Android Permissions Remystified: A Field Study on Contextual Integrity

Primal Wijesekera, *University of British Columbia;* **Arjun Baokar, Ashkan Hosseini, Serge Egelman, and David Wagner,** *University of California, Berkeley;* **Konstantin Beznosov,** *University of British Columbia*

https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wijesekera

This paper is included in the Proceedings of the 24th USENIX Security Symposium

August 12–14, 2015 • Washington, D.C.

ISBN 978-1-939133-11-3

Open access to the Proceedings of the 24th USENIX Security Symposium is sponsored by USENIX

Android Permissions Remystified: A Field Study on Contextual Integrity

Primal Wijesekera[1], Arjun Baokar[2], Ashkan Hosseini[2], Serge Egelman[2], David Wagner[2], and Konstantin Beznosov[1]
[1]*University of British Columbia, Vancouver, Canada,*
{*primal,beznosov*}*@ece.ubc.ca*
[2]*University of California, Berkeley, Berkeley, USA,*
{*arjunbaokar,ashkan*}*@berkeley.edu,* {*egelman,daw*}*@cs.berkeley.edu*

Abstract

We instrumented the Android platform 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 the notion of "contextual integrity," i.e., 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. At least 80% of our participants would have preferred to prevent at least one permission request, and overall, they stated a desire to block over a third of all requests. Our findings pave the way for future systems to automatically determine the situations in which users would want to be confronted with security decisions.

1 Introduction

Mobile platform permission models regulate how applications access certain resources, such as users' personal information or sensor data (e.g., camera, GPS, etc.). For instance, previous versions of Android prompt the user during application installation with a list of all the permissions that the application may use in the future; if the user is uncomfortable granting any of these requests, her only option is to discontinue installation [3]. On iOS and Android M, the user is prompted at runtime the first time an application requests any of a handful of data types, such as location, address book contacts, or photos [34].

Research has shown that few people read the Android install-time permission requests and even fewer comprehend them [16]. Another problem is habituation: on average, Android applications present the user with four permission requests during the installation process [13]. 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 [15].

Nissenbaum posited that the reason why most privacy models fail to predict violations is that they fail to consider contextual integrity [32]. 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.

We contribute the following:

- 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 Related Work

While users are required to approve Android application permission requests during installation, most do not pay attention and fewer comprehend these requests [16, 26]. In fact, even developers are not fully knowledgeable about permissions [40], and are given a lot of freedom when posting an application to the Google Play Store [7]. Applications often do not follow the principle of least privilege, intentionally or unintentionally [44]. Other work has suggested improving the Android permission model with better definitions and hierarchical breakdowns [8]. Some researchers have experimented with adding fine-grained access control to the Android model [11]. Providing users with more privacy information and personal examples has been shown to help users in choosing applications with fewer permissions [21, 27].

Previous work has examined the overuse of permissions by applications [13, 20], and attempted to identify malicious applications through their permission requests [36] or through natural language processing of application descriptions [35]. Researchers have also developed static analysis tools to analyze Android permission specifications [6, 9, 13]. 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 [39]; 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 [25, 42]. Their level of concern varied from annoyance to seeking retribution when presented with possible risks associated with permissions [15]. Some studies employed crowdsourcing to create a privacy model based on user expectations [30].

Researchers have designed systems to track or reduce privacy violations by recommending applications based on users' security concerns [2, 12, 19, 24, 28, 46–48]. Other tools dynamically block runtime permission requests [37]. Enck et al. found that a considerable number of applications transmitted location or other user data to third parties without requiring user consent [12]. Hornyack et al.'s AppFence system gave users the ability to deny data to applications or substitute fake data [24]. 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) [14]. They concluded that the majority of Android permissions can be automatically granted, but 16% (corresponding to the 12 permissions in Table 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 [32]. 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" [32], however, the lack of thorough documentation on the Android permission model makes it easier for programmers to neglect these norms, whether intentionally or accidentally [38]. Deciding on whether an application is violating users' privacy can be quite complicated since "the scope of privacy is wide-ranging" [32]. 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 [14]. 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.

3 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.

3.1 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.

3.1.1 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) [5]. 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) [4].

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 3.1.2. 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.

3.1.2 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 1) at runtime so that users have contextual information to infer why the data might be needed [14]. 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 [29]. 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 [43].

Figure 1 shows a screenshot captured during the study along with its corresponding log entry. The user was playing the Solitaire game while Spotify requested a WiFi scan. Since this permission was of interest (Table 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

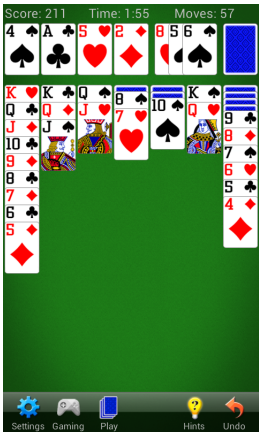
Table 1: The 12 permissions that Felt et al. recommend be granted via runtime dialogs [14]. We randomly took screenshots when these permissions were requested by applications, and we asked about them in our exit survey.

3.2 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. 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 preinstall free applications prior to them visiting our laboratory. (We copied paid applications from their phones, since we could not 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

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



(a) Screenshot

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 - 1412538686641 (Time it was updated)
View	com.mobilityware.solitaire/.Solitaire
History	com.android.phone/.InCallScreen com.android.launcher/com.android.-launcher2.Launcher com.android.mms/ConversationList

(b) Corresponding log entry

Figure 1: Screenshot (a) and corresponding log entry (b) captured during the experiment.

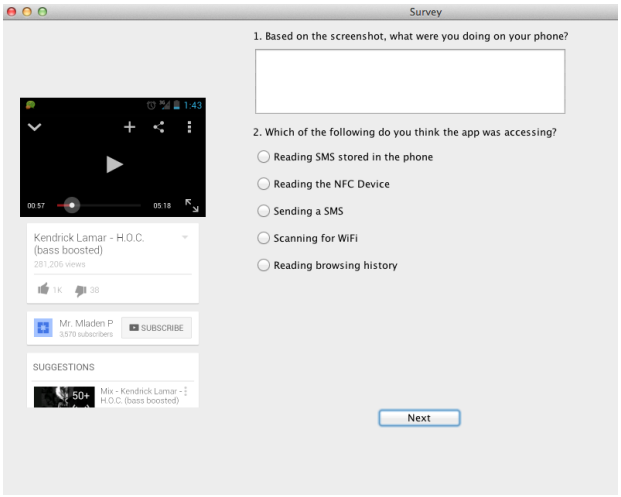
from T-Mobile users). In the email, we noted that due to the space constraints of our experimental phones, we might not be able to install all the applications on their existing phones, and therefore they needed to make a note of the ones that they planned to use that week. The initial setup took roughly 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).

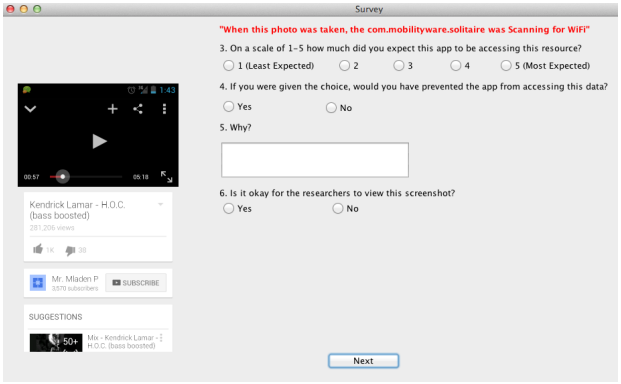
3.3 Exit Survey

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 1 was accessed. Next to the screenshot (Figure 2a), 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 2b). 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



(a) On the first screen, participants answered questions to establish awareness of the permission request based on the screenshot.



(b) On the second screen, they saw the resource accessed, stated whether it was expected, and whether it should have been blocked.

Figure 2: Exit Survey Interface

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 [45], we computed the average of both Buchanan et al.'s Privacy Concerns Scale (PCS) [10] and Malhotra et al.'s Internet Users' Information Privacy Concerns (IUIPC) scale [31].

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.

4 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 [14]).

4.1 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 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.

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

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 [22]. 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 shows the most frequently requested permissions from applications running invisibly to the user (i.e., Categories 2, 4, and 5); Table 3 shows the applications responsible for these requests (Appendix A lists the permissions requested by these applications). We normalized the numbers to show requests per user/day. ACCESS_NETWORK_STATE was most frequently requested, averaging 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 [14]. 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 [42], 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, applica-

tions 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 [17]. 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 [18].

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.

4.2 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 3.1.1), so that the logs would not fill up users' remaining storage or incur performance overhead. Table 4 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 ACCESS_NETWORK_STATE	213.88	956.97
com.facebook.orca ACCESS_NETWORK_STATE	334.78	1146.05
com.google.android.apps.maps ACCESS_NETWORK_STATE	247.89	624.61
com.google.process.gapps AUTHENTICATE_ACCOUNTS	315.31	315.31
com.google.process.gapps WAKE_LOCK	898.94	1400.20
com.google.process.location WAKE_LOCK	176.11	991.46
com.google.process.location ACCESS_FINE_LOCATION	1387.26	1387.26
com.google.process.location GET_ACCOUNTS	373.41	1878.88
com.google.process.location ACCESS_WIFI_STATE	1901.91	1901.91
com.king.farmheroessaga ACCESS_NETWORK_STATE	284.02	731.27
com.pandora.android ACCESS_NETWORK_STATE	541.37	541.37
com.taptu.streams ACCESS_NETWORK_STATE	1746.36	1746.36

Table 4: 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.

4.3 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 [14]. They advocated using runtime dialogs before the following 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 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 develop-

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 5: 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.

ers 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 5 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 4.1, 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 1 resulted in the exposure of sensitive data (Table 5).

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.

5 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 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 (≈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 4.1), whereas 180 were taken while the application was invisible (Category 2 and 4 from Section 4.1). In this section, we describe the situations in which requests

[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.

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.

5.1 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 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.

5.1.1 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).

5.1.2 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)

5.2 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

[3] Both measures were normally distributed.

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 (V=58, $p<0.001$).

Effects of Privacy Preferences on Blocking: Participants completed the Privacy Concerns Scale (PCS) [10] and the Internet Users' Information Privacy Concerns (IUIPC) scale [31]. 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.

5.3 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.

6 Feasibility of Runtime Requests

Felt et al. posited that certain sensitive permissions (Table 1) should require runtime consent [14], but in Section 4.3 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 4.3), 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 pol-

icy, 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.

6.1 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.

6.1.1 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 5. 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 6 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 6 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.

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

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.

6.1.2 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 5. We used our exit survey data as ground truth, and used 5-fold cross-validation to evaluate model accuracy.

We calculated the receiver operating characteristic (ROC) to capture the tradeoff 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) [23]. 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.

7 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 [33]. 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 [14]. 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 4), 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 6.1 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., [1]). 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 [41]. 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.

Acknowledgments

This work was supported by NSF grant CNS-1318680, by Intel through the ISTC for Secure Computing, and by the AFOSR under MURI award FA9550-12-1-0040.

References

[1] ACQUISTI, A., AND GROSSKLAGS, J. Privacy and rationality in individual decision making. *IEEE Security & Privacy* (January/February 2005), 24–30. `http://www.dtc.umn.edu/weis2004/acquisti.pdf`.

[2] ALMOHRI, H. M., YAO, D. D., AND KAFURA, D. Droidbarrier: Know what is executing on your android. In *Proc. of the 4th ACM Conf. on Data and Application Security and Privacy* (New York, NY, USA, 2014), CODASPY '14, ACM, pp. 257–264.

[3] ANDROID DEVELOPERS. System permissions. `http://developer.android.com/guide/topics/security/permissions.html`. Accessed: November 11, 2014.

[4] ANDROID DEVELOPERS. Common Intents. `https://developer.android.com/guide/components/intents-common.html`, 2014. Accessed: November 12, 2014.

[5] ANDROID DEVELOPERS. Content Providers. `http://developer.android.com/guide/topics/providers/content-providers.html`, 2014. Accessed: Nov. 12, 2014.

[6] AU, K. W. Y., ZHOU, Y. F., HUANG, Z., AND LIE, D. Pscout: Analyzing the android permission specification. In *Proc. of the 2012 ACM Conf. on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 217–228.

[7] BARRERA, D., CLARK, J., MCCARNEY, D., AND VAN OORSCHOT, P. C. 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* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 81–92.

[8] BARRERA, D., KAYACIK, H. G. U. C., VAN OORSCHOT, P. C., AND SOMAYAJI, A. 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* (New York, NY, USA, 2010), CCS '10, ACM, pp. 73–84.

[9] BODDEN, E. Easily instrumenting android applications for security purposes. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (NY, NY, USA, 2013), CCS '13, ACM, pp. 1499–1502.

[10] BUCHANAN, T., PAINE, C., JOINSON, A. N., AND REIPS, U.-D. 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 (2007), 157–165.

[11] BUGIEL, S., HEUSER, S., AND SADEGHI, A.-R. Flexible and fine-grained mandatory access control on android for diverse security and privacy policies. In *Proc. of the 22nd USENIX Security Symposium* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association, pp. 131–146.

[12] ENCK, W., GILBERT, P., CHUN, B.-G., COX, L. P., JUNG, J., MCDANIEL, P., AND SHETH, A. N. 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* (Berkeley, CA, USA, 2010), OSDI'10, USENIX Association, pp. 1–6.

[13] FELT, A. P., CHIN, E., HANNA, S., SONG, D., AND WAGNER, D. Android permissions demystified. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (New York, NY, USA, 2011), CCS '11, ACM, pp. 627–638.

[14] FELT, A. P., EGELMAN, S., FINIFTER, M., AKHAWE, D., AND WAGNER, D. How to ask for permission. In *Proceedings of the 7th USENIX conference on Hot Topics in Security* (Berkeley, CA, USA, 2012), HotSec'12, USENIX Association, pp. 7–7.

[15] FELT, A. P., EGELMAN, S., AND WAGNER, D. 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* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 33–44.

[16] FELT, A. P., HA, E., EGELMAN, S., HANEY, A., CHIN, E., AND WAGNER, D. Android permissions: user attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security* (New York, NY, USA, 2012), SOUPS '12, ACM, pp. 3:1–3:14.

[17] FU, H., AND LINDQVIST, J. General area or approximate location?: How people understand location permissions. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society* (2014), ACM, pp. 117–120.

[18] FU, H., YANG, Y., SHINGTE, N., LINDQVIST, J., AND GRUTESER, M. A field study of run-time location access disclosures on android smartphones. *Proc. USEC 14* (2014).

[19] GIBLER, C., CRUSSELL, J., ERICKSON, J., AND CHEN, H. 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* (Berlin, Heidelberg, 2012), TRUST'12, Springer-Verlag, pp. 291–307.

[20] GORLA, A., TAVECCHIA, I., GROSS, F., AND ZELLER, A. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering* (New York, NY, USA, 2014), ICSE 2014, ACM, pp. 1025–1035.

[21] HARBACH, M., HETTIG, M., WEBER, S., AND SMITH, M. 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* (New York, NY, USA, 2014), CHI '14, ACM, pp. 2647–2656.

[22] HARBACH, M., VON ZEZSCHWITZ, E., FICHTNER, A., DE LUCA, A., AND SMITH, M. 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).

[23] HASTIE, T., TIBSHIRANI, R., FRIEDMAN, J., AND FRANKLIN, J. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer 27*, 2 (2005), 83–85.

[24] HORNYACK, P., HAN, S., JUNG, J., SCHECHTER, S., AND WETHERALL, D. 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.* (New York, NY, USA, 2011), CCS '11, ACM, pp. 639–652.

[25] JUNG, J., HAN, S., AND WETHERALL, D. 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* (New York, NY, USA, 2012), SPSM '12, ACM, pp. 45–50.

[26] KELLEY, P. G., CONSOLVO, S., CRANOR, L. F., JUNG, J., SADEH, N., AND WETHERALL, D. A conundrum of permissions: Installing applications on an android smartphone. In *Proc. of the 16th Intl. Conf. on Financial Cryptography and Data Sec.* (Berlin, Heidelberg, 2012), FC'12, Springer-Verlag, pp. 68–79.

[27] KELLEY, P. G., CRANOR, L. F., AND SADEH, N. Privacy as part of the app decision-making process. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2013), CHI '13, ACM, pp. 3393–3402.

[28] KLIEBER, W., FLYNN, L., BHOSALE, A., JIA, L., AND BAUER, L. 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* (New York, NY, USA, 2014), SOAP '14, ACM, pp. 1–6.

[29] LARSON, R., AND CSIKSZENTMIHALYI, M. New directions for naturalistic methods in the behavioral sciences. In *The Experience Sampling Method*, H. Reis, Ed. Jossey-Bass, San Francisco, 1983, pp. 41–56.

[30] LIN, J., SADEH, N., AMINI, S., LINDQVIST, J., HONG, J. I., AND ZHANG, J. Expectation and purpose: understanding users' mental models of mobile app privacy through crowdsourcing. In *Proc. of the 2012 ACM Conf. on Ubiquitous Computing* (New York, NY, USA, 2012), UbiComp '12, ACM, pp. 501–510.

[31] MALHOTRA, N. K., KIM, S. S., AND AGARWAL, J. Internet Users' Information Privacy Concerns (IUIPC): The Construct, The Scale, and A Causal Model. *Information Systems Research 15*, 4 (December 2004), 336–355.

[32] NISSENBAUM, H. Privacy as contextual integrity. *Washington Law Review 79* (February 2004), 119.

[33] NISSENBAUM, H. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009.

[34] O'GRADY, J. D. New privacy enhancements coming to ios 8 in the fall. `http://www.zdnet.com/new-privacy-enhancements-coming-to-ios-8-in-the-fall-7000030903/`, June 25 2014. Accessed: Nov. 11, 2014.

[35] PANDITA, R., XIAO, X., YANG, W., ENCK, W., AND XIE, T. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proc. of the 22nd USENIX Sec. Symp.* (Berkeley, CA, USA, 2013), SEC'13, USENIX Association, pp. 527–542.

[36] SARMA, B. P., LI, N., GATES, C., POTHARAJU, R., NITA-ROTARU, C., AND MOLLOY, I. Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies* (New York, NY, USA, 2012), SACMAT '12, ACM, pp. 13–22.

[37] SHEBARO, B., OLUWATIMI, O., MIDI, D., AND BERTINO, E. Identidroid: Android can finally wear its anonymous suit. *Trans. Data Privacy 7*, 1 (Apr. 2014), 27–50.

[38] SHKLOVSKI, I., MAINWARING, S. D., SKÚLADÓTTIR, H. H., AND BORGTHORSSON, H. 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* (New York, NY, USA, 2014), CHI '14, ACM, pp. 2347–2356.

[39] SPREITZENBARTH, M., FREILING, F., ECHTLER, F., SCHRECK, T., AND HOFFMANN, J. Mobile-sandbox: Having a deeper look into android applications. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (New York, NY, USA, 2013), SAC '13, ACM, pp. 1808–1815.

[40] STEVENS, R., GANZ, J., FILKOV, V., DEVANBU, P., AND CHEN, H. Asking for (and about) permissions used by android apps. In *Proc. of the 10th Working Conf. on Mining Software Repositories* (Piscataway, NJ, USA, 2013), MSR '13, IEEE Press, pp. 31–40.

[41] TAN, J., NGUYEN, K., THEODORIDES, M., NEGRON-ARROYO, H., THOMPSON, C., EGELMAN, S., AND WAGNER, D. 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).

[42] THOMPSON, C., JOHNSON, M., EGELMAN, S., WAGNER, D., AND KING, J. When it's better to ask forgiveness than get permission: Designing usable audit mechanisms for mobile permissions. In *Proceedings of the 2013 Symposium on Usable Privacy and Security (SOUPS)* (2013).

[43] VITTER, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw. 11*, 1 (Mar. 1985), 37–57.

[44] WEI, X., GOMEZ, L., NEAMTIU, I., AND FALOUTSOS, M. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference* (New York, NY, USA, 2012), ACSAC '12, ACM, pp. 31–40.

[45] WOODRUFF, A., PIHUR, V., CONSOLVO, S., BRANDIMARTE, L., AND ACQUISTI, A. 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* (2014), USENIX Association, pp. 1–18.

[46] XU, R., SAÏDI, H., AND ANDERSON, R. Aurasium: Practical policy enforcement for android applications. In *Proc. of the 21st USENIX Sec. Symp.* (Berkeley, CA, USA, 2012), Security'12, USENIX Association, pp. 27–27.

[47] ZHANG, Y., YANG, M., XU, B., YANG, Z., GU, G., NING, P., WANG, X. S., AND ZANG, B. Vetting undesirable behaviors in android apps with permission use analysis. In *Proc. of the ACM Conf. on Comp. and Comm. Sec.* (New York, NY, USA, 2013), CCS '13, ACM, pp. 611–622.

[48] ZHU, H., XIONG, H., GE, Y., AND CHEN, E. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2014), KDD '14, ACM, pp. 951–960.

A 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,

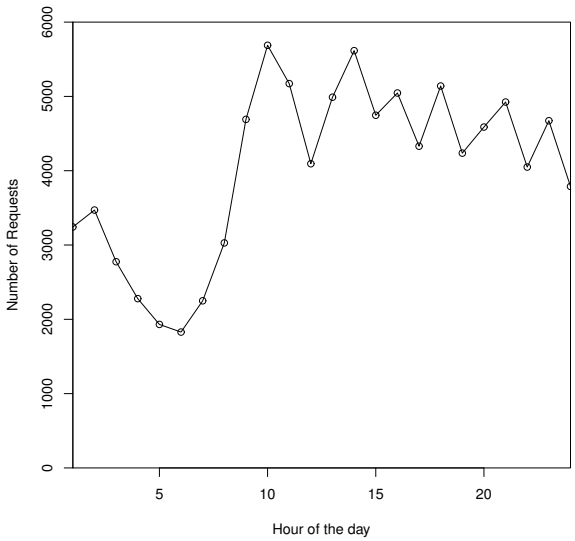
- *Devexpert Weather*—ACCESS NETWORK STATE, INTERNET, ACCESS FINE LOCATION,
- *Tile Game(Umoni)*—ACCESS NETWORK STATE, WAKE LOCK, INTERNET, ACCESS WIFI STATE,

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)

B Distribution of Requests

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



C 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

D 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