Short Text, Large Effect: Measuring the Impact of User Reviews on Android App Security & Privacy

Duc Cuong Nguyen*, Erik Derr*, Michael Backes†, Sven Bugiel†
*CISPA, Saarland University †CISPA Helmholtz Center i.G.

***Abstract*—Application markets streamline the end-users' task of finding and installing applications. They also form an immediate communication channel between app developers and their end-users in form of app reviews, which allow users to provide developers feedback on their apps. However, it is unclear to which extent users employ this channel to point out their security and privacy concerns about apps, about which aspects of apps users express concerns, and how developers react to such security- and privacy-related reviews.**

In this paper, we present the first study of the relationship between end-user reviews and security- & privacy-related changes in apps. Using natural language processing on 4.5M user reviews for the top 2,583 apps in Google Play, we identified 5,527 security and privacy relevant reviews (SPR). For each app version mentioned in the SPR, we use static code analysis to extract permission-protected features mentioned in the reviews. We successfully mapped SPRs to privacy-related changes in app updates in 60.77% of all cases. Using exploratory data analysis and regression analysis we are able to show that preceding SPR are a significant factor for predicting privacy-related app updates, indicating that user reviews in fact lead to privacy improvements of apps. Our results further show that apps that adopt runtime permissions receive a significantly higher number of SPR, showing that runtime permissions put privacy-jeopardizing actions better into users' minds. Further, we can attribute about half of all privacy-relevant app changes exclusively to third-party library code. This hints at larger problems for app developers to adhere to users' privacy expectations and markets' privacy regulations.

Our results make a call for action to make app behavior more transparent to users in order to leverage their reviews in creating incentives for developers to adhere to security and privacy best practices, while our results call at the same time for better tools to support app developers in this endeavor.

I. Introduction

Application markets such as Google's Play or Apple's App Store are core components in mobile software ecosystems. They constitute centralized markets for developers to distribute their apps and for end-users to search, download, and purchase applications. Similar to online retail markets, end-user reviews are a key element to the success of app markets. Users that have used an app can write reviews—short text messages typically including a star-rating—to express their opinion about an app and help other users to choose between similar apps. At the same time, reviews can also be used as a direct feedback channel to app developers, e.g., to express feature requests or to report bugs and security issues. The app developers, in turn, can react to this feedback and reply to their users.

Although user reviews form a direct communication channel betwen users and developers, past research on security and privacy protection has—to the best of our knowledge—not given this channel any attention. Prior research focused instead, for instance, on providing users with support in choosing less risky apps [1], [2] or on helping users making informed decisions whether or not to grant permissions to an application [3], [4], [5]. Although such support is undeniably valuable for helping users, we believe that those also form short-term solutions that do not immediately tackle the root cause of developers releasing apps that disregard privacy best practices. For apps to improve their security- and privacy-related behavior in the long-run, feedback should not only be directed to end-users but also to developers, ideally in a way that the developers have incentives and motivation to update their apps according to the security and privacy concerns of their users. User reviews would seemingly form such an immediate feedback and rating channel for security- and privacy-related user concerns. Unfortunately, the extent to which reviews can provide this kind of feedback and how developers react to such feedback have not yet been investigated.

In this paper, we study the connection between security- and privacy-related reviews (*SPR*) and security- and privacy-related app updates (*SPU*). Concretely, this includes questions like *"To which extent do SPR trigger SPU in apps?"*, *"How often do app developers react to SPR (e.g., due to the fear of follow-up reviews with low ratings and a potential financial loss)?"*, and *"What kind of SPU do app developers do in consequence of SPR?"* To answer those questions, we first build a crawler to collect the complete version histories of the top 2,583 apps (62,838 app versions) on Google Play and their corresponding 4.5M user reviews. We then use supervised learning techniques to identify 5,527 security and privacy relevant reviews. By retargeting the release dates for both the app versions and the reviews, we connect those SPR with the corresponding app version that was mentioned in the SPR. Using static code analysis, we classify the changes between those user-reviewed app versions and their immediate successor versions as SP-relevant when later app versions behave more privacy-friendly. Using recent advances in statically detecting third-party libraries [6], we are able to attribute those SPU to either app or library code changes. Using this data set, we then set out to thoroughly examine the impact of user reviews on the SPU of android applications. We build a statistical regression model that takes different factors into account that could affect the update of an app, including users' variables (e.g., ratio of SPR received, and review star rating) and app variables (e.g., permission mechanism, the ratio of replies to reviews, and app category). By applying our regression model

to our entire data set of reviews and app histories, we are able to show that SPR are significant predictors of SPU in Android applications. This means the more SPR an app version receives, the more likely the subsequent version of the app will be an SPU. Additionally, our results show that of all SPU, only 17.06% could be uniquely attributed to app code while 48.81% could be uniquely attributed to (closed-source) third-party code, meaning that in most cases SPR complained about app behavior that was added to the app through inclusion of third-party code. Furthermore, through statistical testing, we confirm that app versions that use Android's run-time permission dialogs raise more suspicion from users, expressed through a significantly higher rate of SPR for those app versions (1.46 times more than for install-time permissions).

Based on those results, we conclude that SPR indeed have a positive influence on the privacy-related development of apps and that there is a clear call for action to not only support users in making better choices but also making app behavior explicitly more transparent to users to foster higher rates of SPR that express users' privacy attitudes and create incentives for developers to adhere to privacy best practices. Developers, on the other hand, clearly need support in this task, in particular in estimating the impact of included third-party code onto their apps' privacy-critical behavior.
In summary, we make the following contributions:

- We investigate security- and privacy-relevant features in apps that can be perceived by end-users (e.g., permission requests and data accesses) and map them to permission-based functionality that can be extracted from apps.
- We build a longitudinal repository of 2,583 applications and their 4.5M user reviews. We build a classifier to identify SPR with a very good accuracy (mean AUC value of 0.93). By retargeting app release dates, we can map SPR back to their affected app versions in 88.62% of all cases.
- We statically extract permission-based features from apps mentioned in SPR and identified SPU of apps in 60.77% of all SPR. Further, 48.81% of those SPU can be attributed exclusively to (closed-source) libraries.
- We build a statistical regression model to evaluate the impact of different factors on apps' SPU, including users' variables and app variables.
- Our approach reveals that SPRs are a significant predictor of SPU of Android apps and that apps supporting runtime permissions dialogs receive 1.46 times more SPRs than apps with install time permissions.

Outline: This paper is organized as follows. We give an overview of related work in Section II and describe our methodology in Section III. We empirically analyze our data in Section IV and explain our regression model to predict SPU in Section V. We discuss our findings and draw actionable items in Section VI and conclude in Section VII.

II. Related Work

Android security, and in particular application security and the role of developers in the mobile ecosystem, have been studied from different angles in the past. To put our study on user reviews and their connection with the security and privacy evolution of apps into a larger context, we present and discuss in this section briefly related works on using natural language written texts for app classification, app reviews in general and their automatic processing, as well as closest related developments in app security.

Using natural language processing: Past research has successfully mined software artifacts and connected them with the app descriptions regarding security and privacy aspects. For instance, Gorla et al. used the applications' descriptions to examine whether or not the description matches the applications' behavior [2]. The authors proposed *Chabada*, a solution to cluster apps by their topics based on their description, and to identify outliers, i.e., apps whose behavior deviates from the usage of permission protected APIs within each cluster. Further, Pandita et al. [7] proposed *Whyper* and Qu et al. [1] proposed *AutoCog*, two systems that also mine Android application descriptions and then use natural language processing (NLP) to automatically bridge the semantic gap between what applications do and what users expect them to do from their description. All of them, Chabada, Whyper, and AutoCog, work on app descriptions written by developers. On the other hand, our study focuses on reviews written by *users*, which are usually authored on smart phones, and hence often contain typos and do not necessarily follow grammatical structures [8], [9], [10].

Processing app reviews: App reviews play an important role for the success of an app. They are the primary channel through which developers receive feedback about their applications, such as how users perceive their apps, which features users are requesting, or which aspect of the apps users favor. By default, this channel is public and available to current and potential future users. However, inspecting such reviews is a challenging task for developers as apps receive a high number of reviews every day. Prior work by Pagano and Maalej [11] found that iOS apps receive on average about 22 reviews per day and popular apps, such as Facebook, receive magnitudes more reviews. Moreover, reviews are not easy to automatically analyze given their unstructured forms. Existing work by Chen et al. [12] has shown that only about one third of the user reviews are actually informative to developers. Different prior works have focused on automatically identifying useful user reviews for developers. Palomba et al. [13] proposed *ChangeAdvisor* to support app developers in classifying feedback useful for app maintenance. ChangeAdvisor combines NLP, text analysis, and sentiment analysis to automatically classify app reviews written by end users. Fu et al. [14] proposed *WisCom*, a tool that analyzes user comments and ratings in mobile app markets. WisCom uses regression models and latent dirichlet allocation models to analyze the comments' topics. It is able to discover inconsistencies in reviews and determine why users dislike a given app. However, none of these works focuses on the connection between app reviews and the application's security and privacy evolution.

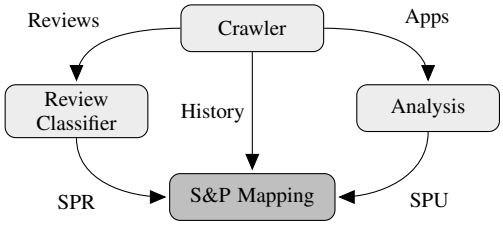

Fig. 1. Overview of our methodology

App security evolution: Calciati et al. [15] studied how the permissions requested by apps evolve across different app versions. Their results show that apps tend to request an increasing number of permissions in their evolution and many newly requested permissions are initially an over-privilege of the app (i.e., a direct violation of the least privilege principle). Violation of least-privilege by app developers is unfortunately a long-standing problem, first identified by Porter Felt et al. [16]. Given the central role of permissions for data protection on Android, past research has also investigated how users should be confronted with permission requests, most noticeably early studies by Porter Felt et al. [17], [18] that investigated users' concerns connected to permission protected resources and that gave different recommendations, respectively, which are partially reflected in a recent paradigm shift of Android's design from install-time to runtime permission delegation. More disruptive proposals try to eliminate the explicit role of the user for permission granting, e.g., through user-driven access control as proposed by Roesner et al. [5] or the use of machine learning as proposed by Wijesekera et al. [4] and Olejnik et al. [3]. Most recently, different works pointed out the risks of third party libraries, in particular of advertisement libraries [19], [20], [21], [22] and of vulnerable libraries [23], [6]. However, to the best of our knowledge, we are the first to study the connection between user reviews and Android application security and privacy evolution.

III. Methodology

In this paper, we automatically identify security- and privacy-related reviews (SPR) and map SPR to security- and privacy-related updates (SPU) of the corresponding applications. Figure 1 gives an overview of our methodology. We collect the dataset for our analysis with a custom built crawler, which mines Android applications and their version history as well as the apps' reviews from Google Play. After having collected the apps and their reviews, a classifier identifies SPR. Once we have the set of SPR, we establish correlations between SPR of apps and the security and privacy relevant changes within the corresponding apps' release history (S&P Mapping). In the following sections, we will describe the different steps of our methodology in details.

A. App and Review Crawler

1) Mining user reviews: The collection of the user reviews from Google Play consists of two steps: collecting the reviews' text as well as their scores, and then pre-processing the text for later classification.

We built a crawler to collect Android application reviews from Google Play. As previous studies [24], [25] have shown that only a small fraction of free applications on Play accounts for the bulk of the application downloads—a so called *superstar market*—we focus our collection of applications on those apps that are most popular among the users of Google Play. Therefore, our crawler collects all Android applications that have at least 50,000,000 downloads, which results in 2,583 distinct applications as of July 2017 when we collected our dataset. It might seem that 2,583 apps is a very small number of applications in comparison to other market studies on Android, but it has to be considered that we also crawl each app's version history and their corresponding reviews. Thus, we trade a large-scale cross-sectional study, as favored in most other studies on Play, for a longitudinal study of apps that allows us to analyze the evolution and influence of SPR on app security and privacy. Since downloading each app's version history easily amplifies the required time for data collection and analysis [6], we chose to limit our data collection, both app version histories and reviews, to apps that have at least 50,000,000 downloads. We explain the technical realization of our app collection further down.

We only crawl reviews that were written in English by selecting the Play web interface language code accordingly. Besides the review text with its rating score, we also gather developer responses (if available). Our dataset as of September 2017 contains 4,547,493 reviews. We will elaborate on how we compiled this list of reviews later on when explaining our training dataset for our review classifier (see Section III-B)

2) Crawling app history: Studying security and privacy relevant changes of applications (SPU) and their connections with user reviews requires building an app repository with historical information about apps, i.e., including all versions of a particular app, which allows analysis of an app's evolution. To this end, we adopt the approach of Backes et al. [6] that used an undocumented market API to query Google Play for older versions of an app. In the following, we will explain how we obtained the complete history of the top 2,583 apps in Play from September 2017, resulting in a repository of 62,838 distinct app versions (i.e., on average 24.33 versions per app in our collection).

a) History collection: The Play API allows us to query for app versions using the app's packagename and version code. However, there is no option to list the available versions for a given app. Thus, building the history of an app requires probing for existing version codes. The version code is an integer number that must be monotonically increased with every app update, but there is no official numbering scheme enforced. Although the majority of app developers simply increases the version code by one, related work [6] has shown that some developers use special date patterns, such as `YYYYMMDDVV` where `VV` is the revision-per-day. Since exhaustively probing for the existing version codes of each app is very time consuming, we set the threshold for version codes

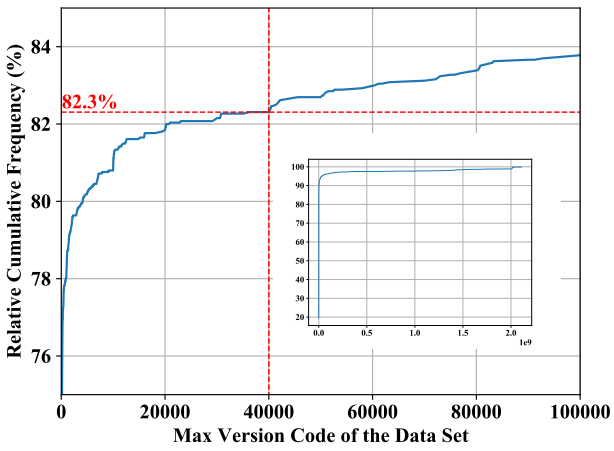

Fig. 2. Relative CFD of maximum version codes of apps in our data set

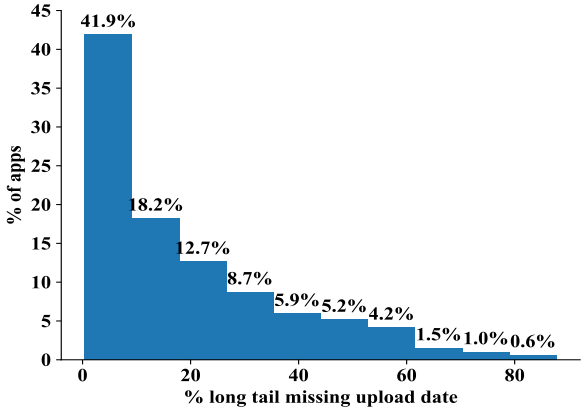

Fig. 3. Distribution of apps missing upload dates

that we test to a maximum of 40k. This gives a coverage of 82.3% for the apps in our data set, i.e., for 2,126 out of 2,583 apps this threshold is higher than their highest version code on Play. Figure 2 illustrates the relative cumulative frequency distribution of the maximum version codes in our data set.

b) Release dates: A second major drawback of the Play API is that it is not possible to query for release/upload dates of old app versions. In order to be able to map reviews to app versions by date, we follow the approach of related work to collect missing release dates from market analysis companies, such as *appannie.com*, *apk4fun.com*, and *appbrain.com*. In total, we were able to recover the upload dates of 81.52% of all app versions in our data set (51,225 of 62,838). For a set of 957 apps we were able to retrieve the complete version history. For the remaining 1,169 apps we have an incomplete set of upload dates, for whose majority (790 of 1,169) we miss the long tail of upload dates, i.e, we could not recover dates for early versions that were published before any of the market analysis services started to collect data. Figure 3 shows

TABLE I
SECURITY- AND PRIVACY-RELEVANT KEYWORDS

Permissions	**Key words**
Account	account access, account
Bluetooth	bluetooth, bluetooth devices
Calendar	read calendar, calendar, write calendar
Contact	read contacts data, write contact, contact
Location	location, track, gps
Mail	mail, voicemails
Media	picture, photo, media, files, take picture, taking picture, camera
Messages	sms, receive mms, send mms, messages, read messages, sms, read sms, send sms, mms, receive sms
Network	network, network state, wifi information, wifi, internet access, internet, network connectivity
Notification	notification, system alert window, system alert
Phone	phone call, phone number, outgoing call, manage call, phone state, call, call log, call's log, log, sip
Sensor	sensor data, sensor, fingerprint, nfc, vibrate
System	package size, install shortcut, delete package, battery info, reorder tasks, boot, boot completed, wap push, run in background, root
Storage	write storage, storage, read storage, sd card, SD card, file
General keywords	permission, access, intrusive, identity, personal info, malware, virus, malicious

the distribution of 790 apps for which we miss the long tail of upload dates. For about 70% of these apps, less than 30% of the whole app history is missing.

B. Review Classifier

A naïve way to identify SPR would be using keywords. However, this is not an easy task, since we cannot study millions of reviews to pick a representative keyword list. Besides, a review written by users can have multiple sentences. If we only use keywords to identify SPR, we may miss other information that comes from the nearby sentences that may contain interesting information but not the predefined keywords. Hence, by using machine learning techniques to learn not only the sentence with keywords but also the nearby ones we can expand our classifier's knowledge. For instance, consider the following review: *"Why do you need access to my location? Why on gods good green earth does your app need access to my location info? One star for the privacy steal."* If we would use keywords, we can only determine the first two sentences as security- or privacy-relevant. However, the last sentence is also an indicator that this app is perhaps doing something fishy. This is an important feature that we can put into a classifier without having to learn the phrase *privacy steal*. Later on, if our classifier encounters similar reviews, even without the presence of privacy-related keywords (here: location), it is still able to classify them as SPR (e.g., *"This app steals your info"*).

a) Training set: Given the large amount of reviews and the anticipated low portion of SPR, it is not feasible to manually label a representative set of reviews while simultaneously

balancing the number of SPR and non-SPR. Therefore, we first look at reviews that mention Android permissions or resources that are by default protected by an Android permission. We then manually examine some SPR to pick further keywords mentioned in such SPR and visit the Android documentation regarding the mentioned permissions to further complement our keyword list with the information from the documentation. We strongly focus on permission-protected resources, because this is the only interaction that end-users can usually observe when they interact with the apps, e.g., install-time permission dialogs (prior to Android 6) or intercepting dialogs for runtime permission requests (Android 6 or later). It is rather uncommon to see layman users that are not security experts using extra analysis tools (e.g., *Xposed* modules [26]) to track data flows within applications for privacy violations or to detect insecure network connections of apps.

Table I shows the list of compiled keywords we use in our analysis. This list results in approx. 1.85M reviews that are potentially security and privacy related. We randomly picked 4,000 reviews to manually label them. We consider a review as SPR if the user mentions the app's requested permissions, keywords related to accessed resources, or other general SPR keywords (see Table I); otherwise we consider the review as non-SPR. After removing some malformed reviews (e.g., we were unable to determine what the reviews meant), our training set contained 3,891 reviews (SPR: 586, non-SPR: 3,305). To account for imbalanced data (SPR vs. non-SPR), we apply SMOTE [27] to over-sample the SPR class.

b) Features extraction: Characters of n-grams are commonly used features in text classification [28], [29], [30]. Character n-gram features for a review are all *n* consecutive letters in that review. For instance, the 5-grams for the review *"Why does this app need access to my location"* are *why d*, *hy do*, *y doe*, *does*, *oes t*, *es th*, [...] , *locat*, *ocati*, *cation*. We use n-grams of characters instead of words, because reviews written by users often contain typos, and by using n-grams of characters, we can reduce the influence of typos onto the classification. Prior work of McNamee et al. [31] showed that $n = 4$ (characters) is a good choice for European language text retrieval, while Dave et al. [32] reported that unigrams ($n = 1$) of words outperform bigrams when conducting text classification of movie reviews. Inspired by their findings, we choose $n = 3, 4, 5$ as our n-gram models, which also yielded the best results during experiments with our training data. Before extracting n-grams of the reviews, we apply different text pre-processing techniques to obtain a better quality data set since user reviews are often written on smart phones, hence they tend to be very short and usually contain grammatical mistakes or typos [8], [9], [10]:

- Remove stopwords: remove articles from the user reviews (e.g "a", "an", "the")
- Stemming: reduce inflectional forms to a common base form of a word (e.g. am, are is $\rightarrow$ be)

c) Machine learning model: Classifying reviews belongs to the task of natural language processing (NLP) and the most common NLP approach for using machine learning to classify text documents is using *Bag of Words* [33]. With bag of words, each text document is represented as the bag of its words regardless of its grammar forms and its orders. Occurrences of each word is used as feature for training classifiers. We use a Support Vector Machine (SVM) Linear kernel for our classifier as it has been shown to be effective for text classification [34], [35], especially for short documents [36]. We form bag of words by splitting the reviews at spaces and punctuation marks, and use n-gram model to extract features for our classification task.

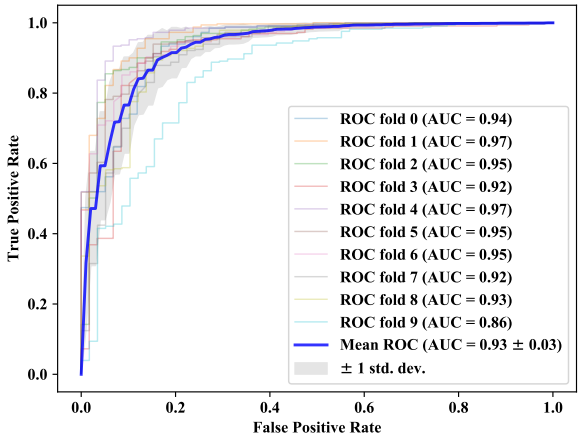

Fig. 4. ROC curves of the 10-Fold cross-valication for our SPR classifier

d) Validation: To validate our approach, we use k-Fold cross validation with $k = 10$, as prior work of Kohavi [37] has shown that this is the best method for cross validation. Besides, we choose AUC (area under the ROC curve [38]) as our classifier evaluation metric because it is not sensitive to imbalanced class distribution (SPR vs. non-SPR) and was widely used in prior work as the metric for imbalanced data classification [39], [40]. Figure 4 shows the AUC values for our 10-Fold cross validation. Our classifier has an AOC's mean value of 0.93 as its accuracy in classifying SPR (a classifier with perfect accuracy would have an AUC of 1.0).

C. Static App Analysis

So far we have built the data model that allows us to map SPR to the enclosing set of app versions by using both app version release dates and the date of the review. In order to measure the effect of an SPR on app security and privacy, we conduct static analysis on the version immediately preceding the SPR and the updated versions after the SPR to find potential SPU. For the majority of end-users the install-time permission list and runtime permission requests are the only information to assess whether the advertised functionality (e.g., via the app description, app category, etc.) seems legitimate. To determine the change of permissions and usage of APIs that require permissions across versions, we leverage the permission lists of the `axplorer` project [41]. Its authors provide mappings of Android SDK APIs to required permissions up to Android version 7.1. In a first step, we

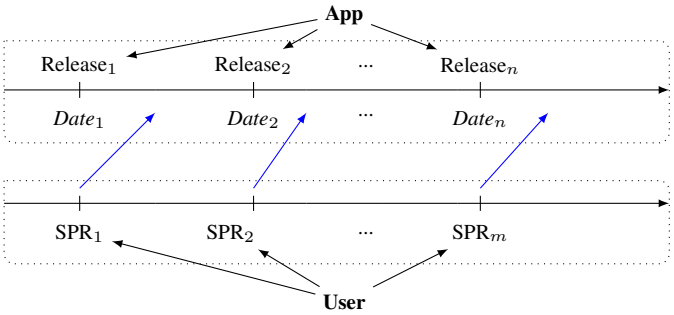

Fig. 5. Mapping SPR to security-/privacy-relevant app updates (SPU)

extract the list of declared permissions from the apps' manifest files. We further extract the target SDK versions to determine whether or not the app supports runtime permissions (target API higher or equal to 23). We subsequently scan the apps' bytecode for APIs that require dangerous permissions.

Attribution is another important aspect of the analysis, i.e., are permissions and their respective APIs used within the app developer code or within some third-party library code. In such cases, we would like to know the exact library (version). To this end, we leverage the open-source tool *LibScout* [6] that is capable of providing this information for a set of 205 commonly used libraries. To cover cases of unknown third-party code, we extend the implementation to classify any code not identified by LibScout into app or library code based on the app package name as a heuristic. We finally add functionality to attribute identified permission API calls to either app code or library code (either detected by LibScout or via our heuristic).

This collected information allows us in the following to identify security and privacy relevant changes as a potentially immediate result of an SPR.

D. Mapping SPR to SPU

The final step in our work-flow (see Figure 1) is to correlate the SPR for an app with the security and privacy related changes of an app. First, we identify potential candidate app versions that might contain relevant app changes in connection with an SPR, afterwards we analyze the candidate versions for security and privacy relevant updates (e.g., in the app manifest or code).

Identifying candidate app versions: Figure 5 illustrates how we map SPR to candidate application updates. For every SPR, we first assign the SPR to the immediate preceding app version, *SPR app version*, released before the SPR. We then look for security and privacy related updates in later versions of the app after the *SPR app version*. In case we do not have the release date of an app version, we skip that version. When an SPU is found, this connection between the SPR and newly found SPU is considered a match (i.e, the SPR potentially influences the SPU).

SPR to SPU version distance: While in ideal scenarios, we would expect SPU right after SPR, there are other factors which may contribute to the reasons why the next update may not be an SPU. For instance, developers are working on a particular feature of the app or they may only read user reviews irregularly (e.g., reviews come in large number [11]). We therefore take the distance between an SPR and an SPU release into account. In particular, if there is an SPR for version$_1$ but an SPU is found at version$_4$, then the distance is 3. The longer the distance is, the less likely the SPU is triggered by the SPR.

TABLE II
SECURITY- & PRIVACY-RELATED REVIEWS PER APP CATEGORY

Category (#apps)	Total #SPR	Mean #SPR/app
Tools (221)	1,343	7.5
Health And Fitness (30)	190	7.04
Shopping (35)	163	6.52
Sports (13)	54	6.0
Business (23)	113	5.95
Productivity (73)	364	5.69
Communication (66)	322	5.55
Media And Video (62)	192	5.33
Social (56)	215	5.12
Transportation (14)	42	4.67
Lifestyle (48)	136	4.53
News And Magazines (13)	47	4.27
Travel And Local (27)	89	4.05
Entertainment (98)	251	3.92
Personalization (112)	310	3.69
Finance (10)	25	3.57
Weather (19)	51	3.4
Photography (141)	228	3.3
Books And Reference (36)	73	3.04
Music And Audio (73)	144	2.94
Games (889)	1,149	2.78
Education (14)	22	2.44

IV. Empirical Analysis

We present the results and findings of our analysis of security- and privacy-relevant reviews on Google Play and the corresponding relevant changes in app updates. We refer to Section VI for a discussion of our findings.

A. Security and Privacy Related Reviews (SPR)

In order to analyze whether SPR trigger security and privacy relevant changes in app updates, it is first necessary to map SPR to features that can be checked with code analysis techniques. To this end, we first map SPR to permissions (groups) and subsequently check for permission-based features in app versions released after the SPR was posted.

1) Mentioned permissions: Our review classifier identified 5,527 SPR (0.12% of a total of 4,547,493 reviews) belonging to 1,269 distinct apps. Each of these apps has an average of 4.36 SPR (median = 2), where certain app categories received more SPR (e.g., Tools) while others received less (e.g., Games). Table II lists the number of SPR per category. For 2,898/5,527 SPR, we are able to identify 4,180 permission-related statements that can be assigned to 15 distinct permission groups. This implies that some SPR refer to multiple permissions. The remaining 2,629 security and privacy related reviews cannot unambiguously be mapped to permissions

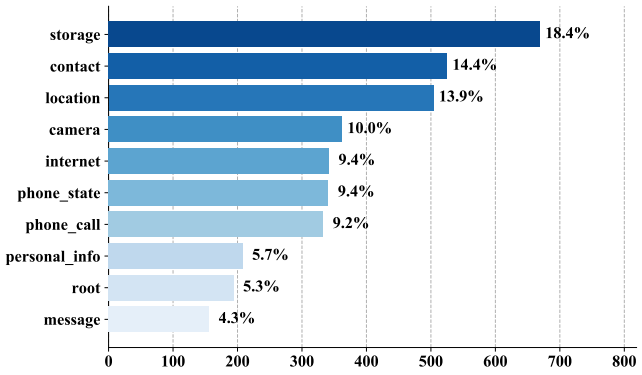

Fig. 6. Ten most mentioned permissions in SPR

without extra knowledge, e.g., *"Worked fine, but removing due to permission change without saying why...and if it is just for ads say that"* and *"A Nice game, but ridiculous permissions the game is very good, but the permissions in the last update is ridiculous."* Figure 6 presents the permissions mentioned most in SPR. The list is headed by the permissions to access external storage, contacts and location. We created a separate category "personal information" for SPR when users complain about such data without mentioning specific permissions.

2) Runtime permissions vs. install-time permissions: In October 2015, Google officially released Android 6.0 (API level 23) and shifted from an install-time permission model to a runtime permission delegation in which apps request dangerous permissions dynamically at runtime. For the 2,126 applications for which we built the version history, 1,073 (50.5%) have adopted runtime permissions in their latest version as of September 2017. Among the 1,269 apps that have at least one SPR, a similar fraction (49.7%) has adopted runtime permissions. To empirically investigate the effect of runtime permission requests on users' perception, we calculate the percentage of SPR over the total number of app reviews *before* and *after* an app adopted runtime permissions. To this end, we check whether the *targetSDK* argument from the apps' manifests is set to API level 23 or higher. We then conducted a t-test to compare the ratio of SPR per total reviews of app versions with runtime permission and with install-time permission. We found that there is significant difference between the SPR ratio of apps with install-time permission (mean = 0.001) and SPR ratio of apps with runtime permission (mean = 0.0025) with a p-value of 0.02. This suggests that apps with runtime permissions receive a significantly higher number of SPR.

3) Developer responses: Some SPR are written by users that do not understand context-specific permission requests or do not have sufficient knowledge to assess the necessity of a request [42]. Incomplete or missing app descriptions that provide an intuition about the apps' permission usage is one contributing factor. To allow interaction with users, Google offers a *Reply to Reviews API* [43]. To analyze to which extent app developers make use of this feature, we crawl any developer replies that have been made to the set of 5,527 SPR. In total, we found 673 replies. With respect to the 5,527 SPR, developers also implicitly reacted in 3,359 cases with SPU in the subsequent app version. In 256 cases, the developer replied (without making SPU) and in 417 cases we could observe both replies and SPU. We manually examined these replies and grouped them into the following categories:

- **Explain** (397): Developers explain the necessity of the mentioned permissions
- **Contact** (130): Developers asked the user to contact them and to provide more information
- **Fix** (96): Developers confirmed the SPR and reported that a fix is already published or in progress.
- **Pre-defined generic** (50): Developers replied with pre-defined generic answer templates

In about 56% of the cases, the developer explained the necessity of permissions. Besides missing app descriptions, install-time permissions are one of the factors that make it difficult for the user to make the connection between permission and functionality. In our data set, the developers replied with explanations for 234 app versions that were using install-time permissions, in contrast to only 163 explanations for apps with runtime permissions. Oftentimes, the developers ask users to provide more information via mail. The reason for this is the 350 character limit imposed by Google for both reviews and replies. This severely impedes providing comprehensive and detailed information about a specific issue. In 96 cases the developer confirmed the user observation and reported that the issue has already been fixed or the fix is in progress. For 78/96 cases, we could identify SPU in the subsequent version of the respective app. For these cases, we can be very certain that the SPU has been an immediate effect of an SPR. In 50 cases, the developer simply replied with a pre-defined template without responding to details of the review.

Figure 7 gives three examples for typical scenarios where developers reply to SPR. In the first example, the developer acknowledges the issues and announces a fix without providing more details. In the second example, the developer explains the necessity of the location permission. In this case, no SPU are to be expected in subsequent versions. In the final example, the developer announces a switch to runtime permissions in a future version to provide more context for requests.

B. Security and Privacy Relevant App Updates (SPU)

We consider changes in the permission usage of an app as SPU, since these changes would reflect what the user might perceive in terms of security and privacy. Our static analysis found the following SPU between all consecutive app releases in our data set:

- Requested permissions removed from app: 1,608
- Permission-protected API calls removed: 1,085
- Lib calls removed that trigger protected APIs: 940

In the following, we analyze those changes further.

1) Permission changes: Figure 8 shows the top 10 permissions removed from the apps' manifests with app updates after

Acknowledge → Fix

USER: New permissions Why would this app need access to my location?

DEVELOPER: Fixed in version 2.6.2. Update will be available in some hours.

Explanation → No Update

USER: Why do you need access to my location? Why on gods good green earth does your app need access to my location info? One star for the privacy steal.

DEVELOPER: Location used for showing and maintenance ads only. App doesn't use/save/share your location.

Explanation → Update

USER: Why does it wait until you install it to tell you it's a trial Why does it need access to my photos and videos

DEVELOPER: Hello. We need an access to SD card only for the specific situations like saving configuration files etc. We are working on new version with runtime permissions support, so with upcoming version we will request the permissions only when it will be necessary. Team (*removed*).

Fig. 7. Examples of user reviews and developers' responses

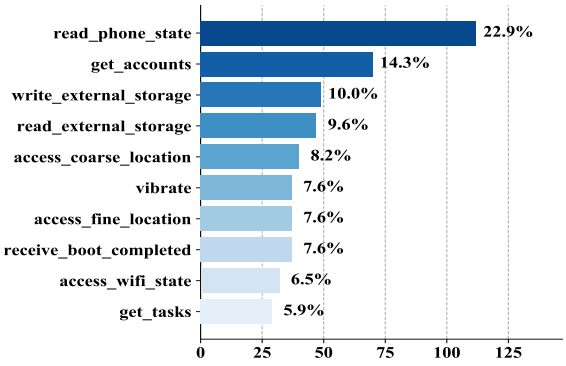

Fig. 8. Top 10 permissions removed from application manifests.

an SPR was posted on Play. Reading the device's phone state, access to user accounts, and access to the external storage are the most frequently removed permissions, with external storage also being the top mentioned permission in SPR (see Section IV-A1). The majority of removed permissions allow access to sensitive data, thus indicating a raised privacy awareness of users. Figure 9 shows the top 10 permissions from permission-protected API calls that were removed from the apps. However, removing permission-protected calls does not necessarily mean that the app does no longer require that permission.

2) Change attribution: We identify the root causes for the different results that we observe for removed permissions and permission APIs. An important aspect is statically included third-party code. Figure 10 lists the top 10 removed permissions required by permission-protected API calls triggered by calls to third-party libraries. Many of these permissions allow to retrieve data suitable for user tracking, which has been found in a variety of tracking and advertisement libraries [44]. An interesting case is the *WAKE_LOCK* permission. A study about wake lock misuse [45] showed that improper usage of

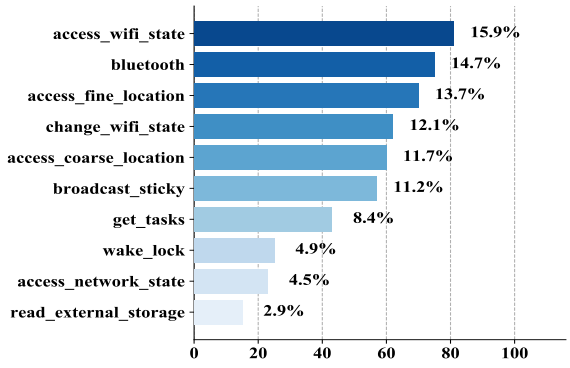

Fig. 9. Top 10 permission-protected API calls removed from applications

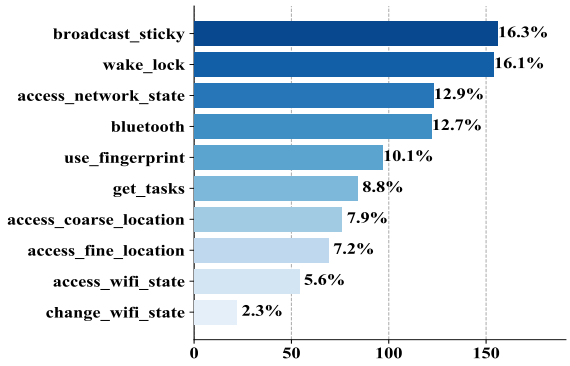

Fig. 10. Top 10 permissions removed from third party libraries

this permission often manifests in battery drain, crashes, and app instabilities. Besides permission requests, these are events that can be observed by the user as well. We also found that in 98.4% (925) of the cases the complete library was removed as part of the app update. In only 14 cases merely the library functionality that required the permission was removed. Figure 11 shows the frequently removed libraries without the extremely common *Play Service* and *Android support libraries*. Half of these libraries constitute advertising libraries that require at least the *INTERNET* permission, typical uses often include *ACCESS_NETWORK_STATE* and location permissions as well. For apps that target install-time permissions, the *INTERNET* permission was frequently mentioned in SPR (228 instances), in particular when the app's core functionality, e.g., calculator or flashlight, obviously did not require network access. With Android 6, Google downgraded this permission to a *normal* protection level. As a consequence, it is granted automatically and no longer shown to the user by default. SPR complaining about the *INTERNET* permission for app versions targeting Android 6 or higher dropped to just 38 instances.

We further used the results of the analysis to attribute SPU to app developer and library code. To this end, we checked for each permission mentioned in the review whether the permission-protected API usage in the subsequent version is exclusively located in app code, library code, or used in both app and library. We found that in only 17% (72) of cases the permission-protected APIs were exclusively used by app code,

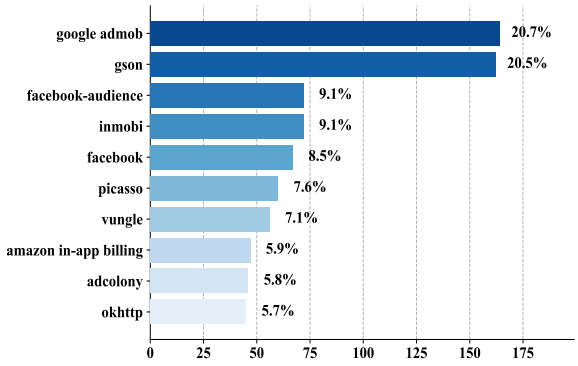

Fig. 11. Top 10 libraries removed from apps (w/o Play and Support libs)

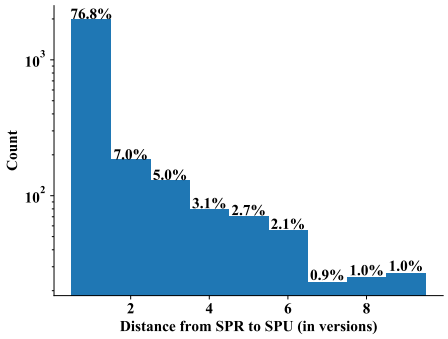

Fig. 12. Distribution of distance to SPU for 3,359 SPR. Count in log-scale.

while in 48.8% (206) of cases the APIs were exclusively used by library code. In the remaining 144 cases the permissions were used by both the application and the included libraries.

C. SPR to SPU Mapping

In this section, we report the results of mapping SPR to SPU and discuss factors that may influence the results.

a) Mapped SPR to SPU: We are able to unambiguously map 4,898/5,527 SPR to the affected app versions. Only 629 SPR (11.38%) could not be mapped back to the affected versions, because the review was posted before 2012 for which we could not recover app upload dates. For 3,359/4,898 SPR (68.6%) we could map the SPR to SPU identified in one of the subsequent app versions. For the remaining 1,539 SPR, we could not detect security and privacy relevant changes in app updates. Figure 12 shows the distribution of the app version distance from SPR to detected SPU for the set of 3,359 SPR. In 76.8% of the cases we can observe SPU in the app version immediately following the SPR. If we consider the interval from one to five versions, this value increases to 94.4%. The likelihood that SPU in an app $version_x$ were triggered by the SPR gradually decreases with the number of new app versions released between SPR and $version_x$ and other external factors may have triggered the SPU instead.

b) SPU without SPR: To evaluate to which extent our classifier misses SPR, we generate a backward mapping from SPU to reviews. For the 5,994 SPU our static code analysis found, 2,666 changes were observed without the presence of *any* review and 1,488 changes could be mapped to SPR. This leaves 1,840 SPU without SPR. Other reasons include external factors, such as updated libraries, internal code reviews, and developer notifications via different channels such as email.

c) SPR without SPU: To further validate our approach, we also seek to find answers for the cases in which we identify SPR but no SPU in the subsequent app versions. When excluding the SPR that could successfully be mapped to SPU (3,359) and the SPR that were posted prior to 2012 (629), this leaves 1,539 SPR for which we could not detect SPU. Reasons for this include: 1) Replies to reviews in which the developers explains the necessity of permissions or acknowledges the report without modifying the application. In these cases, no SPU are to be expected. 2) Limitations of our static analysis, e.g., LibScout can detect about 205 popular libraries, but there might exist more libraries that could have an effect on the security and privacy of an application. In addition, our analyzer checks for permission-protected APIs only. This misses permission-protected content providers such as *Contacts* for which an additional API argument analysis would be necessary. 3) The app is no longer maintained. To investigate why developers did not respond—neither with SPU nor replies—we check whether developers still update their apps after an SPR was posted. To this end we consider an app no longer maintained if its last version is older than one year starting from the day our crawler checked the latest version of the app). This threshold is reasonable since prior work [46], [47] has shown that most apps—in particular top apps—release updates biweekly or monthly. We found 728/1,539 (47.3%) SPR belonging to 244 unmaintained apps. Since these apps are still available on Play, they receive new reviews but new updates should not be expected.

D. Summary of Findings

We briefly summarize our findings from the empirical analysis. We first crawled 4.5M reviews for 2,583 distinct apps and identified 5,527 SP related reviews (SPR), out of which 2,898 SPR could be mapped to permissions. With our static code analysis we could identify 5,994 SPU in app versions following an SPR. In 60.8% of cases, we could successfully map SPR to SPU. If we consider corresponding SPU (changes to 3,359 SPR) and replies (273) as responses from developers and exclude SPR of unmaintained apps (728), we can calculate the developer response rate (RR) to SPR as follows:

$$\text{RR} = \frac{\text{\#SPR with SPU + \#SPR with replies}}{\text{\#SPR} - \text{\#SPR of unmaintained apps}} = 75.68\%$$

Despite a small overall number of SPR compared to the total number of reviews and the limited number of allowed text (350 characters), we can observe that these short texts are an effective means to trigger fast responses from developers. In almost 76% of cases, the app developer responded to a SPR.

V. Modeling Security and Privacy Updates

To examine the impact of different factors on Android application updates (SPU and non-SPU), we conducted multiple

regression models that predict whether an app update will be security-/privacy-related or not. Our models include the effects of user reviews, user rating, app's permission mechanism (install-time or runtime), developer reply ratio, and app category. To account for possible effects of multiple updates of the same application, we use mixed models in which the updates are attributed to their application (i.e., nested data) and include random effects by allowing the intercepts to vary at application level but aggregating them over all applications. We compared a model with random effects against a model without random effects, and our results show that the model with random effects is significantly superior in its predictions.

Data set: From the collected app history, static analysis results, and the identified SPR we built a data set of app updates (including both SPU and non-SPU). In this data set, we consider every change between two app version codes for which at least one review is available as a data point for our regression model, which yields 15,835 data points in total (12,540 non-SPU and 3,295 SPU) when excluding the "Comics" and "Libraries and Demo" categories since they only have one and three apps, respectively. We consider the following variables for every data point (i.e., app update) as predictors in our regression models:

- *SPR ratio*: ratio of SPR over the total number of reviews
- *Average score*: average rating score that the corresponding app version received since the last app update
- *Permission mechanism*: permission mechanism (runtime or install-time) used by the app version
- *App category*: as defined in Google Play
- *Reply ratio*: the ratio of developer replies over the total number of reviews since the previous app version

We consider SPR ratio and average score to be *user variables*, while permission mechanism, app category, and reply ratio are *app variables*. To account for SPR and average score that can potentially have an impact on later versions of the app but not the immediate version (see Section IV-C), we included the impact of SPR and average score of the previous versions within the *version distance* into the final SPR ratio and final average score, respectively. The *version distance* between a version (version_i) that has the SPR ratio and average score, and the version that is being considered (version_j) is calculated by the number of versions between version_i and version_j for the analysis. Version_i is here a preceding version of version_j. The final SPR ratio and final average score of the currently being considered version are the cumulation of all of the previous SPR ratios divided by their corresponding *version distance*, and the cumulation of all previous average scores divided by their corresponding *version distance*, respectively.

A. Correlation Analysis

Since the coefficient estimates of mixed models can be unstable and difficult to interpret if the model has multicollinear variables, we first performed a correlation analysis of the independent variables, such as ratio of SPR over total reviews, reply ratio, and average score. The analysis showed that there is no significant multicollinearity between any variables of SPR ratio, reply ratio, average score in our data set. We did not include a variable that accounts for the SPU location, i.e, whether security and privacy issues mentioned in an SPR were located in application code or library code, since this variable is derived from SPRs, which would violate the requirement for regression analysis that predicting variables must be measured independently. In our data set, the location of those issues are completely dependent on SPRs, which is already represented by the *SPR ratio* variable. We therefore excluded such a location variable from our model.

TABLE III
GOODNESS OF FIT FOR THE MODELS PREDICTING SPU. AIC = AKAIKE INFORMATION CRITERION; DF = DEGREE OF FREEDOM; LOGLIK = LOG LIKELIHOOD; PR(>CHISQ) QUANTIFIES STATISTICAL SIGNIFICANCE. STATISTICALLY SIGNIFICANT VARIABLES ARE SHADED.

	AIC	**logLik**	**Df**	**Pr(>Chisq)**
simple regression	16198.23	-8098.12		
mixed base regression	15654.05	-7825.03	1	<0.001
+ user variables	15570.16	-7781.08	2	<0.001
+ app variables	14830.06	-7388.03	23	<0.001
+ interaction	14831.54	-7387.77	1	0.471

B. Building the Models

To have a quality model, we need to only include variables that are necessary and can account for as much of the variance in the empirical data as possible. We start with a base model without any independent variables and then subsequently extend it with more predictors. Table III presents the goodness of fit for the relevant steps in building the corresponding models. Since the dependent variable of our analysis is binary—either an app update is SPU or non-SPU—we use logistic regression.

Moreover, to verify that a mixed model suits our data better than a simple base model, we tested the base model without any independent variables against the mixed models. The result is that mixed models fit our data significantly better. In particular, we extend the base model as follows:

- Start with base model with a random effect to account for effects from updates of the same app
- Include variables at user level: SPR ratio, average score
- Include variables at app level: permission mechanism, app category, developer's reply ratio
- Include interaction between SPR ratio and average score

In each step, we calculated the model fit and used log likelihood model fit comparison to check whether the later model fits our data significantly better than the previous one. For the final model, we chose the one with the best fit that was significantly better in explaining our data than the previous. This is a well established approach for model selection [48], [49], [50], [51].

We compared all models according to their corresponding *Akaike information criterion* (AIC), see Table III, which estimates the relative quality of statistical models for a given set of data. Smaller AIC scores indicate a better fit. Moreover, we also used likelihood-ratio tests, which are evaluated using Chi-squared distribution, to compare the models.

TABLE IV
LOGISTIC REGRESSION MIXED MODEL PREDICTING SP CHANGES. STATISTICALLY SIGNIFICANT VARIABLES ARE HIGHLIGHTED. PM = PERMISSION MECHANISM; CAT = APP CATEGORY

	Estimate	**Std. Error**	**z value**	**Pr($>\|z\|$)**
(Intercept)	-1.092	0.315	-3.465	<0.001
SPR ratio	2.568	0.796	3.225	0.001
avg_score	-0.094	0.006	-15.093	<0.001
reply_ratio	-0.420	0.157	-2.678	0.007
pm:run-time	1.360	0.054	25.276	<0.001
cat:Books_Reference	0.096	0.367	0.261	0.794
cat:Business	-0.039	0.374	-0.103	0.918
cat:Communication	0.098	0.347	0.281	0.779
cat:Education	-0.403	0.418	-0.965	0.335
cat:Entertainment	0.526	0.334	1.573	0.116
cat:Finance	0.012	0.462	0.026	0.980
cat:Games	0.601	0.316	1.903	0.057
cat:Health_Fitness	0.428	0.362	1.184	0.237
cat:Lifestyle	-0.105	0.346	-0.303	0.762
cat:Media_Video	-0.035	0.346	-0.102	0.919
cat:Music_Audio	0.324	0.339	0.956	0.339
cat:Personalization	0.047	0.331	0.141	0.888
cat:Photography	0.374	0.326	1.146	0.252
cat:Productivity	0.131	0.342	0.384	0.701
cat:Shopping	0.205	0.362	0.567	0.571
cat:Social	-0.043	0.341	-0.127	0.899
cat:Sports	-0.195	0.391	-0.498	0.618
cat:Tools	0.168	0.323	0.520	0.603
cat:Transportation	0.065	0.420	0.155	0.876
cat:Travel_Local	0.115	0.400	0.288	0.773
cat:Weather	0.216	0.392	0.551	0.581

From Table III, we can see that the model with user and app variables and without interaction has the lowest AIC score (14830.06) and explains the data statistically significantly better than other models. For permission mechanisms and category, we choose *install-time* and *News And Magazines* category respectively as base lines for categorical variables: permission mechanism is a binary variable (either runtime or install time) and we want to see to which extent changing from install-time to runtime permission affects the interaction between user and Android application; and *News And Magazines* category's average number of reviews per app coincides with the global average number of reviews per app among all categories (see Table II).

C. Results and Interpretation

Table IV presents our regression model that examines the effect of different predictors for SPU. We can see that, in comparison to install-time permission mechanism, runtime permission dialogs have significantly positive impact on SPU (odds ratio of 3.9). This means that updates of applications (including app versions) whose permission mechanism is runtime are significantly more likely to be relevant to security and privacy. This further supports our earlier results that runtime permissions raise users' suspicions. Moreover, in comparison to apps of News And Magazines category, apps belonging to other categories do not differ significantly with regards to SPU. This indicates that SPU of an app seemingly do not depend on the app's category. Most importantly, we can see that SPR ratio is a significantly positive and strong predictor of SPU (odds ratio: 13.04). This indicates that the more SPR the app developers receive, the more likely they will release SPU. In contrast to SPR ratio, reply ratio has negative impact on SPU (odds ratio: 0.66), indicating that if developers reply to a review, the less likely the following app updates are security- and privacy-related. When we consider SPR, we see that most of the developers' replies are *Explanation* (see Section IV-A3) why such permissions are needed. This is further supported by our regression model. We argue that if permission requests of Android apps are more transparent (e.g., better explanation, request in context), users would understand why such requests are indeed reasonable, hence developers would not need to explain themselves in their replies. Finally, the average score has a negative impact on SPU. More precisely, if an app is receiving high scores (on average), then the next updates are less likely to be related to security/privacy (odds ratio 0.91)

VI. DISCUSSION

We discuss shortcomings of our approach and interpret our findings. Then, we highlight future work and a call for action.

A. Threats to Validity and Future Work

Our approach relies on the ability to map SPR back to app versions in order to measure possible app changes as reaction to user reviews. Similar to related work [6], we could not retarget the upload dates for all versions of our dataset. In particular, for app versions released before 2012 there exists no reliable third-party source that can be queried for upload dates. As a consequence, we failed to map 629/5,527 SPR (11.38%) back to app versions and therefore cannot assess the impact of these SPR on the app's security and privacy.

Further, we use static code analysis to identify security and privacy related changes in app versions (immediately) following an SPR. This empirical evidence is a strong indicator that these changes have been made as a (direct) consequence of the SPR. Reasons for SPU range from following the principle of least privilege to protecting users' privacy to monetary reasons due to bad ratings and a decline in the number of app installations. For the small number of SPRs to which the developer replied and confirmed the issues, we can directly verify our findings. However, in general, collecting the ground truth would require conducting a developer survey to ask directly for the incentive of these changes. Prior studies [52], [53] have shown that recruiting a reasonable number of developers in Google Play for a survey is challenging without direct infrastructure support of the market (i.e., response rates <1%). We abstained from conducting a survey, as we only have a limited set of 2,583 top apps, with an even smaller number of distinct app developers and, hence, given prior experiences [52], [47], a too small expected number of responses.

Another improvement would include adding a sentiment analysis to our binary review classifier (SPR/non-SPR). This could help in understanding ambiguous SPR where users complain about requested permissions but still like the application or when users complain but are explicitly fine with a good explanation of the permission usage.

Lastly, we focused in our study on the top apps in Google Play, for which a higher level of maintenance and developer responsiveness to reviews would be expected. Our results might not apply to the long tail of apps on Play. However, since the top apps account for the bulk of the app downloads on Play [24], [25], our results apply to the apps with the highest impact on Android's user base.

B. The Effect of SPR

Previous work has not given much attention to the influence of end-users on security and privacy of apps via app reviews (see Section II). Our results show that end-user complaints based on observable evidence (permissions, crashes, or anomalies like unusual battery drain) often lead to app changes that improve security and privacy aspects. In cases where the issues can be attributed to closed-source components (see Section IV-B2), the developers might not even have been aware of these problems without an involved code analysis, e.g., when the library documentations miss important details.

User reviews can also force app developers to react quickly to issues due to the snowball effect. In many cases it is not a single SPR that triggers app changes but a series of SPR by different users (*SPR ratio* in regression model) or SPR followed by a series of follow-up reviews with low star ratings agreeing with the initial SPR (*avg_score* between app versions). Developers are then forced to react due to a fear of losing reputation (star rating) and user base that typically manifests in significant impact on revenue. As a result, developers either try to quickly resolve the problem by providing a better explanation to end-users or by addressing the issue with an SPU.

Although our results emphasize the positive effect of SPR in general, reviews could be much more informative and effective without the current size limitation of 350 characters for both reviews and replies imposed by Google. Such limits force users to omit important details in reviews and make them use alternative, unrestricted communication channels, such as email (see developer responses in Section IV-A3). This also prevents comprehensive app reviews as we know it from consumer reviews for shops, such as Amazon. Although Google is aware of this problem for years [54], no improvements have been made to remedy the situation.

C. The Effect of Runtime Permissions

Permissions are one of the most important security and privacy indicators of apps that can be perceived even by less tech-savvy users. However, the way how permission requests are presented to the user greatly affects their effectiveness (e.g., habituation effects, user understanding, etc., see Section II). The most drastic recent change in Android's permission system is the switch from install-time to runtime permissions, from which we can also observe a ripple effect onto users' reviews. Before Android 6, install-time permissions provided a one-time decision possibility without context. Without an explicit connection from permissions to functionality, users have to resort to (frequently missing or incomplete) app descriptions for permission decisions. With the introduction of runtime permissions, permission requests are (typically) shown in context and end-users may decide differently when the same request is displayed on different occasions. Further, developers have the possibility to augment permission requests with information to explain the necessity of a permission in a given situation. With the introduction of runtime permissions in Android 6, Google did also change the protection levels of a significant number of permissions. Before Android 6 (API level < 23) there have been 38 dangerous permissions that were prominently shown at install-time [41]. Starting with API level 23, Google refactored the permission system and specified only 20 dangerous permissions. The remaining 18 permissions have either been downgraded to normal permissions (that are granted automatically and are not shown to the user by default) or have been deprecated. Among the most prominent examples are the `INTERNET` permission, used by the vast majority of apps, and `READ|WRITE_PROFILE`. In addition, one single permission `READ_EXTERNAL_STORAGE` was upgraded to dangerous. This is also the top-mentioned permission in SPR (16%, see Figure 6).

Our regression model (see Section V-C) suggests that applications adopting runtime permissions are significantly more likely to perform SPU compared to apps that stick to install-time permissions. But at the same time, the results indicate that for apps with runtime permissions there is still a high number of developer replies of type *Explanation* (163) in comparison to apps with install-time permissions (234). This suggests that many app developers do not follow runtime permission best practices [55], i.e., adding explanations for permission requests and requesting permissions in context rather than requesting permissions on app launch. In terms of transparency for the users, requiring the developers to add explanations in permission dialogs should be opt-out instead of opt-in by default. We think that our results support further investigation of how app developers use the runtime permissions.

Google recently announced that in the second half of 2018, Play will require new apps and app updates to adopt runtime permissions, i.e., to target an API level ≥ 23 [56]. This will likely allow end-user privacy assessments for a larger number of apps, i.e., in our dataset about 45% of the top apps have not adopted runtime permissions in their latest version. According to our results this will generate more SPR and, as a result, more SPU in apps.

D. User's Perception of Risks and Privacy Incidents

Compared to related work, our results from studying users' security- and privacy-related app reviews also suggest a change in the user's concerns over the last years. A large scale survey on the perceived risks of smartphone users by Felt et al. [18] indicated that sending premium messages, dialing premium numbers, and deleting contacts were among the top risks in 2012. Contacts are still in the Top 2 mentioned permissions (see Figure 6), but reading external storage and location—now the top perceived risks—have previously been among the lowest-ranked risks. This is partly because of additional

security features that impede using monetary services without the user's explicit consent and due to raised privacy awareness of end-users. Past incidents have shown that simple flashlight [57], [58] or wallpaper apps [59] misused access rights to spy on the user or to exfiltrate personal data. As a result, Google specified both privacy policy for apps [60] and a general *Unwanted Software Policy* [61]. Developers were notified that, by end of March 2017, *"Google Play requires developers to provide a valid privacy policy when the app requests or handles sensitive user or device information."* In future work, we consider evaluating to which extent users' SPR can be used to create trend analyses of users' attitudes, in particular in response to regulatory (e.g., policies) and system changes (e.g., refactoring of permissions). For instance, in our data set, the downgrade of the *INTERNET* permission lead to a sharp decline in the number of SPR for that permission.

Moreover, a recent large-scale investigation of hidden tracking behavior in Android apps revealed that misuse of sensitive data by third-party advertisement and tracking libraries is even larger than ever [62]. In consequence, Google extended its *Unwanted Software Policy* and additionally requires that *"if an app collects and transmits personal data unrelated to the functionality of the app then, prior to collection and transmission, the app must prominently highlight how the user data will be used and have the user provide affirmative consent for such use."* [63] This implies that adhering to the new policy requires transparency for all included third-party components. Related work [44], [19], [20], [64] indicated that particularly advertising and tracking libraries are the main source of privacy violations and also our results (see Section IV-B2) show that the majority of SPRs complain about behavior that apps inherited from included libraries. However, most of these third-party components are distributed as closed-source binaries and many are not explicit about their usage of permissions and end-user data. An open question will be how app developers can handle these problems, as this kind of libraries is often used as the main monetization factor. A related study [47] showed that app developers need more support in handling third-party libraries, both with better development tools and a dedicated package manager for libraries. Similar assistance will be necessary for developers to make educated decisions on the choice of libraries to adhere to the new policy and pro-actively avoid negative SPR on markets.

E. Call for Action

The strict enforcement of the 350 character limit, prevents comprehensive and high quality reviews and forces users to omit additional information. Increasing the limit will give users the opportunity to write meaningful reviews and report issues without having to resort to alternative, non-public communication channels such as email. Both end-users and developers could also benefit from dedicated reviewer programs, such as *Amazon Vine* [65], which promote trusted reviewers that provide high-quality app assessments for incentives, such as paid apps for free. Particularly developers of top apps receive a high number of reviews every day, but only few of them include a call for action. Approaches such as ours—as standalone-tool or directly integrated into the developer console—can effectively reduce the number of reviews that have to be considered, hence making the time-consuming, manual triage process more effective. Additionally, the process of writing a review needs to be simplified to engage a higher number of users to participate. Currently, writing a review with a device constitutes a multi-step process via the Google Play app. This could be optimized by extending app launchers to provide such an option as an app shortcut [66] as a default for all apps hosted on Play. Moving to runtime permissions is a valuable step towards increasing the risk awareness. The latest beta version of Android P continues this path by disallowing idle apps to access the microphone and camera [67]. Any attempt is shown to the user as symbol in a notification. It has to be shown whether this is already effective or whether such accesses should be highlighted in the status bar more prominently.

VII. Conclusion

In this paper, we empirically studied the impact of user reviews on Android application security and privacy features. We automatically classified reviews into SPR and non-SPR. We mapped SPR to mentioned app versions and conduct a static code analysis to extract security- and privacy-related code changes for these versions (SPU). We find that in 60.77% of all cases the SPR triggered an SPU. The majority of these changes can be attributed to (closed-source) third-party code like advertising or tracking libraries. Furthermore, we built a regression model to evaluate the impact of different factors on SPU. With our regression model, we showed that SPR are significant predictors for SPU. In the majority of cases, app developers directly change the respective app code or publicly reply to users and explain why certain permissions are required. We have further seen that the adoption of runtime permissions has a significant positive effect on users' privacy perception. With the announced enforcement of runtime permission adoption, the absolute number of SPR is likely to increase in the near future, which in turn will help to improve app security and privacy in general.

Our results make a call for action to further increase the transparency of apps to foster more SPR as way to increase privacy-friendly app behavior; but also call for better tools to support developers in adhering to privacy regulations and their users' privacy preferences. Lastly, our approach might inspire future research to employ user reviews as a way to measure the effects of changes in regulations or Android's design.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) (FKZ: 16KIS0345, 16KIS0656). We would like to thank Yang Zhang for his valuable feedback on the text classification technique in this project.

REFERENCES

[1] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, "Autocog: Measuring the description-to-permission fidelity in android applications," in *Proc. 21st ACM Conference on Computer and Communication Security (CCS'14)*. ACM, 2014.

[2] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *ICSE'14: Proceedings of the 36th International Conference on Software Engineering*, 2014.

[3] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, and J. P. Hubaux, "Smarper: Context-aware and automatic runtime-permissions for mobile devices," in *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)*. IEEE Computer Society, 2017.

[4] P. Wijesekera, A. Baokar, L. Tsai, J. Reardon, S. Egelman, D. Wagner, and K. Beznosov, "The feasibility of dynamically granted permissions: Aligning mobile privacy with user preferences," in *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)*. IEEE Computer Society, 2017.

[5] F. Roesner, T. Kohno, A. Moshchuk, B. Parno, H. J. Wang, and C. Cowan, "User-driven access control: Rethinking permission granting in modern operating systems," in *Proc. 33rd IEEE Symposium on Security and Privacy (SP'12)*. IEEE Computer Society, 2012.

[6] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proc. 23rd ACM Conference on Computer and Communication Security (CCS'16)*. ACM, 2016.

[7] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "Whyper: Towards automating risk assessment of mobile applications." in *USENIX Security Symposium*, 2013, pp. 527–542.

[8] X. Gu and S. Kim, ""what parts of your apps are loved by users?" (t)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. IEEE Computer Society, 2015, pp. 760–770.

[9] M. K. Dalal and M. A. Zaveri, "Opinion mining from online user reviews using fuzzy linguistic hedges," *Appl. Comp. Intell. Soft Comput.*, vol. 2014, pp. 2:2–2:2, Jan. 2014.

[10] A. Ciurumelea, A. Schaufelbühl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 91–102.

[11] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *Requirements Engineering Conference (RE), 2013 21st IEEE International*. IEEE, 2013, pp. 125–134.

[12] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, "Ar-miner: Mining informative reviews for developers from mobile app marketplace," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 767–778.

[13] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. Gall, F. Ferrucci, and A. De Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, pp. 106–117.

[14] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh, "Why people hate your app: Making sense of user feedback in a mobile app store," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '13. ACM, 2013, pp. 1276–1284.

[15] P. Calciati and A. Gorla, "How do apps evolve in their permission requests?: a preliminary study," in *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 37–41.

[16] A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proc. 18th ACM Conference on Computer and Communication Security (CCS '11)*. ACM, 2011.

[17] A. Porter Felt, S. Egelman, M. Finifter, D. Akhawe, and D. Wagner, "How to ask for permission," in *Proceedings of the 7th USENIX Conference on Hot Topics in Security (HotSec'12)*. USENIX Association, 2012.

[18] A. Porter Felt, S. Egelman, and D. Wagner, "I've got 99 problems, but vibration ain't one: A survey of smartphone users' concerns," in *Proc. 2nd ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'12)*. ACM, 2012.

[19] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *WISEC'12*. ACM, 2012.

[20] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Investigating user privacy in android ad libraries," in *MoST'12*. IEEE, 2012.

[21] S. Demetriou, W. Merrill, W. Yang, A. Zhang, and C. A. Gunter, "Free for all! assessing user data exposure to advertising libraries on android," in *Proc. 23rd Annual Network & Distributed System Security Symposium (NDSS '16)*. The Internet Society, 2016.

[22] S. Son, G. Daehyeok, K. Kaist, and V. Shmatikov, "What mobile ads know about mobile users," in *Proc. 23rd Annual Network & Distributed System Security Symposium (NDSS '16)*. The Internet Society, 2016.

[23] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, "Execute this! analyzing unsafe and malicious dynamic code loading in android applications," in *Proc. 21st Annual Network and Distributed System Security Symposium (NDSS'14)*. The Internet Society, 2014.

[24] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 221–233, Jun. 2014.

[25] N. Zhong and F. Michahelles, "Where should you focus: Long tail or superstar?: An analysis of app adoption on the android market," in *SIGGRAPH Asia 2012 Symposium on Apps*, ser. SA '12. New York, NY, USA: ACM, 2012, pp. 10:1–10:1.

[26] "Xposed module repository," http://repo.xposed.info/.

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *J. Artif. Int. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622407.1622416

[28] W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization," in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 161–175.

[29] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[30] J. Houvardas and E. Stamatatos, "N-gram feature selection for authorship identification," in *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*. Springer, 2006, pp. 77–86.

[31] P. McNamee and J. Mayfield, "Character n-gram tokenization for european language text retrieval," *Information Retrieval*, vol. 7, no. 1, pp. 73–97, Jan 2004.

[32] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW '03. New York, NY, USA: ACM, 2003, pp. 519–528.

[33] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in twitter to improve information filtering," in *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2010, pp. 841–842.

[34] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*. Springer, 1998, pp. 137–142.

[35] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[36] A. Basu, C. Walters, and M. Shepherd, "Support vector machines for text categorization," in *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. IEEE, 2003, pp. 7–pp.

[37] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'95. Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143.

[38] J. A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (roc) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[39] N. V. Chawla, "Data mining for imbalanced datasets: An overview," in *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 875–886.

[40] R. C. Prati, G. E. Batista, and M. C. Monard, "Class imbalances versus class overlapping: an analysis of a learning system behavior," in *Mexican international conference on artificial intelligence*. Springer, 2004, pp. 312–321.

[41] M. Backes, S. Bugiel, E. Derr, P. McDaniel, D. Octeau, and S. Weisgerber, "On demystifying the android application framework: Re-visiting android permission specification analysis," in *Proc. 25th USENIX Security Symposium (SEC'16)*. USENIX Association, 2016.

[42] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: User attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*, ser. SOUPS '12. ACM, 2012, pp. 3:1–3:14.

[43] "Reply to reviews api," https://developer.android.com/guide/topics/ui/shortcuts.htm://developers.google.com/android-publisher/reply-to-reviews.

[44] T. Book, A. Pridgen, and D. S. Wallach, "Longitudinal analysis of android ad library permissions," in *MoST'13*. IEEE, 2013.

[45] Y. Liu, C. Xu, S.-C. Cheung, and V. Terragni, "Understanding and detecting wake lock misuses for android applications," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. ACM, 2016, pp. 396–409.

[46] B. Carbunar and R. Potharaju, "A longitudinal study of the google app market," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. ACM, 2015, pp. 242–249.

[47] E. Derr, S. Bugiel, S. Fahl, Y. Acar, and M. Backes, "Keep me updated: An empirical study of third-party library updatability on android," in *Proc. 24th ACM Conference on Computer and Communication Security (CCS'17)*. ACM, 2017.

[48] D. Bates, M. Maechler, B. Bolker, S. Walker *et al.*, "lme4: Linear mixed-effects models using eigen and s4," *R package version*, vol. 1, no. 7, pp. 1–23, 2014.

[49] R. H. B. Christensen, "ordinal—regression models for ordinal data," *R package version*, vol. 22, 2010.

[50] A. Field, *Discovering statistics using IBM SPSS statistics*. sage, 2013.

[51] J. J. Hox, M. Moerbeek, and R. van de Schoot, *Multilevel analysis: Techniques and applications*. Routledge, 2010.

[52] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *Proc. 37th IEEE Symposium on Security and Privacy (SP'16)*. IEEE, 2016.

[53] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl, "A stitch in time: Supporting android developers in writing secure code," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1065–1077.

[54] "Google play help forum – topic: Increase word count for reviews," https://productforums.google.com/forum/#!topic/play/OTjIUVr4g9I.

[55] "App permissions best practices," https://developer.android.com/training/permissions/usage-notes.html.

[56] "Improving app security and performance on google play for years to come," https://android-developers.googleblog.com/2017/12/improving-app-security-and-performance.html, 2017.

[57] "Android flashlight app tracks users via gps, ftc says hold on," https://www.techrepublic.com/blog/it-security/why-does-an-android-flashlight-app-need-gps-permission/, 2013.

[58] "Why does my flashlight app need access to all these things?!" http://www.sudosecure.com/why-does-my-flashlight-app-need-access-to-all-these-things/, 2013.

[59] "Does an android live wall paper really requires any special permission? beware of android app permissions," http://initpage.com/post/Does-an-Android-Live-Wall-Paper-really-requires-any-Special-Permission-Beware-of-Android-App-Permissions, 2013.

[60] "Privacy, security, deception," https://play.google.com/about/privacy-security-deception/.

[61] "Google's unwanted software policy," https://www.google.com/about/unwanted-software-policy.html, 2015.

[62] "Android users: To avoid malware, try the f-droid app store," https://www.wired.com/story/android-users-to-avoid-malware-ditch-googles-app-store/, 2018.

[63] "Additional protections by safe browsing for android users," https://security.googleblog.com/2017/12/additional-protections-by-safe-browsing.html, 2017.

[64] J. Ren, M. Lindorfer, D. J. Dubois, A. Rao, D. Choffnes, and N. Vallina-Rodriguez, "Bug fixes, improvements, ... and privacy leaks – a longitudinal study of pii leaks across android app versions," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*, ser. NDSS '18. The Internet Society, 2018.

[65] "Amazon vine," https://www.amazon.com/gp/vine/help.

[66] "App shortcuts," https://developer.android.com/guide/topics/ui/shortcuts.html.

[67] "In android p, notifications for apps running in the background show if they're using the camera or microphone," https://www.androidpolice.com/2018/05/09/android-p-notifications-apps-running-background-show-theyre-using-camera-microphone.