on par-

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

ticipants' 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 the Android 6.0 model employs. 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. [5] Moving forward, we focus our analysis only on AOFU-AP (i.e., the current standard).

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. In contrast, the decisions of *Contextuals* vary due to other factors beyond just the requesting application 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 factors affecting their decisions, such as behav-

[5] 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.

ioral tendencies and contextual cues. As Table 3.3 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.

The context in which users are prompted under AOFU might be a factor affecting its ability to predict subsequent instances. In the previous chapter [117], we found that the visibility of the requesting application is a strong contextual cue users use to vary their decisions. During the study period, under the AOFU-AP policy, 60% of the prompts could have occurred when the requesting application was visible to the participant—these prompts had an accuracy of 83.3% in predicting subsequent instances. In instances where participants were prompted when the requesting application was running invisibly to the user, AOFU-AP had an accuracy of 93.7% in predicting subsequent instances. A Wilcoxon signed-ranks test, however, did not reveal a statistically significant difference ($p = 0.3735$).

Our estimated accuracy numbers for AOFU may be inflated because AOFU in deployment (Android 6 and above) does not filter 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 avoiding any false positives), a distinction that AOFU in Android does not make. Thus, an Android user would see a permission request prompt when the application examines the list of location providers, and if the permission is granted, would not subsequently see prompts when location data is actually captured. In our previous field study, we found that 79% of first-time permission requests do not reveal any sensitive information [117], 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, AOFU's average accuracy is likely less than the numbers presented in Table 3.3. We therefore consider our estimates of

AOFU to be an upper bound.

3.5 Learning Privacy Preferences

Table 3.3 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. 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 3.8.

3.5.1 Feature Selection

Using the behavioral, contextual, and aggregate features shown in Table 3.2, we constructed 16K candidate features, formed by combinations of specific applications and actions. We then selected 20 features by measuring Gini importance through random forests [75], 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 3.4 lists the 20 features used in the rest of this work.

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 (A)	Average denial rate for (A1) application:permission:visibility	Numerical
	Average denial rate for (A2) application_F:permission:visibility	Numerical

Table 3.4: The complete list of features used in the ML model evaluation. All the numerical values in 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.

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, permission requested, and time of day of the request. Initially, we included the user ID to account for user-to-user variance, but as we discuss later, we subsequently removed it. Surprisingly, the application requesting the permission was not predictive, nor were other features based on the requesting application, such as application popularity.

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 perceives the appropriateness of a given request in order to accurately predict 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 may partly capture users' privacy preferences. We compute the average denial rate for each unique combination of *application:permission:visibility* (*A1*) and of *application$_F$*[6]*:permission: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.

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

Feature Set	Contextuals	Defaulters	Overall
R1	69.30%	95.80%	83.71%
R2 + B	69.48%	95.92%	83.93%
R2 + A	75.45%	99.20%	92.24%

Table 3.5: The median accuracy of the machine learning model for different feature groups across different sub populations.

3.5.2 Inference Based on Behavior

One of our main hypotheses is that passively observing users' behaviors helps infer users' future privacy decisions. To this end, we instrumented Android to collect a wide array of behavioral data, listed in Table 3.2. 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 (§3.5.1), we found that only locking behavior, audio habits, and web-browsing habits correlated with privacy behaviors. Appendix B.2 contains more information on feature importance. All the numerical values under the behavioral group were normalized per day.

We trained an SVM model with an RBF kernel on only the behavioral and runtime features listed in Table 3.4, 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 of using behavioral data is that it can be observed passively without any active user involvement (i.e., no prompting).

We use leave-one-out cross validation 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). 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 and runtime information on the request, without ever prompting that new user. 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 $3\times$ 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 56% 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.

3.5.3 Inference Based on Contextual Cues

Our SVM model with an RBF kernel produced the best accuracy. The results in the remainder of this section are trained and tested with five-fold cross validation with random splitting for an SVM model with an 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. We used one-hot encoding for categorical variables. We normalized numerical variables by making each one a z-score relative to its own average. Table 3.5 shows how the median accuracy changes with different 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 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*, *appli-cation*$_F$*: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 3.5 suggest, this variance is correlated with some contextual cues. The high predictive power of aggregate features indicates that they may be capturing the contextual cues, used by *Contextuals* to make decisions, to a greater extent.

The fact that both *application:permission:visibility* and $application_F$*:permission:visibility* are highly predictive (Appendix B.1) indicates that user responses for these combinations are consistent. The high consistency could relate to the notion that the visibility and the foreground application (application$_F$[7]) are strong contextual cues people use to make their privacy decisions; the only previously studied contextual cue was the visibility of the application requesting the sensitive data

[7]Even when the requesting application is running visible to the user, the foreground application could still be different from the requesting application since the only visible cue of the requesting application could be a notification in the notification bar.

– which based on our data from the previous field study [117]. 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 Section 3.8.

The model trained on feature sets *R2*, *A1*, and *A2* had the best accuracy (and the fewest privacy violations). For the remainder of the chapter, we will refer to this model unless otherwise noted. We now compare AOFU-AP (the status quo as of Android 6.0 and above, presented in Table 3.3) and our model (Table 3.5). Across all users, our model reduced the error rate from 15.38% to 7.76%, nearly a two-fold improvement.

Mispredictions (errors) in the ML model were 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 (0.936 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 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 4.2% with the ML model. Privacy violations are likely more costly to the user than functionality loss: denied data can always be granted at a later time, but disclosed data 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 preferences 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.[8] 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 96.2%, whereas predictions with low confidence were only predicted with an average accuracy of 72%. §3.6.2 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 24. 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 capture user preferences under different contextual circumstances, it risks habituation, which may eventually reduce the reliability of the user responses.

The evaluation setup mentioned in the current section does not have a specific strategy to select the training set. It randomly splits the data set into the 5 folds and picks 4 out of 5 as the training set. In a real-world setup, the platform needs a strategy to carefully select the training set so that the platform can learn most of the user's privacy preferences with a minimum number of prompts. The next section presents an in-depth analysis on possible ways to reduce the number of

[8]To calculate the class probabilities, we used the KSVM library in R. It employs a technique proposed by Platt et al. [70] to produce a numerical value for each class's probability.

prompts needed to train the ML model.

3.6 Learning Strategy

This section presents a strategy the platform can follow in the learning phase of a new user. The key objective of the learning strategy should be to learn the user's privacy preferences with minimal user involvement (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.

3.6.1 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 §3.5 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, based on our first field study data, we found that individuals' privacy preferences varied a lot [117], 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 §3.2.1. 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-APV policy. That is, this model is bootstrapped using data from other users and then adopts the AOFU-APV policy to further learn the current user's preferences. The experiment was conducted using the same set of features mentioned in §3.5.1 ($R2 + A1 + A2$ and an SVM with a RBF kernel). The test set only contained prompt responses collected after the last AOFU prompt to ensure chronological consistency.

Figure 3.3 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 12. We chose this upper bound

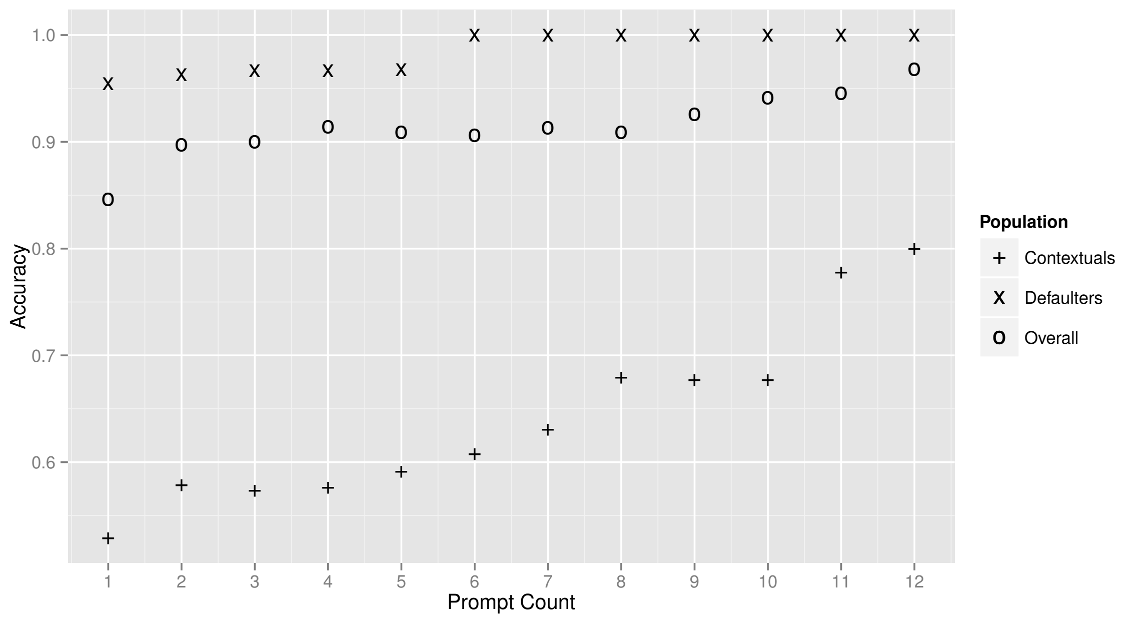


Figure 3.3: How the median accuracy varies with the number of seen prompts

because, on average, a participant saw 12 different unique *application:permission* combinations during the study period—the current permission model in Android. AOFU relies on user prompts for each new combination. The proposed ML model, however, has the advantage of leveraging data collected from other users to predict a combination not seen by the user; it can significantly reduce user involvement in the learning phase. After 12 prompts, accuracy reached 96.8% across all users.

Each new user starts off with a single model shared by all new users and then moves onto a separate model trained with AOFU prompt responses. We 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 3.3 shows that our model trained with AOFU permission-prompt responses outperforms AOFU from the very beginning. The model starts off with 96.6% accuracy (before it reaches close to 100% after 6 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 12 permission prompts, same as needed for AOFU-AP, the new approach had reduced the error rate by 43% over AOFU-AP (the status quo). The number of prompts needed to reach this level of accuracy in the new approach is 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. Overall, the proposed learning strategy reduced the error rate by 80% after 12 user prompts over AOFU-AP. Given, *Defaulters* plateau early in their learning cycle (after only 6 prompts), the proposed learning strategy, on average, needs 9 prompts to reach its maximum capacity, which is a 25% reduction in user involvement over AOFU-AP.

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 decisions based on different contextual cues and require more user involvement for the model to learn the cues used by each user and how do they affect their decisions. 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 prompts.

Our new hybrid approach of using AOFU-style permission prompts in the bootstrapping phase to train our model can achieve higher accuracy than AOFU, with significantly fewer prompts. Having a learning strategy (use of AOFU) over random selection helped to minimize user involvement (24 vs. 9) while significantly reducing the error rate (7.6% vs. 3.2%) over a random selection of the training set.

3.6.2 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-to-predict users and reduce permission prompting for those users.

While running the experiment in §4.2.1, 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 $>$96% accuracy for our fully-trained model ($\approx$25 prompts per user) for high-confidence decisions, but this is a tunable threshold. Thus, in the remainder of our analysis in this chapter, 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 10 users (7.63% of 131 users) had at least one decision predicted with low confidence. The remaining 92.37% 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 17.63% (median = 16.67%) out of all their predicted decisions. With a sensitive permission request once every 15 seconds [117], prompting even for 17.63% of predictions is not practical. Users who had low-confidence predictions had a median accuracy of 60.17%, compared to 98% accuracy for the remaining set of users with only high-confidence predictions. Out of the 10 users who had low-confidence predictions, there were no *Defaulters*. This further supports the observation in Figure 3.3 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 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 user 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 accuracy could be a better option for that specific user, to avoid excessive prompting. However, depending on which group the user is in (*Contextual* or *De-*

faulter), 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 participants with no low-confidence predictions, we had a median error rate of 2% (using the new hybrid approach after 12 AOFU prompts); for the same set of users, AOFU could only reach a median error rate of 13.3%. 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 9 prompts on average (*Defaulters* require far fewer prompts, dropping the average); the model only requires 60% of the prompts that AOFU requires. Even with far fewer prompts in the learning phase, the ML model achieves a 84.61% reduction in error rate relative to AOFU.

While our model may not perform well for all users, it does seem to work quite well for the majority of users (92.37% 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.

3.6.3 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) it distributes the overhead of training increasing the practicality of locally training the classifier on the phone itself.

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 KB to store. With our current model, each feature and weight vector are at most 3 KB each, resulting in at most 6 KB 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, and any features sent to the server are *scaled*; they are reported in standard deviations from the mean, not in raw values. Furthermore, using an online model with incremental training allows us to periodically train the model on the phone (i.e., nightly, when the user is charging her device) to eliminate the need for a trusted server.

3.7 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. 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 §3.5.3 based on the contextual integrity framework proposed by Barth et al. [20].

Contextual integrity provides a conceptual framework to better understand how users make privacy decisions; we use Barth et al.'s formalized model [20] as a framework in which to view 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 are combinations 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. The relationship between the agent sending the information and the role of the agent ((p, r)) receiving the information must follow these contextual norms.

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), 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 data), 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 difficult for 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 is likely an intractable problem. For our purposes, however, 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 most useful for this purpose, from our collected dataset.

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 provide 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 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 3.3, 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 contextual integrity framework suggests, the permission model has to take both the role and the current context into account before making an accurate decision. AOFU (and other models that neglect context) 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 better protect user privacy, more work is needed along these lines to fully understand how people use context to make decisions in the Android permission model. Nevertheless, we believe we contribute a significant initial step towards applying contextual integrity to improve smartphone privacy by dynamically regulating permissions.

3.8 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) that 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.

3.8.1 Limitations of Permission Models

Our field study confirms that users care about their privacy and are wary of permission requests that violate their expectations. We observed that 95% of participants chose to block at least one permission request; in fact, the average denial rate was 60%—a staggering amount given that the AOI model permits all permission requests for an installed application.

While AOFU improves over the AOI model, it still violates user privacy around one in seven times, as users deviate from their initial responses to permission requests. This amount is significant because of the high frequency of sensitive permission requests: a 15% error rate yields thousands of privacy violations per user—based on the latest dataset, this amounts to a potential privacy violation every minute. It further shows that AOFU's correctness assumption—that users make binary decisions based only on the *application:permission* combination—is incorrect. Users take a richer space of information into account when making decisions about permission requests.

3.8.2 Our ML-Based Model

We 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 surrounding 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 97% accuracy is a substantial improvement over AOFU's 85% and AOI's 25%; the latter two of which comprise the *status quo* in the Android ecosystem.

Our research shows 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 what the foreground application the user is engaged with. 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.

3.8.3 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 an 81% reduction in prediction errors while reducing user involvement by 25%. The significance of this observation is that by reducing the risk of habituation, it increases reliability when user input is needed.

3.8.4 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.

3.8.5 Attacking the ML Model

Attacking the ML model to get access to users' data without prompting is a legitimate concern [17, 61, 112]. There are multiple ways an adversary can influence the proposed permission model: (i) imposing an adversarial ML environment [76]; (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 manipulating 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 a scenario exists where the adversary is able to access sensitive resources without prompting the user first. Our preliminary analysis suggests that such attacks may be non-trivial, but more work is needed to study and prevent such attacks, particularly examining adversarial ML techniques and feature brittleness.

3.8.6 Experimental Caveat

We repeat a caveat about our experimental data: users were free to deny permissions without any consequences. 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. In fact, researchers have noted that many applications crash when permissions are denied [44]. If these consequences are imposed on users, they may decide that the functionality is more important than their privacy decision.

If we actually denied permissions, users' decisions may skew towards a decreased denial rate. The denial rates in our experiments therefore represent the actual privacy preferences of users and their *expectations* of reasonable application behavior—not the result of choosing between application functionality and privacy. We believe that how people react when choosing between functionality and privacy preferences is an important research question beyond the scope of this paper. Such a change, however, will not limit this contribution, since our proposed model was effective in guarding resources of the users who are selective in their decision making—the proposed classifier reduced the error rate of *Contextuals* by 44%.

We believe that there are important unanswered questions about how to solve the technical hurdles surrounding enforcing restrictive preferences with minimal usability issues. As a first step towards building a platform that does not force users to choose between their privacy preferences and required functionality, we must develop an environment where permissions appear—to the application—to be allowed, but in reality only spurious or artificial data is provided.

3.8.7 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*. While visibility was an important factor in this division, there may be others that are significant and relevant. More work needs to be done to explore how *Contextuals* make decisions and which behaviors correlate with their decisions.

3.8.8 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 examination. While we do not expect any system to be able to obtain perfect accuracy, we do expect that our 97% 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. This gives users recourse to correct undesirable decisions. 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 to change it accordingly. A user may also use this to adjust their settings as their privacy preferences evolve over time.

3.8.9 The Cost of Greater Control

A more restrictive platform means users will have greater control over the data being shared with third parties. Applications that generate revenue based on user data, however, could be cut off from their primary revenue source. Such an effect could disrupt the current eco-system and force app developers to degrade app functionality based on the availability of the data. We believe the current eco-system is unfairly biased against users and tighter control will make the user an equal stakeholder. While more work is needed to understand the effects of a more restrictive platform, we believe it is imperative to let the user have greater control over their own data.

3.8.10 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. Users may allow permissions some times, while denying them at others, implying 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 had a comparable error rate to AOFU and benefited from not requiring any user prompting. Another of our models required some user prompts—less than is required by AOFU—and achieved a reduction of AOFU's error rate by 81%.

Chapter 4

Implementation

In the previous chapter we trained an offline classifier by using participants' responses to runtime prompts. While we demonstrated that the ML approach holds promise: the training data was solely based on participants' stated privacy preferences, without considering the impact it might have on app functionality. That is, if participants state they would like to deny an unexpected permission request, but then discover that permission denial impacts app functionality, they may wish to reconsider that decision and grant the permission. Thus, the realtime classifier approach requires validation through real-world usage, which is the basis for this chapter.

We implemented and evaluated the usability of a novel mobile privacy management system that builds heavily on prior prediction work. To resolve the long-standing challenges of mobile privacy management, in the previous chapter we proposed applying machine-learning (ML) to dynamically manage app permissions, whereas Tsai et al. [111] proposed a user interface design for that system. Both proposals applied Nissenbaum's theory of Privacy as Contextual Integrity by enabling users to vary their privacy preferences based on contextual factors [83], but neither has been heretofore implemented and evaluated on real users *in situ*. We implemented these systems on the Android platform and performed a field study to evaluate their effectiveness at aligning app privacy behaviors with users' expectations. The machine-learning (ML) model runs entirely on the device and uses infrequent user prompts to retrain and improve its accuracy over time. When the ML model makes a mistake, the user interface is available to support the user in reviewing and modifying privacy decisions, thereby retraining the ML.

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

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

- We implemented the first contextually-aware permission system that performs permission denial dynamically, which is an advancement over prior work that only performed offline learning or did not regulate permissions in realtime.
- We show that AOFU not only lacks context, but it also fails to match users' privacy expectations 25% of the time, substantially hampering its ability to protect user data.
- We show opportunities and practical limitations for future work on more usable mobile app permission systems.

4.1 Related Work

Substantial previous work has shown the ineffectiveness of mobile phone permission systems. For ask-on-install (AOI) prompts, earlier investigations showed that users frequently did not pay attention to prompts or comprehend them. Users also failed to understand the resources being protected [49, 55, 65, 116]. This lack of understanding hinders users' ability to address potential risks that arise from allowing access to sensitive resources; some users were even found wanting retribution when the possible risks were revealed [48]. Another critical flaw with the AOI model is that the user lacks any sense of context about how applications might exercise the permissions granted to them. For example, users were surprised to learn that applications can continue to access those resources even when not being actively used [64, 108].

Prior research has used taint analysis and other information flow tracking techniques to understand how applications use sensitive data in the wild [43, 53, 67]. While these techniques shed light on how applications access and share sensitive data, none gave users a mechanism to indicate their preferences regarding the access of sensitive data. Other approaches did involve users, but those efforts required such a high degree of manual involvement as to overwhelm the average user and risk habituation [5, 60, 100].

Nissenbaum's theory of "contextual integrity" suggests that permission models should focus not on sensitive resources, but rather on *information flows*—from source to destination—that are likely to defy the user's expectations [82]. In an attempt to systematize Nissenbaum's theory, Barth et al. [20] extended the theory to smartphones. They suggest that it is important to consider the context in which the resource request is made, the role played by the requesting app under the current context, and the type of resource being accessed. To the best of our knowledge, we are the first to perform a field study to understand how users perceive sensitive resource usage by apps in the wild. We found that users consider the visibility of the requesting application in deciding whether a particular information flow is acceptable.

Machine learning has recently gained traction as a promising approach to predict user privacy decisions. Machine learning can significantly reduce the user's involvement in decision-making and therefore reduce *habituation*—the problem where users see so many notifications that they become desensitized to future requests, thereby making poor decisions. Previous work in this direction developed techniques to cluster users [72, 73, 97] and built recommender systems [125]. Liu et al. clustered users by privacy preferences, then subsequently predicted user preferences to applications' future permission requests using the inferred cluster [74]. The authors developed a privacy assistant to recommend privacy settings users should adopt based on their inferred cluster. The biggest drawback in these works, however, is their lack of consideration for the rich signals that context provides, which we found to be a significant factor in decision making [117, 118].

Access Control Gadgets (ACGs) are a proposed mechanism to more closely associate sensitive resource accesses to particular UI elements [78, 94, 95], so users are more aware when those accesses occur. Examples of this are the "file chooser" and "photo picker" widgets. While such an approach helps the users be better aware of resource accesses, it has two main limitations. First, applications are known to legitimately access resources when the user is not actually using the device, and therefore the user cannot receive any visual cues. Second, based on our data in the first field study, the frequency of permission requests made by smartphone apps makes systematic use of ACGs impractical [117].

After Android's switch to the AOFU permission model, research followed that investigates how users perceive it. Bonn et al. looked into understanding what motivates users to (i) install an app, (ii) allow or deny an AOFU prompt, and (iii) uninstall an app [25]. Other works investigate the consistency of user decisions across application categories under this permission model [7, 8]. Closely related are two works that proposed using contextual cues to better predict and protect user data [84, 118]. In both of these works, contextual cues are used to build a machine-learning-based permission model to increase the accuracy of the system as compared to the default Android permission model. However, Olejnik et al. [84] only focused on a selected set of applications and resources.

4.2 Implementation

We implemented a complete ML pipeline that includes: mechanisms to allow users to review and redress their decisions based on Tsai et al. [111]; ways to mask resource denial from apps so that apps continue to run, even when permissions are denied (unless those permissions were critical to their functionality); and finally, a classifier that takes surrounding contextual signals to predict user preferences for each permission request. This means our usability study is a more accurate assessment of how the system behaves in the wild than the previous investigations, which relied on user expectations rather than consequential privacy decisions.

4.2.1 A Local SVM Classifier

In the previous chapter, we implemented an offline model and suggested this could be deployed as a remote web-accessible service in order to shift compute costs from the mobile device to a more powerful dedicated server [118]. We note, however, that this design requires sending privacy-relevant data beyond the smartphone, which creates a larger attack surface and increases system costs and complexity. It also creates significant security risks if the server responsible for making decisions is compromised or is trained with spurious data.

To mitigate these security and privacy issues, we implemented and integrated the full SVM classifier into the Android operating system as a system service. We ported the open-source implementation of *libsvm* to Android 6.0.1 (Marshmallow) [30], and built two additional system-level services to interface with the SVM: the *SVMTrainManager*, which trains the model using user-provided privacy preferences through prompts (See Figure 4.1); and the *PermissionService*, which uses the SVM to regulate applications accessing permission-protected resources and issues a prompt for the user for cases when the model produces low-confidence predictions. The *SVMTrainManager* notifies the *PermissionService* when the model is trained and ready for use. These two new services are implemented into the core Android operating system, and neither are accessible to third-party apps. On average, model training takes less than 5 seconds. We instrumented all Android control flows responsible for sharing sensitive permission-protected data types to pass through this new pipeline.

Bootstrapping

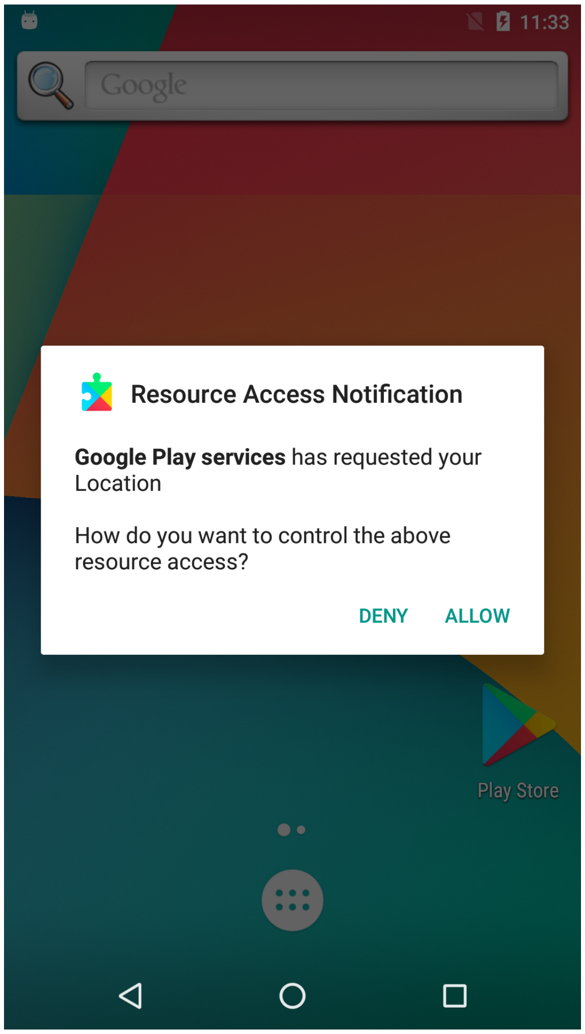
We deployed our trainable permission system along with a generic model that was pre-trained with the real-world permission decisions of 131 users, collected from our previous chapter [118]. This ensured that a new user has an initial model for making privacy decisions. This initial model, however, is inadequate for accurately predicting any particular individual user's preferences, because it simply has no knowledge of that particular user. Despite that, we, in the previous chapter, showed that our model only needs 12 additional user-provided permission decisions before the model attains peak accuracy. Given this, our system requires that the user make 12 decisions early on to train the initial model to that particular user's preferences.

The initial 12 decisions are selected based on weighted reservoir sampling. We weigh the combination of *application:permission:visibility*[1] by the frequency that these are observed; the most-frequent combinations are the likeliest to produce a permission request prompt (Figure 4.1). The intuition behind this strategy is to focus more on the frequently occurring permission requests over rarer ones. We used these same prompts for validating our classifier during the field study.

Feature Set

Our model considers the name of the application requesting the permission, the application in the foreground at the time of the request (if different than the application making the request), the requested permission type (e.g., Location, Camera, Contacts), and the visibility of the application making the request. In a pilot study (discussed later), our system implemented the full feature set described in the

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


Resource Access Notification

Google Play services has requested your Location

How do you want to control the above resource access?

Figure 4.1: A screenshot of a permission request prompt.

previous chapter. This design, however, resulted in a noticeable reduction in device responsiveness as reported by multiple study participants. We subsequently removed the “time of request” feature for the second phase of our study. The removal of the time feature from the ML enabled the platform to cache higher number of ML decisions saving the overhead stemming from running the ML for each different permission request.

4.2.2 Sensitive Resources

Previous work by Felt et al. argued that certain permissions should be presented as runtime prompts, as those permissions guard sensitive resources whose use cases typically impart contextual cues indicating why an app would need that resource [47]. Beginning with Android 6.0 (Marshmallow), the OS designated certain permissions as “dangerous” [54], and prompts the user to grant or deny permission when an app tries to use it for the first time. The user’s response to this prompt then carries forward to all future uses of that resource by the requesting application.

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

We extended the frameworks we used in the last two chapters to monitor and regulate all attempts by apps to resources protected by any of the 24 permissions we monitored. We avoid false positives by monitoring both the requested permission and the returned data type.

4.2.3 Permission Denial

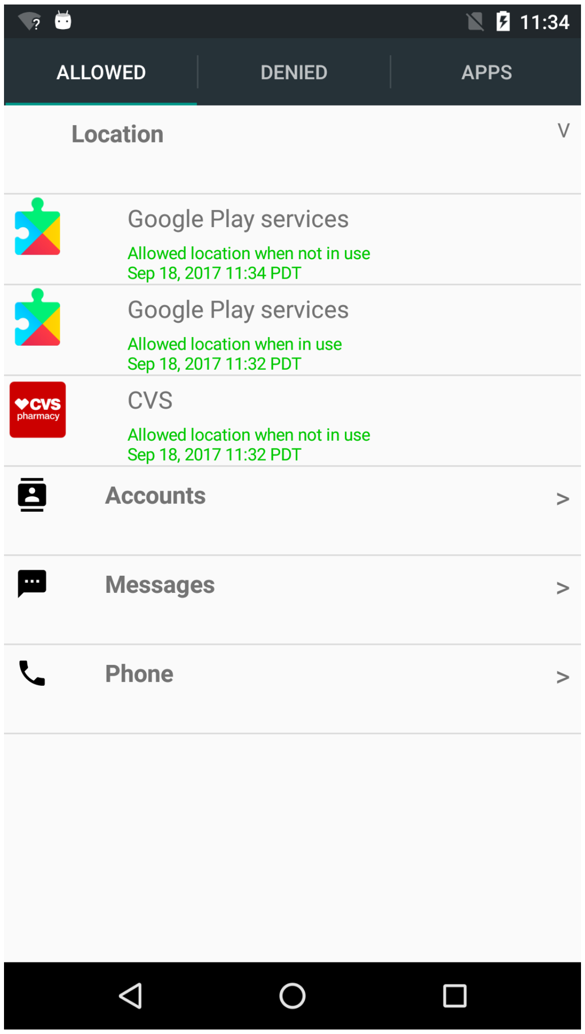
Making changes to the permission system carries the risk of app instability, as apps may not expect to have their resource requests denied [44]. If denying permissions results in frequent crashes, then users are likely to become more permissive simply to improve app stability. We therefore designed our implementation with this concern in mind: rather than simply withholding sensitive information in the event of a denied permission, our system supplies apps with well-formed but otherwise non-private "spoofed" data. This enables apps to continue functioning usefully unless access to the permission-protected resource is critical to the app's correct behavior.

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

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

4.2.4 Contextually Aware Permission Manager

We recognize that our classifier is bound to make mistakes. Therefore, it is crucial to provide a mechanism for users to review and amend decisions made by the permission model on their behalf. Mobile operating systems have configuration panels to manage app permissions, but these fail to provide users key information or options to make informed decisions. However, recent work by Tsai et al. [111] proposed a new interface to solve this problem. The authors evaluated


11:34

Location

Google Play services
Allowed location when not in use
Sep 18, 2017 11:34 PDT

Google Play services
Allowed location when in use
Sep 18, 2017 11:32 PDT

CVS
Allowed location when not in use
Sep 18, 2017 11:32 PDT

Accounts >

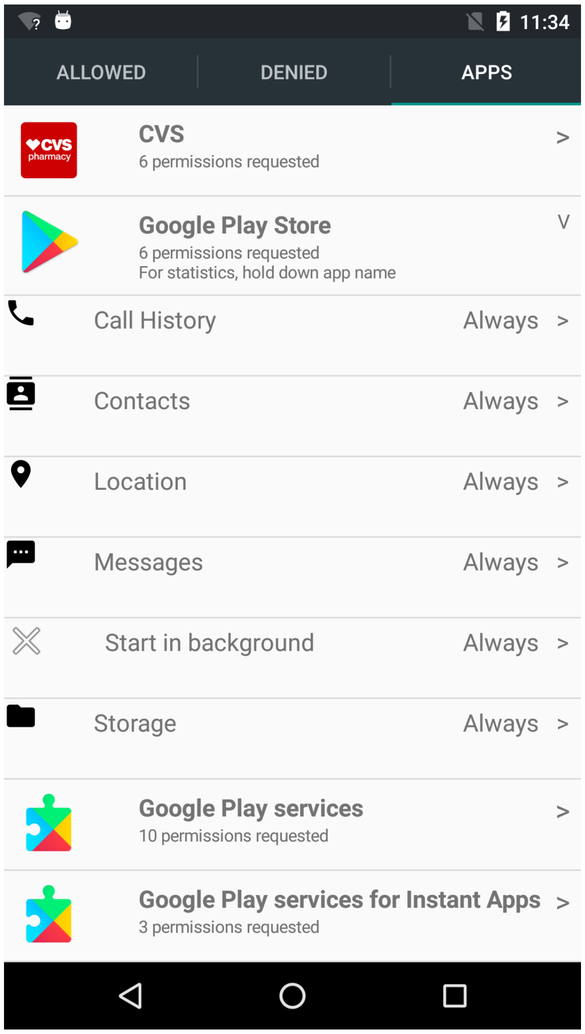
Messages >

Phone >

Figure 4.2: The recent-allowed app activity

their design using interactive online mock-ups and found that the design significantly improved user experience over the stock configuration panel. We followed the authors' recommendations in the design of our own permission manager implementation.

We built our contextual permission manager as a system-space app, similar



CVS
6 permissions requested

Google Play Store
6 permissions requested
For statistics, hold down app name

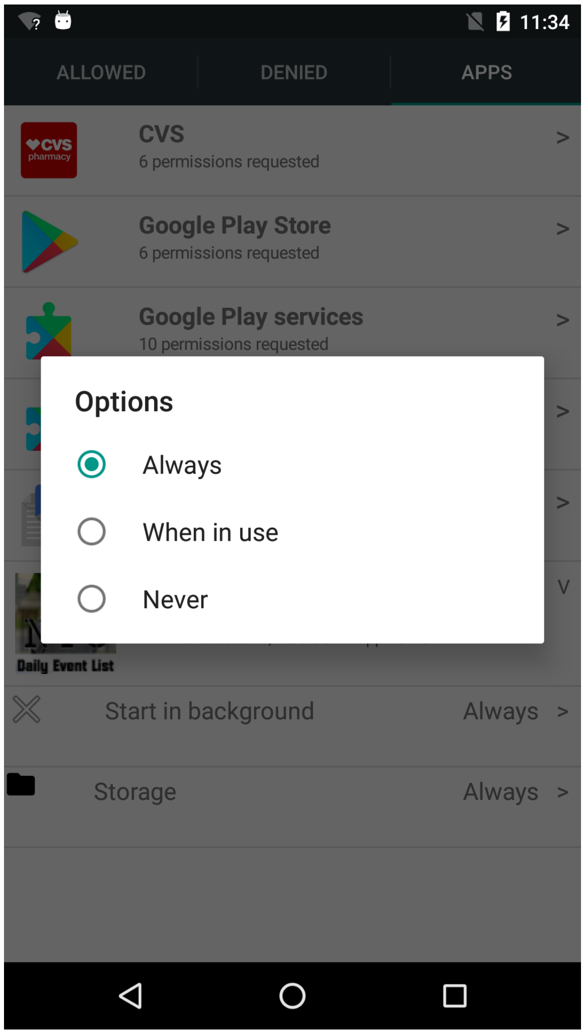
Call History	Always
Contacts	Always
Location	Always
Messages	Always
Start in background	Always
Storage	Always

Google Play services
10 permissions requested

Google Play services for Instant Apps
3 permissions requested

Figure 4.3: A list of installed apps and their associated permissions

to Android's *Settings* app (Figure 4.2, Figure 4.3, Figure 4.4). Our permission manager has three main objectives: (i) to display all recent permission requests and the corresponding "allow" or "deny" decisions from the ML model; (ii) to allow users to review and change app permissions; and (iii) to display all the resources an app can access.



Options

[✓ Always]

When in use

Never

Start in background	Always
Storage	Always

Figure 4.4: Permissions can be *always* granted, granted only *when in use*, or *never* granted (bottom).

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

4.3 Validation Methodology

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

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

4.3.1 Participant's Privacy Preferences

We used the Experience Sampling Method (ESM) to understand how participants want to control certain sensitive resource accesses [59]. ESM involves repeatedly questioning participants *in situ* about a recently observed event; in our case, the event is an app requesting access to a sensitive resource. We probabilistically asked them about an application's recent request to access to data on their phone, and how they want to control future similar requests (Figure 4.1). We treated participants' responses to these ESM prompts as our main dependent variable, which we used to validate the accuracy of the decisions that the classifier was automatically making.

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

4.3.2 Recruitment

We recruited participants in two phases: a pilot in May 2017 and the full study in August 2017. We placed a recruitment ad on Craigslist under "et cetera jobs" and "domestic gigs."[2] The title of the advertisement was "Smartphone Research Study," and it stated that the study was about how people interact with their smartphones. We made no mention of security or privacy. Interested participants downloaded a screening app from the Google Play store, which asked for demographic information and and collected their smartphone make and model. We screened out applicants who were under 18 years of age or used CDMA providers, since our experimental phones were only GSM-compatible. We collected data on participants' installed apps, so that we could pre-install free apps prior to them visiting our laboratory. (We only encountered paid apps for a few participants, and those apps were installed once we setup their Google account on the test phone.)

We scheduled a time with participants who met the screening requirements to do the initial setup. Overall, 63 people showed up to our laboratory, and of those, 61 qualified (2 were rejected because our screening application did not identify some CDMA carriers). The initial setup took roughly 30 minutes and involved transferring their SIM cards, helping them set up their Google and other accounts, and making sure they had all the applications they used. We compensated each participant with a $35 gift card for showing up.

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

[2]Approved by the UC Berkeley IRB under protocol #2013-02-4992

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

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

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

4.3.3 Exit Interview

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

- **New Permission Manager UI**—We asked participants to show us how they would use the UI (Figure 4.2, Figure 4.3, Figure 4.4) to block a given application from accessing background location data, as well as how difficult they found it. We also checked our data to see how they interacted with the UI during the study period, and asked them about the circumstances for those interactions. The objective of this task was to validate the design objectives of the UI, including whether they use it to resolve issues stemming from resource denial.

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

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

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

4.4 Results

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

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

4.4.1 Status Quo Problems

In the "ask-on-first-use" (AOFU) model, the user receives prompts to grant individual permissions, but only the first time an app requests them. Requesting these permissions at runtime allows the user to infer the potential reason for the request, based on what they were doing when the request occurred (i.e., context). AOFU's shortcoming, however, is that it naïvely reapplies the user's first-use decision in subsequent scenarios, without adapting to different contexts. In the previous chapter (Sec 3.4), we show that failing to account for changing contexts produces high error rates.

We note that, in the previous study, we attempted to measure the accuracy of the AOFU model by merely collecting users' responses to runtime permission prompts, without actually enforcing them by denying apps access to data [118]. Thus, the accuracy rates reported by that study may not actually be accurate, since users may elect to change their permission-granting preferences, if they result in a loss of application functionality. Thus, we evaluated the performance of the AOFU approach (in current use by Android and iOS) by presenting participants with permission prompts that *actually* resulted in the denial of application permissions.

During the study period, each participant responded to combinations of *application:permission* more than once. As AOFU is deterministic, we can simulate it by comparing a user's first response to a *application:permission* combination to future responses to the prompts for the same app and permission. We use this data to measure how often AOFU matches the user's preference in subsequent requests.

Our data show that the AOFU permission model has a median error rate[3] of 20%: in more than one-fifth of app requests for permission-protected resources, participants changed their initial response for the same *application:permission* combination. Of the 37 participants, 64% had at least one such discrepancy between the first-use and subsequent preferences. This refutes AOFU's core assumption that only few users will deviate from their initial preferences in future cases. This observation corroborates our previous data. [118], in which 79% of 131 participants were shown to deviate from their initial responses in subsequent cases.

The errors shown in AOFU, could be either privacy violations or losses of functionality. A privacy violation occurs when the system grants an app access to a protected resource, contrary to the user's preference, had she been prompted. Loss of functionality occurs when the permission system denies access to a protected resource, which the user would have otherwise permitted. We consider privacy violations to be the more severe type of error, as the user is unable to

[3]The median value is reported because the error rate is not normally distributed among participants.

take back sensitive information once an app has acquired it and transmitted it to a remote server. However, loss of functionality is still undesirable because those errors might incentivize the user to be overly permissive in order to regain that functionality. From our data, we found that 66.67% of AOFU errors were privacy violations; the remaining 33.33% were losses in functionality.

AOFU User Expectations

Errors in permission systems could arise from a variety of reasons. Mismatched user expectations and lack of comprehension are two critical ones, which could hamper any permission model's utility. User comprehension is critical because users may make suboptimal decisions when they do not fully understand permission prompts, hindering the ability of the permission system to protect sensitive system resources. Users must be able to comprehend the decision they are making and the consequences of their choices. Recent work on AOFU has examined the motives behind users' decisions and how it varies between categories of applications, as well as how people adapt their behavior to the new model [7, 8, 25].

In our study, the participants had, on average, 5 years of experience with Android. This indicates that most of our participants have experienced both install-time permissions—the permission model prior to Android 6.0, released in 2015—and runtime "ask-on-first-use" permission prompts. The majority of participants said they noticed the shift to AOFU prompts, and they were aware that these prompts are a way to ask the user for consent to share data with an app. A large minority of participants (≈40%), however, had an inadequate understanding of how AOFU works, which could substantially hinder that permission model's effectiveness in protecting user data.

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

> *"When I see prompts, I feel like they want to know something about*

me, not that they want to protect anything." (P21)

One *possible* explanation is that some users grew accustomed to install-time prompts, and subsequently perceived the change to runtime prompts as a new way for Android to collect user data. Although it is impractical to project how prevalent this sentiment is in the general population, we cannot reject its existence. Hence, more work is needed to measure its impact and explore the potential solutions.

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

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

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

4.4.2 Classifier Accuracy

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

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

Taken as a whole, these 31 participants responded to 640 total prompts in the validation phase. Our contextual permission model produced a median accuracy of 90%, compared to 80% under AOFU for the same population. The classifier reduced AOFU's error rate by 50%, with the majority of classifier errors consisting of privacy violations (i.e., access granted when the user would have denied it).

Offline Learning

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

The offline model had a median accuracy of 94.74% across the 37 participants. By comparison, AOFU had a 80% accuracy for the same population. This represents a 75% error reduction in the offline contextual model compared to AOFU. These numbers corroborate prior findings [118]. We stress the significance of this corroboration, because the results hold in the presence of actual resource denial, which was not examined in the prior study. This suggests that users will continue to indicate their true preferences in response to prompts, even when those preferences are enforced, potentially resulting in unanticipated app behavior.

We note the accuracy difference between the SVM classifier we integrated into Android and the *R* model (90% vs. 94.74%, respectively). This is due to how the Android SVM implementation performs the bootstrapping. This issue is not inherent to integrating an SVM classifier into Android. An updated implementation has the potential to reach the maximum accuracy observed in the offline model.

Decision Confidence

In the previous chapter we demonstrated how to use decision confidence to determine for which *application:permission:visibility* combinations users should be prompted in the validation phase(Sec 3.6.2). The rate of decision confidence is also a measure of the extent to which the classifier has learned the user's preferences. The authors suggested that if this rate does not decrease over time, then AOFU will likely be a better system for those users.

In addition to the prediction, our classifier also produced a class probability, which we used as the measure of decision confidence. The classifier produced a binary result (i.e., allow or deny) with a cutoff point of 0.5. A decision probability close to the cutoff point is a less confident result than one far from it. We used the 95% confidence interval as a threshold to determine which decisions were low-confidence and which ones were not.

Only 4 of our field study participants experienced low-confidence classifier decisions that caused a prompt to appear after the bootstrapping period. Each of these participants had just one such low-confidence prompt appear. These prompts retrained the classifier, so the lack of any subsequent low-confidence prompts indicates that the classifier produced high-confidence predictions for the same *application:permission:visibility* combination in future cases.

The lack of additional training prompts also suggests that users are less likely to become habituated to prompting. The 4 participants who each received one additional prompt saw a total of 13 prompts (including the 12 prompts during the training phase). The remaining 27 participants saw just the 12 training phase prompts. Had our participants been subject to AOFU instead of our contextual permission system, they would have received a median of 15 prompts each, with a quarter of the participants receiving more than 17. Instead, we achieved a 75% error reduction (80% vs. 94.74%) and reduced user involvement by 20% (12 prompts vs. 15) through the use of classifier-driven permissions, compared to AOFU.

4.4.3 Impact on App Functionality (and Stability)

Previous research has shown that many applications do not properly handle cases where they are denied permission to access a protected resource [44]. One core objective of our work was to measure how apps responded to a stricter permission model than AOFU. For example, the system will be unusable if it causes erratic application behavior, through the use of dynamically granted permissions.

In the field study, our platform instrumentation recorded each application crash and its corresponding exception message. This information allowed us to identify the *possible* root cause of the crash and whether it was related to resource denial.

We observed 18 different exceptions classes, such as `SecurityException`, `RuntimeException`, and `NullPointerException`. For the remainder of this section, we will only discuss `SecurityExceptions`, as this class is directly related to resource denials. Almost all (98.96%) of the recorded `SecurityExceptions` were observed on the devices of just two participants. Each of the remaining participants encountered, on average, 18 SecurityExceptions during the study period (i.e., roughly 3 SecurityExceptions per day per participant).

Almost all (99.93%) SecurityExceptions were caused when apps attempted to read subscriber information (i.e., the READ_PHONE_STATE permission, used to obtain the phone number). In the event of a READ_PHONE_STATE denial, we designed our implementation to not supply the app with any phone number data. We had considered supplying a randomly-generated phone number, but decided against it due to potential risks, if the generated number were a valid phone number belonging to someone else.

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

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

4.4.4 User Reactions to Prompts

The use of runtime prompts was initially proposed as a mechanism to obtain better-informed consent from users. At the end of the study period, we conducted exit interviews with each participant in order to determine the extent to which these assumptions were met.

We measured how much participants were surprised to see the prompts during the course of the study period (on a scale of 1="not surprised" to 5="very surprised”). Participants expressed an average rating of 2.7. Almost half (44%) of the participants indicated that the prompts surprised them, and among them, 70% were surprised at the frequency with which the prompts appeared (up to 4 times per day), though few participants expressed annoyance by that frequency (8.33%).

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

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

From these observations, we believe that the new contextual permission system is a positive step toward improving user awareness. We believe this enables users to make better privacy decisions for themselves. Although additional work is needed to address some negative sentiments about the current implementation, this system has shown to be in the right direction overall.

4.4.5 User Reactions to Controls

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

In our implementation, we built a user interface based on prior work by Tsai et al. [111]. This system allowed our participants to view automated permissions decisions made by the classifier, as well as set privacy preferences with respect to context (i.e., the visibility of the requesting app). We included this user interface as part of the operating system, as a panel within the system settings app.

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

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

We also asked participants to demonstrate how they would change the settings of a familiar app to only be able to access their location when they are using that app. We based this task off of one of the evaluation tasks performed by Tsai et al. [111], when they performed an online study to evaluate a low-fidelity prototype of the design on which we based our user interface. All but two of our participants were able to correctly complete this task using the user interface. Participants rated the average ease of the task as 1.15 (on a scale from 1="very easy" to 5="very hard"). We conclude that participants are able to understand the permission interface after having used it for a week, and without special instructions.

The permission manager also enables users to diagnose application crashes that result from a resource denial (a feature not present in the original design on which we based it). In exit interviews, we examined how participants responded to app crashes in their experiences with the device. The majority of participants reported that their first step was to restart the app that had crashed. If that was unsuccessful, they would then restart their phone. This informs the design of a future system: if an app crashes as a result of a resource denial, the platform should clearly communicate this to users or otherwise automatically adjust the permissions on their behalf. This could be communicated through a dialog or in the notification bar.

4.5 Discussion

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

Consequential Denial

Overall, participants denied 24% of all prompted permission requests. This is a 60% reduction in denials compared to the results from the previous chapter, who did not enforce the user's decision to deny a permission and prompted the user using only hypothetical language: "given the choice, would you have denied...?" The decreased denial rate we observed is therefore unsurprising given that participants were now actually making a tradeoff between functionality and privacy, instead of expressing the degree to which privacy is important to them. Our results show that even in the presence of consequential resource denial, contextual cues helped to predict users' privacy decisions and better aligned permission settings with their expectations, as compared to the status quo.

Ask on First Use

Our results corroborate our previous two chapters in showing that AOFU's inability to capture the context surrounding users' decisions is a cause of AOFU's significant error rate. We also found that a significant portion of participants do not have an adequate understanding of how AOFU works, which further limits AOFU's utility: 11 participants did not realize that their prompt responses for AOFU are taken as permanent decisions; and 4 participants interpreted the prompts as yet another mechanism for collecting user data instead of as a privacy-protection mechanism. While the actual impact of these inaccurate beliefs is yet to be explored, we believe that these issues need to be fixed in the future, in order to increase Android's ability to predict and protect user data effectively.

Implementation Limitations

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

Purpose

While our new permission model outperforms AOFU, it still does not explain to the user *why* an app needs to use a permission. In our exit interviews, we observed that 14% of participants expressed the desire to know *why* apps made a request in the first place. Previous work has shown that app functionality is a key factor in permission decisions [25]. If users were properly informed of the functionality requirement behind a permission request, then they might be better positioned to make decisions that meet their privacy and functionality expectations.

We believe that there are ways to extend contextual permission systems by incorporating the actual purpose of the request. For example, after introducing AOFU permissions, Android started encouraging app developers to provide the reason behind their permission requests so that the user can include that in the decision-making process [40]. Tan et al. [107] showed that similar prompts on iOS actually resulted in users being more permissive about granting permissions to apps. Similarly, prior work has attempted to use static analysis to automatically incorporate inferred purpose [71, 74].

Resource Denial

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

The platform, therefore, is responsible to ensure that apps handle resource denials gracefully. To their credit, when Android introduced AOFU, it implemented some permission denials to appear like a lack of available data or the non-existence of hardware, instead of throwing a `SecurityException`. In our implementation, we take the extra step of supplying apps with generic but well-formed data in the event of a denial. We observed that our participants tended to deny more permissions as they progressed through the study period (on average 20% denial in the learning phase versus a 26% denial rate during the validation phase). Those participants also experienced a low rate of app failures due to resource denials. In the future, platforms should implement measures to reduce functionality losses stemming from having stricter privacy preferences. Failing to do so might otherwise compel users to compromise on their privacy preferences for the sake of functionality.

Remedying Unexpected Behavior

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

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

Conclusion

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

Chapter 5

Discussion

The primary goal of this thesis is to develop a permission system that is capable of protecting user's private smartphone data – so that user's privacy preferences are better met compared to current systems. Reaching that goal required understanding how users make privacy decisions, understanding different circumstances under which mobile applications access resources, and devise ways a mobile platform can align platform protection with user privacy preferences.

We started the project with three hypotheses: a) currently deployed permission controls are not efficient in protecting user's private data, b) users are likely to take advantage of finer grained permission controls, and if such options are available, c)the context surrounding the permission decision will play a critical role in user's privacy decisions. To test our hypothesis, we conducted two (one-week long) field studies and one (six-week long) longitudinal field study. We have collected 6K privacy decisions using Experience Sampling – to collect their in-situ privacy preferences– from more than 200 real-world users and also collected more than 200M real-world data points on resource usage and contextual information on how participants used the phone during the study period.

More than 80% of our participants wanted to block at least one sensitive permission request during the study period showing their desire to have finer grained permission systems compared to currently deployed Android permission system.

While making privacy decisions (allowing or denying), nearly half of the participants took the context surrounding each permission request into account – the remaining half, based on the factors we measured, didn't seem to take the context into account. It could also be the case that, we are not fully aware of the actual factors remaining participants are considering.

During the period of the thesis, Android switched from Ask-on-install to Ask-on-first-use reducing the potential error rate from 75% to 15%, measured using the data collected from the three field studies. While AOFU demonstrated a significant improvement, it does not fully capture the user decision when it confronts the user to seek their consent on sharing their private data with an application. AOFU falsely assumes once a user makes a privacy decision, it lasts forever regardless of the surrounding context – this false assumption is solely responsible for the error rate of AOFU. Based on our collected dataset, 70% of participants have at least changed their initial decision in subsequent cases nullifying the AOFU core assumption.

Based on our collected privacy decisions, we designed and developed a contextually aware permission system that reduces the user confrontation and significantly increases the protection by taking the surrounding context into account. In the new permission system, once a user makes a privacy decision, the context in which the user made the decision is taken into account when it reuses the user decision in subsequent cases. The new permission model has the potential to have an 80% lower error rate compared to AOFU. The new permission model was first tested as an offline simulation, and then in the real world, interacting with real-time user decisions and with data-hungry applications. While achieving the increased protection and reduced user involvement, it also preserves the app functionality to a greater extent.

5.1 Mismatched Personas

The threat model of the new permission system is not to protect sensitive data from malicious applications that exploit security vulnerabilities of the mobile platform.

It is to protect sensitive data from benign applications that access sensitive data when it is not expected by the user but through legitimate channels implemented in Android. Majority of the unexpected requests come from benign applications with a legitimate functionality. While understanding the motives behind developer practices is beyond the scope of this work [13, 15], it is an important part of the bigger effort of building a more privacy-friendly ecosystem in mobile.

In product development, personas have been a widely used technique where developers create hypothetical characters based on scenarios to guide the design process of the application [34, 90]. While this approach has been a success in designing an application that is functionally efficient, personas can have a different impact on the privacy side of the application. The use of personas helps designers and developers design the application in a way a potential user from the targeted audience would use it. This entails making assumptions about how their potential audience usage of the applications – these assumptions, inadvertently, include how the potential user would share their data, with the outside world, in the process of using the application.

How practical is it to assume potential users sharing practices? We show in previous chapters [117, 118], each participant has a unique way of expecting data accesses which corroborates with previous work [2, 72, 103, 119]. Contextual Integrity also suggests that their expectations are based on their surroundings - none of the one-size-fits-all models worked because of this. It is impractical to envision each user's different contexts and the dependent expectations when the application is designed – which is what is done by using personas. This explains why a majority of users were surprised to see certain data accesses during the study which lead them to deny a significant portion of it.

As an application developer, it is infeasible to assume each different potential user's contextual privacy expectations under different contexts in which the application will be used. We believe the best an application developer can do is to let the user decide what to share and when to share it so that users can make their own privacy decisions and contextual preferences. Users, however, already have a plethora of decisions to make in the day to day digital world [86, 93, 105]; so we must find a way to engage the user without habituating or annoying them while getting the required privacy consent. Given the number of applications each user is using, the number of privacy decisions they have to make can quickly get out of control.

From the beginning, to their credit, mobile platforms took care of the burden of seeking user consent, removing that responsibility from the application developers. Platforms, however, were not efficient at asking the correct question at the correct time, which has the serious consequences of not giving enough context to the user for before forcing them to make a privacy decision. These issues led users to make sub-optimal privacy decisions risking the protection of their own sensitive data.

Fixing the developer ecosystem to be more privacy-friendly is a very important, yet, beyond the scope of this work. More work is needed along the lines of helping developers to make better privacy choices while developing applications. We, however, fix the issues of helping users to make better privacy decisions so that when applications request data users can make optimal privacy decisions [1, 2].

5.2 Arms Race

Regardless of their design and development methodologies, developers were used to receive all the resource they requested – in earlier Android versions users were forced to make an all-or-nothing choice (Ask-on-install), so developers could safely assume that an installed application is allowed to access all requested resources. Android 6.0 introduced Ask-on-first-use, giving users finer grained access control – users could just only allow the data types they were comfortable with sharing with the app while denying the rest. Previous work, however, showed that applications are unlikely to take resource denial gracefully, leaving the possibility of a critical usability chaos [44]. Furthermore, work has shown that if applications start to crash, users are likely to adopt a less restrictive, more permissive privacy policy, trading privacy for the desired usability and functionality [6].

With the introduction of AOFU, Google has been warning developers to be prepared to have resource requests denied, and to gracefully handle resource denials [39]. No work recent work suggests developers are adapting to the new guidelines, or otherwise. Regardless, platforms need to make sure that users are free from the burden of worrying about usability issues when they make privacy decisions. If not, the objective of a more restrictive, more user-aligned permission system is lost, much like the previous regime of permission systems.

People have looked into ways to feed carefully articulated data to prevent leaking sensitive information [21, 29, 60, 124] – none of these techniques were tested in a real-world setup. A closely related work tested on real world applications but do not report a quantitative analysis on it's impact [84]. Our last field study look into ways to feed obfuscated data in the event of a resource denial so that applications will continue serving their functionality based on allowed data without crashing. Our data suggest such obfuscated data feeding has been largely successful – we only observe a median of 2 application crashes per day per participant. compared to over 2000 resource denials observed for each user during the study period. While two crashes per day are not ideal, we believe this is a correct step towards making sure users are free to make privacy decisions. 99% of the crashes came from a data resource for which we did not have an obfuscation implemented – this shows a) without data obfuscation applications are likely to create a usability issue, and b) currently implemented data obfuscation techniques are successful to a sizable extent.

This, however, will be a short-lived success. When more users start to adopt the finer grain permission systems, applications are likely to get more obfuscated data. Depending on the nature of the applications, data incentives, and for other reasons that probably require further research to understand, applications may start to become more aggressive towards detecting obfuscated data. Therefore, it is imperative that the platform takes measures to not let the applications know about resource denial. We, however, believe this is going to be an arms race between platforms trying to protect user data and applications wanting to get the desired resources.

5.3 Purpose Matters

From the developer perspective, one way to avoid having an essential resource denied, as Google, Apple and previous work explain [37, 40, 107], is to provide the users with a reason for the access request. Tan et al. showed that users are more likely to be permissive if a reason for the request is present [107]. During our first field study [117], participants comprehension of the resource's necessity for desired functionality of the app played a major role in their decision to allow or deny a resource request; a more recent work corroborates these findings [25]. Furthermore, some of our participants of the last study also mentioned the need to know the actual reason for a permission request before making a privacy decision. Participants could be making decisions based on ill-informed perceptions i.e., they could be denying a request that could actually be needed for the core functionality of the application, assuming they are not related or vice versa.

Previous research has looked into inferring the purpose of a given request based on crowdsourcing [71], static analysis, and dynamic analyis [69, 115]. In previous work, authors were relying on the previously collected data sets, labels and machine learning techniques to figure out the purpose of each request. Liu et. al. looked into incorporating the request's purpose when informing the users about sensitive access requests [74]. The reliance of previously accumulated data could be a problem since a) applications are likely to update their code regularly, which could make historical data obsolete, and b) techniques such as crowdsourcing could be subjective to the participants. The new permission model presented, however, in this thesis does not take the (inferred or actual) purposed into account when making the decisions.

From the point of view of the users, the reason for a request could be a binary value indicating whether the request is related to the core app code or whether it is a third-party library. Third party libraries usually serve nonfunctional needs such as analytics, ads, monetization. Previous work has already looked into runtime library detection based on stack analysis [31, 115]. It would be an interesting exploration to examine how users react to permission requests when they have information about the actual source of the request – whether it is the actual code or a library. It will be an interesting and helpful question to examine how the purpose affects their contextual preferences. We believe such information can be a useful feature to add to our current contextual permission system which could further increase its accuracy.

5.4 User driven Privacy

Data privacy is a multi-faceted dilemma faced by different fronts in computer science. There is a long line of work focusing on the data leakage aspect of it, which belongs to more system side computer science [43, 53, 67, 110]. In a different vein than the permission system proposed in this thesis – which focuses on both systems and the user aspect of the problem, information flow tracking, taint analysis, provenance analysis focus on low-level information passing that occur between a known source (originating point e.g., location) and a sink (where the data flow ends e.g., a socket connection).

Taint analysis will provide fine-grained information on how exactly applications are using and/or sharing the requested data. Some of these methods incur a significant of overheard bearing it from practical usage [43]. Another important difference would be not having the user in the pipeline before deciding an information flow is inappropriate or not. This could risk false positives since not all data flows leaving the phone are privacy violations – privacy violations only occur if a flow defies user expectations. The work presented in the thesis monitors data flows from system spaces to the app space. The current instrumentation mediates before an information(a sensitive data) flow occur from system to app space.

The thesis of this work is based on having the user playing the central role in deciding what flow is appropriate and what is not. Having a data point leaves a phone does not necessarily defy user expectations. We believe it is essential for users to have the final say about what is appropriate and what is not. Previous work [41] and our own last study found that leaving users completely out of the loop might not be the best solution. During that last study, we found participants saying that they felt inferior to the phone given how applications use data and not all participants were comfortable with letting the platform make decisions for them. One way to elevate user confidence is by involving the users in the process (but at the same time not to habituate them).

The question of who is well equipped to make the decision is an interesting avenue to work on. While the platform has every bit of information, potentially, needed to make the decision, the platform doesn't have the user perspective. While the user owns the data, so far users were not able to make optimal privacy decisions to protect their data. One of the contributions of the new permission system is to empower users to make informed decisions so that they can better protect their own sensitive data. We believe this is an important distinction between the more systems-level taint-tracking system and our new permission system. Moving forward, the correct direction would be to investigate how to merge these two different approaches to better serve users.

5.5 Contextualization

One of the core hypothesis of this work is users make contextual decisions which was first put forward by the notion of Contextual Integrity [82]. Over the course of the project, we found that visibility of the requesting application and the foreground application at the time of the request play a critical role in user decisions – we believe these are two contextual factors used by users to make contextual preferences. Not only these two factors are affecting their decisions, as we show previously in the thesis (Chapter 3), users are more likely to be consistent with these different factors as demonstrated by the high predictive power of these two factors in the new predictive model.

While we identify factors that are likely to be contextual cues used by users to make privacy decisions, we do not infer the *context* – automatically inferring the current context could be an intractable problem. For the purpose of the mentioned problem statement, what is more important is the identification of tangible factors and impact of those factors towards users decision. Such knowledge will help the platform to detect when the context changes so that platform can change their decisions accordingly.

In chapter 3, based on the conceptual framework proposed by Barth et al. [20], we explain why visibility and foreground are critical contextual factors. We posit that the visibility of the requesting application could be helping the user to grasp the role played by the requesting application (requesting actor). With the recent advancement in the discovery of data sharing practices [91, 92, 102], it is plausible that the actual data recipient might not be the application requesting the data. If the data flow ends up in a third party server, then the user has an incorrect understanding of the data recipient.

One approach to extending the current permission systems would be to inform the user about potential third parties other than the requesting application that might receive the data so that the user is better informed about the actual data recipient. A recent work shows that users are likely to vary their decision if they know who is the actual data recipient is [113]. While previous work has incorporated *potential purpose* of a request using offline data [74], no one has yet to incorporate the actual third parties that might receive the data from a request. We believe by incorporating such information, the new permission system can achieve even higher accuracy giving finer-grained information to the user's decision.

We believe, that more work is needed to understand how the context impact user decisions. There could be more contextual factors than just the visibility and the foreground application. These new factors could help to further improve the efficiency of the proposed permission system.

5.6 Conclusion

We show that by accounting for the surrounding context, mobile platforms can increase the privacy protection by better aligning the platform protection with how users expect to control sensitive data if they had an option. Rather than enforcing a blanket allowance or denial – which risks both privacy violations and unwarranted functionality loss – the platform should devise ways to dynamically take decisions based on the surrounding context. Dynamically controlling the access entails learning user privacy preferences under different circumstances; we show that use of machine learning not only helps to understand how users want to react under different contextual circumstances, but it also, helps to reduce the user involvement significantly.

Bibliography

[1] A. Acquisti. Nudging Privacy: The Behavioral Economics of Personal Information. *IEEE Security and Privacy*, 7(6):82–85, 2009. http://www.computer.org/portal/web/computingnow/1209/whatsnew/securityandprivacy. → pages

[2] A. Acquisti and J. Grossklags. Privacy and rationality in individual decision making. *IEEE Security & Privacy*, 3(1):26–33, January–February 2005. → pages

[3] A. Acquisti, L. Brandimarte, and G. Loewenstein. Privacy and human behavior in the age of information. *Science*, 347(6221):509–514, 2015. → pages

[4] Y. Agarwal and M. Hall. Protectmyprivacy: Detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 97–110, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1672-9. doi:10.1145/2462456.2464460. URL http://doi.acm.org/10.1145/2462456.2464460. → pages

[5] 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*, CODASPY '14, pages 257–264, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2278-2. doi:10.1145/2557547.2557571. URL http://doi.acm.org/10.1145/2557547.2557571. → pages

[6] H. Almuhimedi, F. Schaub, N. Sadeh, I. Adjerid, A. Acquisti, J. Gluck, L. F. Cranor, and Y. Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proc. of the 33rd Annual*

ACM Conference on Human Factors in Computing Systems, pages 787–796. ACM, 2015. → pages

[7] P. Andriotis, M. A. Sasse, and G. Stringhini. Permissions snapshots: Assessing users' adaptation to the android runtime permission model. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec 2016. doi:10.1109/WIFS.2016.7823922. → pages

[8] P. Andriotis, S. Li, T. Spyridopoulos, and G. Stringhini. *A Comparative Study of Android Users' Privacy Preferences Under the Runtime Permission Model*, pages 604–622. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-58460-7_42. → pages

[9] Android Developers. Content Providers. http://developer.android.com/guide/topics/providers/content-providers.html, 2014. Accessed: Nov. 12, 2014. → pages

[10] Android Developers. Common Intents. https://developer.android.com/guide/components/intents-common.html, 2014. Accessed: November 12, 2014. → pages

[11] 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*, CCS '12, pages 217–228, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi:10.1145/2382196.2382222. URL http://doi.acm.org/10.1145/2382196.2382222. → pages

[12] C. Auerbach and L. B. Silverstein. *Qualitative data: An introduction to coding and analysis*. NYU press, 2003. → pages

[13] R. Balebako and L. Cranor. Improving app privacy: Nudging app developers to protect user privacy. *IEEE Security & Privacy*, 12(4):55–58, 2014. → pages

[14] R. Balebako, J. Jung, W. Lu, L. F. Cranor, and C. Nguyen. "little brothers watching you": Raising awareness of data leaks on smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 12:1–12:11, New York, NY, USA, 2013. ACM. ISBN

978-1-4503-2319-2. doi:10.1145/2501604.2501616. URL http://doi.acm.org/10.1145/2501604.2501616. → pages

[15] R. Balebako, A. Marsh, J. Lin, J. I. Hong, and L. F. Cranor. The privacy and security behaviors of smartphone app developers. 2014. → pages

[16] R. Balebako, F. Schaub, I. Adjerid, A. Acquisti, and L. Cranor. The impact of timing on the salience of smartphone app privacy notices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 63–74. ACM, 2015. → pages

[17] 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*, pages 16–25. ACM, 2006. → pages

[18] D. Barrera, H. G. u. c. Kayacik, P. C. van Oorschot, and A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. In *Proc. of the ACM Conf. on Comp. and Comm. Security*, CCS '10, pages 73–84, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0245-6. doi:http://doi.acm.org/10.1145/1866307.1866317. URL http://doi.acm.org/10.1145/1866307.1866317. → pages

[19] D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 81–92, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1666-8. doi:10.1145/2381934.2381949. URL http://doi.acm.org/10.1145/2381934.2381949. → pages

[20] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proc. of the 2006 IEEE Symposium on Security and Privacy*, SP '06, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2574-1. doi:10.1109/SP.2006.32. URL http://dx.doi.org/10.1109/SP.2006.32. → pages

[21] A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0649-2. doi:10.1145/2184489.2184500. URL http://doi.acm.org/10.1145/2184489.2184500. → pages

[22] 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*, UbiComp '13, pages 657–666, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1770-2. doi:10.1145/2493432.2493510. URL http://doi.acm.org/10.1145/2493432.2493510. → pages

[23] E. Bodden. Easily instrumenting android applications for security purposes. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '13, pages 1499–1502, NY, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi:10.1145/2508859.2516759. URL http://doi.acm.org/10.1145/2508859.2516759. → pages

[24] R. Böhme and J. Grossklags. The security cost of cheap user interaction. In *Proceedings of the 2011 New Security Paradigms Workshop*, pages 67–82. ACM, 2011. → pages

[25] B. Bonné, S. T. Peddinti, I. Bilogrevic, and N. Taft. Exploring decision making with android's runtime permission dialogs using in-context surveys. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 195–210, Santa Clara, CA, 2017. USENIX Association. ISBN 978-1-931971-39-3. URL https://www.usenix.org/conference/soups2017/technical-sessions/presentation/bonne. → pages

[26] T. D. Breaux and A. Rao. Formal analysis of privacy requirements specifications for multi-tier applications. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 14–23, July 2013. doi:10.1109/RE.2013.6636701. → pages

[27] T. Buchanan, C. Paine, A. N. Joinson, and U.-D. Reips. Development of measures of online privacy concern and protection for use on the internet.

Journal of the American Society for Information Science and Technology, 58(2):157–165, 2007. → pages

[28] S. Bugiel, S. Heuser, and A.-R. Sadeghi. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Proc. of the 22nd USENIX Security Symposium*, SEC'13, pages 131–146, Berkeley, CA, USA, 2013. USENIX Association. ISBN 978-1-931971-03-4. URL http://dl.acm.org/citation.cfm?id=2534766.2534778. → pages

[29] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Millar, and M. B. Srivastava. ipshield: A framework for enforcing context-aware privacy. In *NSDI*, pages 143–156, 2014. → pages

[30] C.-C. Chang and C.-J. Lin. Libsvm – a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Accessed: September 11, 2017. → pages

[31] S. Chitkara, N. Gothoskar, S. Harish, J. I. Hong, and Y. Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 1(3):42, 2017. → pages

[32] E. K. Choe, J. Jung, B. Lee, and K. Fisher. Nudging people away from privacy-invasive mobile apps through visual framing. In *Human-Computer Interaction–INTERACT 2013*, pages 74–91. Springer, 2013. → pages

[33] M. Conti, V. T. N. Nguyen, and B. Crispo. Crepe: Context-related policy enforcement for android. In *ISC*, volume 10, pages 331–345. Springer, 2010. → pages

[34] A. Cooper et al. *The inmates are running the asylum:[Why high-tech products drive us crazy and how to restore the sanity]*. Sams Indianapolis, IN, USA:, 2004. → pages

[35] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. Kaynar, and A. Sinha. *Understanding and Protecting Privacy: Formal Semantics and Principled Audit Mechanisms*, pages 1–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. doi:10.1007/978-3-642-25560-1_1. URL https://doi.org/10.1007/978-3-642-25560-1_1. → pages

[36] A. Developer. Requesting permissions. https://developer.android.com/guide/topics/permissions/requesting.html, . Accessed: March 18, 2017. → pages

[37] A. Developer. App programming guide for ios. https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ExpectedAppBehaviors/ExpectedAppBehaviors.html#//apple_ref/doc/uid/TP40007072-CH3-SW7, . Accessed: September 25, 2017. → pages

[38] G. Developer. Distribution of android versions. http://developer.android.com/about/dashboards/index.html, . Accessed: March 15, 2017. → pages

[39] G. Developer. Permissions usage notes. https://developer.android.com/training/permissions/usage-notes.html, . Accessed: September 24, 2017. → pages

[40] G. Developer. Requesting permissions at run time. https://developer.android.com/training/permissions/requesting.html, . Accessed: September 16, 2017. → pages

[41] W. K. Edwards, E. S. Poole, and J. Stoll. Security automation considered harmful? In *Proceedings of the 2007 Workshop on New Security Paradigms*, pages 33–42. ACM, 2008. → pages

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

[43] 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*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1924943.1924971. → pages

[44] 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, 2016. ACM. → pages

[45] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi:http://doi.acm.org/10.1145/2046707.2046779. URL http://doi.acm.org/10.1145/2046707.2046779. → pages

[46] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM, 2011. → pages

[47] A. P. Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner. How to ask for permission. In *Proc. of the 7th USENIX conference on Hot Topics in Security*, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2372387.2372394. → pages

[48] A. P. Felt, S. Egelman, and D. Wagner. I've got 99 problems, but vibration ain't one: a survey of smartphone users' concerns. In *Proc. of the 2nd ACM workshop on Security and Privacy in Smartphones and Mobile devices*, SPSM '12, pages 33–44, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1666-8. doi:10.1145/2381934.2381943. URL http://doi.acm.org/10.1145/2381934.2381943. → pages

[49] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: user attention, comprehension, and behavior. In *Proc. of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1532-6. doi:10.1145/2335356.2335360. URL http://doi.acm.org/10.1145/2335356.2335360. → pages

[50] 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. → pages

[51] H. Fu and J. Lindqvist. General area or approximate location?: How people understand location permissions. In *Proceedings of the 13th*

Workshop on Privacy in the Electronic Society, pages 117–120. ACM, 2014. → pages

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

[53] 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*, TRUST'12, pages 291–307, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-30920-5. doi:10.1007/978-3-642-30921-2_17. URL http://dx.doi.org/10.1007/978-3-642-30921-2_17. → pages

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

[55] 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*, ICSE 2014, pages 1025–1035, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. doi:10.1145/2568225.2568276. URL http://doi.acm.org/10.1145/2568225.2568276. → pages

[56] M. Harbach, M. Hettig, S. Weber, and M. Smith. Using personal examples to improve risk communication for security & privacy decisions. In *Proc. of the 32nd Annual ACM Conf. on Human Factors in Computing Systems*, CHI '14, pages 2647–2656, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi:10.1145/2556288.2556978. URL http://doi.acm.org/10.1145/2556288.2556978. → pages

[57] M. Harbach, E. von Zezschwitz, A. Fichtner, A. De Luca, and M. Smith. It'sa hard lock life: A field study of smartphone (un) locking behavior and risk perception. In *Symposium on Usable Privacy and Security (SOUPS)*, 2014. → pages

[58] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005. → pages

[59] S. E. Hormuth. The sampling of experiences in situ. *Journal of personality*, 54(1):262–293, 1986. → pages

[60] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi:http://doi.acm.org/10.1145/2046707.2046780. URL http://doi.acm.org/10.1145/2046707.2046780. → pages

[61] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011. → pages

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

[63] L. K. John, A. Acquisti, and G. Loewenstein. Strangers on a plane: Context-dependent willingness to divulge sensitive information. *Journal of consumer research*, 37(5):858–873, 2010. → pages

[64] 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*, SPSM '12, pages 45–50, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1666-8. doi:10.1145/2381934.2381944. URL http://doi.acm.org/10.1145/2381934.2381944. → pages

[65] P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.*, FC'12, pages 68–79, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-34637-8. doi:10.1007/978-3-642-34638-5_6. URL http://dx.doi.org/10.1007/978-3-642-34638-5_6. → pages

[66] P. G. Kelley, L. F. Cranor, and N. Sadeh. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3393–3402, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. doi:10.1145/2470654.2466466. URL http://doi.acm.org/10.1145/2470654.2466466. → pages

[67] 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*, SOAP '14, New York, NY, USA, 2014. ISBN 978-1-4503-2919-4. doi:10.1145/2614628.2614633. URL http://doi.acm.org/10.1145/2614628.2614633. → pages

[68] R. Larson and M. Csikszentmihalyi. New directions for naturalistic methods in the behavioral sciences. In H. Reis, editor, *The Experience Sampling Method*, pages 41–56. Jossey-Bass, San Francisco, 1983. → pages

[69] Y. Li, Y. Guo, and X. Chen. Peruim: Understanding mobile application privacy with permission-ui mapping. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 682–693. ACM, 2016. → pages

[70] H.-T. Lin, C.-J. Lin, and R. C. Weng. A note on platt's probabilistic outputs for support vector machines. *Machine learning*, 68(3):267–276, 2007. → pages

[71] J. Lin, N. Sadeh, S. Amini, J. Lindqvist, J. I. Hong, and J. Zhang. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proc. of the 2012 ACM Conf. on Ubiquitous Computing*, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1224-0. doi:10.1145/2370216.2370290. URL http://doi.acm.org/10.1145/2370216.2370290. → pages

[72] J. Lin, B. Liu, 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)*, pages

199–212, Menlo Park, CA, 2014. USENIX Association. ISBN 978-1-931971-13-3. → pages

[73] B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 201–212, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2744-2. doi:10.1145/2566486.2568035. URL http://doi.acm.org/10.1145/2566486.2568035. → pages

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

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

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

[77] N. K. Malhotra, S. S. Kim, and J. Agarwal. Internet Users' Information Privacy Concerns (IUIPC): The Construct, The Scale, and A Causal Model. *Information Systems Research*, 15(4):336–355, December 2004. → pages

[78] K. Micinski, D. Votipka, R. Stevens, N. Kofinas, J. S. Foster, and M. L. Mazurek. User interactions and permission use on android. In *CHI 2017*, 2017. → pages

[79] S. Mirzamohammadi and A. Amiri Sani. Viola: Trustworthy sensor notifications for enhanced privacy on mobile systems. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 263–276. ACM, 2016. → pages

[80] 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*, pages 1–6. ACM, 2013. → pages

[81] M. Nauman, S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-936-7. doi:http://doi.acm.org/10.1145/1755688.1755732. URL http://doi.acm.org/10.1145/1755688.1755732. → pages

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

[83] H. Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009. → pages

[84] K. Olejnik, I. I. Dacosta Petrocelli, J. C. Soares Machado, K. Huguenin, M. E. Khan, and J.-P. Hubaux. Smarper: Context-aware and automatic runtime-permissions for mobile devices. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017. → pages

[85] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proc. of the 22nd USENIX Sec. Symp.*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association. ISBN 978-1-931971-03-4. URL http://dl.acm.org/citation.cfm?id=2534766.2534812. → pages

[86] S. Parkin, K. Krol, I. Becker, and M. A. Sasse. Applying cognitive control modes to identify security fatigue hotspots. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, 2016. → pages

[87] Path. We are sorry. http://blog.path.com/post/17274932484/we-are-sorry. Accessed: September 25, 2017. → pages

[88] Path. We are sorry. http://blog.path.com/post/17274932484/we-are-sorry, February 8 2012. Accessed: February 26, 2016. → pages

[89] G. Petracca, A.-A. Reineh, Y. Sun, J. Grossklags, and T. Jaeger. Aware: Preventing abuse of privacy-sensitive sensors via operation bindings. In *26th USENIX Security Symposium 17)*, pages 379–396. USENIX Association, 2017. → pages

[90] J. Pruitt and J. Grudin. Personas: Practice and theory. In *Proceedings of the 2003 Conference on Designing for User Experiences*, DUX '03, pages 1–15, New York, NY, USA, 2003. ACM. ISBN 1-58113-728-1. doi:10.1145/997078.997089. URL http://doi.acm.org/10.1145/997078.997089. → pages

[91] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson. Haystack: In situ mobile traffic analysis in user space. *arXiv preprint*, 2015. → pages

[92] J. Ren, A. Rao, M. Lindorfer, A. Legout, and D. Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016. → pages

[93] K. Renaud. Blaming noncompliance is too convenient: What really causes information breaches? *IEEE Security & Privacy*, 10(3):57–63, 2012. → pages

[94] T. Ringer, D. Grossman, and F. Roesner. Audacious: User-driven access control with unmodified operating systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 204–216. ACM, 2016. → pages

[95] F. Roesner and T. Kohno. Securing embedded user interfaces: Android and beyond. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 97–112, 2013. → pages

[96] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan. User-driven access control: Rethinking permission granting in modern operating systems. In *2012 IEEE Symposium on Security and Privacy*, pages 224–238. IEEE, 2012. → pages

[97] 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. → pages

[98] N. Sadeh, J. Hong, L. Cranor, I. Fette, P. Kelley, M. Prabaker, and J. Rao. Understanding and capturing people's privacy policies in a mobile social networking application. *Personal and Ubiquitous Computing*, 13(6): 401–412, 2009. → pages

[99] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 13–22, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1295-0. doi:10.1145/2295136.2295141. URL http://doi.acm.org/10.1145/2295136.2295141. → pages

[100] B. Shebaro, O. Oluwatimi, D. Midi, and E. Bertino. Identidroid: Android can finally wear its anonymous suit. *Trans. Data Privacy*, 7(1):27–50, Apr. 2014. ISSN 1888-5063. URL http://dl.acm.org/citation.cfm?id=2612163.2612165. → pages

[101] I. Shklovski, S. D. Mainwaring, H. H. Skúladóttir, and H. Borgthorsson. Leakiness and creepiness in app space: Perceptions of privacy and mobile app use. In *Proc. of the 32nd Ann. ACM Conf. on Human Factors in Computing Systems*, CHI '14, pages 2347–2356, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2473-1. doi:10.1145/2556288.2557421. URL http://doi.acm.org/10.1145/2556288.2557421. → pages

[102] Y. Song and U. Hengartner. Privacyguard: A vpn-based platform to detect information leakage on android devices. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 15–26. ACM, 2015. → pages

[103] S. Spiekermann, J. Grossklags, and B. Berendt. E-privacy in 2nd generation e-commerce: privacy preferences versus actual behavior. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 38–47. ACM, 2001. → pages

[104] 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, SAC '13, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi:10.1145/2480362.2480701. URL http://doi.acm.org/10.1145/2480362.2480701. → pages

[105] B. Stanton, M. F. Theofanos, S. S. Prettyman, and S. Furman. Security fatigue. *IT Professional*, 18(5):26–32, 2016. → pages

[106] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen. Asking for (and about) permissions used by android apps. In *Proc. of the 10th Working Conf. on Mining Software Repositories*, MSR '13, pages 31–40, Piscataway, NJ, USA, 2013. IEEE Press. ISBN 978-1-4673-2936-1. URL http://dl.acm.org/citation.cfm?id=2487085.2487093. → pages

[107] J. Tan, K. Nguyen, M. Theodorides, H. Negron-Arroyo, C. Thompson, S. Egelman, and D. Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, 2014. → pages

[108] 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. → pages

[109] S. Thurm and Y. I. Kane. Your apps are watching you. The Wall Street Journal. Accessed: January 21, 2016. → pages

[110] O. Tripp and J. Rubin. A bayesian approach to privacy enforcement in smartphones. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 175–190, San Diego, CA, 2014. USENIX Association. ISBN 978-1-931971-15-7. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/tripp. → pages

[111] L. Tsai, P. Wijesekera, J. Reardon, I. Reyes, S. Egelman, D. Wagner, N. Good, and J.-W. 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. ISBN 978-1-931971-39-3. → pages

[112] J. Tygar. Adversarial machine learning. *IEEE Internet Computing*, 15(5): 4–6, 2011. → pages

[113] M. Van Kleek, I. Liccardi, R. Binns, J. Zhao, D. J. Weitzner, and N. Shadbolt. Better the devil you know: Exposing the data sharing practices of smartphone apps. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 5208–5220. ACM, 2017. → pages

[114] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, Mar. 1985. ISSN 0098-3500. doi:10.1145/3147.3165. URL http://doi.acm.org/10.1145/3147.3165. → pages

[115] H. Wang, Y. Li, Y. Guo, Y. Agarwal, and J. I. Hong. Understanding the purpose of permission use in mobile apps. *ACM Transactions on Information Systems (TOIS)*, 35(4):43, 2017. → pages

[116] 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*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1312-4. doi:10.1145/2420950.2420956. URL http://doi.acm.org/10.1145/2420950.2420956. → pages

[117] 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)*, pages 499–514, Washington, D.C., Aug. 2015. USENIX Association. ISBN 978-1-931971-232. → pages

[118] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov. The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1077–1093, May 2017. doi:10.1109/SP.2017.51. → pages

[119] A. Woodruff, V. Pihur, S. Consolvo, L. Brandimarte, and A. Acquisti. Would a privacy fundamentalist sell their dna for $1000...if nothing bad happened as a result? the westin categories, behavioral intentions, and consequences. In *Proceedings of the 2014 Symposium on Usable Privacy*

and Security, pages 1–18. USENIX Association, 2014. ISBN 978-1-931971-13-3. URL https://www.usenix.org/conference/soups2014/proceedings/presentation/woodruff. → pages

[120] 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. → pages

[121] R. Xu, H. Saïdi, and R. Anderson. Aurasium: Practical policy enforcement for android applications. In *Proc. of the 21st USENIX Sec. Symp.*, Security'12, pages 27–27, Berkeley, CA, USA, 2012. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2362793.2362820. → pages

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

[123] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang. Vetting undesirable behaviors in android apps with permission use analysis. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.*, CCS '13, pages 611–622, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi:10.1145/2508859.2516689. URL http://doi.acm.org/10.1145/2508859.2516689. → pages

[124] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh. Taming information-stealing smartphone applications (on android). In *Trust and Trustworthy Computing*, pages 93–107. Springer, 2011. → pages

[125] 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, 2014. ACM. ISBN 978-1-4503-2956-9. doi:10.1145/2623330.2623705. URL http://doi.acm.org/10.1145/2623330.2623705. → pages

Appendix A

Resource Usage

This section presents observations made during the first field study. Specifically related to applications accessing resources under varying contextual circumstances.

A.1 Invisible requests

Following list shows the set of applications that have requested the most number of permissions while executing invisibly to the user and the most requested permission types by each respective application.

- *Facebook App*— ACCESS NETWORK STATE, ACCESS FINE LOCATION, ACCESS WIFI STATE ,WAKE LOCK,
- *Google Location*—WAKE LOCK, ACCESS FINE LOCATION, GET ACCOUNTS, ACCESS COARSE LOCATION,
- *Facebook Messenger*—ACCESS NETWORK STATE, ACCESS WIFI STATE, WAKE LOCK, READ PHONE STATE,
- *Taptu DJ*—ACCESS NETWORK STATE, INTERNET, NFC
- *Google Maps*—ACCESS NETWORK STATE, GET ACCOUNTS, WAKE LOCK, ACCESS FINE LOCATION,

- *Google (Gapps)*—WAKE LOCK, ACCESS FINE LOCATION, AUTHENTICATE ACCOUNTS, ACCESS NETWORK STATE,
- *Fouraquare*—ACCESS WIFI STATE, WAKE LOCK, ACCESS FINE LOCATION, INTERNET,
- *Yahoo Weather*—ACCESS FINE LOCATION, ACCESS NETWORK STATE, INTERNET, ACCESS WIFI STATE,
- *Devexpert Weather*—ACCESS NETWORK STATE, INTERNET, ACCESS FINE LOCATION,
- *Tile Game(Umoni)*—ACCESS NETWORK STATE, WAKE LOCK, INTERNET, ACCESS WIFI STATE,

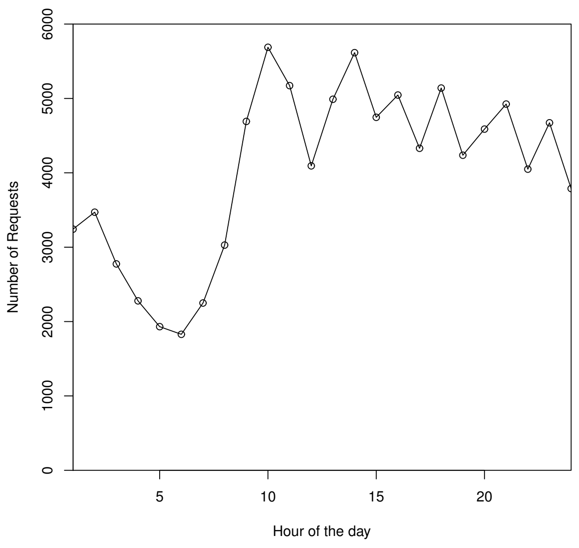
Following is the most frequently requested permission type by applications while running invisibly to the user and the applications who requested the respective permission type most.

- *ACCESS_NETWORK_STATE*— Facebook App, Google Maps, Facebook Messenger, Google (Gapps), Taptu - DJ
- *WAKE_LOCK*—Google (Location), Google (Gapps), Google (GMS), Facebook App, GTalk.
- *ACCESS_FINE_LOCATION*—Google (Location), Google (Gapps), Facebook App, Yahoo Weather, Rhapsody (Music)
- *GET_ACCOUNTS*—Google (Location), Google (Gapps), Google (Login), Google (GM), Google (Vending)
- *ACCESS_WIFI_STATE*—Google (Location), Google (Gapps), Facebook App, Foursqaure, Facebook Messenger
- *UPDATE_DEVICE_STATS*—Google (SystemUI), Google (Location), Google (Gapps)
- *ACCESS_COARSE_LOCATION*—Google (Location), Google (Gapps), Google (News), Facebook App, Google Maps
- *AUTHENTICATE_ACCOUNTS*—Google (Gapps), Google (Login), Twitter, Yahoo Mail, Google (GMS)

- *READ_SYNC_SETTINGS*—Google (GM), Google (GMS), android.process.acore, Google (Email), Google (Gapps)
- *INTERNET*—Google (Vending), Google (Gapps), Google (GM), Facebook App, Google (Location)

A.2 Distribution of Requests

The following graph shows the distribution of requests throughout a given day averaged across the data set.



A.3 Permission Type Breakdown

This table lists the most frequently used permissions during the study period. (per user / per day)

Permission Type	**Requests**
ACCESS_NETWORK_STATE	41077
WAKE_LOCK	27030
ACCESS_FINE_LOCATION	7400
GET_ACCOUNTS	4387
UPDATE_DEVICE_STATS	2873
ACCESS_WIFI_STATE	2092
ACCESS_COARSE_LOCATION	1468
AUTHENTICATE_ACCOUNTS	1335
READ_SYNC_SETTINGS	836
VIBRATE	740
INTERNET	739
READ_SMS	611
READ_PHONE_STATE	345
STATUS_BAR	290
WRITE_SYNC_SETTINGS	206
CHANGE_COMPONENT_ENABLED_STATE	197
CHANGE_WIFI_STATE	168
READ_CALENDAR	166
ACCOUNT_MANAGER	134
ACCESS_ALL_DOWNLOADS	127
READ_EXTERNAL_STORAGE	126
USE_CREDENTIALS	101
READ_LOGS	94

A.4 User Application Breakdown

This table shows the applications that most frequently requested access to protected resources during the study period. (per user / per day)

Application Name	**Requests**
facebook.katana	40041
google.process.location	32426
facebook.orca	24702
taptu.streams	15188
google.android.apps.maps	6501
google.process.gapps	5340
yahoo.mobile.client.android.weather	5505
tumblr	4251
king.farmheroessaga	3862
joelapenna.foursquared	3729
telenav.app.android.scout_us	3335
devexpert.weather	2909
ch.bitspin.timely	2549
umonistudio.tile	2478
king.candycrushsaga	2448
android.systemui	2376
bambuna.podcastaddict	2087
contapps.android	1662
handcent.nextsms	1543
foursquare.robin	1408
qisiemoji.inputmethod	1384
devian.tubemate.home	1296
lookout	1158

Appendix B

Decision Prediction

This section presents how each different feature set was contributing to the final accurateness of the respective predictive models.

B.1 Information Gain of Contextual Features

	Contextuals	Defaulters	Overall
A1	0.4839	0.6444	0.5717
A2	0.4558	0.6395	0.5605
Permission	0.0040	0.0038	0.0050
Time	0.0487	0.1391	0.0130
Visibility	0.0015	0.0007	0.0010

Table B.1: Feature Importance of Contextual Features

B.2 Information Gain of Behavioral Features

Feature	**Importance**
Amount of time spent on audio calls	0.327647825
Frequency of audio calls	0.321291184
Proportion of times screen was timed out instead of pressing the lock button	0.317631096
Number of times PIN was used to unlock the screen.	0.305287288
Number of screen unlock attempts	0.299564131
Amount of time spent unlocking the screen	0.29930659
Proportion of time spent on loud mode	0.163166296
Proportion of time spent on silent mode	0.138469725
Number of times a website is loaded to the Chrome browser	0.094996437
Out of all visited websites, the proportion of HTTPS-secured websites.	0.071096898
Number of times Password was used to unlock the screen	0.067999523
Proportion of websites requested location through Chrome	0.028404167
Time	0.019799623
The number of downloads through Chrome	0.014619351
Permission	0.001461635
Visibility	0.000162166

Table B.2: Feature Importance of Behavioral Features