Reducing Permission Requests in Mobile Apps

Sai Teja Peddinti, Igor Bilogrevic, Nina Taft
Martin Pelikan, Úlfar Erlingsson, Pauline Anthonysamy, Giles Hogben
Google Inc.

ABSTRACT

Users of mobile apps sometimes express discomfort or concerns with what they see as unnecessary or intrusive permission requests by certain apps. However encouraging mobile app developers to request fewer permissions is challenging because there are many reasons why permissions are requested; furthermore, prior work [25] has shown it is hard to disambiguate the purpose of a particular permission with high certainty.

In this work we describe a novel, algorithmic mechanism intended to discourage mobile-app developers from asking for unnecessary permissions. Developers are incentivized by an automated alert, or "nudge", shown in the Google Play Console when their apps ask for permissions that are requested by very few functionally-similar apps—in other words, by their competition. Empirically, this incentive is effective, with significant developer response since its deployment. Permissions have been redacted by 59% of apps that were warned, and this attenuation has occurred broadly across both app categories and app popularity levels. Importantly, billions of users' app installs from the Google Play have benefited from these redactions.

CCS CONCEPTS

• **Security and privacy → Domain-specific security and privacy architectures**; *Economics of security and privacy*; • **Information systems →** *Data mining*; • **Computing methodologies →** Machine learning;

KEYWORDS

Mobile Apps, Permissions

ACM Reference Format:
Sai Teja Peddinti, Igor Bilogrevic, Nina Taft and Martin Pelikan, Úlfar Erlingsson, Pauline Anthonysamy, Giles Hogben. 2019. Reducing Permission Requests in Mobile Apps. In *Internet Measurement Conference (IMC '19), October 21–23, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3355369.3355584

1 INTRODUCTION

The Android ecosystem and Google Play are popular platforms with over 2 million apps and 2 billion active devices worldwide. Many apps require access to private or protected data on users' devices, which they request via the Android permissions system. Recent versions of Android (6.0 and higher) organize individual permissions into groups, such as *Storage*, *Contacts*, and *Location*. Users are asked to grant or deny a permission for an app at the level of these groups via a runtime prompt. These prompts are usually surfaced at the moment when an app needs a permission, and are therefore made in context.

There are a number of reasons why an app may request permissions outside of those needed for its core functionality, such as for analytics, personalization, testing, performance assessment, advertising (especially for free apps), or support for (unused) functionality in libraries that the app includes. Prior research has shown that many mobile apps request potentially unnecessary permissions [21, 27, 29] or permissions that are not directly related to their core functionality [2, 8, 17, 24, 28, 35], or use permissions in unexpected ways [21]. This has also been reported by the press [12, 33, 37]. Furthermore, several studies have documented user frustration with what is viewed as unnecessary permission requests [11, 18, 34, 35] and how this can lead to a feeling of erosion of privacy [10, 12].

Permission usage can differ greatly, as mobile-app developers comprise a large community with varying experience and disparate working environments. A 2018 survey that reached over 40,000 developers from 160 countries, showed that 49% had less than 5 years experience and 40% worked for organizations with less than 50 employees [9]. Small- to medium-sized businesses, or businesses with limited experience, may be less likely to have privacy experts on their teams, or understand the tradeoffs of designing with privacy in mind [7]. For example, well meaning developers that include third-party libraries in their code, may not realize that their app's manifest need not request all the permissions requested by a library, and may be unaware of the privacy implications of different permission requests. Indeed, the study in [19] showed that developers mostly use the default configuration of ad libraries, choose libraries based on popularity and ease-of-use rather than risk assessment, and feel themselves unable to address the risks. Our aim is to help such developers become more privacy-aware in their handling of app permissions.

Developers who are aware of how their users perceive excessive permission requests may be motivated to refrain from using permissions that aren't strictly needed, e.g., for the sake of their reputation. A perusal of app reviews can reveal comments about invasive and unnecessary permissions. Beyond complaints in app reviews, there are other reasons why a developer may remove a previously requested permission, including: (i) a change in a library they use; (ii) an update to the APIs associated with a permission such that the permission is no longer required; (iii) a change in the app functionality; (iv) Google's developer outreach efforts [4]; and (v) in response to negative press articles [37].

Uncovering whether a permission request is necessary or not, with certainty, has proved challenging; even powerful techniques like static and dynamic analysis methods do not offer comprehensive answers [25]. For instance, dynamic analysis has code coverage

IMC '19, October 21–23, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6948-0/19/10.
https://doi.org/10.1145/3355369.3355584

issues, and static analysis cannot examine code downloaded during runtime. Thus, the decision as to whether or not to ask for a permission should ultimately be left up to each developer.

Our first contribution is an automated algorithmic approach that uncovers situations in which there is a significant chance a permission request is unnecessary. We inform developers via a nudge. Nudges are a well-known technique from behavioral economics which have been used in many contexts to encourage positive behavior without being punitive [32]. Recently researchers have started studying nudges surfaced to users to assist them with decision making [3, 18]. We focus on nudges that target developers.

To incentivize developers to act on our nudge, we employ the following metric. For a given app, we let the developer know when other apps with very similar functionality refrain from requesting a particular permission. For each specific app, we compute a 'peer group' of functionally-similar apps, automatically determining similarity from textual app descriptions and Google Play user-behavior data (as described in Section 2.2). If a specific app requests a permission that nearly none of its peers request, then we inform the developer of this and also remind the developer of research showing that users are more likely to select apps that request fewer permissions, when given a choice [15].

Previous research studies have explored other ways to give developers privacy related feedback, such as giving apps privacy grades [23], giving privacy policies grades [31, 36], or providing privacy risk metrics on public websites [6, 26]. We chose to explore giving developers feedback via the tools they use to create and manage their apps, such as Android Studio, Play Console, GitHub, and the Gradle build system, to name just a few. These tools provide opportunities to surface nudges. We deployed our privacy warning in August 2017 as part of the Pre-Launch Report shown in the Play Console.

Our second contribution is an assessment on the effectiveness of our nudge. After the live deployment of the nudge, we observed that 59% of apps who received a warning did indeed remove permissions. These removals occurred across all app categories, all app popularity levels, and over a broad set of permission types. This demonstrates developers' willingness to remove permissions when it is pointed out to them. We show that the removal of these permission requests, in aggregate, affected over 55 billion app installs. We also show that the existing permission redaction activity that happens for other reasons is significantly boosted by our warnings.

Our warning is one component of Google's larger strategy to protect users and help developers achieve good security and privacy practices. One component focuses on device security, with services such as Play Protect[1] that offer malware protection services for Android. A second component focuses on robust enforcement of Google Play's user data policies, which require developers to provide clear notice and control over collection of data in their apps, as well as recent policy changes further limiting developers' ability to request access to certain permissions. For instance, Google Play announced (in October 2018) further limitations of apps' ability to request Call Log and SMS permissions on Android devices [30] . A third component aims at educating developers to adopt better practices. In addition to our privacy warning, other signals have also been incorporated into the Play Console, such as warnings that discourage use of HTTP and permanent identifiers. Also, Lint warnings are surfaced in the Android Studio IDE to alert developers if their app is using a version of a library that has been identified by the library developer as a potential source of privacy and/or security risks [1]. In isolation, each approach has its own benefits and limitations, yet together they're all complementary.

[1] https://www.android.com/play-protect/

2 NUDGING DEVELOPERS

2.1 Our Approach

Developers interact with Google Play via the Play Console [5], both before and after launching an app. This console includes a 'Pre-Launch Report', accessible to developers who submit apps for testing, that surfaces the results of automated tests on app APKs (e.g. that identify performance issues and many other things) before the app is published on Google Play. Our approach to incentivize developers to avoid requesting permissions that aren't strictly necessary is to show them a motivating warning in this Pre-Launch Report.

Consider a developer who asks for a specific permission in their new app. We compute a set of functionally similar apps and check if this set of apps also ask for the same specific permission. The permissions in the set of *functionally-similar* apps, or *peer* apps, provide a baseline for the set of permissions needed for an app, as well as a baseline for user expectations about which permissions make sense for the app to request. Hence if nearly all of their competition does *not* ask for the same permission, then we let developer know. We thus make it easier for developers to assess their needs as compared to their peers. We leave the decision to the developer as we recognize there may be other specific reasons for the permission.

Your app is requesting the permission, <permission _name>, which is used by less than X% of functionally similar apps. *<number> functionally similar apps which initially requested <permission_name> have stopped requesting it.* *Users prefer apps that request fewer permissions and requesting unnecessary permissions can affect your app's visibility on Google Play. If these permissions aren't necessary, you may be able to use alternative methods in your app and request fewer permissions. If they are, we recommend providing an explanation to users of why you need the permissions. Learn more.*

Table 1: Privacy warning shown to developers

The warning message we show developers is depicted in Table 1. We point out a few properties of this warning. First, developers can choose to adhere to or ignore it. If a developer ignores the warning, then it will re-appear in the report for the next version of their app as long as the conditions for the signal remain true (e.g. peer groups would be recomputed at that time). Second, we further motivate developers by reminding them that users prefer apps with fewer permissions and that the perception of a permission request being unnecessary could affect their installs. Third, we recommend

providing an explanation for the permission request. Fourth, parameter X is a design choice that influences how conservative the warning aims to be. A very small value, such as 1%, ensures that the developers receiving the warnings are highly unlikely to need the permission.

2.2 Finding Similar Apps

The simplest approach would be to identify peer groups using Google Play app categories; but this is too coarse-grained as apps within the same category can offer very different functionality. For example, the category 'Travel and Local' contains navigation apps but also hotel reservation apps and tour guides. The category 'Auto' contains car software as well as apps that help users buy a car.

Another potential method for identifying similar apps could be based on user behavior while browsing in Google Play. When users look at a particular app on Google Play, suggestions of other apps that users may want to install are also shown. When users click on these suggestions, it may indicate that the user thought the clicked app is similar or related to the original one they were viewing. A method based on clustering analysis of these user co-clicks, (called UBC for *user behavior clustering* method), iteratively improves over time. However, the UBC data alone is not sufficient for our task for a number of reasons. First, the UBC method is optimized to find interesting suggestions, and not to find functionally-similar apps; for example, for a game app the UBC method may suggest a game discussion-forum app, or a game media editor. Second, to model user preferences, the UBC method favors apps in the same primary language, designed for the same locale. We do not want to limit our peer app assessment by language since app functionality is often independent of language. For example, the peer group of a French email app could include a Japanese email app. Finally and perhaps most importantly, the UBC method fails to provide any suggestions for brand-new apps or very unpopular apps, since Google Play has no user-behavior data for these cases.

Prior work has suggested analyzing the app description text via LDA analysis to identify apps with similar functionality [14, 16]. Our approach follows this direction but, as explained below, we use a different model and supplement the basic app descriptions with user co-click data. Further below we compare our approach to a pure LDA-based one.

Design of our App Peer Group Mechanism: To determine when to surface a nudge in a Pre-Launch Report, we must be able to compute app peer groups for all apps, including brand-new apps, unpopular ones, and do so for apps in any language. We have developed a deep-learning algorithm that creates an *embedding* based on word2vec for each app, thereby mapping apps into a high-dimensional space where closeness in distance corresponds to similarity [13, 20].

The key input data used to compute our embedding is textual, notably the apps' description, as well as the app name and category. If needed, this text is translated to English, to handle apps in any of the 100+ languages supported by Google Translate. In addition to text, the embedding is also computed based on UBC data; this is done by incorporating the top 10 UBC-related apps for each app, unless no such UBC-related peers are available. We note that the UBC data we work with is highly aggregated and contains no personal information. The data is in the form of app pairs with an accompanying probability on the likelihood that the second app would be clicked if the first is already shown.

The training data contains several rows of text for each app and are created as follows. Each sentence from the app description constitutes one row. The app title and category each form a separate row. Each app is given a unique identifier token (a unique word) that is prepended to all rows associated with that app. To incorporate the UBC ranking data, we include a single row with the identifier tokens of the top 10 apps considered most related by the UBC algorithm. Each English word is also a unique token. Thus, the complete set of training-data tokens includes both unique tokens representing apps, as well as vocabulary tokens for the words in the app descriptions, titles, and categories.

The training data is used to train a skip-gram language model, which uses a log-linear classifier with softmax to estimate the probability distribution of tokens that come within a certain range before and after the current token [20]. Thus, the input of the model is a token and the output is the probability distribution of the context in the form of surrounding words. Each token is represented by an n-dimensional vector and all these vectors comprise the projection matrix with v rows and n columns, where v is the vocabulary size (number of all tokens). For an input word w, the probability of a word c being in the context (surrounding words) decreases exponentially fast with cosine distance of w and c. A univariate model defines the probability of several context words as the product of the individual probabilities of these words. The training optimizes the vectors so that the product of all the context probabilities is maximized for the training data. Thus, words that are often located near each other end up with vectors that are close to each other according to the cosine distance. Similarly, vectors of closely related apps end up close to each other. For example, chess and checkers apps will be close to each other in the vector space because words such as *board*, *game*, and *piece* co-occur near each other and are also close to both app tokens in the vector space.

We use 200-dimensional vectors to represent all Android app tokens and also other tokens in the training data (words from descriptions, titles, and categories). The vocabulary size v (number of tokens) is in the millions. Our models are trained using stochastic gradient descent. To make the training of an embedding model for millions of tokens feasible, we use negative sampling as described in [20].

To measure the similarity between two apps, we use the cosine distance of their app vectors from the word2vec algorithm. The peers of an app are its nearest neighbors based on this distance. For example, with our model, the closest peer of Gmail is another email app with cosine distance of 0.41. Weakly related Android apps have greater cosine distances from Gmail: such as 0.68 for a messenger app, 0.80 for an internet browser, and 0.85 for a chess app. Our peer identification algorithm works well across all app popularity levels as the distribution of peer similarity scores is very similar.

We believe that developers have little motivation to circumvent our system. Because our warning signal is shown only to the developer, there is no public effect and thus no impact to their reputation. In theory, developers could attempt to influence the warning by creating a large number of apps, each of which asks for all the permissions they want, and has similar text descriptions. That said,

developers would not know a priori how many such apps are required to affect our algorithm.

Evaluating App Similarity within our Peer Groups. We compare our mechanism against two approaches: the UBC algorithm alone and a Latent Dirichlet Allocation (LDA) using 50 topics on normalized app descriptions [14]. We conducted two rounds of evaluation. First, we had 17 human evaluators participate in a survey. We split apps into 5 buckets based on the number of installs, and then select several apps from each bucket. This ensures that our analysis provides feedback across app popularity levels. For each selected app, we chose the top 3 peers from each of the 3 approaches (if available), removing duplicates. These at most 9 peers for each app are then randomly shuffled and presented to raters without any information about the source model or the original rank of each peer. Raters are shown a pair of apps and asked how similar they think the apps are, on a scale from 1 (not related) to 4 (nearly identical). We normalize the feedback values to $[0, 1]$ with higher values indicating higher similarity. These evaluators rated over 400 app pairs. Each pair of apps was rated by at least 3 evaluators. Our evaluators were all Google employees recruited via internal advertisement, which resulted in employees from numerous groups. They were told the purpose of the evaluation - to compare the quality of app similarity algorithms - but were not told how they work and had no way to know which algorithm produced which pair of apps presented. The evaluators were told they could use any publicly information available on Google Play to inform their decision about the similarity of the presented app pairs.

Note that the app pairs should all have high similarity since we pair each app with one that was a top-3 match from one of the models. The average app similarity for app pairs proposed by our word2vec-based algorithm was 0.81. Our algorithm significantly outperformed the LDA method that received an average app similarity of 0.51. The UBC method received an average similarity of 0.79 which is comparable to our approach, however, as explained earlier, it has much lower coverage ; in particular, it didn't provide any suggestions for approximately one in ten of the apps in our test set (because they were either new or unpopular).

To further compare our proposed model with UBC method, we conducted a second round of evaluation. For each app selected we picked the top 3 peers plus 2 other peers randomly selected from the closest 30 peers (if available), from each method, resulting in at most 10 app pairs. Similar to the earlier setup, these app pairs are shuffled and presented to the evaluators, who are asked to rate how similar the two apps are on a scale from 1 to 4. To increase the app coverage, each app pair is only rated by 1 evaluator in this round. Our evaluators rated over 1450 app pairs. The average similarity based on the normalized feedback values is 0.82 for our proposed model, compared to 0.66 for the UBC model. This further re-enforces that our deep-learning algorithm outperforms prior approaches. Beyond using a powerful approach such as deep-learning, it is intuitive that our approach works best because LDA relies solely on text descriptions and the UBC approach relies solely on co-clicks, whereas our approach uses both of these input signals.

3 EFFECTIVENESS OF NUDGES

Using our algorithm above to detect similar apps, we implemented the privacy nudge described in Section 2.1 for Android Platform related permissions and released it in the Play Pre-Launch Report, along with a blog post, in August 2017 [22]. We imposed a cutoff on the peer similarity scores to produce between 150-200 high quality peers per app. The distribution of peers per app is shown in Figure 1, and as is seen, a very small fraction of apps have less than 190 peers.

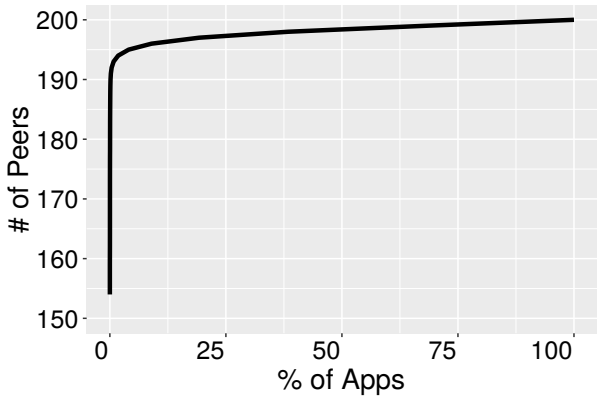

Figure 1: Number of peers per app

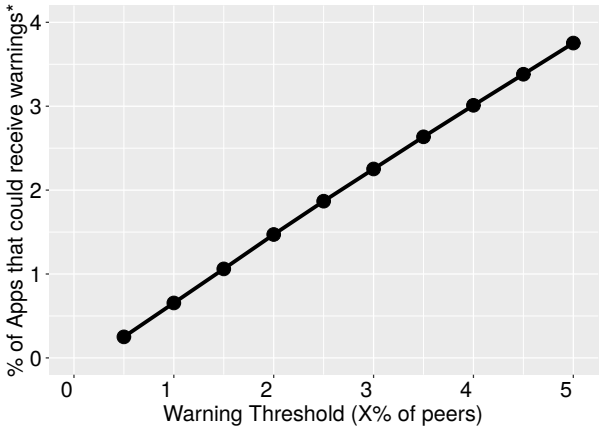

Figure 2: Percentage of apps that could receive privacy warnings at different thresholds (*only apps with peers and certain minimum installs are considered)

To guide the choice of the threshold ('X'% of peers not using a permission) that determines when to raise the privacy warning, we estimated the percentage of apps with peers that could receive a warning at different thresholds between 1% to 5%. Note that not all apps are included in the calculation of peers sets; for instance, we require an app to have a minimum number of installs to be included. The results are outlined in Figure 2. We initially elected to use a highly conservative threshold of 'less than 1%' of peer apps using a permission to raise the privacy warning. We opted for this conservative threshold for two reasons. First this ensures that each warning signal has high fidelity, i.e. developers are nudged only when we are quite confident that their behavior is significantly out of the ordinary. Second, we elected to start conservatively to

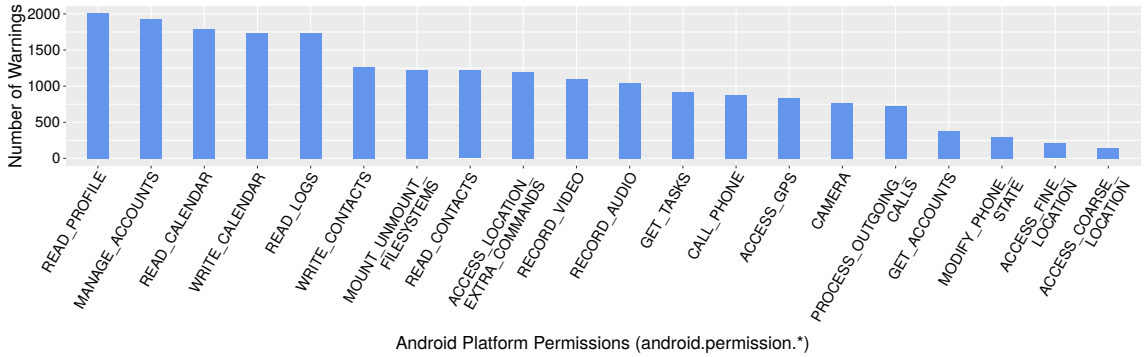

Figure 3: Permissions warned (20 permissions with more than 100 warnings each are shown)

explore whether developers would respond to such a warning (we wanted to avoid de-sensitizing them). Based on positive developer reaction, later in September 2018, we relaxed the threshold to 3%. Since August 2017, every new test APK submission to Google Play, among the millions participating in our testing framework, has been evaluated by our system. We evaluate our threshold's effectiveness on an ongoing basis and may consider raising the threshold further in the future.

Overall Response. Between August 2017 and Feb 2019 (inclusive), privacy warnings were raised for 28,446 permissions requested by 19,215 apps, which came from 15,645 developers. Among the warned apps, we observed that 11,289 apps (59%) apps, developed by 9313 developers, removed at least one permission after seeing our warnings. If we look at individual permissions, of the 28,446 warned permissions, only 5,725 (20%) were removed. Interestingly we observed that developers of warned apps removed many permissions they had not been explicitly warned about. In particular, our 19,215 warned apps removed 45,866 unwarned permissions. This hints that our warning may have encouraged developers to revisit their decisions about permissions beyond the one(s) they were warned about. The total number of permissions removed by these apps was 51,591 including both warned and unwarned permissions.

Response by Permission Type. Figure 3 shows a breakdown of the number of warnings surfaced for 20 example Android Platform permissions. Some of these permissions were deprecated in recent Android SDK versions, but we include them as some apps can be targeting older Android SDKs. These permissions had at least a 100 warnings each. While overall 20% of individual permission warnings were adhered to, there was variation across the permissions: 7% of apps receiving warnings for Camera removed the permission, where as 65% of them receiving a warning for Get Accounts removed the permission. Because our warnings are soft, developers can elect to keep a permission—perhaps rightly so, if their needs do not fully correspond to those of their peers.

In addition to these warned permission removals, app developers have removed many more unwarned permissions. These unwarned removals are much more prominent across popular permissions, such as Location and Storage, which naturally generate fewer warnings. For instance, Read External Storage was only warned 29 times but has seen 1374 removals across the 19K apps. It could be that developers realize they don't always need these, and a warning for any permission could serve as a reminder. Many permissions beyond the 20 presented here, including those defined by other apps or services to control access to their resources, were removed by developers after they saw our privacy warnings. Note that when developers remove a permission they were not warned about, it does not remove our warning; hence the behavior of removing unwarned permissions should not be interpreted as a way to try to remove our warning. All these (non) platform permission removals sum up to the 51K removals reported earlier.

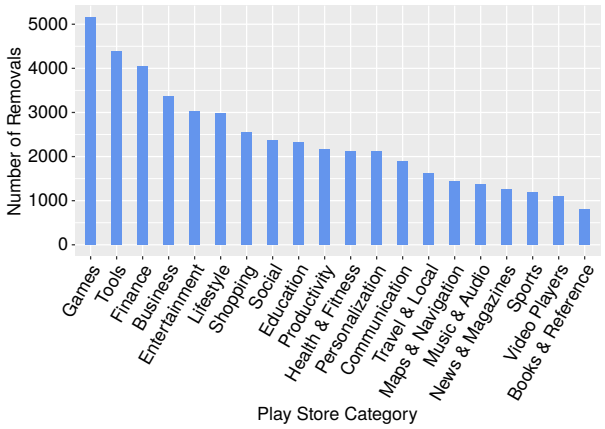

Figure 4: Categories and their permission removals

Response by App Category and Popularity. We now examine whether these removals are occurring only in a niche portion of Google Play or more broadly. We first look at Google Play categories of the apps who remove permissions. We observe removals in *all* categories. For example, as shown in Figure 4, within the top 20 categories, we observe between 1000-5000 permission removals per category. Hence developers across a broad set of app categories are responsive to our nudges. Next we look at the popularity of the 11.3K responsive apps, shown in Table 2. We notice that apps from all popularity levels have received warnings and removed permissions. These observations are encouraging, as they indicate that permissions are being removed broadly from many different kinds of apps across the popularity spectrum.

Estimating Impact. Ideally we'd like to assess the impact of these permission removals on the number of users affected. We cannot directly measure this. Our data only includes app descriptions, metadata and install counts; we have no user or device level data,

Installs per app	Number of apps that removed permissions
< 10K	3887
10K–50K	2555
50K–100K	893
100K–1M	2326
1M–5M	912
5M–10M	278
10M–100M	371
100M+	67

Table 2: Popularity of responsive warned apps

thus we cannot know, for example, how many of these responsive apps may be installed on the same device. However, we can approximate the impact via a lower bound on the number of installs as follows. Table 2 shows that there are 67 warned apps with more than 100 million installs each, thus a permission removal by each of these apps improves privacy for at least 100 million devices. This is clearly an underestimation as there are a few hundred other apps each affecting 10s of millions of devices. We can also assess the cumulative impact by looking at the number of installs of an app. For this estimate, we assume that all users update to the latest app version. For each app in this table, we count its number of installs; summing these counts we find that the 51K permission removals, resulting from our warnings, impacted more that 55 billion app installs in aggregate.

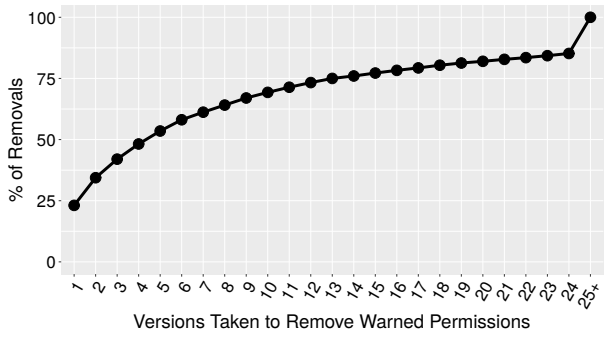

Figure 5: Versions taken to remove permissions

Time/Versions Taken to Respond. For the 5725 permissions that were warned and later removed, we analyzed the time taken by the developer to remove the permissions. For computing these metrics, we consider the earliest version of the app that was flagged by our privacy warning and the first version released later without the warned permission. We found that about 5% of the removals happened within a day of surfacing the warning, and 50% occurred within a 3 month period. Since permission removals require an app version update, and developers each have their own app development cycle, it is natural that the time to respond would vary significantly. We also looked at the number of app versions taken to remove the warned permission, and Figure 5 shows the CDF plot. 25% of removals happen in the immediate version release, and 70% take less than 10 version releases. Many developers have more than one app version lined up for release next, and the permission removal change may be included in any one of these release versions.

Supplementary Effect. Many developers publish more than one app, and so we investigate if showing a warning across one of the apps would induce changes across other unwarned apps published by the same developer. Our analysis showed that of the 15,645 developers who saw our permission warnings, 3758 developers revisited permissions requested by their unwarned apps. These developers removed an additional 60,993 permissions from 18,997 apps they published. These results indicate that our privacy warning, though being shown conservatively due to the chosen thresholds, may be influencing developer behavior more widely than the scope of our warnings.

4 DO OUR WARNINGS CHANGE DEVELOPER BEHAVIOR?

There can be many reasons why a developer may remove a permission (see Section 1). To try to understand if developers remove permissions due to our warning or due to other reasons, we compared the permission removal statistics of our warned apps to that of apps in two control groups.

- **Warned Apps:** The 19K apps that received our warnings.
- **Control Group-A:** Apps for which our metrics did not detect any unnecessary permissions, and therefore did not receive any warnings.
- **Control Group-B:** Apps for which our metrics detected unnecessary permission requests, but which were not notified since their developers do not receive the Pre-Launch Report .

All 3 groups have similar proportions of (un)popular apps. Control group-A has more than a million apps. This group could be removing permissions for any of the 5 reasons we outlined in Section. 1. Note that any changes in APIs, or in a library, would require all updating apps to incorporate these changes. Since group-A is removing permissions not due to anomalous permissions (as per our definition), we view them as a proxy for permission removal activity that occurs for other reasons. We observed that 9% of apps in Control Group-A remove permissions. This hints that possibly 9% of the apps in our warned app set remove permissions for reasons not related to our warning.

Control group-B only contains 12K apps, that exhibit the same anomalous permissions behavior as our warned apps, however they do not receive a warning. In this group, 45% of the apps removed permissions. Since this is larger than 9%, it could be that developers in this group are more sensitive to negative press, or may be aware of the broad issue surrounding potentially unnecessary permission requests. While we cannot confirm that, we note that nevertheless, our warning boosts the permission removal activity an additional 31% beyond what these developers do on their own.

We also point out that Group-A removed on average 0.3 permissions/app, Group-B removed 1.7 permissions/app and our warned apps removed 2.7 permissions per app. So even compared to other apps with similar anomalous permission behavior, our warning appears to increase the number of permissions removed *per app* by 60%.

Figure 6 shows the percentage of apps that removed permissions across the three app groups, for 20 permissions. These percentages for each group are calculated based on the number of apps that

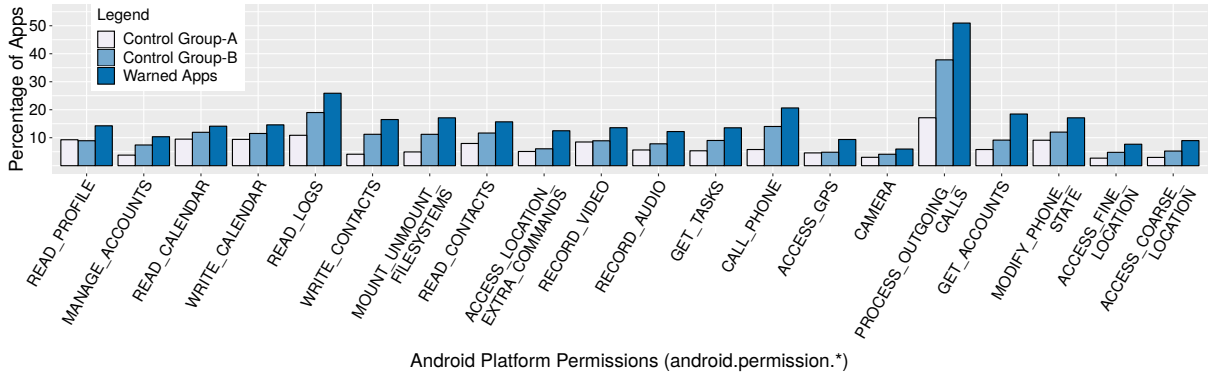

Figure 6: Percentage of apps that removed permissions (warned apps and control groups; 20 permissions shown)

request the permission in that group. As is clearly seen, many more apps in the warned group removed each permission type compared to apps in either of the control groups. A Kruskal-Wallis test conducted on the number of apps that have removed each of the 60 platform permissions across the three groups indicates that the differences are statistically significant ($H = 51.2, p < .01$).[2] Furthermore, the Mann-Whitney U tests show that the pairwise differences between the warned apps and the two other control groups (A and B) are also significant ($U = 854, p < .01$ for the warned apps vs. control group A, and $U = 1212, p < .01$ for the warned apps vs. control group B, after applying the Bonferroni correction). This evidence indicates that our privacy warning is effective in nudging developers towards removing more permissions than they would without the warning, and this influence extends beyond the specific permissions warned about, thereby boosting the overall redaction activity both directly and indirectly.

5 LIMITATIONS

Although changes due to our warning affect billions of app installs, we acknowledge that our feedback signal only reaches a small portion of the overall Google Play ecosystem in terms of the number of apps affected. This is because our conservative threshold discourages the presentation of warnings that we are not strongly confident about. This behavior also indirectly limits false positives; indeed, we view the high developer response as indicative that the signal is of good fidelity. If we showed warnings too frequently, or with many false positives, developers may become desensitized and ignore them.

As mentioned in Section 2.1, when the majority of apps in the peer set do request the same permission as a specific developer, we interpret this as an indication that the permission is genuinely needed by this type of app. If it should arise that more than 97% of the apps in a peer set are simultaneously requesting an unnecessary permission, then our approach does not flag these. Instead dynamic analysis is more promising for such cases.

6 CONCLUSIONS AND FUTURE WORK

We showed that even with a conservative approach (3% threshold) to surfacing privacy nudges, our deep learning approach is able to influence a significant number of apps towards the more privacy-friendly behavior of refraining from requesting unnecessary permissions. Overall, 59% of warned apps adhered to our warnings by removing permission requests. Moreover, this occurred not just for a niche group of developers or apps, but rather broadly across Google Play—as evidenced by removals from all app categories, across all popularity levels and many permission types. Our nudges encourage additional permission removals beyond the ones we warn about, for example, we observe a boost of 60% in the number of permissions removed per app, as compared to a control group that captures normal background permission removal activity due to other reasons.

Since developers are responsive to nudges, we believe it is promising to explore the design of other nudges in future work. Warnings could be surfaced in other developer tools such as Android Studio, Gradle, and more. It may be worthwhile to explore nudging SDK and library developers, as prior work has shown that a small number of libraries (~30) are used by the vast majority of apps [8]. Finding ways to incentivize this group of developers is a challenge but would have significant impact. To better understand why interventions such ours have the intended effect, controlled experimentation could be used to compare multiple variations of the warnings. Surveying developers as to why and when they remove permissions, and to understand their response to our warnings, could help to further clarify the effectiveness of nudges such as ours - especially in terms of their supplemental effects.

[2]The null hypothesis is that the number of apps that remove each of the 60 platform permissions across the three groups come from the same distribution.

7 ACKNOWLEDGEMENTS

Development and deployment of the privacy warning involved many people. We would like to thank Qiang Yan, Fergus Hurley, Bruno Buss, Olivier Gaillard, Marcin Oczeretko, and Richard Gaywood.

REFERENCES

[1] 2019. Android Studio Project Site: Android Lint Checks. http://tools.android.com/tips/lint-checks. (2019).
[2] Y. Agarwal and M. Hall. 2013. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceedings of MobiSys*.
[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 CHI*. ACM.

[4] Android Developers 2019. App permissions best practices. https://developer.android.com/training/articles/user-data-permissions.html?hl=en. (2019).

[5] Android Developers [2019]. Google Play Console. https://developer.android.com/distribute/console/. ([2019]).

[6] AppCensus 2019. AppCensus: Learn the privacy cost of free apps. https://www.appcensus.mobi/. (2019).

[7] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. 2014. The privacy and security behaviors of smartphone app developers. *Workshop on Usable Security (USEC)* (2014).

[8] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. 2017. Does This App Really Need My Location?: Context-Aware Privacy Management for Smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 3 (Sept. 2017).

[9] Developer Economics 2019. Slash Data: Developer Economics 2018 Survey. https://graph.developereconomics.com/?survey=de15. (2019).

[10] Federal Trade Commission 2013. Android Flashlight App Developer Settles FTC Charges It Deceived Consumers. https://www.ftc.gov/news-events/press-releases/2013/12/android-flashlight-app-developer-settles-ftc-charges-it-deceived. (2013).

[11] 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)*. ACM, 12.

[12] Tom Fox-Brewster. 2014. Check the permissions: Android flashlight apps criticised over privacy. https://www.theguardian.com/technology/2014/oct/03/android-flashlight-apps-permissions-privacy. *The Guardian* (Oct. 2014).

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[14] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*. ACM, 11.

[15] Marian Harbach, Markus Hettig, Susanne Weber, and Matthew Smith. 2014. Using Personal Examples to Improve Risk Communication for Security & Privacy Decisions. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI)*. ACM, 10.

[16] Suman Jana, Úlfar Erlingsson, and Iulia Ion. 2015. Apples and Oranges: Detecting Least-Privilege Group Analysis. *CoRR* abs/1510.07308 (2015). arXiv:1510.07308 http://arxiv.org/abs/1510.07308

[17] Jialiu Lin, Shahriyar Amini, Jason I. Hong, Norman Sadeh, Janne Lindqvist, and Joy Zhang. 2012. Expectation and Purpose: Understanding Users' Mental Models of Mobile App Privacy Through Crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp)*. ACM, 10.

[18] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun (Aerin) Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. 2016. Follow My Recommendations: A Personalized Privacy Assistant for Mobile App Permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS)*. USENIX Association.

[19] Abraham H. Mhaidli, Yixin Zou, and Florian Schaub. 2019. "We Can't Live Without Them!" App Developers' Adoption of Ad Networks and Their Considerations of Consumer Risks. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*. USENIX Association.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *CoRR* abs/1301.3781 (2013). arXiv:1301.3781 http://arxiv.org/abs/1301.3781

[21] Elleen Pan, Jingjing Ren, Martina Lindorfer, Christo Wilson, and David R. Choffnes. 2018. Panoptispy: Characterizing Audio and Video Exfiltration from Android Applications. *Proceedings of Privacy Enhancing Technologies Symposium (PETS)* (2018).

[22] Martin Pelikan, Giles Hogben, and Úlfar Erlingsson. 2017. Identifying Intrusive Mobile Apps Using Peer Group Analysis. https://security.googleblog.com/2017/07/identifying-intrusive-mobile-apps-using.html. (2017).

[23] PrivacyGrade 2019. PrivacyGrade: Grading The Privacy Of Smartphone Apps. http://privacygrade.org/. (2019).

[24] Zhengyang Qu, Vaibhav Rastogi, Xinyi Zhang, Yan Chen, Tiantian Zhu, and Zhong Chen. 2014. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (November 2014).

[25] A. Razaghpanah, R. Nithyanand, N. Vallina-Rodriguez, S. Sundaresan, M. Allman, C. Kreibich, and P. Gill. 2018. Apps, Trackers, Privacy and Regulators: A Mobile Study of the Global Tracking Ecosystem. In *Proceedings of NDSS*.

[26] ReCon 2019. ReCon: Bug Fixes, Improvements, ..., and Privacy Leaks: A Longitudinal Study of PII Leaks Across Android App Versions. https://recon.meddle.mobi/appversions/index.html. (2019).

[27] Jingjing Ren, Martina Lindorfer, Daniel J Dubois, Ashwin Rao, David Choffnes, and Narseo Vallina-Rodriguez. 2018. Bug Fixes, Improvements, and Privacy Leaks. A Longitudinal Study of PII Leaks acorss Android App Versions. In *NDSS*.

[28] F. Shih, I. Liccard, and D. Weitzner. 2015. Privacy tipping points in smartphones privacy preferences. In *ACM CHI*.

[29] Anastasia Shuba, Evita bakopoulou, and Athina Markopoulou. 2018. Privacy Leak Classification on Mobile Devies. In *Workshop on Signal Processing Advances in Wireless Communication (SPAWC)*. IEEE.

[30] Ben Smith. 2018. Project Strobe: Protecting your data, improving our third-party APIs, and sunsetting consumer Google+. https://www.blog.google/technology/safety-security/project-strobe/. (2018).

[31] Welderufael B. Tesfay, Peter Hofmann, Toru Nakamura, Shinsaku Kiyomoto, and Jetzabel Serna. 2018. PrivacyGuide: Towards an Implementation of the EU GDPR on Internet Privacy Policy Evaluation. In *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics (IWSPA '18)*. ACM, New York, NY, USA, 15–21.

[32] Richard H. Thaler and Cass R. Sunstein. 2008. *Nudge - Improving Decisions About Health, Wealth, and Happiness*.

[33] New York Times. 2016. Keep Tabs on Android App Permissions. https://www.nytimes.com/2016/10/06/technology/personaltech/keep-tabs-on-android-app-permissions.html. (October 2016).

[34] Lynn Tsai, Primal Wijesekera, Joel Reardon, Irwin Reyes, Serge Egelman, David Wagner, Nathan Good, and Jung-Wei Chen. 2017. Turtle Guard: Helping Android Users Apply Contextual Privacy Preferences. In *Symposium on Usable Privacy and Security (SOUPS)*.

[35] Timothy Vidas, Nicolas Christin, and Lorrie Cranor. 2011. Curbing android permission creep. In *Proceedings of the Web*, Vol. 2. 91–96.

[36] Shomir Wilson, Florian Schaub, Aswarth Abhilash Dara, Frederick Liu, Sushain Cherivirala, Pedro Giovanni Leon, Mads Schaarup Andersen, Sebastian Zimmeck, Kanthashree Mysore Sathyendra, N. Cameron Russell, Thomas B. Norton, Eduard Hovy, Joel Reidenberg, and Norman Sadeh. 2016. The Creation and Analysis of a Website Privacy Policy Corpus. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 1330–1340.

[37] Wired. 2018. App permissions don't tell us nearly enough about our apps. https:https://www.wired.com/story/app-permissions. (April 2018).