SecuRank: Starving Permission-Hungry Apps Using Contextual Permission Analysis

Vincent F. Taylor
Department of Computer Science
University of Oxford
Oxford, United Kingdom
vincent.taylor@cs.ox.ac.uk

Ivan Martinovic
Department of Computer Science
University of Oxford
Oxford, United Kingdom
ivan.martinovic@cs.ox.ac.uk

ABSTRACT

Competition among app developers has caused app stores to be permeated with many groups of general-purpose apps that are functionally-similar. Examples are the many *flashlight* or *alarm clock* apps to choose from. Within groups of functionally-similar apps, however, permission usage by individual apps sometimes varies widely. Although (run-time) permission warnings inform users of the sensitive access required by apps, many users continue to ignore these warnings due to conditioning or a lack of understanding. Thus, users may inadvertently expose themselves to additional privacy and security risks by installing a more permission-hungry app when there was a functionally-similar alternative that used less permissions. We study the variation in permission usage across 50,000 Google Play Store search results for 2500 searches each yielding a group of 20 functionally-similar apps. Using fine-grained contextual analysis of permission usage within groups of apps, we identified over 3400 (potentially) over-privileged apps, approximately 7% of the studied dataset. We implement our contextual permission analysis framework as a tool, called SecuRank, and release it to the general public in the form of an Android app and website. SecuRank allows users to audit their list of installed apps to determine whether any of them can be replaced with a functionally-similar alternative that requires less sensitive access to their device. By running SecuRank on the entire Google Play Store, we discovered that up to 50% of apps can be replaced with preferable alternative, with free apps and very popular apps more likely to have such alternatives.

1. INTRODUCTION

The smartphone ecosystem is centered around large repositories of apps, called app stores (or app marketplaces), that provide millions of apps covering a broad spectrum of functionality and quality. In order to provide their prescribed functionality, apps may need access to sensitive resources and private data on a smartphone. Some apps handle a user's sensitive data respectfully, but other privacy-invasive apps and greyware/malware are known to abuse their granted permissions to pilfer data or profile users [13, 35]. Many attempts to address the problem of permission-abuse by apps have been explored in the literature, including ways to determine suspicious apps from the permissions they use [30, 26] and by analysing whether an app's store description reflects the permissions it asks for [24, 27, 31].

When a user searches for an app of particular functionality, the search engine returns a variety of suitable apps for the user to choose from. Users, however, are known to disproportionately favour the highest ranking search results [18], and may inadvertently expose themselves to additional security or privacy risks, by choosing a permission-hungry app when there was a functionally-similar alternative that required less sensitive access to their device. For this reason, we study the Google Play Store to understand the extent to which apps request only as many permissions as they need to provide their functionality. That is, we study the extent to which apps follow the *principle of least privilege* [28]. On Android, following the principle of least privilege limits opportunities for malicious actors to perform privilege escalation, app collusion, and confused deputy attacks [7].

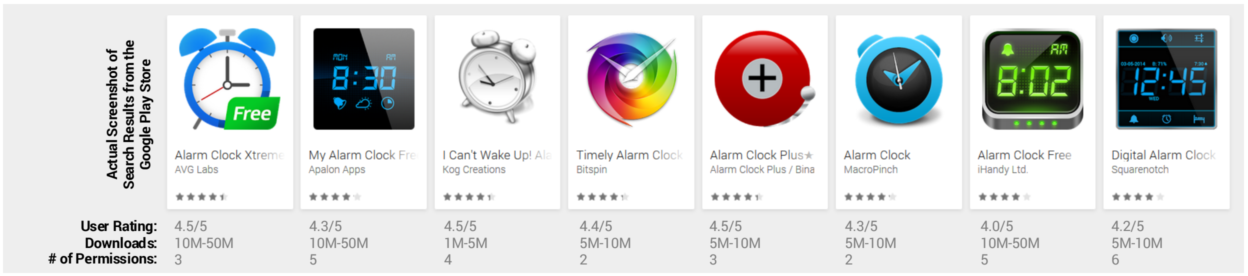
On both Android and iOS, sensitive system APIs and data are guarded by a permissions-based security model. With the release of Android 6.0 (API level 23, code-name *Marshmallow*), the Android platform now resembles that of Apple's iOS where users are no longer forced to grant all permissions at install time, but must now allow or deny permission requests at runtime[1] [4]. This added power offers users additional options for protecting their privacy and security, but many users continue to blindly accept warnings because of conditioning and/or a lack of understanding of the ramifications of their actions [14, 12]. Indeed, Eling et al. [10] show that 40.4% of users accept fine-grained, intrusive and unnecessary *run-time* permission requests for minimal reward. Moreover, a majority of users are known to ignore permissions, privacy policies, and terms of agreement altogether [9], with some users placing higher emphasis on the short-term benefits derived from information disclosure [1, 2]. Studying app over-privilege continues to be important, as users' intentions are known to diverge from their actual behaviour when it comes to protecting their privacy [23, 3].

To illustrate the problem of permission-hungry apps, we show the details of the Top 8 search results for the query

[1]Note that app developers can nullify the switch-over to run-time permissions by targeting their apps to API level 22 or lower.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPSM'16, October 24 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4564-4/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2994459.2994474


Actual Screenshot of Search Results from the Google Play Store

	Alarm Clock Xtreme	My Alarm Clock Fre	I Can't Wake Up! Ala	Timely Alarm Clock	Alarm Clock Plus★	Alarm Clock	Alarm Clock Free	Digital Alarm Clock
	AVG Labs	Apalon Apps	Kog Creations	Bitspin	Alarm Clock Plus / Bina	MacroPinch	iHandy Ltd.	Squarenotch
User Rating:	4.5/5	4.3/5	4.5/5	4.4/5	4.5/5	4.3/5	4.0/5	4.2/5
Downloads:	10M-50M	10M-50M	1M-5M	5M-10M	5M-10M	5M-10M	10M-50M	5M-10M
# of Permissions:	3	5	4	2	3	2	5	6

Figure 1: Snapshot of the Top 8 Google Play Store results for the search query "alarm clock". All apps have very high ratings and provide similar functionality, but the most permission-hungry app uses three times as many permissions as the least permission-hungry app.

"alarm clock" in Fig. 1. Overall, there is a great disparity (the most permission-hungry result uses three times as many permissions as the least permission-hungry result) in permission usage across the Top 8 results, in spite of the fact that each app in the group provides similar functionality. Moreover, all of the apps are highly rated and provide similar textual app descriptions, so it is difficult for (and unreasonable to expect) the average user to make an informed choice every time they install a new app. We are motivated to determine whether this disparity in permission usage is an isolated occurrence for the "alarm clock" query, or evidence of a much larger phenomenon affecting app stores. The utility of our work hinges on the fact that app store searches are the most popular method of app discovery [22].

In this paper, we present SecuRank, a framework and tool that identifies and suggests functionally-similar apps that follow the principle of least privilege. Throughout the paper, we focus exclusively on the usage of *dangerous permissions*, i.e., the 24 system permissions listed in the Android developer API as allowing an app access to a user's confidential data [5]. For this reason, we refer to *dangerous permissions* as simply *permissions*.

Our work is motivated by the following key observations:

O-1: *App store searches return many apps with similar functionality but dissimilar permission usage.* The average user cannot be expected to analyse and understand the security and privacy trade-offs in choosing one app over another every time they want to install an app.

O-2: *Over-privileged apps can be more profitable to app developers and advertisers.* With greater access to a device's data and other resources (such as location), advertisers can more effectively profile users for the purposes of advertisement targeting to the benefit of the app developer.

O-3: *Threats to user privacy and security coming from over-privileged apps are not static.* App stores evolve as new apps (both benign and malicious) are added and thus automatic and continuous analysis is required to identify apps that seek inappropriate access to a device.

It is *critical* to note that our work only focuses on *general-purpose apps*. By general-purpose apps, we mean those that provide generic functionality, with many apps competing in the app store to provide that functionality. Examples of general purpose apps include alarm clocks, flashlights, calorie counters, etc. Non-general-purpose apps, such as *Facebook* or *Bank of America*, are considered to be out of scope, as often times there will be only one particular app that matches what the user is looking for.

Our work is guided by the following research questions:

Q-1: *Can permission-hungry apps be identified by comparing their permission usage to other apps that provide similar functionality?*

Q-2: *Are the rankings of search results from app stores stable over time?*

Q-3: *Can permission-hungry apps be replaced with functionally-similar alternatives of good quality that follow the principle of least privilege?*

Contributions. To the best of our knowledge, we are the first to make the following contributions to the literature:

- A methodology for determining popular app searches in app stores.
- An understanding of the extent to which users can obtain less permission-hungry apps of good quality to replace the ones they currently use.
- A framework and tool, called SecuRank, that assists users in replacing apps with less permission-hungry alternatives. We have made our tool available to the general public in the form of a website[2] and an Android app[3].

Outline. The rest of this paper is organised as follows: Section 2 explains our data collection methodology; Section 3 presents an overview of our dataset; Section 4 explains how contextual permission analysis works; Section 5 presents SecuRank and the results of running it on the Google Play Store; Section 6 discusses our work; Section 7 surveys related work in the area; and finally Section 8 concludes the paper.

[2]SecuRank is freely available to use online at https://securank.me/
[3]The SecuRank app can be downloaded for free from the Google Play Store.

2. DATA COLLECTION METHODOLOGY

Our overall goal is to analyse permission usage within groups of functionally-similar apps. To identify such groups, we leverage general-purpose search queries on the Google Play Store[4] that each return a sufficient number of functionally-similar apps as search results. Data relating to the Google Play Store and search results for particular search queries was retrieved in early 2016 from an IP address in the United Kingdom. Our data collection methodology is depicted in Fig. 2 and has three major steps that are discussed in Section 2.1, 2.2, and 2.3.

2.1 Collect Google Play Store App Data

Google does not make full app store data available, so we first had to leverage the Google Play Store Crawler project [19]. This project is concerned with making a simple, scalable crawler that retrieves data (such as app title, description, and popularity) from all apps in the Google Play Store. Unfortunately, this project does not collect information about the permissions an app uses, so we had to perform additional steps:

1. Extract all app names from the Google Play Store Crawler project dataset and add them to a local database.

2. Build a second crawler that scrapes the HTML page of an app on the Google Play Store website and parse it to obtain the complete data (including permission usage) of an app.

3. Add this complete app data to the local database.

2.2 Generate Popular Search Queries

The Google Play Store does not provide statistics of the most popular search queries, so we had to derive our own list. An app's description is used heavily by the app store search ranking algorithm to return relevant results. Frequently occurring (*non-stop*) words in the descriptions of the most popular apps can give some idea of what popular searches in that store may look like. We leverage this insight to generate popular search queries by performing the following steps (as shown in *Stage 1* of Fig. 2):

1. Obtain app store data (from the local database) of all apps reported as having more than 100,000 installs.

2. Parse and filter the app store descriptions of these apps to obtain written text containing Roman alphabet characters only.

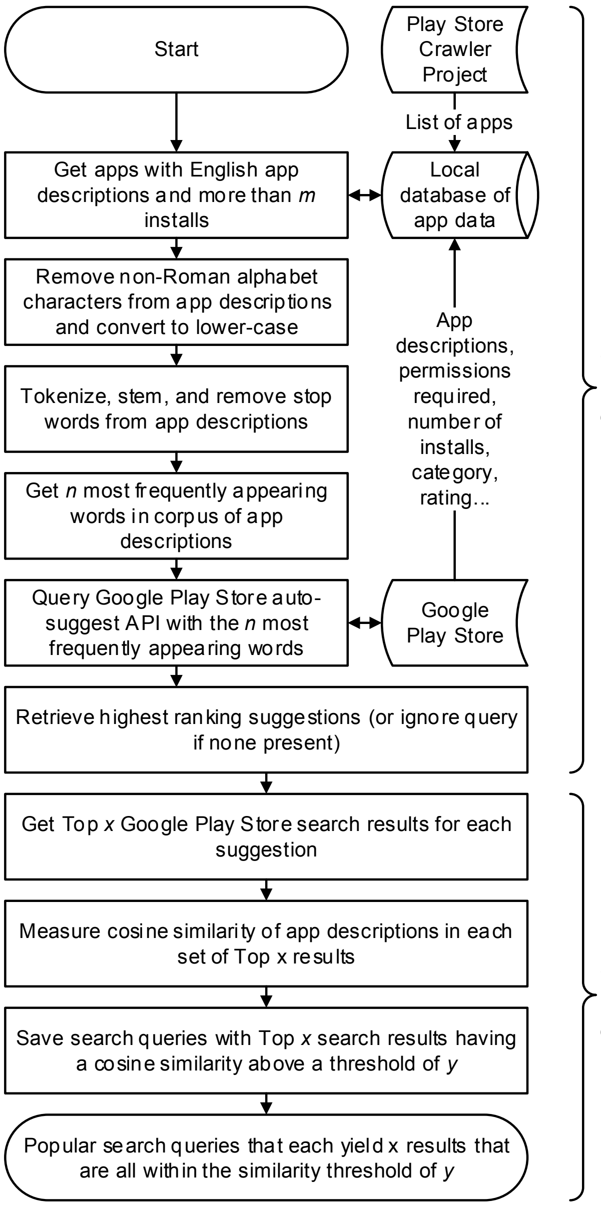
3. Pre-process these app store descriptions using the natural language processing (NLP) techniques of tokenization, stemming, and *stop word* removal.

4. Combine pre-processed descriptions into a corpus of text, and extract the 20,000 most frequently appearing words.

5. Ensure that these 20,000 most frequently appearing words correspond to actual searches of the app store by querying[5] the Google Play Store search form auto-suggest API with each word.

6. Add the highest ranking suggestion to a list of popular search queries.

[4]Throughout this paper, we use the terms *Google Play Store*, *app store* and *Store* interchangeably.



Start
Play Store Crawler Project
List of apps
Get apps with English app descriptions and more than *m* installs
Local database of app data
Remove non-Roman alphabet characters from app descriptions and convert to lower-case
Tokenize, stem, and remove stop words from app descriptions
App descriptions, permissions required, number of installs, category, rating...
Get *n* most frequently appearing words in corpus of app descriptions
Query Google Play Store auto-suggest API with the *n* most frequently appearing words
Google Play Store
Retrieve highest ranking suggestions (or ignore query if none present)
Stage 1
Get Top *x* Google Play Store search results for each suggestion
Measure cosine similarity of app descriptions in each set of Top x results
Save search queries with Top *x* search results having a cosine similarity above a threshold of *y*
Popular search queries that each yield x results that are all within the similarity threshold of *y*
Stage 2

Figure 2: Flow chart showing the various stages in our data collection methodology.

2.3 Ensure App Similarity

The list of popular search queries derived needs to be filtered, to obtain only those search queries that each yield a sufficient number functionally-similar apps. For our analysis, we considered only those search queries whose resulting

[5]To prevent our script from being blocked, we rate-limited our queries to one per second and sent a random, valid User-Agent string with each query.

Top 20 apps were all similar in functionality and had descriptions written in English. The set of 20 apps returned by each search query is hereafter called the *search result set* for that query.

We used an app's textual description as a proxy for the functionality it provides. We believe this to be a valid strategy because app developers write the descriptions of their apps to advertise functionality and enable the app store to effectively rank the app based on keywords in the description. We used a similarity measure to compare the text descriptions of apps within each search result set. The similarity measurement function compares the app descriptions of search results `#2-20` to that of search result `#1` (since we assume that search result `#1` is the best match for the query). If the similarity of app descriptions across a search result set was above a certain threshold, we assumed that the group of apps all provide similar functionality. We validate this assumption in Section 4.

Similarity measures for short segments of text, such as app descriptions, have been widely studied in the literature [20, 21]. Some approaches are lexical, meaning that they rely solely on matching terms, while others are probabilistic and involve language modelling frameworks. A common similarity measurement approach involves transforming documents using *Term Frequency-Inverse Document Frequency* (TF-IDF) in a so-called vector space model. This model does not work well for measuring documents of different lengths. To get around this problem, the cosine similarity measure is often used since the vectors are normalised before measurement. Cosine similarity is easy to understand and implement but suffers from the fact that it does not take sentiment into account. We do not consider understanding sentiment to be critical is this case, since the app store search ranking algorithm already provided apps that were thought to be related. Our similarity measure, then, is only needed to identify and reject those search queries containing one or more genuinely unrelated apps. Cosine similarity has been used previously for Android malware detection [29].

For our similarity measure, we used the Python NLTK library [6] to build a *cosine similarity* measurement function that leveraged a Porter stemmer for text pre-processing. We filtered (as shown in *Stage 2* of Fig. 2) the popular search queries obtained from the previous step (*Stage 1* of Fig. 2) to obtain only those that delivered 20 similar apps. We expand on choosing a similarity threshold in Section 3.1. After following the aforementioned steps, we had a list of popular search queries of the Google Play Store that each returned a search result set of 20 functionally-similar apps.

3. ANALYSIS OF THE SEARCH QUERY DATASET

We first measured the stability (or time invariance) of search rankings in the Google Play Store to understand the extent of changes in rankings over time. To do this, we retrieved the Top 20 results from the Google Play Store for the queries in our search query dataset several times, with periods of two-weeks and two-months between retrievals. The results of this analysis are presented in Fig. 3. The plots show what percentage of search queries have the same apps in particular positions in the search results. The *2-week difference* plot shows that 86% of the queries had the same app in position `#1` and 71% had the same pair of apps in positions `#1-2`. The *2-month difference* plot shows that 75% of search queries had the same app in position `#1` and 51% had the same pair of apps in positions `#1-2`. Over the longer period, and for lower ranks of apps, the search results are less stable. At this point, it is important to reiterate that the highest ranking search results are the ones that command the significant majority of traffic (and thus app downloads) from organic[6] searches [18]. Thus high-ranking apps, if permission-hungry, are a persistent cause for concern since their search rankings are very stable.

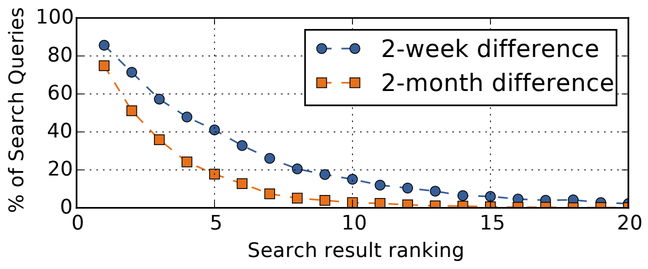


Figure 3: Difference in search results over a 2-week/2-month period. The most highly ranked apps tend to hold on to their exact search rankings or were slightly reshuffled.

3.1 Generating the Search Query Dataset

To generate the *search query dataset* to be used in our analysis, we had to choose an appropriate value for the *cosine similarity threshold* to ensure that the search queries each delivered 20 similar apps. Manual inspection showed that a cosine similarity threshold of 0.25 yielded queries with search result sets of functionally-similar apps. Thus we used a threshold of 0.25 for the remainder of the tests. At a threshold of 0.25, we had 2620 search queries that delivered 20 similar apps. We randomly selected 2500 of these queries for our analysis of the Google Play Store. These 2500 queries (and their corresponding 50,000 search results) constitute our *search query dataset* (not to be confused with a *search result set* of 20 apps).

3.2 App Ratings and Permission Usage

Within each search result set, we measured how the quality of an app varied with its rank in the search results. We used the average user rating for an app as a proxy for the quality of that app. Our findings are shown in Fig. 4. We found that the average rating of an app did not fall off as the ranking in the search results went down. This suggests that competing apps within groups of functionally-similar apps are also of high quality.

Fig. 5 shows how the mean number of permissions required by apps varied based on the app's rank in the search results. The `#1` ranked apps required (noticeably) more permissions than the apps that were ranked below them. Single-factor ANOVA confirmed that this was a statistically significant result. On average, eight apps from a search result set required more permissions than the mean for that search result set. Our analysis shows that six of these eight apps,

[6]Organic search results are those that are derived from their relevance to queries, as opposed to those resulting from advertisement campaigns.

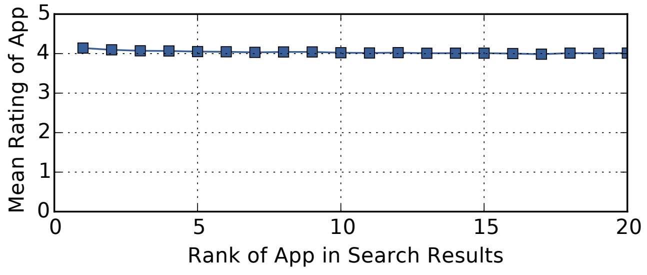


Figure 4: How the mean user rating per app varies with the rank of that app.

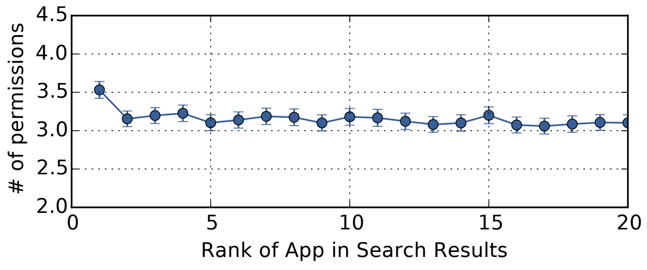


Figure 5: How the mean number of permissions required per app varied with the rank of the app in the search results.

on average, were ranked in the Top 10 for their search result set. This observation suggests that more highly-ranked apps, in general, are more permission hungry, and underscores the utility of a framework that can identify and suggest functionally-similar alternatives that use less permissions.

Our analysis so far has yielded three key insights:

1. Highly-ranked apps tend to be more permission-hungry.
2. The highest-ranking search results are fairly stable, meaning that high-ranking apps that are overly permission-hungry pose a persistent challenge.
3. Less highly-ranked apps are still highly rated by users, pointing to the availability of high-quality, less permission-hungry alternatives.

Based on these insights, we analyse the extent to which we can suggest suitable alternative apps to users that still satisfy their demand for particular functionality. In this way, we incentivise the development of less permission-hungry apps.

4. FINE-GRAINED CONTEXTUAL PERMISSION ANALYSIS

Fine-grained contextual permission analysis involves looking at permission usage across apps within each search result set. For this analysis, we use *permission prevalence*, a metric that measures how many other apps within a search result set use the same permission. A similar metric was used by Sarma et al. [30] for malware classification. The idea is that if all the apps in a search result set provide similar functionality, any deviations from the norm in terms of permission usage may be a useful indicator of an app being overly permission-hungry (for an app providing that functionality). We define the Individual Permission Prevalence (IPP) of each permission as the fraction of apps in a search result set using that permission. We define App Overall Permission Prevalence (AOPP) as the mean of the IPPs of those permissions used by an app. Algorithm 1 outlines more formally how IPP and AOPP are calculated.

Algorithm 1: Calculate IPP and AOPP for a list of apps

```
Input: List of apps $\beta = \beta_1, \ldots, \beta_n$
Output: IPP, AOPP per app
$permList \leftarrow [\,]$
foreach $app$ in $\beta$ do
    $permList \leftarrow permList + getPermissions(app)$
$IPP \leftarrow \emptyset$
foreach $perm$ in $GetUniqueItems(permList)$ do
    $IPP[perm] \leftarrow permList.count(perm) \div len(\beta)$
$AOPP \leftarrow \emptyset$
foreach $app$ in $\beta$ do
    $temp \leftarrow [\,]$
    foreach $perm$ in $getPermissions(app)$ do
        $temp \leftarrow temp + IPP[perm]$
    $AOPP[app] \leftarrow mean(temp)$
return $AOPP, IPP$
```

As a concrete example, consider four apps: $app_1 = \{A, B, C\}$, $app_2 = \{A, B\}$, $app_3 = \{A, C\}$, $app_4 = \{A\}$, where A, B and C are permissions. IPP and AOPP are calculated as follows:

1. Make list of all permissions, i.e., [A, B, C, A, B, A, C, A].
2. Count occurrences of each permission in list, i.e., $p_A = 4$, $p_B = 2$, $p_C = 2$.
3. IPP is the fraction of occurrences of a permission to the number of apps, i.e., $IPP_A = \frac{4}{4} = 1.0$, $IPP_B = \frac{2}{4} = 0.5$, $IPP_C = \frac{2}{4} = 0.5$.
4. AOPP is the mean of the IPPs for those permissions used by an app, i.e., $AOPP_1 = 0.66$, $AOPP_2 = 0.75$, $AOPP_3 = 0.75$, $AOPP_4 = 1.0$.

4.1 Identifying Rare Permissions

We classified a permission as a *rare permission* if it had an IPP of 5%, i.e., only one app in a search result set of 20 used this permission. Rare permission usage should be considered suspicious, since among a set of 20 functionally-similar apps, only one app used this permission. Approximately 6% of apps that were ranked `#1` for their search query used one or more rare permissions. The most common rare permissions used by `#1` ranked apps were `ACCESS_FINE_LOCATION` (10.5%), `CAMERA` (9%), and `READ_CONTACTS` (8.5%). Apps using rare permissions are not automatically malicious, but should certainly be the subject of more intensive static or dynamic analysis by app stores to detect malfeasance. Across our search query dataset, we analysed which permissions were rare permissions. Our results are shown in Fig. 6. From the figure, approximately 12% of the time the offending permission was `RECORD_AUDIO`. Other 'popular' rare permissions allowed apps to get a user's location, use their camera, or read their contacts.

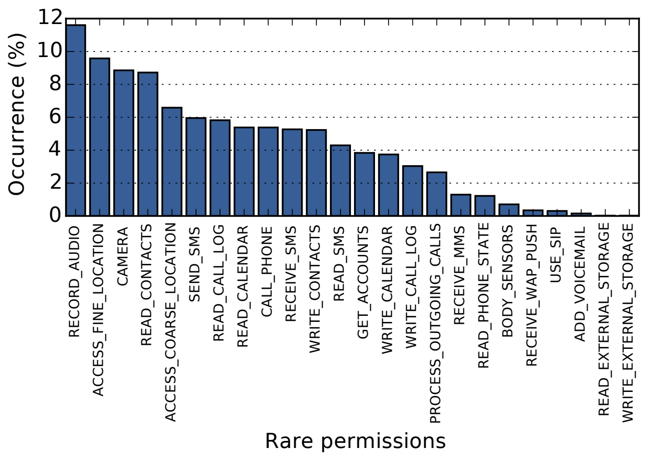


Figure 6: Breakdown of the usage of rare permissions, i.e., permissions with an IPP of 5%.

4.2 Case Study of Rare Permissions

Fig. 6 shows the rare permissions observed in our dataset. We did a manual inspection on 125 of the apps that used rare permissions to determine whether they were dissimilar apps that got into each search result set and were flagged for rare permission usage for that reason (false positives), or they were indeed functionally-similar apps that were using rare permissions (true positives). For the manual inspection, we considered permissions that would allow apps to capture audio/video from a device, access the user's exact location, or cost the user money. Specifically, we randomly chose 25 occurrences from each of `RECORD_AUDIO`, `CAMERA`, `ACCESS_FINE_LOCATION`, `SEND_SMS`, and `CALL_PHONE` to analyse.

Table 1 shows a summary of the results of our manual analysis. We found that 92.8% of the apps were relevant to the search query that they were returned from. That is, we confirmed that the vast majority of apps in this set were not flagged because they were irrelevant search results. Moreover, we consider this 92.8% to represent the lower bound on relevance, because our manual analysis was focused on apps that were flagged for rare permission usage. These apps are the most likely ones to have been false positives in the first place, since their permission usage was so drastically different from the other apps in their search result set. Across our entire dataset, we expected relevance to be near perfect, and we validated that this was the case.

We manually determined whether apps (that were relevant to their query) justified their use of rare permissions. To infer this, we analysed app store information for each app including app description, screenshots, feature listing, and "What's New" section to glean insight as to why the app would need the rare permission. Of the apps that were manually audited, rare permission usage was justified only 22.4% of the time. Worryingly, this 22.4% seems to be inflated by apps using the `CAMERA` permission, whose use was justified in 61% of cases. Apps using the more surreptitious (and dangerous) `CALL_PHONE` and `SEND_SMS` permissions only justified their need for doing so 4% and 9% of the time respectively. These two permissions may have financial consequences as some malicious apps are known to make calls and send SMS messages to premium rate numbers [34, 13].

Table 1: Results of the manual analysis on 125 apps that used rare permissions. Apps were highly relevant to their search query and a significant majority did not justify their rare permission use.

Permission	Relevant?	Justified?
ACCESS_FINE_LOCATION	96%	8%
CALL_PHONE	96%	4%
CAMERA	92%	61%
RECORD_AUDIO	92%	30%
SEND_SMS	88%	9%
Average	**92.8%**	**22.4%**

We used historical snapshots[7] of the Google Play Store to analyse the apps in our dataset that used rare permissions. After removing duplicates (apps appearing more than once for the same rare permission), we had 3452 apps, i.e., approximately 7% of the search query dataset. We discovered that 8.7% (301) of apps using rare permissions added their rare permission within the last 10 months. Worryingly, 31.6% (95) of these apps did not have a corresponding update to their app descriptions to explain any added functionality warranting the use of these permissions. Of these 95 apps, 77 of them had more than 100,000 installs, and 36 had more than 1,000,000 installs.

5. SECURANK FRAMEWORK AND TOOL

Given the prevalence of apps using rare permissions in our dataset, we applied the concept of fine-grained contextual permission usage to analyse the entire Google Play Store. We performed this measurement using a framework and tool called SecuRank[8] (*Secu*rity *Rank*). SecuRank finds functionally-similar apps then determines the ones that most closely seem to be following the principle of least privilege. To infer least privilege, SecuRank uses an app's AOPP as defined in Section 4. The AOPP metric penalises apps that use permissions that are uncommon within their group of functionally-similar apps. Thus, given an input app, SecuRank can identify and suggest alternative apps that more closely obey the principle of least privilege. We used SecuRank to analyse the entire Google Play Store, to report on the extent to which apps had functionally-similar alternatives that used less privileges on users' devices. There are two main steps to populate the database that SecuRank uses:

1. Use a similarity measurement function (similar to the one described in Section 2.3) to compare app descriptions across the Google Play Store to identify and group functionally-similar apps.

2. Compare permission usage within groups to identify whether apps had less permission-hungry alternatives.

Across the Google Play Store, we consider an app to be less permission-hungry (or more closely following the principle of lease privilege), if it has a higher AOPP than a functionally-similar app. Fig. 7 shows the Store-wide percentage of apps having less permission-hungry ('preferable')

[7]Our historical snapshots of the Google Play Store are available upon request.

[8]SecuRank is freely available to use online at https://securank.me/ and as an Android app.

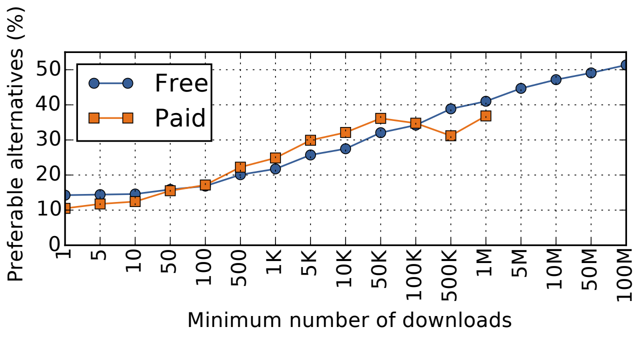


Figure 7: Percentage of apps having 'preferable' alternatives. A preferable alternative is a functionally-similar app with a higher AOPP. ***Note that (at the time of writing) there were no paid apps with five million downloads or more.***

Table 2: Breakdown of the store-wide likelihood of an app to have a preferable alternative depending on whether the app was free or paid.

Cost	Has alternatives	No alternatives
Free	2408 (13.4%)	15508 (86.6%)
Paid	189 (9.1%)	1895 (90.9%)

alternatives, broken down by popularity and cost (free or paid). There is a positive correlation between the minimum number of downloads of an app and the number of alternatives it had, regardless of the cost of the app, i.e., free or paid. The likelihood of a free app having a preferable alternative monotonically increased with the minimum number of downloads of that free app. Approximately one in five apps (free or paid) with 500 downloads or more had a preferable alternative. At the upper end of popularity (50 million or more downloads), approximately one in two free apps had a preferable alternative. This is concerning since it means the more likely an app is to have been installed by users in the first place, is the more likely it is to have a preferable alternative that could have been installed instead.

Across the entire Google Play Store, we tested the overall likelihood of an app to have a preferable alternative depending on whether it was a free or paid app. Data was collected using random sampling with a sample size of 20,000. Table 2 shows the results of our random sample. Free apps were more likely (13.4%) than paid apps (9.1%) to have preferable alternatives. A 2-proportion z-test confirmed that this observation was statistically significant: $p < 0.01$.

The results so far demonstrate that many apps have less permission-hungry alternatives. However, care must be taken to ensure that these alternatives are of good quality if they are to be suggested to users to replace the originals. We use the average user rating of apps (measured out of 5-stars) in the Google Play Store as a proxy for their quality. We analysed the app recommendations made by SecuRank to understand how they compared to the app they were intended to replace. Table 3 presents the result of our analysis. From the table, 53.2% of alternative apps had the same rating (4.9%) or higher (48.3%) than the app they were intended to replace. Of the alternative apps that had a lower rating, 37% of them were within 0.25 stars of the rating of the original app. Thus, a fair portion of alternative apps that had a lower overall rating were still of reasonably close quality to the original. For this reason, we believe that in most cases SecuRank would be able to recommend alternative apps to users that would be considered satisfactory. This is critical if SecuRank is to be adopted.

Table 3: Breakdown of the rating of suggested replacement apps. A majority of replacement apps were more highly rated than the apps they replaced.

Rating	Number of apps
Higher than original	100,760 (48.3%)
Same as original	10,303 (4.9%)
Lower than original	97,714 (46.8%)
Total	**208,777**

6. DISCUSSION

In this section, we identify three stakeholders in the smartphone app ecosystem and discuss the potential barriers to adopting SecuRank from the perspective of each of the stakeholders.

- **Smartphone users** can immediately use SecuRank to find functionally-similar alternative apps that are less permission-hungry.
- **App stores** can use SecuRank to increase the security of the app store by identifying and scrutinising apps that use unusual permissions for the functionality they provide.
- **App developers** can obtain useful feedback on how their apps compare to other apps that provide similar functionality.

6.1 Smartphone Users

SecuRank has the potential to greatly reduce the attack surface of smartphones. The main obstacles that stem from the use of SecuRank involve inaccuracies in app suggestions and users' willingness to use SecuRank and follow its suggestions.

In Section 5, we showed that the alternative apps suggested by SecuRank are typically of similar or better quality than the apps they are intended to replace. However, alternative apps are discovered using similarity metrics, and while manual inspection of the results shows high accuracy, inaccurate suggestions are inevitable. To combat this problem, future versions of SecuRank could incorporate user feedback about the suitability of app suggestions, which could then be used to inform and improve the similarity measurement algorithms. Additional characteristics of apps such as libraries used, API calls, and network endpoints could also be leveraged to more accurately determine which apps are similar.

Various tools have been proposed to help improve security and privacy on smartphones. While these tools prove useful, user adoption can be a challenge, and SecuRank is not immune to such a challenge. We believe, however, that SecuRank may not suffer as much as other tools since it is straightforward to use and provides valuable feedback every use, without itself needing to be run frequently. Moreover, users could opt-in to have SecuRank applied for them at the app store level, to have their search results automatically tailored to apps that are less permission-hungry.

6.2 App Stores

App stores can improve their profile by using SecuRank to identify permission-hungry apps. This use of SecuRank would have the greatest impact on the health of the ecosystem since permission-hungry apps would be penalised at the highest level. The main obstacle to the adoption of SecuRank by app stores comes from the reduction in the freedom of app developers during app development. We must make it clear that permission-hungry apps are not automatically malicious. Indeed, many successful apps achieved success from using permissions in novel ways to provide attractive features over their competitors. In other words, app developers may benefit from being free to use and request permissions as they see fit. However, competition has led to the current situation where competing apps are continuously trying to outdo each other in terms of features provided. While competition is good for users in one sense, these same users now suffer from the fact that their smartphones are filled with permission-hungry apps. Unsurprisingly, permission-hungry apps are very attractive targets to attackers, as they provide an additional (and sometimes more effective) avenue to privilege escalation on a smartphone.

We propose a compromise in the use of SecuRank, to satisfy the requirements of the freedom of app developers while at the same time assisting those users who want less permission-hungry apps. As mentioned above, app stores could use SecuRank internally to provide less permission-hungry apps to those users that select the option. Additionally, app stores could automatically suggest (during app search ranking) less-privileged apps to users running outdated smartphones to help minimise their already inflated attack surface. These two strategies remove the effort required by informed users to manually find less-privileged apps, and transparently assist less informed and more vulnerable users to reduce their attack surface. Additionally, app stores can use SecuRank to identify apps that use permissions that are ‘suspicious’ for the functionality that they provide. These apps can then be more thoroughly vetted before being made available for public consumption.

6.3 App Developers

In the current model, app developers may be hesitant to adopt SecuRank because there is no penalty for developing highly-privileged apps. In fact, developers stand to benefit from highly-privileged apps since an app’s bundled libraries, such as ad libraries, inherit the permissions granted to the app. Ad libraries are already known to leverage as many permissions as are available to them, ostensibly for better advertisement targeting. We envisage app stores that use SecuRank in their ranking algorithms (in some cases) to recommend less permission-hungry apps to users. The existence of such app stores may provide the necessary incentive to develop apps that use less privileges.

6.4 Limitations and Future Work

SecuRank, in its current form, uses an app’s description from the app store as a proxy to determine the app’s functionality. Given that app developers (presumably) are trying to market their app in a way that maximises their number of downloads, we believe that an app’s description indeed describes most of the functionality provided by the app. However, in comparing two apps for similarity of function, we relied on a cosine similarity measure of the app descriptions. Cosine similarity may not be robust enough to capture all the required information from an app since it may misunderstand context and semantics. If this is the case, then more advanced NLP techniques would need to be employed to solve the problem. Additionally, we plan to integrate SecuRank with other sources of information such as network endpoints, libraries used, and API usage within apps. This is expected to improve the overall precision of SecuRank when grouping apps since false positives will be reduced. Additionally, we plan to integrate SecuRank with existing tools such as AutoCog [27] so that after identifying rare permission usage, we can automatically determine if an app has justified its use of these permissions in its app description.

7. RELATED WORK

Felt et al. [11] took an early look at the permissions required by Android apps. They studied over-privilege in Android apps and built a tool, called Stowaway, that detects whether an app required too many permissions. The authors applied their tool to 940 Android apps and found one-third of them to be requesting more permissions than they needed. They then investigated the reason for this over-privilege, and found that the developers seem to be following the principle of least privilege, but are sometimes confused due to insufficient API documentation. This work is important, since it shows that the Android ecosystem contains a significant number of over-privileged apps. Complementary to this, we evaluate the official app marketplace itself to determine the extent to which users can be assisted in avoiding permission-hungry apps altogether.

To determine whether the Android permission system was effectively protecting users, Felt et al. [14] conducted two usability studies to determine whether users paid attention to or understood the permission information they were presented during app installation. Worryingly, the authors discovered that only 17% of participants paid attention to permissions at install-time and a paltry 3% were able to correctly answer three comprehension questions. In a similar study, Kelley et al. [17] also found that people are generally unaware of the security risks from mobile apps. This is concerning, since the security and privacy-protection mechanisms are in place (most recently run-time permissions), but users are typically unable or unwilling to make the most of them. By contrast, we do not focus on quantifying the extent to which users are effectively guided by permissions. Rather, we examine the ways in which users can be assisted in finding less-privileged apps to install in the first place, in a way that is potentially transparent to them.

Sarma et al. [30] tackle the problem from a direction similar to ours, in that they not only analyse the permissions used by an app, but they leverage additional information such as the category that an app belongs to, and the permissions requested by apps in the same category. The authors show that machine learning techniques utilising Support Vector Machines (SVMs) can effectively identify malware by understanding the rareness of certain permissions in apps in general. Their work boasts a malware detection rate of up to 81% with a warning rate of only 8%. Peng et al. [26] do a similar analysis, but they use probabilistic models in their risk scoring schemes and show that their models outperform existing approaches. In a similar vein, Gorla et al. [16] use clustering by description topic to identify outliers with respect to permission usage. By contrast, we do not look at the general characteristics of an app to attempt to justify its use of permissions. Rather, we gain greater contextual insight through our large-scale analysis of permission usage across apps that are functionally-similar.

Several authors consider using permission request and usage patterns to detect malfeasance. Frank et al. [15] used a clustering based approach and determined that low-reputation apps deviated from the typical permission request patterns of apps with a high reputation. Chia et al. [8] find that some apps attempt to mislead users into granting permissions. Papamartzivanos et al. [25] propose a cloud-based crowdsourcing solution to detect privacy violations by apps. Zhang et al. [33] propose a platform called VetDroid that leverages permission use analysis to identify information leaks. All these authors look at permission request patterns at a high level without the context of app functionality.

More recently, Wijesekera et al. [32] studied the extent to which apps accessed resources that are protected by permissions. They use the concept of contextual integrity to explore usage of protected resources when smartphone users were not expecting it. They conducted surveys and discovered that 80% of their respondents had a desire to block over one-third of all requests. Eling et al. [10] show that 40.4% of users still accepted glaring run-time permission requests for minimal reward. This points to the need for assisting users in finding lower-permission app alternatives in the first place, and our work fills this gap.

Finally, a number of authors leveraged natural language processing to understand how advertised app functionality related to an app's use of permissions. One of the earliest work that did this was WHYPER from Pandita et al. [24]. They used NLP techniques to determine whether they could identify sentences from an app's description that hinted at the need for particular permissions. Qu et al. [27] subsequently outperformed WHYPER and highlight the general problem of "low description-to-permission fidelity". Watanabe et al. [31] propose a framework, ACODE, and find that it gives comparable performance to WHYPER. These pieces of work suffer from the fact that they will unnecessarily flag permissions as being suspicious simply because an app's description does not allude to their need. By contrast, we do not use app descriptions to infer the need for permissions. Instead, we use app descriptions to find and group apps that are functionally-similar, then leverage contextual permission usage within groups to identify rare permission usage.

8. CONCLUSION

In this paper, we presented SecuRank, a framework and tool that identifies and suggests functionally-similar alternatives to replace permission-hungry apps. SecuRank leverages app descriptions to identify groups of apps with similar functionality, before using contextual permission analysis within groups to identify unusual permission usage. Within a sample of the Google Play Store, we observed that 7% of apps used permissions that were rare for apps of that functionality. Across the entire Google Play Store, we found that free apps and popular apps were more likely to have a less permission-hungry replacement. Worryingly, we found a positive correlation between the likelihood of an app having a less permission-hungry alternative and the popularity of that app. As smartphone and app usage continues to skyrocket, there is an increasing concern about the access that third-parties have to our data. SecuRank is a well-needed tool in the arsenal for taming permission-hungry apps and unscrupulous app developers.

9. REFERENCES

[1] A. Acquisti. Privacy in Electronic Commerce and the Economics of Immediate Gratification. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, pages 21–29, New York, NY, USA, 2004. ACM.

[2] A. Acquisti and J. Grossklags. Losses, Gains, and Hyperbolic Discounting: An Experimental Approach to Information Security Attitudes and Behavior. In *2nd Annual Workshop on Economics and Information Security (WEIS)*, volume 3, pages 1–27, 2003.

[3] A. Acquisti and J. Grossklags. Privacy and Rationality in Individual Decision Making. *IEEE Security and Privacy (S&P)*, 3(1):26–33, Jan 2005.

[4] Android. Requesting Permissions at Run Time. http://developer.android.com/training/permissions/requesting.html, December 2015.

[5] Android. System Permissions. http://developer.android.com/guide/topics/security/permissions.html, December 2015.

[6] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[7] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.-R. Sadeghi, and B. Shastry. Towards Taming Privilege-Escalation Attacks on Android. In *NDSS*, 2012.

[8] P. H. Chia, Y. Yamamoto, and N. Asokan. Is This App Safe?: A Large Scale Study on Application Permissions and Risk Signals. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 311–320, New York, NY, USA, 2012. ACM.

[9] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring User Confidence in Smartphone Security and Privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 1:1–1:16, New York, NY, USA, 2012. ACM.

[10] N. Eling, S. Rasthofer, M. Kolhagen, E. Bodden, and P. Buxmann. Investigating Users' Reaction to Fine-Grained Data Requests: A Market Experiment. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 3666–3675, Jan 2016.

[11] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

[12] A. P. Felt, S. Egelman, and D. Wagner. I'Ve Got 99 Problems, but Vibration Ain't One: A Survey of Smartphone Users' Concerns. In *Proceedings of the 2nd ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 33–44, New York, NY, USA, 2012. ACM.

[13] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A Survey of Mobile Malware in the Wild.

In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

[14] 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 8th Symposium on Usable Privacy and Security (SOUPS 2012)*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[15] M. Frank, B. Dong, A. Felt, and D. Song. Mining Permission Request Patterns from Android and Facebook Applications. In *IEEE 12th International Conference on Data Mining (ICDM)*, pages 870–875, Dec 2012.

[16] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking App Behavior Against App Descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035, New York, NY, USA, 2014. ACM.

[17] P. Kelley, S. Consolvo, L. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A Conundrum of Permissions: Installing Applications on an Android Smartphone. In J. Blyth, S. Dietrich, and L. Camp, editors, *Financial Cryptography and Data Security*, volume 7398 of *Lecture Notes in Computer Science*, pages 68–79. Springer Berlin Heidelberg, 2012.

[18] J. Lee. No.1 Position in Google Gets 33% of Search Traffic [Study]. http://searchenginewatch.com/sew/study/2276184/no-1-position-in-google-gets-33-of-search-traffic-study, June 2013.

[19] M. Lins. Google Play Apps Crawler. GitHub Repository https://github.com/MarcelloLins/GooglePlay AppsCrawler, February 2014.

[20] D. Metzler, S. Dumais, and C. Meek. Similarity Measures for Short Segments of Text. In G. Amati, C. Carpineto, and G. Romano, editors, *Advances in Information Retrieval*, volume 4425 of *Lecture Notes in Computer Science*, pages 16–27. Springer Berlin Heidelberg, 2007.

[21] R. Mihalcea, C. Corley, and C. Strapparava. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press, 2006.

[22] Nielsen. The State Of Mobile Apps. http://www.nielsen.com/content/dam/ corporate/us/en/newswire/ uploads/2010/09/NielsenMobileAppsWhitepaper.pdf, June 2010.

[23] P. A. Norberg, D. R. Horne, and D. A. Horne. The Privacy Paradox: Personal Information Disclosure Intentions versus Behaviors. *Journal of Consumer Affairs*, 41(1):100–126, 2007.

[24] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13, pages 527–542, Berkeley, CA, USA, 2013. USENIX Association.

[25] D. Papamartzivanos, D. Damopoulos, and G. Kambourakis. A Cloud-based Architecture to Crowdsource Mobile App Privacy Leaks. In *Proceedings of the 18th Panhellenic Conference on Informatics*, PCI '14, pages 59:1–59:6, New York, NY, USA, 2014. ACM.

[26] H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Using Probabilistic Generative Models for Ranking Risks of Android Apps. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, CCS '12, pages 241–252, New York, NY, USA, 2012. ACM.

[27] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen. AutoCog: Measuring the Description-to-permission Fidelity in Android Applications. In *Proceedings of the 21st ACM Conference on Computer and Communications Security*, CCS '14, pages 1354–1365, New York, NY, USA, 2014. ACM.

[28] J. H. Saltzer. Protection and the control of information sharing in multics. *Commun. ACM*, 17(7):388–402, July 1974.

[29] B. Sanz, I. Santos, X. Ugarte-Pedrero, C. Laorden, J. Nieves, and P. G. Bringas. Anomaly detection using string analysis for android malware detection. In *International Joint Conference SOCO '13 CISIS '13 ICEUTE '13*, pages 469–478, 2014.

[30] B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy. Android Permissions: A Perspective Combining Risks and Benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 13–22, New York, NY, USA, 2012. ACM.

[31] T. Watanabe, M. Akiyama, T. Sakai, and T. Mori. Understanding the Inconsistencies between Text Descriptions and the Use of Privacy-sensitive Resources of Mobile Apps. In *Proceedings of the 11th Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 241–255, Ottawa, July 2015. USENIX Association.

[32] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. Android Permissions Remystified: A Field Study on Contextual Integrity. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security 15)*, pages 499–514, Washington, D.C., Aug. 2015. USENIX Association.

[33] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang. Vetting Undesirable Behaviors in Android Apps with Permission Use Analysis. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, pages 611–622. ACM, 2013.

[34] Y. Zhou and X. Jiang. Dissecting Android Malware: Characterization and Evolution. In *IEEE Symposium on Security and Privacy (IEEE S&P)*, pages 95–109, May 2012.

[35] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *NDSS*, 2012.