User Interactions and Permission Use on Android

Kristopher Micinski, Daniel Votipka, Rock Stevens, Nikolaos Kofinas, Michelle L. Mazurek, and Jeffrey S. Foster

University of Maryland, College Park
{micinski, dvotipka, rstevens, nkofinas, mmazurek, jfoster}@cs.umd.edu

ABSTRACT

Android and other mobile operating systems ask users for authorization before allowing apps to access sensitive resources such as contacts and location. We hypothesize that such authorization systems could be improved by becoming more integrated with the app's user interface. In this paper, we conduct two studies to test our hypothesis. First, we use AppTracer, a dynamic analysis tool we developed, to measure to what extent user interactions and sensitive resource use are related in existing apps. Second, we conduct an online survey to examine how different interactions with the UI affect users' expectations about whether an app accesses sensitive resources. Our results suggest that user interactions such as button clicks can be interpreted as authorization, reducing the need for separate requests; but that accesses not directly tied to user interactions should be separately authorized, possibly when apps are first launched.

ACM Classification Keywords

H.5.m Information Interfaces and Presentation (e.g. HCI): Miscellaneous

Author Keywords

Android; Permissions; Contextual Security; Apps

INTRODUCTION

Android has a *permission system* that asks users for authorization before an app uses sensitive resources such as contacts or GPS location. A key challenge in such authorization systems is balancing user interruptions with making sensitive resource use transparent. We hypothesize that Android's existing authorization systems (install-time permission lists or run-time dialog boxes, depending on the version) could achieve a better balance by integrating with the app's user interface (UI), because the UI deeply informs the user's mental model of the app's behavior, including security-relevant behavior.

In particular, in this paper we ask whether *user interactions*—button clicks, page changes, dialog boxes, etc.—can be taken as evidence of authorization to use certain sensitive resources. If so, this could reduce the need for separate authorization requests. Conversely, we ask whether sensitive resource use without an associated interaction suggests a need for additional authorization requests. Note that while our studies are heavily influenced by Android, we believe our results will generalize to related mobile OS's and similar settings like web apps.

To answer these questions, we conducted two related studies. First, we reviewed 150 popular Android apps to determine whether sensitive resource uses are related to user interactions in existing apps. If so, an authorization mechanism integrated with the UI could work well with existing app designs. To carry out this study, we developed AppTracer, a dynamic analysis tool that instruments Android apps to log UI actions and resource uses, and then visualizes the logs as graphs that show temporal ordering of logged events. We used AppTracer to determine whether each observed resource use in each app was *interactive*, meaning either it was immediately preceded by a related UI event (e.g., accessing contacts after clicking a button marked "contacts"), or it was the main focus of the current screen (e.g., using location on a map screen).

We found that, across our subject apps, several resources (microphone, camera, external storage, and calendar) are used almost exclusively interactively; several others (including bluetooth and phone state) are used mostly non-interactively (which we call in the *background* even if the app itself is on screen); and several resources (most notably contacts and location) exhibit a mix of interactive and background uses. These results suggest interactive and background uses may call for different authorization mechanisms, and that these mechanisms cannot necessarily be divided strictly by resource.

These results informed the design of our second study, a 961-participant online survey investigating participants' expectations about interactive and background permission uses. Each participant viewed a slideshow of two usage scenarios for a mock mobile app, where each scenario shows a short interaction (e.g., launching the app, clicking a button, etc.) and then asks if the participant expects microphone, location, and/or contacts to be used after the interaction. We chose these resources to reflect a range of interactivity as measured in our app study. We aimed to gain insight into how different factors affect user expectations, and therefore which authorization mechanisms might be appropriate for different usage patterns.

As we anticipated, we found that users are much more likely to expect resources to be accessed after a related interaction than in the background. However, we also found that seeing one interactive use does not prime the user to expect a future background use, indicating a potential weakness in the Android M request-on-first-use authorization model. In contrast, our findings show that an authorization request at launch

CHI 2017, May 06–11, 2017, Denver, CO, USA.

ACM 978-1-4503-4655-9/17/05 ... $15.00
DOI: http://dx.doi.org/10.1145/3025453.3025706

does increase expectations for both interactive and background accesses, perhaps because it better conveys the idea that the resource could be accessed at any time.

Drawing on the results of our studies, we make three design recommendations. First, resource uses should be made after associated interactions as much as possible. Given the current makeup of apps, this should be achievable for many commonly used resources without extensive effort. Second, separate authorization dialogs might be unnecessary for resources that are accessed mostly interactively (see, e.g., [27]). Finally, authorization for background resource uses should be distinct from authorization for interactive uses, and these background authorizations may be most effective when the app is launched.

BACKGROUND AND RELATED WORK

In earlier versions of Android, users were presented with a list of permissions requested by an app at install time. The user could then either grant the app full use of those permissions or not install the app at all. This model had a number of problems: few users comprehended or even read the list of permissions [12], and many apps requested more permissions than they used [10]. Because of these issues, Android M [14] switched to a model where apps ask for a permission the first time it is needed; the permission is then granted indefinitely.

In our work, we ask whether authorization systems similar to Android's can be improved by taking the user interface into account. Note that our work is orthogonal to the question of whether permissions are at the right level of granularity [6, 17] or protect the right resources [11].

Contextual Security on Mobile Devices

The motivation for our work, that authorization can be better integrated with the UI, exemplifies *contextual security* [22], which suggests security decisions should take the context into account. Several researchers have studied contextual security on mobile devices. Almuhimedi et. al [3] showed users historical data about how apps accessed their locations. They found 95% of users reassessed the apps' need for location, with 58% of those users further restricting location access. King [19] found users are more likely to expect sensitive resource accesses when suggested by the context. Felt et. al [1] proposed a process for deciding the appropriate authorization mechanism for a permission based on the a permissions' use in context. Several researchers [5, 13, 33] found users are surprised by some sensitive resource accesses that occur when apps are in the background. Most closely related to this paper, in a field study Wijesekera et al. [33] found that context is an important factor in determining expectation of resource use. Our work builds on this finding by using a controlled experiment to distinguish how different contextual factors, including consecutive interactions, contribute to user expectations.

The works just mentioned mainly define context as whether the app is on or off the screen. In contrast, we use a much richer notion of context based on sequences of UI actions.

Enforcing Contextual Security

Many systems have been proposed to enforce contextual security in apps. Chen et. al [8] present Pegasus, a static analysis system for analyzing apps and enforcing policies based on *permission event graphs* (PEGs). For example, Pegasus can check that contacts are only accessed after clicking a certain button. PEGs inspired the design of AppTracer. However, AppTracer uses dynamic (rather than static) analysis to reduce issues of false positives—every behavior AppTracer logs occurred in an actual run, whereas static analysis may report sensitive resource accesses that can never actually occur.

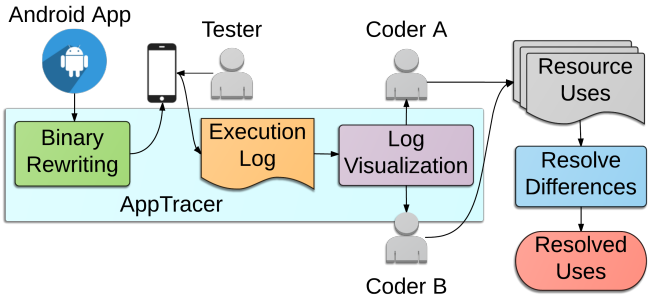

Figure 1: App Measurement Survey Procedure.

Yang et al. [34] presented AppIntent, which uses symbolic execution to determine sequences of UI events that lead to information leakage. Micinski et. al [21] use symbolic execution to enforce secure information-flow properties based on UI events. While both systems are promising, in practice symbolic execution is difficult to run at scale on Android apps due to the complexity of modeling the Android framework.

Stiegler et al. developed CapDesk [30] and later Polaris [29], two capability-based desktop system that utilize user interaction to drive access control. However, CapDesk and Polaris's focus is limited to file access. Roesner et. al [27] expand user-driven access control with *Access Control Gadgets* (ACGs), which tie resource accesses to certain UI elements, e.g., an ACG might allow location to be used only after a specific button is clicked. ACGs were later expanded to work on Android, with and later without modifying the operating system [25, 26]. The original ACG paper includes a user study measuring expectations related to interactive permission uses; our work expands on this idea to study a broader variety of factors and use cases. While the current paper does not directly implement or measure ACGs, our findings do support the use of ACGs.

APP MEASUREMENT SURVEY METHODOLOGY

In our first study, we reviewed a set of popular Android apps to determine how UI actions and resource uses are related. Figure 1 gives a high-level overview of our review process, which ultimately produces a set of *resource use codes* that indicate, for each resource use, what event (if any) caused the use. For example, we might determine that in some app, contacts were accessed just after a button click, or location was used immediately when the app launched.

The next subsections walk through the review process in detail.

Binary Rewriting and Execution Logging

The first step of our process uses AppTracer, a dynamic analysis tool we developed. AppTracer instruments the subject app

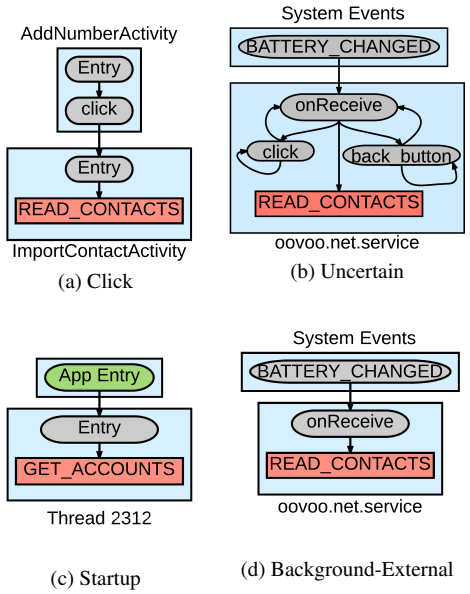

Figure 2: AppTracer graphs and corresponding resource access patterns in our codebook.

so that, when run, it writes a log of UI events and permission uses. AppTracer adds instrumentation using Redexer [17], a rewriting tool for Dalvik bytecode (the language to which Android apps are compiled). Generally speaking, AppTracer instruments code by appending a log method to the bytecode and inserting calls to log at the beginning of UI-related callbacks (e.g., onClick handlers for button events) and just before calls to permission-protected methods (e.g., getLastLocation).

We identified methods associated with the UI using the Android documentation. We identified permission-protected methods using the PScout dataset [4], which attempts to list every security-sensitive method and its associated permission. PScout also includes information about sensitive content providers, intents, etc. Note that while PScout is reasonably thorough, we found several cases it missed. For example, we observed several visual indicators of SD card use that had no corresponding log entry AppTracer. After investigating the apps' decompiled code, we found the SD card was used via Java IO methods omitted by PSCout. Whenever we found such cases, we added the missing methods to our copy of the PScout database and reran our evaluated apps.

Once an app has been instrumented, a tester runs it to generate a log. (Note that one log is usually sufficient because in most apps, any app state is reachable from any other app state.) We elide the details of the log, but at a high level, for each UI event, AppTracer records the type of event and the corresponding parameters (e.g., what button was clicked, which menu option was selected, etc.). For each permission use, AppTracer logs the name of the called method and its arguments.

Log Visualization

After the tester produces a log, the next step is log visualization. Figures 2a–2b show example portions of AppTracer's graph-based visualization, redrawn for compactness and to omit most package names. Here, light blue boxes represent *activities*, which are the "screens" of the app. Within each activity, gray ovals represent the beginning of that activity ("entry," which usually corresponds to the onStart handler), UI events (e.g., "click"), or system events (e.g., BATTERY_CHANGED). Red rectangles indicate calls to methods protected by the named permission. There is an edge from node A to node B if A occurs immediately before B in the log. For example, in Figure 2a, contacts were read immediately after the ImportContactActivity activity was started.

Since logs can be quite lengthy, AppTracer heuristically merges nodes that arise from the same position in the bytecode. This sometimes results in ambiguity. For example, in Figure 2b, the single onReceive node actually represents calls to an Android broadcast receiver that AppTracer coalesced. We discuss this more below.

Finally, AppTracer also allows the user to directly view the log file entries corresponding to a node in the graph. This is useful to retrieve more detailed information about the node. For example, when reviewing Figure 2a, we looked in the log to determine that the clicked button had text associated with contacts. As another example, we used the logs to distinguish SD card accesses to user files from accesses to the app's own storage. We do not count the latter as a sensitive resource access because it accesses data owned by the app.

Resource Uses

The next step is to examine the AppTracer graph and record a set of codes that accurately categorize various resource accesses. More precisely, for each red node in the graph, the coder assigns a pair of the form $(resource, pattern)$, where *resource* indicates what is protected by the permission and *pattern* is one of six different UI patterns, discussed next.

To keep our results understandable, we grouped together resources according to Google's permission groups [24]. For example, the single SMS resource includes more fine-grained permissions such as READ_SMS and SEND_SMS.

We developed an initial codebook for UI patterns based on our knowledge of Android app development. We then iteratively applied our codebook to sets of five apps (not in our evaluation set) at a time and adjusted the codes as necessary. After evaluating a total of twenty apps, we felt we had reached a codebook with a minimal set of orthogonal patterns.

The six access patterns in our codebook are grouped into three categories. First are the *interactive* patterns *Click* and *Page*. The code *Click* indicates resource use preceded by a UI event (including non-clicks such as swipes). For example, in Figure 2a, we coded READ_CONTACTS as *Click* because the contacts were accessed on a straight-line path from the click of a button labeled as importing contacts. *Page* codes uses that are clearly associated with an activity but are not associated with exactly one click. For example, *Page* would code the use of location during an activity that shows a list of nearby stores.

Second, in the *background* patterns *Startup*, *Bg-App*, and *Bg-Ext*, the resource use has no obviously connected user action.

Startup codes a resource accessed right after app launch but before any screen appears (and is thus disjoint from *Page*), e.g., the use of accounts in Figure 2c, which occurs in a thread created at app launch. *Bg-App* codes a resource used while the app is in the foreground, but with no clearly related user action. For example, a graph similar to Figure 2a would be coded as *Bg-App* if a button labeled for importing contacts was followed by the GET_ACCOUNTS permission rather than READ_CONTACTS. *Bg-Ext* denotes a resource used due to a system event, meaning it would be used whether or not the app is on screen. For example, in Figure 2d, accounts are read after the BATTERY_CHANGED system event.

Finally, *Uncertain* denotes a resource access that did not fit any of the previous patterns. In Figure 2b, we see a path from a system event to onReceive to READ_CONTACTS, which should be coded as *Bg-Ext*. But, the onReceive node also has an incoming edge from the click handler and the back button. These uses would be coded as *Bg-App*. (The *Click* and *Page* codes do not apply because the associated buttons do not indicate contacts will be used.) Because there are multiple possibilities, we code this case as *Uncertain*.

If desired, an AppTracer user can subsequently review the logs to try to resolve the uncertainty. For example, here the log entry associated with BATTERY_CHANGED is followed by a call to onReceive and then READ_CONTACTS. Thus, the permission use is coded as *Bg-Ext*. We then separately looked at the log entries for click and back_button, and found they were followed by an onReceive that was *not* followed by READ_CONTACTS. Thus, examining the logs allowed us to distinguish paths that we merged in AppTracer.

Note that for each app we only count each $(resource, pattern)$ pair once, no matter how often it occurs in the log. A stronger notion of frequency would be hard to interpret, since it would depend on how AppTracer heuristically coalesces graph nodes as well as to how the tester explored the app.

Coding Apps and Resolving Differences

Coding the resource uses of an app is inherently complex. Thus, two coders reviewed apps independently in sets of 15 and met after each set to resolve differences. Coders took approximately 10–20 minutes to code each app, and resolving differences for a set of 15 apps took approximately 30 minutes.

Most differences between coders were due to one coder overlooking a path or a resource in the AppTracer graph. In almost all such cases, when the other coder pointed out the omission, the first coder would quickly reach the same conclusion as the other coder. The remaining differences were caused by disagreements about whether the resource use was interactive. For example, one app read a user's accounts within an activity for filling a form. One coder recorded this as *Click*, since the accounts were read after a click. The other coder recorded this as *Bg-App*, because there was no observable use of the account data, such as pre-filling the form with data from the user's existing accounts. After encountering several such cases, the coders decided on a general principle of coding uses as *Bg-App* unless the UI explicitly mentioned that the resource could be used—hence, this example was resolved as *Bg-App*.

Inter-rater reliability between our coders for the non-visualization-error disagreements was Krippendorff's $\alpha = 0.897$, indicating close agreement [15].

App Selection

We drew our subject apps from a larger set comprising the 20 top downloaded free apps[1] from the 27 non-gaming categories on Google Play. We excluded gaming apps because they typically use native code that AppTracer cannot analyze. This yielded 503 apps (note there is overlap between categories).

We then randomly selected 150 apps to evaluate, subject to the constraint that no more than two apps were from the same developer. (We wanted to avoid bias due to overrepresentation of apps coded in the same style.). We excluded any apps that could not be run with AppTracer, replacing them with new randomly drawn apps to maintain an evaluation set of 150.

We excluded 48 apps because Redexer fails when rewriting them and 23 apps because either they refuse to run if modified (due to internal or system signature checks) or they are primarily implemented in native code. In most cases, we created accounts when signup was required to fully exercise an app, but we excluded 16 apps because they require accounts that are hard to set up online or require a fee (e.g., bank accounts).

Limitations

There are several potential limitations to this study. First, the tester may miss some app behavior, leading to a log that omits some possible resource uses and contexts. We tried to alleviate this concern by exercising as much of the app as possible. However, we did not use app features that required payment.

Second, AppTracer has limitations mentioned above: it may miss UI events or permission-protected methods, and it merges nodes that correspond to the same position in the bytecode. We tried to address the former issue by looking for cases where we expected an event to be in a log but it was not, and then adding the missing events or methods. We addressed the latter issue by manually disambiguating some of the uncertain cases.

Third, our study reviewed only popular Android apps. Observing that a resource is already accessed interactively in popular apps would indicate that changing the Android framework or system to implement interactivity-related protections for those accesses may be reasonable. These apps also represent the common case and likely help to set user expectations about apps and permissions. However, we note that the apps we examine likely differ from the long tail of other apps in important ways; popular apps are likely implemented to high standards and unlikely to be malicious. We leave similar measurements on a broader set of apps as future work.

APP MEASUREMENT SURVEY RESULTS

Figure 3 summarizes the resource use patterns we found. For example, across all apps, the camera was used in the *Click* pattern 57 times. Orange-shaded bars indicate interactive use patterns, and blue shades represent background uses. Resources are sorted by percent of interactive patterns.

[1] as of June 11, 2016

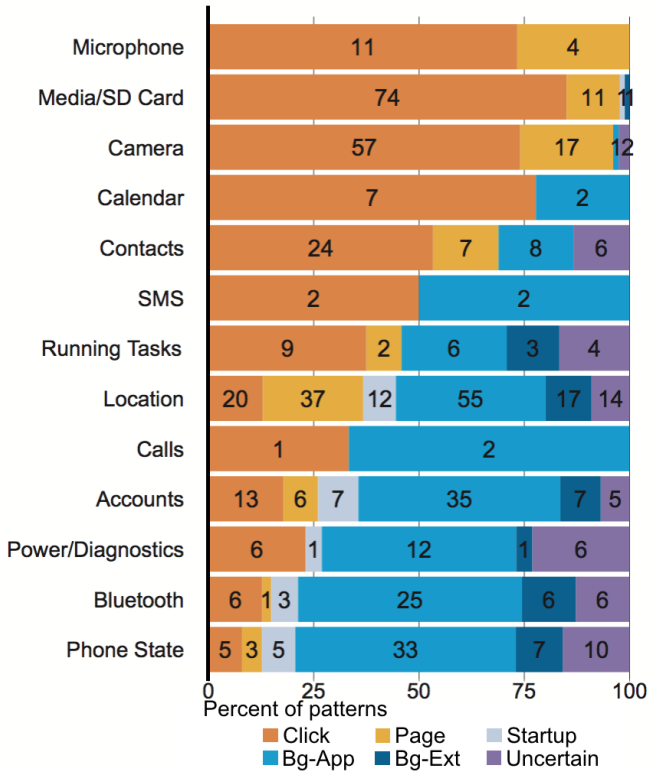

Figure 3: Percent of observed patterns of each type, per resource. Bar labels indicate how many apps we observed using each pattern, non-normalized.

We see that, to a rough approximation, the more sensitive the resource the more likely it is used interactively. Indeed, microphone, media, camera, and calendar were accessed almost exclusively interactively. We investigated the background and *Uncertain* uses for these resources. One calendar use and the two SD card uses were due to periodic background tasks (calendar syncing or scanning for files). One camera use took a picture after the passcode was entered incorrectly three times. One camera use and one calendar use did not seem to access sensitive resources—the camera use accessed configuration information, and the calendar use got the current date (which could be done without accessing the calendar). Finally, after disassembling and reading the bytecode, we found one *Uncertain* camera use could actually only happen interactively.

We next see that contacts, SMS, tasks, location, and calls are used in a mix of interactive and non-interactive ways. We investigated the background and *Uncertain* uses of these resources as well. Contacts were used in the background mainly to pre-fetch or sync the contact list. Two apps used SMS in the background to listen for a registration code after the user signed up for an account. Currently running tasks were polled in the background to monitor battery usage, scan for malicious apps, look for apps by the same developer (to communicate with them), or for analytics and tracking purposes. Call-related permissions were used in the background to block calls from a user-supplied blacklist. While there were too many background location uses to examine them all, those we did examine frequently had no obvious reason (as expected [3, 13]), even in apps that elsewhere used location for a clear purpose.

Resource	# Apps	Resource	# Apps
Location	75	Microphone	14
Media/SD Card	69	Tasks	13
Camera	69	Power/Diag.	12
Phone State	43	Calendar	5
Accounts	39	SMS	4
Bluetooth	31	Calls	2
Contacts	30		

Table 1: Number of apps that used each resource.

Finally, four resources—accounts, power, bluetooth, and phone state information—were mostly accessed in the background. We believe this is either because developers believe users care less about these accesses [11], the uses are hard to explain clearly to non-experts, or the uses are not naturally associated with an immediately preceding interaction.

Looking in more detail at the breakdown between *Click* and *Page*, we see that for most resources, *Click* is a clear majority of the interactive uses. The exception is location, which has more *Page* uses. This was mostly due to location use for map screens or lists of nearby places. Breaking down the background uses, we see the use of resources at *Startup* and in *Bg-Ext* becomes much more common lower in the chart.

Resource Usage Across Apps

We also measured the number of apps that used each resource at least once, as a rough approximation for how familiar each resource is to users. Table 1 shows the results. We found a wide range across resources, with little correlation between frequency of appearance in apps and usage patterns. For example, location was used frequently, and Figure 3 shows that it was often used in the background, whereas media/SD card was also used frequently, but was rarely seen in the background.

Discussion

The results of our app survey suggest several possibilities. Given the large amount of interactive resource use overall, there seems to be a clear opportunity for better integrating authorization into the UI. However, the question remains to what extent interactions make resource use apparent to users. We try to answer this question in the next two sections.

Our app survey also shows that many resources are used in the background, and many of these uses seem reasonable, at least to the authors. Moreover, apps sometimes use the same resource both in the foreground and in the background, to different purposes. This suggests interactive and background uses should be authorized separately. Thus, in the next two sections, we also try to answer questions about how background uses should be authorized, and about whether users' expectations of interactive and non-interactive uses are related.

USER EXPECTATIONS STUDY METHODOLOGY

After our app survey, we conducted an online study to elicit users' expectations about resource use as different user actions

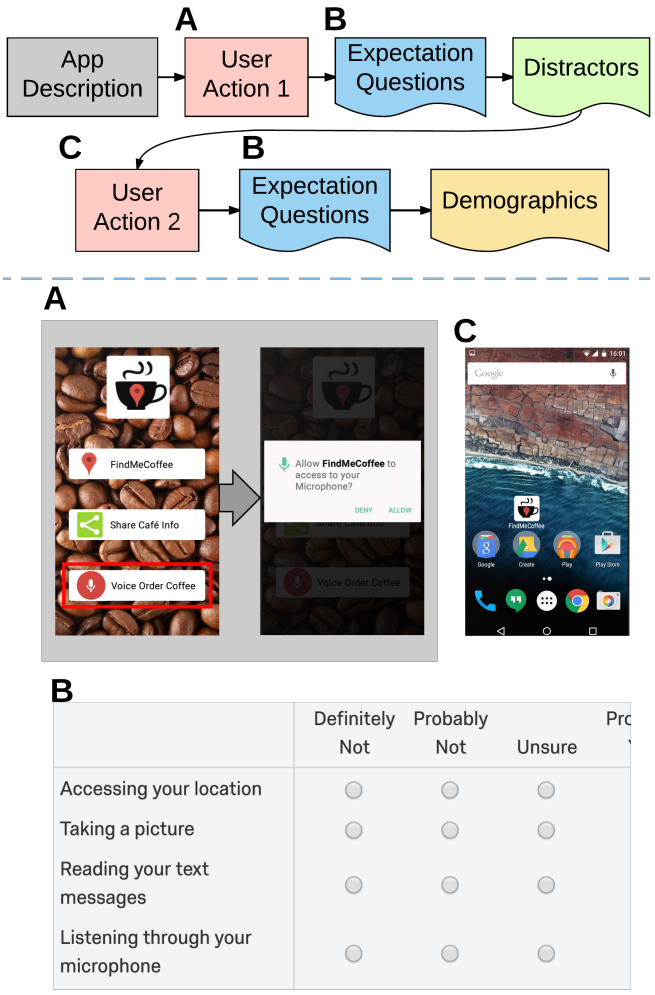

Figure 4: User expectations study procedure. Lower portion shows partial examples of user actions and survey questions.

are performed on a smartphone. Our goal was to understand to what extent user expectations align with the interactivity patterns we observed in our app study. Concretely, our user study examined the following three hypotheses:

H1. Users are more likely to expect resource accesses with an interactive use pattern than without.

H2. The more apps that use a resource (as measured in our app survey), the more likely users are to expect less-interactive uses of that resource.

H3. Users are more likely to expect resource accesses they have seen before.

Note that *H1* and *H3* have implications for the Android M permission system. Specifically, if *H1* is true, users already expect a resource to be accessed due to user action, so an explicit authorization request may be unnecessary. Additionally, if *H3* is false, then users granting authorization for a resource in one access pattern does not cause them to expect that resource to be used later in a different pattern. Thus, requesting authorization only on first use may be insufficient.

H2 is a proxy for a more general hypothesis: that users are more likely to expect accesses to resources with which they are familiar. Familiarity is likely affected by many factors including how many apps use the resource and how often they do so, whether passive notifications are present (as for location), coverage in the news media, etc. For simplicity, we use the frequencies in Table 1 as a metric of familiarity.

Study Overview

We recruited participants through Amazon's Mechanical Turk crowdsourcing service. All participants were at least 18 years old and located in the United States. Participants were paid $1.00 for completing the survey. The survey was approved by the University of Maryland IRB. Participants were instructed that we wanted their opinions about an app; privacy and sensitive resources were not explicitly mentioned.

Figure 4 gives a flowchart of the procedure followed by each study participant. First, the participant reads a short description of an app. We used two mock apps in our study: FindMeCoffee (*Coffee*) and HealthyFit Tracker (*Fitness*). FindMeCoffee locates nearby coffee shops, allows users to share the location of favorite coffee shops with friends, and supports ordering coffee via voice command. HealthyFit Tracker tracks workouts, allows sharing workouts with friends, and allows posting audio "smack talk" on the user's profile. Note while these apps demonstrate different categories, we do not attempt to fully study the effect of app type on user perceptions.

Next the participant views one sequence of user interactions with the app, shown as a slideshow of app screenshots. To avoid confusion with terminology in the next section, we refer to such a sequence as a *user action*. For example, in the user action in Figure 4A, the user clicks the bottom-most button (outlined in red), and the app then displays an authorization request dialog. In Figure 4C, which is only shown partially, the user exits the app and returns to the device home screen.

After viewing a user action, the participant answers five-point Likert questions such as those in Figure 4B, which ask whether a resource is "Definitely not" to "Definitely yes" accessed immediately after the user action. To avoid priming the user, the survey asks about camera, SMS, flashlight, "accessing credit card information," and "looking up coffee shop reviews" (*Coffee*) or "reading your heart rate" (*Fitness*) as well as the three resources we study.

Next, the participant answers five distractor questions, views another user action, and answers the same Likert questions about the second user action. The distractors are designed to induce a cognitive break and ensure the two access patterns are treated as separate events. For example, one distractor asks users "Which button would you press if you wanted to find a new coffee shop?" We did not measure the effectiveness of distractors. The participant concludes the survey by answering demographic questions. Participants took 4 minutes and 45 seconds on average to complete the survey.

To understand the effect of resource access patterns on user expectation, we analyze responses about the first user action each participant viewed. We use responses about the second user action to examine how prior exposure affects expectation.

Conditions

Participants were assigned round-robin to one of 42 conditions, which varied across four variables: the *app*, the *resource* being accessed, the *authorization pattern*, and the pair of *interaction (int) patterns*.[2] Table 2 lists possible values for each variable, and conditions comprise one value from each column.

As mentioned earlier, our study used two mock apps, FindMeCoffee and HealthyFitTracker, and three resources, chosen to cover a range of int. patterns observed in our app survey: Microphone (*Mic*, only interactive accesses observed), Contacts (*Con*, mixed interactive and background uses observed), and Location (*Loc*, also mixed).

Our study considered three different authorization patterns. First use (*Fst*) mimics Android M, presenting an authorization dialog during the first user action but not the second. Launch (*Lch*) presents an authorization dialog immediately after the app's home page is shown, but before any further screenshots. We noticed anecdotally that a few apps in our study used this strategy. Never (*Nvr*) does not show any authorization dialog at run time. This mimics older versions of Android.

We examined three different int. patterns, also drawn from the app survey: Button Click (*Clk*), Background with Notification (*Bn*, uses a resource without interaction, but displays an icon and short message in the notification drawer indicating some condition is met, such as being located near a coffee shop) and Background Only (*Bg*, uses a resource in a way not clearly shown in the UI). We can order these from most (*Clk*) to least (*Bg*) interactive. We always label the button clearly for its use (no deception). We do not test UI widgets besides buttons.

Regardless of a participant's assigned condition, the survey always asks about expectations for all resources and auxiliary actions. As a result, we implicitly collect data about users' expectations for the *Bg-Bg* int. pattern pair, with authorization pattern *Nvr*, for the two non-targeted resources in each condition. For example, a participant assigned to the *Coffee-Mic-Fst-Clk-Clk* condition also answers Likert questions about contacts and location that are analyzed within the *Coffee-Con-Nvr-Bg-Bg* and *Coffee-Loc-Nvr-Bg-Bg* conditions.

Testing the full-factorial combination of all variables was infeasible, so we eliminated conditions that were redundant or less relevant to our hypotheses, resulting in 42 final conditions. In more detail: We exclude conditions where the second int. pattern is more interactive than the first, as we assume the participant's second expectation would be dominated by the second int. pattern, rather than by the combination of patterns. We use *Bn* only in the first user action, because we are primarily interested in its effect on user expectations for the second user action. We assume expectations for *Bn* itself will depend entirely on whether the participant notices the passive cue, a topic that has been well studied [28, 31] but is somewhat orthogonal to our work. These two rules limit the int. pattern pairs we study to those in the last column of Table 2.

We exclude *Nvr-Bg-Bg* conditions because they provide no evidence of resource use, and are therefore identical to the implicit scenarios discussed above. We also exclude *Fst-Bg-Bg* because, in our experimental design, they are indistinguishable from *Lch-Bg-Bg*. In Table 2, *Bg-Bg* is highlighted in blue to indicate that it is only used with the *Lch* authorization pattern.

Because we do not comprehensively consider the effect of app type, we limit *Fitness* conditions to those we anticipated would have the largest variation in expectations. Specifically, we consider only *Loc* and *Mic* and only conditions where the two user actions exhibit different int. patterns. The *Fitness* scenarios therefore include all combinations of variables in Table 2 that not are not highlighted in blue or yellow.

[2]Through this section we will use the abbreviation int. pattern to avoid confusion with the interactions in our regression analysis.

App	Resource[1]	Authorization[2]	Int. Patterns[3]
Coffee	*Mic*	*Fst*	*Clk-Clk* [4]
Fitness	*Con* [4]	*Lch*	*Clk-Bg*
	Loc	*Nvr*	*Bn-Bg*
			Bg-Bg [4,5]

[1] *Mic* - Microphone, *Con* - Contacts, *Loc* - Location
[2] *Fst* - First, *Lch* - Launch, *Nvr* - Never
[3] *Clk* - Click, *Bn* - Background w/Notif., *Bg* - Background Only
[4] Only used with *Coffee*
[5] Only used with *Lch*

Table 2: Possible values for each variable in tested conditions.

Statistical Analysis

To test *H1* and *H2*, we primarily consider the expectations expressed by participants after the first user action. We use a logistic regression, appropriate for ordinal Likert data, to estimate the effect of the app, resource, authorization pattern, and int. pattern on participants' expectation. To test *H3*, which concerns the effect of the prior user action, we also use a logistic regression, with participants' expectation for the second user action as the outcome variable. We use the same input factors as before, this time including both int. patterns.

Each regression includes multiple observations of each participant: the explicit condition plus two implicit *Nvr-Bg-Bg* responses. As is standard, we include a mixed-model random effect that groups observations from the same participant [16].

Our initial model for each regression included the input variables plus all possible two-way interaction terms. To prevent overfitting, we tested all possible combinations of these inputs and selected the model with minimum Akaike Information Criterion (AIC), a standard measure of model quality [2]. We present only the final model for each regression.

Ecological Validity and Limitations

We use a controlled experiment with mock apps. While this limits ecological validity, it allows us to reason statistically about the effect of specific factors on participants' expectations, and to disregard factors such as participants' history with an app or reputation of the app's developer. In a study environment, participants may be less suspicious than if their real data were at risk. To partially account for this, we ask about expectation rather than comfort level [23].

By limiting our survey to two apps and restricting the int. patterns and resources tested, we likely miss factors, and particularly combinations of factors, that influence expectations. As one example, users likely expect SMS and Calls to have different usage patterns, and these expectations may vary with app type. However, based on the results of our app survey, we believe the variables chosen still provide useful insights.

Each participant sees two user actions in a relatively short time period. We do not study the effect of longer sequences of actions or long-term use (e.g., over days or weeks) of an app.

As is typical for online studies and self-reported data, some participants may not take the survey seriously, and some may try to complete the survey multiple times. We limit repeat participants using MTurk ID and browser cookies. While MTurk has generally been validated for high-quality data [7, 9, 20, 32], U.S. MTurkers are somewhat younger and more male, tech-savvy, and privacy-sensitive than the general population, which may limit the generalizability of our results [18].

These limitations apply similarly across all conditions; we therefore consider comparisons among conditions to be valid.

USER EXPECTATIONS STUDY RESULTS

We now present the results of our online user study. We found that *H1* holds: users were the most likely to expect a resource use when shown a more interactive int. pattern. In contrast, we observed that while resource type does affect user expectation, *H2* was not strongly supported. Finally, we found that *H3* does not hold. However, our results indicate that both background notification (*Bn*) and on-launch authorization requests (*Lch*) increase user expectation of future resource accesses.

For each of our hypotheses, we present key findings from our regression analysis. Summaries of our regressions are shown in Tables 3 and 4. Each table shows the included input variables and their values. Each variable includes a *base case* value (identified by dashes in the remaining columns). The odds ratio (OR) shows the observed effect of each value relative to the base case, measured as odds of expectation increasing one unit on our Likert scale. We also provide the 95% confidence interval (CI) and p-value for each measurement.

For example, the third row of Table 3 shows that switching from *Bg* to *Clk* in the first user action, all other variables held constant, is associated with a 106.3× likelihood of increasing one unit of expectation. The CI expresses 95% confidence that the "true" odds ratio is between 63.6 and 177.7. A p-value less than 0.05 is interpreted as statistically significant.

The second half of each table shows interactions between value pairs, given as the two values separated by a colon. These odds ratio indicate the change in likelihood when the two variables co-occur, relative to considering them independently. For example, the *Loc*:*Clk* odds ratio of 0.2 in Table 3 suggests the combined effect of these two values is *subadditive*: only 20% as strong as predicted by their individual effects.

Demographics

A total of 961 participants completed the study. Participants' ages ranged from 18 to 70+ years, with 37% age 18-29. Fifty-three percent reported being male and 47% female. ("Prefer not to answer" and "other" options for gender were provided.) Participants were required to be from the United States. Forty-five percent of participants reported holding a college degree, and 25% reported having "far above average" smartphone expertise. Each condition had at least 20 unique participants. Twenty people dropped out partway through the survey, distributed evenly across conditions.

Variable	Value	Odds Ratio	CI	p-value
App	*Coffee*	–	–	–
	Fitness	1.3	[0.96, 1.78]	0.086
Int	*Bg*	–	–	–
	Clk	**106.3**	**[63.6, 177.7]**	**< 0.001***
	Bn	**4.1**	**[2.6, 6.7]**	**< 0.001***
Res	*Mic*	–	–	–
	Loc	**17.5**	**[13.4, 22.9]**	**< 0.001***
	Con	0.8	[0.6, 1.0]	0.056
Auth	*Nvr*	–	–	–
	Fst	**2.2**	**[1.2, 4.0]**	**0.008***
	Lch	**1.9**	**[1.2, 3.2]**	**0.008***
App:Res	*Coffee*:*Mic*	–	–	–
	Fitness*:*Loc	**0.4**	**[0.3, 0.6]**	**< 0.001***
	Fitness:*Con*	1.1	[0.8, 1.7]	0.546
Res:Auth	*Mic*:*Nvr*	–	–	–
	Con*:*Lch	**3.2**	**[1.5, 6.7]**	**0.002***
	Con:*Fst*	1.5	[0.6, 3.6]	0.41
	Loc:*Lch*	0.8	[0.4, 1.6]	0.487
	Loc:*Fst*	0.5	[0.2, 1.3]	0.166
Res:Int	*Mic*:*Bg*	–	–	–
	Loc*:*Bn	**2.4**	**[1.2, 5.0]**	**0.021***
	Con*:*Bn	**5.0**	**[2.3, 11.3]**	**< 0.001***
	Loc*:*Clk	**0.2**	**[0.1, 0.4]**	**< 0.001***
	Con*:*Clk	**0.2**	**[0.1, 0.4]**	**< 0.001***

*Significant effect – Base case (OR=1, definitionally)

Table 3: Summary of regression over participant expectations after the first user action.

H1 – Interactivity v. Expectation

We found that *H1* holds: the more interactive the int. pattern, the more likely the user is to expect the resource access. In fact, interactivity (specifically *Clk*) is the strongest indicator of expectation we measured.

Table 3 shows that both *Clk* and *Bn* significantly increase the likelihood the user expects a resource access compared to *Bg*. The effect of *Clk* is particularly strong (OR 106.3, $p < 0.001$). Because the confidence intervals of *Clk* and *Bn* do not overlap, we can also conclude *Clk* generates significantly more expectation than *Bn*. Table 4 confirms that the strong association between *Clk* and expectation holds for the second user action as well (OR 810.4, $p < 0.001$). Figure 5a illustrates this finding, showing that 90% of participants who saw a *Clk* int. pattern first definitely or probably expected the associated resource use, compared to 72% for *Bn* and 25% for *Bg*.

Explicit authorization, which is by definition interactive, is also associated with a significant increase in expectation. Compared to *Nvr*, both *Fst* and *Lch* have odds ratio effects of about

Variable	Value	Odds Ratio	CI	*p*-value
Int 2	*Bg*	–	–	–
	Clk	**810.4**	**[352.2, 1864.9]**	**< 0.001***
Int 1	*Bg*	–	–	–
	Clk	1.0	[0.8, 1.4]	0.9
	Bn	**2.1**	**[1.5, 2.8]**	**< 0.001***
Res	*Mic*	–	–	–
	Loc	**20.0**	**[15.5, 25.9]**	**< 0.001***
	Con	1.0	[0.8, 1.3]	0.768
Auth	*Nvr*	–	–	–
	Fst	1.4	[0.9, 2.4]	0.174
	Lch	**1.7**	**[1.1, 2.7]**	**0.027***
Res:Auth	*Mic*:*Nvr*	–	–	–
	Con*:*Lch	**4.4**	**[2.2,8.5]**	**< 0.001***
	Con:*Fst*	2.1	[1, 4.5]	0.056
	Loc:*Lch*	0.8	[0.4, 1.5]	0.445
	Loc*:*Fst	**0.5**	**[0.2, 0.9]**	**0.029***
Res:Int 2	*Mic*:*Bg*	–	–	–
	Loc*:*Clk	**0.1**	**[0.03, 0.3]**	**< 0.001***
	Con*:*Clk	**0.03**	**[0.01, 0.1]**	**< 0.001***

*Significant effect – Base case (OR=1, definitionally)

Table 4: Summary of regression over participant expectations after the second user action.

2× for the first user action (Table 3), both significant. This result makes intuitive sense, as the dialogs for both (*Fst* and *Lch*) occur very closely in time to the first int. pattern.

H2 – Real-World Frequency v. Expectation

We observed an inconsistent relationship between real-world frequency and expectation. Location was the most frequently seen resource in the app study (75 apps according to Table 1) and was also the most expected resource we tested. As shown in Tables 3 and 4, *Loc* was associated with 17-22× higher likelihood of expectation compared to *Con* (seen in 30 apps) or *Mic* (seen in 14 apps). Non-overlapping CIs and *p*-values less than 0.05 indicate these effects are significant. This is consistent with *H2*; however, as our frequency measurement is only a very rough approximation of user familiarity, we note that this effect could relate to existing passive notifications for location, the higher volume of academic and media coverage of the location permission, or other factors.

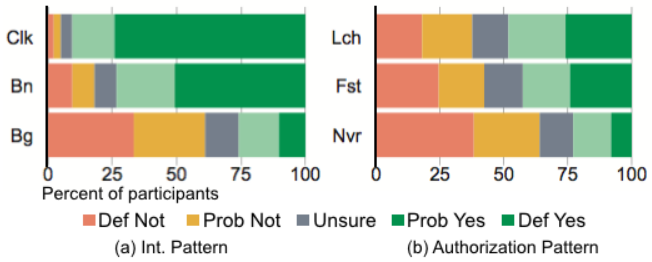

Figure 5: Likert-scale expectation responses for (a) the first user action, organized by int. pattern, and (b) the second user action, organized by authorization pattern.

However, *H2* does not hold when comparing *Con* and *Mic*. While *Con* was seen about twice as often in the app study, there was no significant difference in expectation between the two resources, at either the first or second user action. This could be because our app frequency metric does not sufficiently capture differences (or in this case, similarities) in users' overall exposure to each resource.

While there was no significant difference between the main effects of *Con* and *Mic*, we found evidence that users rarely expected *Mic* to be used with *Bg*. Looking at the *Res:Int* interaction in Table 3, *Loc*:*Bn* and *Con*:*Bn* both show significant superadditive results, indicating that these combinations are even more expected, relative to the baseline *Mic*:*Bg* combination, than those factors' main effects would predict. We hypothesize this difference is driven more by low expectation for background microphone access than high expectations for the other combinations. The interactions involving *Clk* are significantly subadditive, but we suspect this is a ceiling effect: expectations for *Clk*, which as shown in Figure 5a are very high, cannot increase beyond the end of our Likert scale. We see similar effects for the *Clk* interactions in Table 4.

H3 – Effect of Previously Seen Accesses

To examine the effects of prior accesses on participants' expectations, we focus on Table 4. Overall, we find that *H3* does not hold: participants are not more likely to expect resource accesses they have seen before. In particular, the int. pattern variable *Int1:Int2* does not appear in the final model, suggesting the combination of int. patterns does not meaningfully influence expectation at the second user action.

However, participants did appear more likely to expect a background access if they had previously seen an indication that background accesses might be used. Table 4 shows expectation at the second user action was significantly higher when the first int. pattern was *Bn* (OR 2.1, $p < 0.001$) or the *Lch* authorization request was shown (OR 1.7, $p = 0.027$), both of which indicate that background access may occur. Figure 5b illustrates this finding, showing that in the second user action, more participants definitely or probably expected resource usage in *Lch* conditions (47%) than in *Fst* (42%) or *Nvr* (22%). Additionally, the superadditive relationship between *Lch* and *Con* in both regressions may suggest a *Lch* request primes users to expect non-interactive accesses to contacts.

In contrast, we found evidence that the authorization pattern *Fst* implies only a single access. Table 4 shows that *Fst* did not have a significant effect ($p = 0.174$) compared to *Nvr*, indicating that an authorization associated with the first user action is no more effective than no authorization when thinking about a second user action. The subadditive relationship between *Loc* and *Fst* (OR 0.5, $p = 0.029$) also suggests that authorization requests associated with a specific event train users to only expect a resource access after an authorization request. This decreases the otherwise relatively high expectation of background location access. These relationships, while not particularly strong, are notable because they imply that the Android M model may in some cases be counterproductive; we explore this further in the Discussion section below.

App v. Expectation

Finally, we observed that the app type did have some effect on the way other variables were perceived. We found no significant difference between the two mock apps with respect to the first user action ($p = 0.086$), and app effects did not appear in the final model for the second user action, suggesting no meaningful relationship with expectation. However, we do observe in Table 3 a significant, subadditive relationship between *Fitness* and *Loc*, indicating that location accesses were less expected in the fitness app than the coffee app. We expect that across a wider variety of apps, further relationships between app type and expected resource usage would be observable.

CONCLUSIONS AND DESIGN RECOMMENDATIONS

Based on our app survey and user study, we propose several ways to improve Android's and similar authorization systems.

Access Resources As Interactively As Possible

Our app study found that camera, microphone, media, and calendar are already used almost exclusively interactively in popular apps. Moreover, our user study found that users overwhelmingly expect resource access after an explicit click on a related item. We speculate that users would also have higher expectations of resource access under other interactive uses.

Thus, we recommend expecting (or perhaps even requiring) most or all accesses to these four resources to be interactive. Other resources are more frequently used in the background, and it is not always obvious how to associate their uses with a foreground interaction. However, we argue that developers should prioritize (and the Android framework should encourage) making background uses more interactive when possible. For example, a social media app that periodically pulls the user's contacts to recommend new friends could tie this background access to a foreground interaction by asking the user to "turn on" this feature at launch and provide a settings menu where the user could turn the feature off at a later time. While this design is similar to the authorization mechanism provided by Android M, implementing it in the app provides context, which we have found is important to decision making.

We also recommend that when resources that are more typically used interactively will be used in the background, these uses should be documented explicitly in the app's description or a similar user-visible location.

Use Interactions to Grant Authorization

Our results further suggest that if a resource use is interactive, then a separate authorization dialog can be eliminated. We speculate that removing explicit authorization requests in these cases could reduce potential user confusion (e.g., "I just clicked 'Import Contacts,' why is it asking me if I want the app to access contacts?"). In addition, removing these requests could reduce annoyance and habituation, potentially helping the user to focus on other, less clear authorization decisions. Eliminating request dialogs for interactive use cases could also help motivate developers to prioritize interactivity, as mentioned above. Of course, to handle potentially malicious apps we must be sure the preceding interaction is clearly related to the resource use. For example, clicking on a location icon should not cause the camera to be used.

We envision two main approaches to enforce this principle. One idea is access-control gadgets [26, 27], which we discussed with Related Work. Another approach would be to leave apps as they are, but use a program analysis to ensure they conform. For example, we could use AppTracer to do so, in one of two possible modes. AppTracer could be run ahead of time, e.g., by an app market gatekeeper, to examine app behavior. Even though it would not necessarily observe all app behavior, results from analyzing behaviors users actually encounter would still be useful. AppTracer could also be used for auditing: power users and security experts could use AppTracer to log an app's behavior as they use it and then retroactively check the AppTracer graphs for any suspicious behavior, which could then be reported to the broader public.

Handle background authorization separately

We found that users were much less likely to expect background resource access if authorization dialogs were presented after a prior user action or were not presented at all. Thus, we recommend requesting authorization separately and explicitly for background uses. Based on our study, it may be preferable to do so when apps are first launched. However, because our study showed the increase in expectation is small (especially compared to the expectation after a click), it may be important to also show background notifications of use (which also increased expectation) so users remain aware. We note that while authorization on launch informs many users that background accesses should be anticipated, Figure 5 suggests there are others who do not recognize this possibility. Further research into the best approach is still needed.

Resources that have a broad mix of interactive and background uses—such as contacts—might particularly benefit from separate background authorization and limited requests for interactive uses. This could help avoid potential misconceptions about interactive uses being the only uses.

Future work could shed light on how differences in background use cases affect user expectations and preferences. For example, users might be expected to react differently when contacts are accessed in the background for pre-fetch (a use case identified in our app survey) compared to advertising. We also expect that the frequency of access will impact the user's decision. For example, it may be possible to tie high-frequency background uses to some foreground passive notification (e.g., a notification tray icon), similarly to the design presented by Balebako et al. for informing the user of data leakage [5]. This could make the user aware of the accesses without requiring additional authorization effort. Program analysis tools such as AppTracer could potentially be used to separate these cases and apply different authorization policies accordingly.

ACKNOWLEDGMENTS

We thank the anonymous reviewers, Sascha Fahl, Yasemin Acar, and members of HCIL for their helpful feedback. This research was supported in part by NSF CNS-1064997, a UMIACS contract under the partnership between the University of Maryland and DoD, and a Google Research Award.

REFERENCES

1. 2012. How to Ask for Permission. In *Presented as part of the 7th USENIX Workshop on Hot Topics in Security*. USENIX, Berkeley, CA. https://www.usenix.org/conference/hotsec12/workshop-program/presentation/Felt
2. Hirotugu Akaike. 1974. A new look at the statistical model identification. *Automatic Control, IEEE Transactions on* 19, 6 (06 Dec. 1974), 716–723. DOI: http://dx.doi.org/10.1109/tac.1974.1100705
3. Hazim Almuhimedi, Florian Schaub, Norman Sadeh, Idris Adjerid, Alessandro Acquisti, Joshua Gluck, Lorrie Faith Cranor, and Yuvraj Agarwal. 2015. Your Location Has Been Shared 5,398 Times!: A Field Study on Mobile App Privacy Nudging. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 787–796. DOI: http://dx.doi.org/10.1145/2702123.2702210
4. Kathy Wain Yee Au, Yi Fan Zhou, Zhen Huang, and David Lie. 2012. PScout: Analyzing the Android Permission Specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12)*. ACM, New York, NY, USA, 217–228. DOI:http://dx.doi.org/10.1145/2382196.2382222
5. Rebecca Balebako, Jaeyeon Jung, Wei Lu, Lorrie Faith Cranor, and Carolyn Nguyen. 2013. "Little Brothers Watching You": Raising Awareness of Data Leaks on Smartphones. In *Proceedings of the Ninth Symposium on Usable Privacy and Security (SOUPS '13)*. ACM, New York, NY, USA, Article 12, 11 pages. DOI: http://dx.doi.org/10.1145/2501604.2501616
6. Sven Bugiel, Stephen Heuser, and Ahmad-Reza Sadeghi. 2013. Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security '13)*. USENIX, Washington, D.C., 131–146. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/bugiel
7. Michael Buhrmester, Tracy Kwang, and Samuel D. Gosling. 2011. Amazon's Mechanical Turk: A New Source of Inexpensive, Yet High-Quality, Data? *Perspectives on Psychological Science* 6, 1 (2011), 3–5. DOI:http://dx.doi.org/10.1177/1745691610393980
8. Kevin Zhijie Chen, Noah M. Johnson, Vijay D'Silva, Shuaifu Dai, Kyle MacNamara, Thomas R. Magrino, Edward XueJun Wu, Martin Rinard, and Dawn Xiaodong Song. 2013. Contextual Policy Enforcement in Android Applications with Permission Event Graphs. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013*. Internet Society, San Diego, California, USA. http://internetsociety.org/doc/contextual-policy-enforcement-android-applications-permission-event-graphs
9. Julie S. Downs, Mandy B. Holbrook, Steve Sheng, and Lorrie Faith Cranor. 2010. Are Your Participants Gaming the System?: Screening Mechanical Turk Workers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 2399–2402. DOI: http://dx.doi.org/10.1145/1753326.1753688
10. Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. 2011. Android Permissions Demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS '11)*. ACM, New York, NY, USA, 627–638. DOI: http://dx.doi.org/10.1145/2046707.2046779
11. Adrienne Porter Felt, Serge Egelman, and David Wagner. 2012a. I've Got 99 Problems, but Vibration Ain't One: A Survey of Smartphone Users' Concerns. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '12)*. ACM, New York, NY, USA, 33–44. DOI: http://dx.doi.org/10.1145/2381934.2381943
12. Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012b. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security (SOUPS '12)*. ACM, New York, NY, USA, Article 3, 14 pages. DOI: http://dx.doi.org/10.1145/2335356.2335360
13. Huiqing Fu, Yulong Yang, Nileema Shingte, Janne Lindqvist, and Marco Gruteser. 2014. A field study of run-time location access disclosures on android smartphones. In *Workshop on Usable Security (USEC)*. Internet Society, San Diego, California, USA.
14. Google 2016. *Requesting Permissions at Run Time*. Google. https://developer.android.com/training/permissions/requesting.html
15. Andrew F Hayes and Klaus Krippendorff. 2007. Answering the call for a standard reliability measure for coding data. *Communication methods and measures* 1, 1 (2007), 77–89. http://dx.doi.org/10.1080/19312450709336664
16. Donald Hedeker. 2008. Multilevel models for ordinal and nominal variables. In *Handbook of multilevel analysis*. Springer, 237–274. http://link.springer.com/chapter/10.1007/978-0-387-73186-5_6
17. Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. 2012. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*. ACM, Raleigh, NC, USA, 3–14.
18. Ruogu Kang, Stephanie Brown, Laura Dabbish, and Sara Kiesler. 2014. Privacy Attitudes of Mechanical Turk Workers and the U.S. Public. In *Proceedings of the 23rd*

USENIX Security Symposium (USENIX Security '14). USENIX Association, San Diego, California, USA, 37–49. https://www.usenix.org/conference/soups2014/proceedings/presentation/kang

19. Jennifer King. 2012. "How Come I'm Allowing Strangers To Go Through My Phone?": Smartphones and Privacy Expectations. In *Proceedings of the Ninth Symposium on Usable Privacy and Security (SOUPS '12)*. USENIX, Washington, DC, USA. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2493412
20. Aniket Kittur, Ed H. Chi, and Bongwon Suh. 2008. Crowdsourcing User Studies with Mechanical Turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 453–456. DOI: http://dx.doi.org/10.1145/1357054.1357127
21. Kristopher Micinski, Jonathan Fetter-Degges, Jinseong Jeon, Jeffrey S. Foster, and Michael R. Clarkson. 2015. *Checking Interaction-Based Declassification Policies for Android Using Symbolic Execution*. Springer International Publishing, Vienna, Austria, 520–538. DOI: http://dx.doi.org/10.1007/978-3-319-24177-7_26
22. Helen Nissenbaum. 2004. Privacy as Contextual Integrity. *Washington Law Review* 79 (2004), 119–157.
23. Ashwini Rao, Florian Schaub, Norman Sadeh, Alessandro Acquisti, and Ruogu Kang. 2016. Expecting the Unexpected: Understanding Mismatched Privacy Expectations Online. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO, 77–96. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/rao
24. Android Developers Reference. 2016. Manifest.permission_group. (2016). https://developer.android.com/guide/topics/security/permissions.html (Accessed 9-16-2016).
25. Talia Ringer, Dan Grossman, and Franziska Roesner. 2016. AUDACIOUS: User-Driven Access Control with Unmodified Operating Systems. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*. ACM, Vienna, Austria. http://tlringer.github.io/pdf/audacious.pdf
26. Franziska Roesner and Tadayoshi Kohno. 2013. Securing Embedded User Interfaces: Android and Beyond. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security '13)*. USENIX, Washington, D.C., 97–112. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/roesner
27. Franziska Roesner, Tadayoshi Kohno, Alexander Moshchuk, Bryan Parno, Helen J. Wang, and Crispin Cowan. 2012. User-Driven Access Control: Rethinking Permission Granting in Modern Operating Systems. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy (SP '12)*. IEEE Computer Society, Washington, DC, USA, 224–238. DOI: http://dx.doi.org/10.1109/SP.2012.24
28. Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The Emperor's New Security Indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*.
29. Marc Stiegler, Alan H. Karp, Ka-Ping Yee, Tyler Close, and Mark S. Miller. 2006. Polaris: Virus-safe Computing for Windows XP. *Commun. ACM* 49, 9 (Sept. 2006), 83–88. DOI:http://dx.doi.org/10.1145/1151030.1151033
30. Marc Stiegler and Mark S. Miller. 2002. *A Capability-based Client: The DarpaBrowser*. Technical Report Technical Report, Focused Research Topic 5. Combex Inc, Meadowbrook, PA. http://www.combex.com/papers/darpareport/index.html
31. Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying Wolf - An Empirical Study of SSL Warning Effectiveness.. In *Proceedings of the 18th USENIX Security Symposium*. {}https://www.usenix.org/legacy/events/sec09/tech/full_papers/sunshine.
32. Michael Toomim, Travis Kriplean, Claus Pörtner, and James Landay. 2011. Utility of Human-computer Interactions: Toward a Science of Preference Measurement. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2275–2284. DOI: http://dx.doi.org/10.1145/1978942.1979277
33. Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David Wagner, and Konstantin Beznosov. 2015. Android Permissions Remystified: A Field Study on Contextual Integrity. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security '15)*. USENIX Association, Washington, D.C., 499–514. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/wijesekera
34. Zhemin Yang, Min Yang, Yuan Zhang, Guofei Gu, Peng Ning, and X. Sean Wang. 2013. AppIntent: analyzing sensitive data transmission in android for privacy leakage detection. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security (CCS '13)*. ACM, New York, NY, USA, 1043–1054. DOI: http://dx.doi.org/10.1145/2508859.2516676