# Security-Centric Ranking Algorithm and Two Privacy Scores To Mitigate Intrusive Apps

Fadi Mohsen[1], Hamed Abdelhaq[2], and Halil Bisgin[3]

[1]Computer Science, Bernoulli Institute, University of Groningen, the Netherlands, f.f.m.mohsen@rug.nl
[2]Computer Science Apprenticeship, An-Najah National University, Palestine, hamed@najah.edu
[3]Computer Science, Engineering, and Physics, University of Michigan-Flint, USA, bisgin@umich.edu

*Abstract*—**Smartphone users are constantly facing the risks of losing their private information to third-party mobile applications. Studies have revealed that the vast majority of users either do not pay attention to privacy or unable to comprehend privacy messages. Developers though have exploited this fact by asking users to grant their apps an enormous number of permissions. In this paper, we propose and evaluate a new security-centric ranking algorithm built on top of the Elasticsearch engine to help users evade such apps. The algorithm calculates an intrusiveness score for an app based on its requested permissions, received system actions and users privacy preferences. As such, we further propose a new approach to capture these preferences. We evaluate the ranking algorithm using a million Android applications, contextual data and APK files, that we collect from the Google Play store. The results show that the scoring and reranking steps add minor overhead. Moreover, participants of the user studies gave positive feedback for the ranking algorithm and the privacy preferences solicitation approach. These results suggest that our proposed system would definitely protect the privacy of mobile users and pushes developers into requesting least amount of privileges. Still, there are many risks that endanger the users' privacy.**

*Keywords*-**Android, Permissions, Apps, Elasticsearch, Pilot study, Privacy, Intrusive, Users, Mobile, Applications, User study**

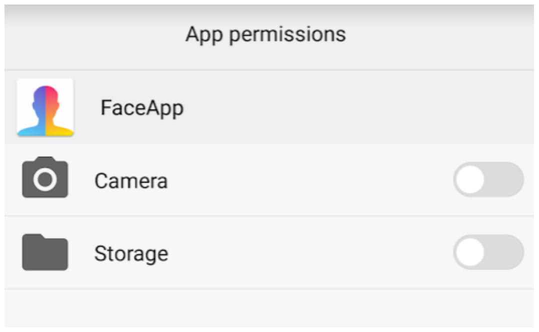
## I. Introduction

Modern mobile ecosystems have been known to support third-party applications essentially to improve users' experience. This is exemplified in the number of apps in online markets such as Google Play and iOS App stores. For instance, the number of available apps in the Google Play store was placed at 2.6 million apps in December 2018 compared to 1 million apps in July 2013 [32]. Although a previous report by the same source [31] stated that the number of apps in the Google Play store was estimated to be 3.5 million apps in December 2017. A drop that was caused by Google removing a large number of apps that had violated its privacy and security policies besides other factors [36]. Prior to this, numerous reports have shed light on the large spread of malicious Android apps that are placed in official and non-official online markets. For example, in Q3 2017 report, Kaspersky lab mobile security products had detected more than 1.5 million malicious installation packages [26]. Consequently, there have been several efforts to detect malicious apps [2], [6], [11], [13]–[17], [19], [22], [24], [28], [29], [35], [37], [41], [42], [44]

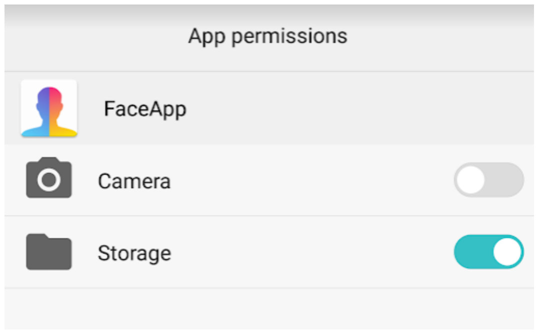
These approaches have been successful in detecting and removing malicious apps that were built intentionally to harm users. However, detecting intrusive applications is more challenging and requires distinct techniques. Intrusive applications are not malicious but instead intrude user's privacy by asking for too many permissions/privileges. These apps, referred to as *intrusive apps*, are most likely offered free-of-charge, and hence, users might be tempted to give up privacy for perceived benefits in using these apps. There are two types of risks pertaining intrusive apps. First, they invade users' privacy by collecting sensitive data from their devices. Second, apps with numerous privileges are more prone to be attacked by malicious apps [25]. Countering intrusive applications have been the focus of many research projects for almost a decade now [8], [12], [23], [30] However, in the past recent years, there has been a shift in countering intrusive mobile apps. The new shift is geared towards helping users make information decisions about the apps they install or keep on their phones [14], [30], [35]. Yet, these works count on the Google Play store to infer the similarity between apps, use only the permissions information to calculate privacy scores, require users to first install an app before the solution offers any alternatives, and finally they do not integrate users' preferences into calculating the privacy scores. In this work, the functionally similar applications are recognized by our implementation instead of relying on the Google Play store. As such, we use the Elasticsearch engine [7] to index over a million Android applications that we collect from the Google Play store. Functionally-similar apps are then retrieved by the engine based on criteria we define. Thus, we are eliminating any kind of bias the Google Play store might introduce upon the retrieval process. Second, the intrusiveness score for the mobile apps is calculated based not only on the permissions but also on the system actions and more importantly on user's privacy preferences. Finally, the formula for calculating the intrusiveness score does not use simple averaging, which results in generating more accurate privacy scores.

Thus, in this work, we are proposing a new security-centric ranking algorithm to mitigate intrusive mobile apps. The ranking algorithm revolves around calculating intrusiveness scores for Android mobile applications based on the permissions they possess, the system actions they receive and the privacy preferences of the user. Therefore, we propose a new approach to solicit user's permissions/privacy preferences. We evaluate this approach by conducting an online user study. We also conduct a performance benchmarking study and a pilot user study to evaluate the accuracy and the efficiency of the ranking algorithm and the Elasticsearch engine. Our dataset is composed of 1,945,056 Android apps, 325,444 of them has the APK files, that we collect from the Google Play store and then index using the Elasticsearch engine. In the last stage, which is the extension of our earlier work [18], we first investigate the intrusiveness score distribution of a large dataset of apps including their APK files, namely, 1,152,676 apps. In addition to the previous 325,444 APKs, we further collect and process about 827,232 APKs. The purpose of this work is to study the degree of alternativity in all genres of the Google Play store. The outcome of this study shall give us an indication on whether our approach is going to be feasible in regards to finding and recommending less intrusive apps. Second, we conduct a new performance benchmarking study to investigate the impact of adding more apps to the corpus on the indexing and the retrieval processes. Last, we compare our work with the work of Taylor and Martinovic [35].

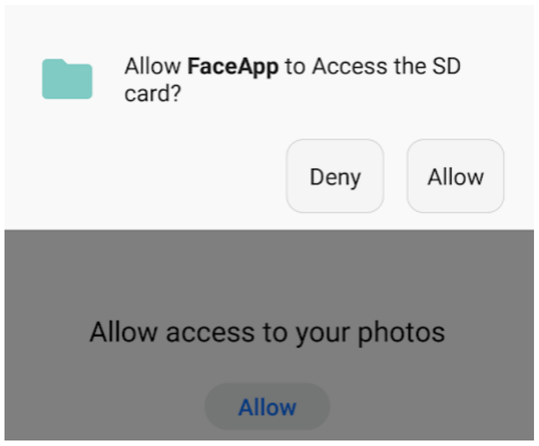
The rest of the paper is organized as follows. Starting with the preliminaries in Section II, Section III discusses the online user study for soliciting users preferences, Section IV discusses the methodology we followed to propose and build the security-centric algorithm on top of Elasticsearch, Section V explains the evaluation studies, Section VI shows the effect of increasing the dataset size and the results of the comparison study, Section VII presents the related works, and finally we end with the conclusion section, Section VIII.



(a) Initially the app has zero dangerous permissions



(b) The app has now access right over *Storage*.


Allow **FaceApp** to Access the SD card?
Deny
Allow
Allow access to your photos
Allow

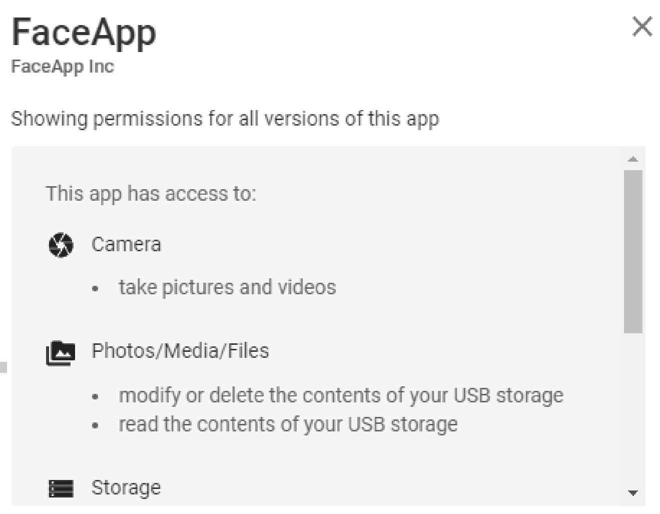
(c) An app permission prompt to the end user.

Fig. 1. Users through *Settings* can control the level of access each app can get or wait for the prompt.

## II. Preliminaries

### A. Permissions

The Android OS uses a permission system [13] to control the level of access each application can have while running on a mobile device. For instance, an application needs to have certain permissions to access a resource or data that resides outside of its sandbox. Application developers must include all the permission requests in the configuration, *AndroidManfiest.xml*, files of their applications. There are four permission types: *normal*, *dangerous*, *signature*, *signature or system*. Unlike dangerous permissions, Android approves the normal permission requests without user's consent. However, to approve the dangerous permission requests, the system either prompts the user upon installing an app or at run-time. If the API level or the SDK of an app is 22 or lower, the system would prompt the user to grant the app the dangerous permissions during the installation. However, if it is 23 or higher, the system would prompt the user to grant the dangerous permissions at run-time. In Table I we show the list of Android dangerous permissions organized into 9 groups. In Listing 1 we are listing some of the Android normal permissions. The signature and signature or system permissions are used for certain



FaceApp
FaceApp Inc

Showing permissions for all versions of this app

This app has access to:

Camera
- take pictures and videos

Photos/Media/Files
- modify or delete the contents of your USB storage
- read the contents of your USB storage

Storage

Fig. 2. Google Play store website displays the list of permissions the FaceApp app requires to be installed.

situations in which multiple apps of the same vendor or multiple vendors need to share specific features or data between them. In addition to introducing the run-time permission granting, beginning with Android 6 (API level 23), the Android system started giving users the ability to revoke permissions from any app at any time. Regardless of these two recent changes in the Android permission system, we believe that our work is going to significantly improve the privacy of mobile users. First, the recent changes has not significantly improved the privacy of users. This is because it follows a ”click once but permanent acceptance” model [1], and users in reality infrequently use these features, as we will show in Section **??**. Second, our work will continue to be relevant to a large spectrum of users due to fragmentation, the biggest security issue of the Android OS, many versions of the Android operating system and the update is quite slow [34].

## B. *Android Manifest*

Android operating systems use Android Package Kit (APK) as the standard package file format for the distribution and installation of mobile and smart TV applications. The APK contains among other things, the executable source code and a configuration file called *AndroidManifest*. The manifest file contains numerous information about the app such as permission requests, activities, services, content providers and broadcast receivers. It also specifies the hardware and software features the application usually requires in order to function properly. In the previous versions of Android and upon installing an Android app on a mobile device, the Android operating system is used to read the app’s manifest file and displays a prompt to the end user requesting her approval of the dangerous permission requests. Nowadays, users may first explore the Google Play store on their computers to locate some apps, read their descriptions, check out the reviews, and for cautious users they might have a look at the permission requests. Users can also view the permissions of an app on the Google Play store by clicking the *View details* link under the *permission* section. They would normally be presented with a screen like the one we shown in Figure 2. Then, if she decides to install the app on her mobile device, the app gets installed with zero dangerous permissions as we shown in Figure 1(a). The user can later grant and revoke certain permissions from/to certain applications. For example, the app we show in Figure 1(b) has the permissions that would allow it to read and write from/to the storage of the device. Developers can also program their apps to display runtime permission prompts like the one we show in Figure 1(c). Navigating between these screens is not a trivial task because most users are unable to interpret the permission requests and unaware of their security and privacy implications [10]. In addition, comparing between different apps requires even more work. Thus, these screens are clearly inadequate to most users and do not help them make informative decisions.

## C. *Broadcast Receivers*

This component allows Android applications to listen to events originating from the system or other applications. For example, an application can register to be notified whenever there is a new SMS, voicemail, or phone call. The Android system allows an unconditional listening to the majority of these events and restricts it for few. The restriction though entails requiring the applications to have mostly normal permissions, which do not need user’s approval. Broadcast receivers can provide applications with a lot of information such as the device state, connection state, and whether the user is present or not. This information might be very useful but it could also pose some privacy risks. The results of a previous research showed that it is more likely for malicious apps to contain a broadcast receiver component in comparison to benign applications [20]. Consequently, in this paper, the broadcast receivers of system events/system actions are being considered when calculating an intrusiveness score for Android applications.

```
ACCESS_LOCATION_EXTRA_COMMANDS
ACCESS_NETWORK_STATE
ACCESS_NOTIFICATION_POLICY
ACCESS_WIFI_STATE
BLUETOOTH
BLUETOOTH_ADMIN
BROADCAST_STICKY
```

Listing 1. Example of Android normal permissions.

## III. User Study

As we mentioned earlier in Section II-A, the Android system places permissions into various groups

TABLE I
THE DANGEROUS PERMISSIONS ARE PLACED INTO 9 CATEGORIES. THE USAGE PERCENTAGE OF EACH PERMISSION IS PLACED BETWEEN PARENTHESIS.

| Group | Permissions (Usage %) |
|---|---|
| CALENDAR | READ_CALENDAR (1.5)<br>WRITE_CALENDAR (1.55) |
| CAMERA | CAMERA (11.8) |
| CONTACTS | READ_CONTACTS (3.9)<br>WRITE_CONTACTS (1.95)<br>GET_ACCOUNTS (13.55) |
| LOCATION | ACCESS_FINE_LOCATION (18.54)<br>ACCESS_COARSE_LOCATION (18.51) |
| MICROPHONE | RECORD_AUDIO (6.44) |
| PHONE | READ_PHONE_STATE (20.72)<br>CALL_PHONE (5.61)<br>READ_CALL_LOG (0.36)<br>WRITE_CALL_LOG (0.22)<br>ADD_VOICEMAIL (0.01)<br>USE_SIP (0.08)<br>PROCESS_OUTGOING_CALLS (0.63) |
| SENSORS | BODY_SENSORS (0.01) |
| SMS | SEND_SMS (2.18)<br>RECEIVE_SMS (1.63)<br>READ_SMS (0.9)<br>RECEIVE_WAP_PUSH (0.05)<br>RECEIVE_MMS (0.08) |
| STORAGE | READ_EXTERNAL_STORAGE (14.21)<br>WRITE_EXTERNAL_STORAGE (45.94) |

according to their sensitivity. Yet, users and developers alike have no control over the content or the dynamic of these groups. Previous research showed that there are significant discrepancies among mobile users with regard to their privacy perceptions, i.e., the value of each smartphone asset (e.g., contacts, pictures, messages) is perceived differently across users [21]. As such, we decided to incorporate users' input into calculating the intrusiveness score of each mobile app. Thus, in this work, we propose a new method for capturing users' privacy perception about the different permission sets. To evaluate this method, we conduct an online user study in which we present the permission groups in a simple format and then ask participants to rate them over multiple rounds. The steps are depicted below:

- Round 1, display a graphical representation of the ten permission groups (9 dangerous permission groups and 1 normal permission group) such as: SMS, STORAGE, and PHONE. Participants are then asked to choose their top seven most sensitive permission groups. The three permission groups that are not picked are excluded and given an importance level of 1.
- Round 2, display the seven permission groups that are picked in round 1 and ask participants to choose five of them. The two remaining groups are excluded and given an importance level of 2.
- Round 3, display the five permission groups that are picked in round 2 and ask participants to choose three. The two remaining groups are given an importance level of 3.
- Round 4, display the three permission groups that are picked in round 3 and ask participants to choose a group. The selected group is assigned the importance level of 5 and the other two are assigned the importance level of 4.
- Round 5, display all the permissions of the most important group and ask participants if they agree to grant all these permissions the same level of importance (level 5). This step is not required for the MICROPHONE and SENSORS groups because each one of them has only one permission.

We believe that taking users' preferences into account when scoring applications is crucial in generating customized and precise scores. The work of [35] for example treats all permissions with the same level of importance, even if some of these permissions are not important to a particular user or to all users. Consequently, all users will get the same results regardless of their attitude towards privacy and usage pattern [4].

In Table II, we show all the permission groups that we used in the online user study, a total of ten groups. Nine of them contain only dangerous permissions. In addition, we created a new group, called *STATE*, to contain a number of normal permissions, which we believe are important and could have an impact on users' privacy. There have been some research efforts that demonstrated the possibility of comprising users' privacy using only normal permissions [20]. For each permission group, we provide a simple description and a representative picture. The STATE category has 5 permissions as we shown in Listing 2. Those permissions allow an app to access and change the WIFI and NETWORK states.

```
ACCESS_WIFI_STATE
ACCESS_NETWORK_STATE
CHANGE_NETWORK_STATE
CHANGE_WIFI_MULTICAST_STATE
CHANGE_WIFI_STATE
```

Listing 2. The permissions of the State category.

Once the user assigns a sensitivity level to a group, all the permissions in that group would get the same sensitivity level, which could be seen as a stretch. However, there is a trade-off between fine-grained permission ranking and keeping the process simple and short for the users. In Table I we show the usage percentages for the dangerous permissions among all the APKs in our dataset, more than a million. Some of these permissions are rarely used, such as the ADD_VOICEMAIL and USE_SIP. On the other hand, there are permissions that are used extensively such as WRITE_EXTERNAL_STORAGE and READ_PHONE_STATE.

## A. Participants

We used Amazon Turk [3] to recruit participants with a level of confidence of about 90%, a qualification that

TABLE II
FOR EACH PERMISSION CATEGORY/GROUP, WE PROVIDE A SINGLE DESCRIPTION THAT SHOWS THE AGGREGATED PRIVILEGES OF ALL PERMISSIONS OF THAT CATEGORY.

| Permission Group | Description |
|---|---|
| CALENDAR | Allows an application to read and write your calendar data. |
| CAMERA | Allows an application access your phone's camera. |
| CONTACTS | Allows an application to read and write your contacts data. Moreover, it allows access to the list of your accounts . |
| LOCATION | Allows an app to access your approximate and precise location. |
| MICROPHONE | Allows an application to record audio. |
| PHONE | Allows an application to have full control over the calling service on your device such as placing and redirecting a call and reading and writing call logs. |
| SENSORS | Allows an application to access data from sensors that you may using to measure what is happening inside your body, such as heart rate. |
| SMS | Allows an application to receive SMS, MMS, and WAP Push messages. It also allows to send and read SMS messages. |
| STORAGE | Allows an application to write and read to/from external storage. |
| STATE | Allows an application to access and change the states of WIFI and NETWORK connections. |

states that over 90% of the assignments a worker has completed have been accepted by the requesters. We paid each participant $0.35. We had a total of 116 participants correctly completed the study. Participation in the study took an average of 6 minutes and 49 seconds. Conducting an effective user study necessitates recruiting participants who represent the target group/end users as close as possible. Participant demographics of this study can be found in Table V. As it appears, the set of study participants is not quite balanced in gender, age, and Android knowledge due to the nature of recruiting participants from Amazon Turk. In addition, our focus was primarily set on obtaining workers with high confidence. This imbalance might have an impact on the precision of the final results for the underrepresented groups.

## B. Study Description

The aim of this study is twofold: first, to test the effectiveness of our proposed permissions preferences solicitation approach; second, to get an idea about user's perceptions and habits with regard to Android permissions before and after installing apps into their mobile phones. In this study, participants are asked to navigate to a website that we developed mainly for this study. The website lets participants setup their permission preferences. At the end of the first part, participants are given a code and a URL to the second part of the study. The second part is a survey, namely a Google web form, which is composed of three sections: the first section gives participants the opportunity to reflect on and evaluate the first stage of the study. The second part is focused on studying participants' understanding and usage of two features pertaining Android permissions: Android run-time permission requests and the revoke features. Finally, a section that is meant to understand participants usage habits of Android mobile devices.

## C. Analysis

We use the time taken by each participant to complete the first part of the questionnaire as a basis to measure the effectiveness of our approach. Additionally, we examine participants' responses to the open ended and closed ended questions.

*1) Effectiveness:* Participants were asked to rate the difficulty level of using the website as well as the time it took them to finish the task. The rating goes from 1 to 7 where 1 means the system was extremely difficult to use and 7 was extremely easy. Participants gave the website an average rating of 5.8 for its simplicity, and 5.7 for time efficiency. This result suggests that our approach is easy to use and does not consume too much time. Participants were also asked if they believe that Google Play store and other online stores should take users' permissions preferences into consideration when recommending apps to them. The average answer to that suggestion was 5.9 out of 7, which suggests a potential for our approach. We further measured the effectiveness of our approach by analysing participants' responses to two open ended questions concerning the things that they liked and disliked the most. We could identify four main topics from these responses: *look*, *feel*, *innovation* and *other*. Twenty-six participants have indicated that the design of the website was the thing they liked the most. Thirty-four participants have indicated that they liked mostly how simple and easy to use the website is. Fifteen participants have indicated that what they liked mostly was the idea of how they were made to state their preferences by narrowing down the selections. Forty-three participants either mentioned irrelevant information or just written "Other". As per the dislikes, we came up with four categories as well: *nothing*, *approach*, *design*, and *other*. Sixty-three participants have indicated that there is nothing to dislike about the approach and/or the website. Twenty participants have submitted critiques and suggestions to improve our approach. For instance, they did not like its repetition aspect, namely, having the same question being asked over and over many times. A participant suggested instead to put all the permission groups in one page and then ask participants

TABLE III
THE AVERAGE, MEDIAN AND STANDARD DEVIATION VALUES FOR THE PERMISSION RATINGS.

| | AVG | STDEV |
|---|---|---|
| Location | 3.45 | 1.28 |
| Contacts | 3.31 | 1.20 |
| Camera | 3.27 | 1.16 |
| Phone | 3.00 | 1.27 |
| SMS | 2.73 | 1.15 |
| Microphone | 2.65 | 1.34 |
| Storage | 2.47 | 1.30 |
| Sensors | 1.81 | 1.10 |
| State | 1.69 | 1.00 |
| Calendar | 1.51 | 0.85 |

to rank them all at once. Eight participants disliked the website design. Twenty-five participants provided unclear and/or irrelevant feedback.

*2) Time Consumption:* We also looked at the amount of time each participant has spent on the website to setup her permissions' preferences. We calculated the average, median, minimum, maximum and standard deviation for all participants. The average time taken to use the website for this population was 127 seconds, median = 101.5, min = 16 seconds, max = 1300 seconds, and standard deviation was 130 seconds. The average and median values suggest that the time needed by an Android user to setup her permission preferences using our approach should be between a minute and a half and two minutes, which sounds a very short time, especially if you combine it with the fact that users shall do this once in a while.

*3) Permission Groups Ratings:* Once a participant is finished from setting up her permission preferences, each permission group must get a rating value from the set $\{1, 2, \ldots, 5\}$, where 1 means least sensitive. To test whether there was any significant difference between the permission groups with regard to their sensitivity level among all participants, a comparison of the repeated measures using Friedman's test was applied [39]. Looking at the test's results, there was a statistically significant difference in participants' assumptions regarding the sensitivity level of each permission group

$$\chi^2(9, N = 116) = 273.254, p<0.001$$

. Then, a post hoc analysis with Wilcoxon signed-rank tests [40] (a non-parametric statistical hypothesis test used when comparing two repeated measurements) was conducted with a Bonferroni correction [38] applied, a method used to counteract the problem of multiple comparisons, resulting in a significance level set at p <0.0001 (the above p value divided by 10, because we have 10 different permissions groups). Median (IQR) sensitivity levels for the Calendar 1 (1 to 2), Camera 4 (2.25 to 4), Contacts 3 (2 to 4), Location 4 (2 to 4.75), Microphone 3 (1 to 4), Phone 3 (2 to 4), SMS 3 (2 to 4), Sensors 1 (1 to 2), State 1 (1 to 2), and Storage 2 (1 to 3.75). Table IV shows the results of running post hoc analysis with Wilcoxon signed-rank tests on the ten permission groups. The word ”Sig” in the table means that there is a significant difference between the two permissions groups, e.g., Calendar and Camera, Camera and Microphone, and Location and SMS. However, if there is no significant difference, the p value is displayed, for instance, the p value for Calendar and Sensors is 0.28. Finally, the "NA" means that the test is inapplicable. Table III shows the average and standard deviation values for the ratings of each permissions group given by all participants. Based on the average, the permission groups can be put into three clusters: the first cluster contains permission groups with an average rating of 3 and more, the second cluster has the permission groups with an average rating greater than 2 but less than 3, and the last cluster contains permission groups with average rating less than 2. From Table III and Table IV, we conclude that the Location, Contacts and Camera are the most sensitive permission groups among all participants because they have the highest average values besides there is a significant difference between them and the permission groups in clusters 2 and 3. Another interesting thing to notice from these tables is the State permission group, the average rating is not the lowest as it supposed to be. In addition, there was no significant difference between it and the Calendar and the Sensor groups.

*4) Android Related Features:* We also included a part in the survey, which tends to acquire participants feedback about similar features that were added recently to Android phones. There were two features, the run time permissions and the revoke permission features. Both of these features are supported on any device that runs Android 6.0 (Marshmallow) or higher. About twenty two percent of participants (22/116) were using mobile devices that run Android 5.1 (Lollipop) or lower, and 76% (88/116) were using mobile devices that run Android 6.0 (Marshmallow) or higher. We asked participants the following question: "If you are currently using or have used a mobile device that is running Android 6.0 (Marshmallow) or higher, have you ever noticed or interacted with the run time permission requests (e.g., the one shown in the below picture)?" Seventy eight percent of participants (90/116) answered "Yes", almost ten percent (11/116) answered ”No", and thirteen percent (15/116) indicated that they never used a device that is running Android 6.0 (Marshmallow) or higher. A similar question was asked about the revoke permission feature, forty eight percent of participants indicated that they have heard and used the feature (56/116), and almost twenty percent (23/116) said that they heard about it but never used it, and 26 percent (30/116) answered "no" but that they would love to try it, and finally six percent (7/116) indicated that they never heard of it and they are not interested in using the feature.

TABLE IV
THE RESULTS OF THE POST HOC ANALYSIS WITH WILCOXON SIGNED-RANK TESTS ON THE DIFFERENT PERMISSION GROUPS.

| | Calendar | Camera | Contacts | Location | Microphone | Phone | SMS | Sensors | Sate | Storage |
|---|---|---|---|---|---|---|---|---|---|---|
| Calendar | NA | Sig | Sig | Sig | Sig | Sig | Sig | .28 | .272 | Sig |
| Camera | Sig | NA | .789 | .567 | Sig | .05 | Sig | Sig | Sig | Sig |
| Contacts | Sig | .789 | NA | .446 | Sig | .052 | Sig | Sig | Sig | Sig |
| Location | Sig | .567 | .446 | NA | Sig | .023 | Sig | Sig | Sig | Sig |
| Microphone | Sig | Sig | Sig | Sig | NA | .076 | .552 | Sig | Sig | .332 |
| Phone | Sig | .05 | .052 | .023 | .076 | NA | .075 | Sig | Sig | .003 |
| SMS | Sig | Sig | Sig | Sig | .552 | .075 | NA | Sig | Sig | .076 |
| Sensors | .28 | Sig | Sig | Sig | Sig | Sig | Sig | NA | .343 | Sig |
| State | .272 | Sig | Sig | Sig | Sig | Sig | Sig | .343 | NA | Sig |
| Storage | Sig | Sig | Sig | Sig | .332 | .003 | .076 | Sig | Sig | NA |

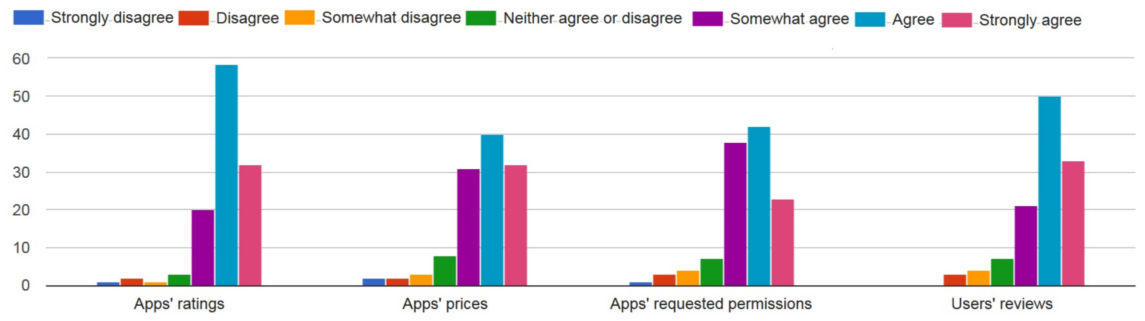


Fig. 3. Summary of participants answers to the question: When choosing between two or more features-identical apps i take in consideration.

*5) Android Related Features (User Satisfaction):* Out of the 116 participants, 90 participants indicated that they have noticed and interacted with the Android runtime permission feature, the average satisfaction rate among them was 5.5. Yet, some participants indicated that they did not like the feature for different reasons. For instance, they did not like the pop ups that show up all the time, which would be irritating. They also criticized the pop ups for not providing any details to help the user make informative decisions. For example, according to the participants, the pop up must explain the consequence of clicking either of the two displayed options. Out of the 116 participants, 56 participants have known and used the revoke permission feature with an average satisfaction rate of 5.8. Very few participants have mentioned reasons for not liking this feature. For instance, one participant did not like the feature because the apps would stop functioning correctly if some of their permissions are revoked, as demonstrated in this work [5]. Another mentioned that the process of revoking a permission requires going through multiple layers of settings and it always takes her a while to get back.

*6) Installation Decision:* Part of the survey aimed at understating the factors that normally influence the users' decisions when choosing between two or more apps with identical features. Four factors were considered: ratings, price, requested permissions, and reviews. In Figure 3 we display a summary of participants' responses to this part of the study. Noticeably, all factors seemed to be important to the majority of participants.

TABLE V
PARTICIPANTS DEMOGRAPHY

| Variable | n(%) |
|---|---|
| **Gender** | |
| Male | 70(60) |
| Female | 46(40) |
| **Age** | |
| 18-24 | 15(13) |
| 25-34 | 71(61) |
| 35-44 | 22(19) |
| 45 or older | 8(7) |
| **Ethnicity** | |
| Asian/Pacific Islander | 67 (58) |
| Black/African-American | 8 (7) |
| White/Caucasian | 33(28) |
| Other | 8(7) |
| **Education** | |
| Some college | 17(15) |
| Associate's degree | 5(4) |
| Bachelor's degree | 60(52) |
| Master's degree | 28(24) |
| Other | 6(5) |
| **Proficiency in Android** | |
| Basic | 17 (15) |
| Intermediate | 66 (57) |
| Advanced | 33(28) |

The numbers were very close, the average values were as follows: 5.92 for ratings, 5.66 for prices, 5.52 for permissions, and 5.80 for reviews. We further conducted a comparison of the repeated measures using Friedman's test, there was no statistically significant difference in participants' assumptions about those factors,

$$\chi^2(3, N = 116) = 6.910, p = .075$$

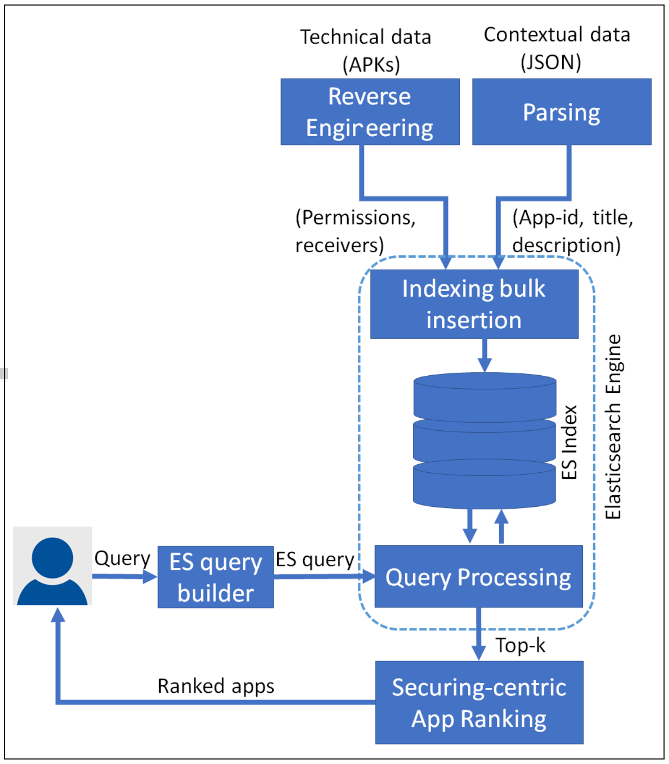


Fig. 4. The overall methodology of our work.

. The result suggests that participants give actually each one of these factors the same importance level. It could also be linked to the way we asked the question. Would the results be different if we had used a different approach like the one we used with the permissions for instance?

*7) Demography:* The distribution of participants' demography was as follows (see Table V): of the 116 participants who used the website and completed the questionnaire, 60% (70 participants) were Male and 40% (46 participants) were Female. When it comes to age, the majority (61%) were ages 25-34, (19%) were between 35-44, (13%) were between 18-24. The remaining (7%) fell between the ages of 45-74. With regard to education, (52%) of them had completed an undergraduate degree, whereas, (24%) had completed a graduate degree, (19%) had completed some college or an associate degree, and the remaining (5%) had either some high school or a high school/GED. Lastly, (28%) of the participants were proficient in using Android phones, (57%) had intermediate proficiency, and the remaining (15%) had basic proficiency.

## IV. Methodology

The overall methodology of our work is depicted in Figure 4. In this section we will go over all the steps in detail. Our security-centric ranking algorithm is based on assigning intrusiveness scores to the apps based on the permissions they possess, the system actions they register to receive, and user's privacy preferences. The intrusiveness score of an app as a function of its system actions is solely calculated in relative to the collection of apps in the result set. On the other hand, the intrusiveness score of an app as a function of its permissions is calculated in relative to the returned apps besides the preferences of the user. Our algorithm is based on an existing method proposed by Taylor and Martinovic [35] with a few major modifications. First, our algorithm incorporates not only the permissions but also the system actions. Second, our algorithm uses a new formula to calculate the final scores instead of simply using the average, which resulted in producing more useful scores as we show in Section VI-E and Section VI-F. Third, our algorithm considers the privacy preferences of users.

### A. Secure Search Engine

Ranking Android applications before displaying the final results to the user might be a computationally-expensive task, in particular when having a huge set of applications. Therefore, an inevitable step to such system is to index the content of these applications. Once indexed, the large space of applications will be pruned, and thus, only those applications relevant to a user's query will be efficiently retrieved and passed on to the ranking task. Thus, we utilized Elasticsearch engine [7], a popular text indexing framework, to index the set of all applications. Since the title and the description of an app is a good indicator of its purpose, we decided to give these two fields more attention. We used the tri-gram indexing scheme to index the content of both fields. By doing so, a user would still be able to obtain the right apps even in case s/he poses a query containing a substring of the app title or description.

### B. Query Processing

When a user submits a query string to search for a certain app, an Elasticsearch query is generated and sent to the index. The most relevant applications will then be efficiently retrieved based on the title and the description fields. An example of an Elasticsearch query is depicted in Figure 5. Our current implementation takes only less than 80 ms to search the 1,945,056 records and retrieve relevant applications, have the query's tokens in either their title or description fields. We then re-rank all the relevant applications based on their intrusiveness scores and returns only the top 20 privacy-preserving apps.

### C. Intrusiveness Score as a Function of Receivers

An Android receiver lets application listen to system actions. The Individual Receiver Prevalence (IRP) of each receiver is defined as the fraction of apps in a search result set using that receiver. The App Overall Receiver Prevalence (AORP) is a function IRPs of those receivers

```
"query": {
  "bool": {
    "should": [ {
      "match": {
        "Title": {
          "query": titleQuery
        }
      }
    },
    {
      "fuzzy": {
        "Title": {
          "value": titleQuery,
          "boost": 3.0,
           "fuzziness": 2,
          "prefix_length": 4,
          "max_expansions": 30
        }
      }
    },
    {
      "match": {
        "Description": {
          "query": descQuery
        }
      }
    },
    {
      "fuzzy": {
        "Description": {
          "value": descQuery,
          "boost": 12.0,
          "fuzziness": 1,
          "prefix_length": 10,
          "max_expansions": 30
        }
      }
    }],
    "minimum_should_match": 1,
    "boost": 1.0
  }
}
```

Fig. 5. A sample Elasticsearch query. Both the title and description fields are used to search for relevant apps.

used by an app and formulated as follows:

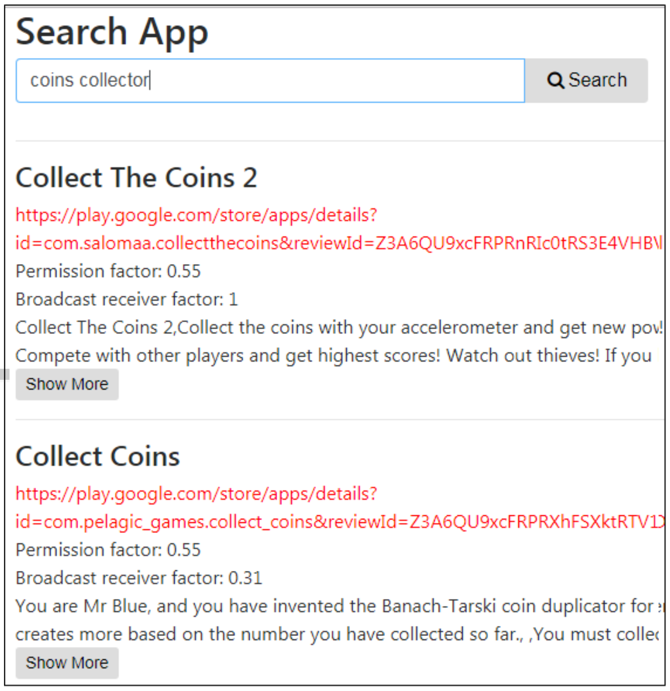
$$AORP_i = \frac{-1}{[c \times \sum_{IRP \in App_i} \log(IRP)] - 1} \tag{1}$$

Each app has none, one or more receiver. For each receiver the IRP is calculated as the fraction of occurrences of a receiver to the number of apps. For instance, if we have a set of 4 apps, and two of them are using *receiver 1* then, the IRP of this receiver is 50% which is high. This implies that this is a popular one, and thus a safe receiver. However, if there are 100 apps and only four of them are using *receiver 1*, then its IRP is 5%, and that implies it is rarely requested and thus a dangerous receiver. In both cases, the IRP value of *receiver 1* is used to calculate the value of AORP for the app that has the receiver in its manifest file. The main motivation behind using a new formula other than the simple average is to prevent receivers with high IRP values from dominating the overall AORP score, since at least one intrusive receiver should eliminate the positive impact of non-intrusive receiver. The parameter $c$ is a declining coefficient that controls the amount of AORP drop when having an app with some low IRPs, i.e., dangerous receivers. The larger the value of $c$, the more sensitive the AORP we get towards IRPs with lower values. For example, assume having an app $i$ that has three receivers with the following IRP values: $IRP_1 = 1$, $IRP_2 = 1$, and $IRP_3 = 0.2$. By applying our equation, Eq. 1, we will get an $AORP_i$ of 0.42. This makes more sense than simply averaging all values, literature's equation, which would result in an AORP of 0.73. Noticeably, the simple average approach is more easily affected by extremely large or small values, which leads into punishing good apps for having only one dangerous receiver, and favoring bad apps for having only one safe receiver. Empirically, we chose a value of $c = 2$ by experimenting with a number of apps and calculating their IORPs using a number of $c$ values.

We chose 5% to be the threshold value for the IRP. We then calculated the IRP value for each receiver and the AORP value for each application. Remember that neither the IRP (5% threshold value) nor the AORP values are used to exclude any app from the list. These values are merely used to re-rank the apps before returning them back to the user. Thus, if an app, e.g., MyApp, contains one or more rare receivers with IRP value of 5% then its AORP value will be lower than those apps containing non-rare receivers. Thus, MyApp would then go to the bottom of the list. In addition to lowering their rank in the list, the information of rare receivers could also be displayed in red font to alert users. We leave this to future work.

## D. Intrusiveness Score as a Function of Permissions

The concepts and the formula from the previous section are also being applied to the permission with one distinction. The permission formula incorporates and use the privacy preferences of users to calculate the final intrusiveness score of an app. Integrating users' preferences in calculation the AORP requires replacing the IRP parameter in the formula with a new version that is updated by the input received from the users' ratings. Our premise is that the rating a permission receives from user is more informative than the statistically-calculated IRP value. Hence, we can view the IRP value



Search App

coins collector

Search

Collect The Coins 2

https://play.google.com/store/apps/details?
id=com.salomaa.collectthecoins&reviewId=Z3A6QU9xcFRPRnRIc0tRS3E4VHB\

Permission factor: 0.55

Broadcast receiver factor: 1

Collect The Coins 2,Collect the coins with your accelerometer and get new pov

Compete with other players and get highest scores! Watch out thieves! If you

Show More

Collect Coins

https://play.google.com/store/apps/details?
id=com.pelagic_games.collect_coins&reviewId=Z3A6QU9xcFRPRXhFSXktRTV1

Permission factor: 0.55

Broadcast receiver factor: 0.31

You are Mr Blue, and you have invented the Banach-Tarski coin duplicator for

creates more based on the number you have collected so far., ,You must colle

Show More

Fig. 6. Secure search engine in action: the first two items have same scores on the permission side but different scores on the receivers side.

just as a weight regularizing the rating on a particular permission. After soliciting users' preferences, we end up having the permission groups divided into 5 levels, where level-5 holds permissions with highest sensitivity. Thus, we map each permission $p_i$ to the inverse of the level of the group that $p_i$ is a member in. For example, if $p_i$ is a member in group $g_5$ (having a level of 5), then

$$g(p_i) = \frac{1}{5}$$

Therefore, our AORP formula can be redefined as:

$$AORP_i = \frac{-1}{[c \times \sum_{IRP \in App_i} \log(IRP \times g(p))] - 1}$$

### E. Overall Score

The overall score will be calculated based on the receiver and the permission scores. Both scores might be averaged to get the final score. Another way to use the two scores would be to first rank the apps based on one of them and then use the other one to break any potential tie. The current prototype implements this approach. Figure 6 shows the first two items of the search results of searching for "coins collector" terms. The two apps have identical score based on the requested permissions, yet, the second score, which is based on the registered receivers, was used to break the tie between them. A comparative study of the two approaches might be done in the future.

| #apps | Type | Collection Date |
|---|---|---|
| 1,945,056 | Contextual | 2017 |
| 325,444 | APKs | 2017-2018 |
| 827,232 | APKs | 2018-2019 |

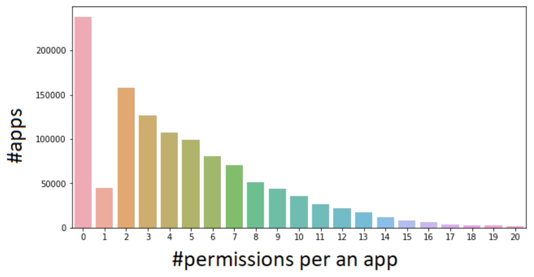
TABLE VI
SUMMARY OF THE CRAWLED DATASET.

## V. Evaluation

The purpose of the evaluation study was twofold: first, to measure the overhead of our ranking algorithm via a benchmarking study. Second, to test the accuracy of the Elasticseareh engine in returning relevant results, using a pilot study. From several fields composing each app record, we utilize only the app id, the app title, and the app description. In addition, for 325,444 applications, we used their requested permissions and received system actions extracted from their APK files. Since the process of crawling APKs is time consuming, in the first phase of the study we only used 325,444 applications, corresponding to a random sample from the entire set of crawled apps. In Section VI, the second part of the study, we used 827,232 further applications. Table VI shows the summary of the dataset we crawled and used in our evaluations. For each app, we maintained two lists of configurations; one containing the set of permissions (137 permissions) and another list for the broadcast receivers (175 receivers). Figure 7(a) shows a histogram depicting the usage distribution of permissions and Figure 7(b) shows the histogram of broadcast receivers. As can be seen, the majority of apps ask for a few number of permissions. For example, about 17.5% of the apps require user to give exactly 2, 3, or 4 permissions, while only 3.8% of the apps ask for more than 10 permissions. In addition, there are some permissions that are requested by a large number of apps. For instance, nearly 38.6% of the apps had requested the INTERNET permission and 35.5% of them ask for ACCESS_NETWORK_STATE permission. One can easily notice that these two permissions are safe, while the permission BIND_CARRIER _MESSAGING_SERVICE that is not requested at all in our dataset is considered dangerous. Therefore, the rank of any app requesting BIND_CARRIER_MESSAGING_SERVICE should be downgraded as described in Section IV-D. These findings suggest that our proposed solution is going to be effective in recommending less intrusive apps given that a large number of the apps in the dataset are configured towards requesting fewer permissions.

### A. Renarking Overhead

Real-time retrieval of relevant apps is key in app recommendation systems. To this end, we evaluated the runtime performance of our app reranking system. For this purpose, we posed 7 queries to the system and


(a) Permissions Distributions Among 99.4% of all apps.

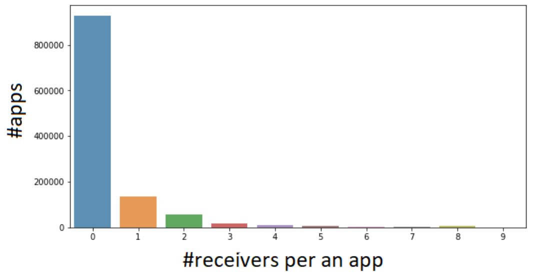

(b) Receivers Distributions among 99.8% of all apps.

Fig. 7. The distribution of the two security related features of the Apps that were collected in the first phase .

TABLE VII
TIME (IN MILLISECONDS) REQUIRED TO RETRIEVE RELEVANT RESULTS FOR UNI-, BI-, AND TRI-GRAM TOKENS.

| | **ES query + reranking** | | **App reranking** | |
|---|---|---|---|---|
| | **Top-10** | **Top-30** | **Top-10** | **Top-30** |
| **Unigram** | 332.29 | 264.28 | 3.33 | 4.66 |
| **Bi-gram** | 452.15 | 765.8 | 3.02 | 3.56 |
| **Tri-gram** | 1038.57 | 1208.5 | 3.04 | 3.371 |

took the average runtime. This experiment is repeated on different parameter settings, namely, (1) uni-, bi-, and tri-grams query tokens, and (2) top-10 and top-30 retrieved results. The experimental platform is based on an Intel(R) Xeon(R) CPU E5-2680 v3 (2.4 GHz) with 16GB memory, running on Windows-7 Professional. The same experiment is repeated after tripling the number of APKs to show if there will be any change, see Section VI.

Table VII shows the collected runtime statistics for the entire retrieval process (ES query and app reranking), and the runtime for the reranking step only. The numbers suggest that reranking the top-k results retrieved from ES is very efficient and is accomplished within less than 5ms.

## B. Pilot Study

Recruitment of participants was done by announcing in classrooms and Blackboard [1], the learning management system used at our campus. We had 28 participants complete the pilot study, consisting of 20 questions. Participants were asked to use the search engine by submitting three different queries and look closely at the top five results. Following that, participants had to answer questions pertaining to their backgrounds, demography, and experience with using the search engine in particular, how relevant the returned results were. Participants were given the choice to make two types of queries, type 1: simple query (e.g., Activity tracker), type 2: query using "with" (e.g., Activity tracker with heart monitor, cloud support). We conducted three types of analysis: overall, between types, and between result items. Overall, the search engine returns relevant results according to all participants. There was a significant difference between type 1 and type 2, type 2 produces less relevant results. No difference in relevance between the 5 items. To test whether there is a significant difference between the result items with regard to their relevance among all participants. We conducted a comparison of the repeated measures using Friedman's test. Looking at the test's results, there is no statistically significant difference.

$$\chi^2(4, N = 84) = 4.959, p > 0.001$$

To test whether there is a significant difference between the two query types with regard to their relevance among all participants. We conducted a comparison of the repeated measures using Friedman's test. Looking at the test's results, there is a statistically significant difference between the two:

$$\chi^2(1, N = 150) = 17.017, p <= 0.001$$

The mean is 5.68 for type 1 and 4.82 for type 2, which shows that type 1 returned more relevant results compared to type 2. However, the number of submitted queries of type 1 are way more than those of type 2, besides, it could be the case that students did not fully understand type 2. As per the overall all 420 submission, 28 participants, 3 queries and 5 research items, the average is 5.32 (N = 420) , which shows that the search engine returns relevant results.

## VI. INCREASING THE DATASET SIZE

For the purpose of studying the effect of the dataset size on the performance of our apps search engine we downloaded a new batch of APK files from the Google Play store, extracted the technical information from their AndroidManifest files, and updated the Elastsearch index. We then re-evaluated the indexing and the search processes to note if there was any discrepancy from the previous evaluation.

### A. More APK files

In this work, two crawlers were built and utilzed to collect the dataset shown in Table VI. Using the first

[1]https://en.wikipedia.org/wiki/Blackboard_Learn

TABLE VIII
TIME (IN MILLISECONDS) REQUIRED TO RETRIEVE RELEVANT RESULTS FOR UNI-, BI-, AND TRI-GRAM TOKENS AFTER ADDING MORE APPS.

| | **ES query + reranking** | | **App reranking** | |
|---|---|---|---|---|
| | **Top-10** | **Top-30** | **Top-10** | **Top-30** |
| **Unigram** | 321.14 | 340.22 | 4.79 | 5.83 |
| **Bi-gram** | 461.24 | 463.64 | 4.94 | 5.31 |
| **Tri-gram** | 937 | 935.3633 | 4.47 | 7.253 |

crawler, a total of 1,945,056 apps were collected from the Google Play store (e.g., id, description, ratings, etc.). The second crawler was used to download the respective APK files, a total of 325,444 APKs were collected and reverse engineered for the technical information (e.g., permissions and Broadcast receivers). In this extension, we ran the second crawler for over a year to get additional apps, more than 827,232 respective APKs were actually collected and processed.

## B. Evaluation

In this section we show the evaluation results of the search engine after updating its index. It is a replica of the evaluation that was discussed in Section V. The focus of this evaluation is on the indexing step as well as on the retrieving and reranking steps, which entails retrieving and reranking the apps before displaying them to the end user. Our results suggest that the retrieving and reranking time has not noticeably changed despite the drastic change in the number of applications/technical information in the index. In Table VIII we show the results of our performance analysis after boosting the size of our corpus. Overall, the overhead caused by tripling the dataset size is not significant and almost negligible. In fact, some numbers have even dropped like in case of the tri-gram top-10 and top-30, the ES query and reranking times dropped down, from 1038.57 to 937, and from 1208.5 to 935.36, respectively. Moreover, for those numbers that have risen, the increase was not actually significant. These results suggest that the dataset size and the speed with which the data is being retrieved and ranked are not correlated. The new update though has somehow altered the time needed to index the entire records. For instance, indexing the 1,945,056 apps where 1,152,676 of them have technical information, under Elasticsearch (ES) took about three hours in comparison to 2.2 hours, previously obtained with less applications, Section V. It is noteworthy to mention that the indexing process is not done frequently. In addition, it is usually done offline, thus, the user experience would not be affected.

## C. Calculating Intrusiveness Scores

In addition to expanding the corpus size, the intrusiveness score values are calculated for 1,152,676 applications. The scores are primarily based on the equation mentioned in Section IV-C with two distinctions. First,

TABLE IX
A SAMPLE OF TEN APPLICATIONS AND THEIR GENRE, DEVELOPER AND INTRUSIVENESS SCORE INFORMATION.

| **ID** | **Genre** | **Developer** | **Score** |
|---|---|---|---|
| app1 | Productivity | zafer ertas | 0.148 |
| app2 | Sports | DropSwitch | 0.645 |
| app3 | Card | ZZZ Solitaire | 0.059 |
| app4 | Card | ZZZ Solitaire | 0.059 |
| app5 | Strategy | zy_emb | 0.243 |
| app6 | Board | ZZZ Solitaire | 0.139 |
| app7 | Books & Reference | ZviSoft | 1.000 |
| app8 | Health & Fitness | Nordavind | 0.064 |
| app9 | Tools | zsoltz | 0.111 |
| app10 | Tools | Nologi (Zakariya ZMK) | 0.055 |

the score of an app is calculated in relative to all applications in its genre instead of all applications in the result set as per the definition. The rationale behind this score is that the apps of the same genre most likely requests similar permissions. Therefore, if an app is requesting permissions that are not common to its genre, its AORP score would certainly be lower. Secondly, the user preference value for each permission is hard coded and calculated as the average rating of all participants for that particular permission, Section III.

## D. Measuring The degree of Alternativity

In this section we show the results of the analysis that we have conducted on the intrusiveness scores of over a million Android applications. The core of our approach relie on the fact that for every query a user submits there shall be a number of functionally-similar applications that would satisfy the query. These applications would most likely differ in their permission needs and expectations. Thus, our solution can be effective in aiding users avoid intrusive apps if there are apps in every genre with a wide range of permission needs. For the purpose of calculating the new scores, applications of the same genre are considered functionally-similar in the context of the equation discussed in Section IV-C

In Table IX we show a sample of ten mobile apps along with their genres, developers, and the newly calculated scores. For example, *app7* has a score of 1, indicating that the app either requests no permissions at all or requests only common permissions among its peers in the ”Books & Reference” genre. On the contrary, *app3*, *app4*, and *app10* have lower intrusiveness scores because of the number and/or type of permissions they request in comparison to their peers in the ”Card” and ”Tools” genres, respectively. Figure 10(a) gives an overall view of the scores distribution for all applications. There is a decent number of apps, between 20-25%, with scores 1, which in fact means they do respect users’ privacy. In Figure 8 we show the distribution of scores for the genres that are not listed in Table X based on a five-number summary.

Overall, the vast majority of genres have no outliers, the red squares. The height of the box plots is an indica-

TABLE X
THE FIRST ROW CONTAINS THE TOP 10 GENRES WITH THE HIGHEST AVERAGE INTRUSIVENESS SCORES AND THE THIRD ROW IS FOR THE LOWEST 10. THE SECOND, FORTH, AND FIFTH ROWS SHOW THE NUMBER OF APPS IN EACH GENRE, MEDIAN, AND STANDARD DEVIATION, RESPECTIVELY.

| **Genre** | **Libraries&Demo** | **Board** | **Arcade** | **Art&Design** | **Puzzle** | **Word** | **Casual** | **Music** | **Educational** | **Trivia** |
|---|---|---|---|---|---|---|---|---|---|---|
| **#Apps** | 3289 | 5334 | 43473 | 3022 | 47335 | 4285 | 43735 | 1562 | 97276 | 8039 |
| **Mean** | 0.491 | 0.406 | 0.397 | 0.394 | 0.397 | 0.369 | 0.368 | 0.368 | 0.366 | 0.364 |
| **Median** | 0.298 | 0.228 | 0.232 | 0.428 | 0.208 | 0.218 | 0.191 | 0.205 | 0.205 | 0.204 |
| **SD** | 0.429 | 0.365 | 0.359 | 0.324 | 0.361 | 0.351 | 0.356 | 0.348 | 0.354 | 0.358 |
| **Genre** | **Travel&Local** | **Communication** | **Auto** | **Events** | **Food** | **Beauty** | **Dating** | **Maps** | **Music** | **Photography** |
| **#Apps** | 38916 | 24009 | 1903 | 1277 | 6863 | 1788 | 703 | 12909 | 54433 | 25499 |
| **Mean** | 0.2959 | 0.298 | 0.298 | 0.298 | 0.297 | 0.296 | 0.295 | 0.92 | 0.279 | 0.25 |
| **Median** | 0.101 | 0.228 | 0.232 | 0.428 | 0.109 | 0.125 | 0.087 | 0.103 | 0.205 | 0.097 |
| **SD** | 0.369 | 0.388 | 0.362 | 0.371 | 0.347 | 0.340 | 0.359 | 0.363 | 0.348 | 0.338 |

*Some of the genres' names were truncated in the table: Food & Drink → *Food*, *Maps&Navigation* → *Maps*, *Music&Audio* → *Music*, *Auto&Vehicles* → *Auto*.

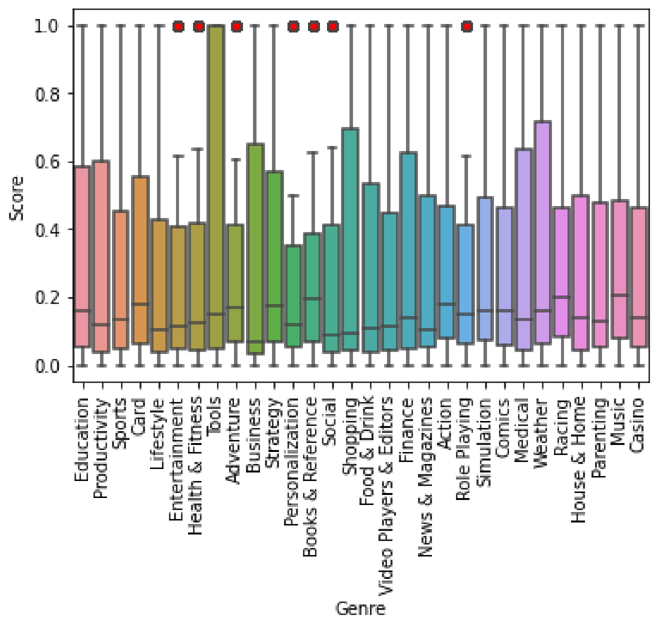


Fig. 8. The intrusiveness score values for the remaining genres excluding the one displayed in Table IX.

tive of the level of agreement among the different apps of the same genre. When the box plot is comparatively short it means that most apps have similar scores, *e.g., Personalization*. On the other hand, when the box plot is comparatively tall it means that most apps have quite different scores, *e.g., Tools*. As the figure clearly shows almost all the genres have apps that are variable in their scores, which serves our goal in providing the user with privacy-preserving alternatives.

Additionally, the average intrusiveness score of all apps in each genre are calculated and sorted in a descending order, we show the top 10 genres and the lowest 10 genres in Table X. In that table, we also show the number of apps in each genre, the median and standard deviation of the intrusiveness scores. The average scores of these genres are slightly different, the same also applies on the standard deviation values, which suggests that the alternativity is quite high.

## *E. Score as a Dependent Factor*

In this section, we study the factors that determines the score of an app. The factors that we considered are obtained from the contextual data that we crawled off the Google Play store app's page. After conducting a thorough experimentation that included considering all factors, sampling and analyzing the variance, the followings are the factors that were selected and further studied; genre, reviews average, number of downloads, content rating, price, android version, and ratings. We then applied the analysis of variance (ANOVA) on these factors to explain the variance in the intrusiveness scores.

In Figure 9 we show the results of conducting the ANOVA test on the entire dataset. While the majority of the variance in the intrusiveness scores cannot be explained by the observed variables, we found that genre, downloads, content rating, and android version contributed significantly higher than other factors such as the reviews, price, and ratings. Those poorly contributing factors such as the level of rating, i.e., number of stars, were excluded from the chart to better illustrate our findings and emphasize their relative effect on the variance. In summary, ANOVA showed that content rating has the greatest impact on the variance and it is followed by genre that accounts for 7% of the variance, which indicates the fluctuation in intrusiveness across different genres, i.e., gaming vs. education. Android version and downloads have the same effect (6%) while the aggregated impact of reviews average, price, and ratings is only 2%. The genre, downloads, content rating, and android version contributed significantly higher than other factors such as the reviews, price, and ratings. Other factors as we mentioned earlier scored very poorly during the initial experimentation that we decided to remove them from the final results. For example, the five-star ratings, four-start ratings, three-start ratings, two-star ratings, and the one-star ratings had almost no influence on the intrusiveness score.

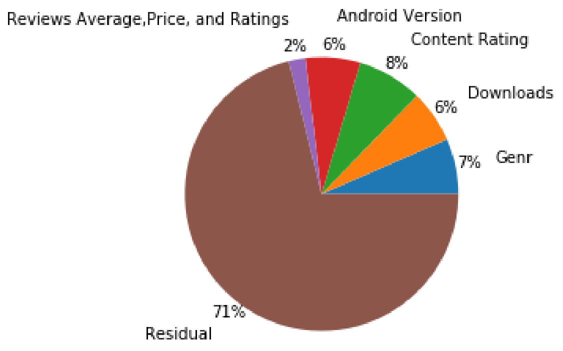


Fig. 9. The results of applying the ANOVA test on the entire dataset.

| Test | Score | One-Star Rating | Two-Star Rating |
|---|---|---|---|
| Pearson | Our Score | -0.0084 | -0.0091 |
| | Taylor Score | -0.0015 | -0.0016 |
| Kendall | Our Score | -0.0004 | -0.0089 |
| | Taylor Score | 0.0092 | 0.0099 |

TABLE XI
THE CORRELATION VALUES BETWEEN THE TWO SCORES AND THE ONE AND TWO STAR RATINGS BASED ON PEARSON AND KENDALL.

### *F. Comparative Analysis*

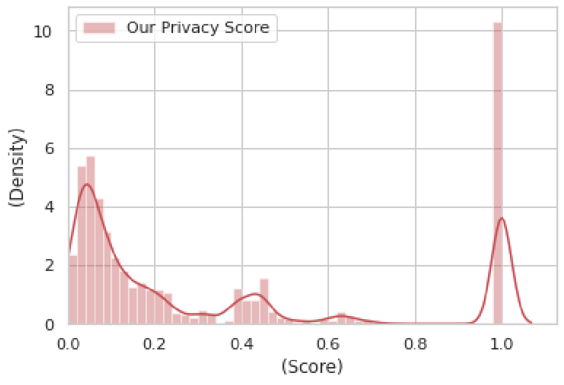
In the previous sections, we explained how our approach is fundamentally more accurate than Taylor and Martinovic [35] approach. In this section, we aim to conduct a comparative analysis between the two approaches. In doing so, we first calculate new scores for the 1,152,676 applications based on Taylor and Martinovic approach. Then, we use the Pearson and Kendall correlations to measure the strength of association between the two scores and the number of users who had given an app low rating (one-star and two-star). The association between the number of users giving an app low rating and any of the two scores must ideally be strong and negative. The Person results indicate that both scores have negative associations with both ratings; however, our score has stronger correlations. The Kendall results though shows that our score still maintains a negative association with the two ratings, however, Taylor's score shows a positive correlation. We show the summary of the results in Table XI. The results provides some evidence into the usefulness of both scores; however, our score is more robust.

Another thing we used to compare between the two scoring approaches is by comparing their distributions. In Figure 10(b) we show the distribution of Taylor and Martinovic scores, and in Figure 10(a) we show the distribution of our scores. While our scoring system pushes several apps below 0.2, Taylor and Martinovic scoring has relatively very few number of apps in that range, which may result in high number of false negatives. In other words, their scoring may let high intrusive apps into the returned result set whereas our approach may act more selectively with the expense of some false positives. Also, the low variation in their scores does not offer alternative apps to users for the same genre as apps that have very similar scores are not well differentiated based on their scores. We make the dataset that contains over a million apps, their contextual features, and the two privacy scores available for researchers to conduct further studies on here [9].

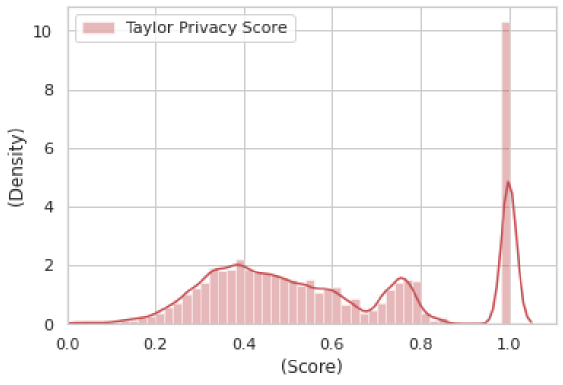
## VII. Related Works

Felt et al. examined whether the Android permissions system is effective at warning users by conducting online and laboratory studies [10]. The results showed that Android permission warnings do not help most users make correct security decisions. As such, many researchers have been trying to propose various kinds of solutions to batch, replace, and/or support the existing permissions system. The work of [14] is considered the first attempt for building the privacy into the App marketplace. The privacy information used in this work is composed of permission ratings that are gathered from human and using automated sources, the privacy information is then presented to the user to aid them make informative decisions. Taylor and Martinovic [35] presented a solution to counter what they call starving permission-hungry apps. The solution offers users alternatives apps to replace currently installed ones. The alternatives apps are determined based on the market search engine. It is worth mentioning that the formal two works are only considering the permissions to calculate the privacy score. Sarma et. al [30] on the other hand used both the permissions an app requests, the category of the app, and the permissions that are requested by other apps in the same category to infer the privacy score that would be used to help users decide on what to install. Wang et. al proposed a framework for quantitative security risk assessment for both Android permissions and apps based on permission request patterns from benign apps and malware. Jing et. al [16] presented a continuous and automated risk assessment framework called *RISKMON* that uses machine-learned ranking to assess risks incurred by users' mobile applications. RISKMON combines users' expectations and runtime behaviors to generate the risks assessment.

There have been numerous researches to bring users to pay more attention to the Android permissions upon installing the apps. For example, the work of [33] developed an Android application, *PermissionWatcher*, which provides users with awareness information about other apps based on the permissions they possess. They conducted a user study to evaluate their approach and the results show that users are actually willing to use secure applications if they are given the privacy information in an easy and understandable way. Rosen et. al [27] proposed a new approach to providing users the



(a) Our privacy score.



(b) Taylor privacy score [35].

Fig. 10. Density plots visualising the distribution of apps over the two privacy scores.

knowledge needed to make informed decisions about the applications they install. Their knowledge base is composed of mappings between API calls and fine-grained privacy-related behaviors. The knowledge base is then used to produce high-level behavior profiles for application behavior. Hengshu et. al [43] proposed to develop a mobile App recommender system that takes on consideration the app's security risks and popularity and users' security preferences. Their design goal was to equip the recommender system with the ability to evaluate the security risk of mobile Apps based on requested permissions.

## VIII. Conclusion

In this work we have proposed and evaluated a new security-centric ranking algorithm built on top of the Elasticsearch engine. The aim of this algorithm is to force least privilege among developers and help users avoid installing intrusive third-party mobile applications. The algorithm calculates an intrusiveness score for each mobile app based on the permissions it possesses, the system actions it has registered to listen to and user's privacy preferences. For the purpose of collecting user's privacy preferences a new approach was proposed and evaluated through an online user study. The majority of participants indicated that the approach is effective and brief.

In evaluating our search engine, we conducted two studies; a pilot study and a benchmarking study. The benchmarking study was conducted twice to measure the effect of increasing the APK dataset size. The results of these studies show that our approach is efficient, scales well with the number of apps, and returns relevant applications.

The new ranking algorithm is considered a great opportunity for mobile users to shop for mobile apps that protect their privacy. If more and more users utilize it, developers will be forced to adjust their behavior to satisfy privacy demands of the users.

## References

[1] E. Alepis and C. Patsakis. Unravelling security issues of runtime permissions in android. *Journal of Hardware and Systems Security*, Oct 2018.
[2] A. Amamra, C. Talhi, and J.-M. Robert. Smartphone malware detection: From a survey towards taxonomy. In *2012 7th International Conference on Malicious and Unwanted Software (MALWARE)*, MALWARE'12. IEEE, 2012.
[3] Amazon. Amazon mechanical turk, 2017.
[4] S. Barth, M. D. de Jong, M. Junger, P. H. Hartel, and J. C. Roppelt. Putting the privacy paradox to the test: Online privacy and security behaviors among users with technical knowledge, privacy awareness, and financial resources. *Telematics and Informatics*, 41:55–69, 2019.
[5] D. Bogdanas, N. Nelson, and D. Dig. Analysis and transformations in support of android privacy. 2016.
[6] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: Behavior-based malware detection system for android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 15–26, New York, NY, USA, 2011. ACM.
[7] Elastic. Elasticsearch. "https://www.elastic.co/", February 2018.
[8] W. Enck, M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 235–245, New York, NY, USA, 2009. ACM.
[9] H. A. Fadi Mohsen and H. Bisgin. More than a million android apps with two privacy scores, 2020.
[10] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. Technical Report UCB/EECS-2012-26, EECS Department, University of California, Berkeley, Feb 2012.
[11] E. Fragkaki, L. Bauer, L. Jia, and D. Swasey. Modeling and enhancing android's permission system. In S. Foresti, M. Yung, and F. Martinelli, editors, *Computer Security – ESORICS 2012*, pages 1–18, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
[12] M. Frank, B. Dong, A. Porter Felt, and D. Song. Mining permission request patterns from android and facebook applications. In *2012 IEEE 12th International Conference on Data Mining*, pages 870–875, Dec 2012.
[13] Google. Android Permission System. "https://developer.android.com/guide/topics/permissions/index.html", April 2017.
[14] S. K. Hannah Quay-de la Vallee, Paige Selby. On a (per)mission: Building privacy into the app marketplace. In *6th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSP '16. ACM, 2016.
[15] S. K. Hannah Quay-de la Vallee, Paige Selby. A survey on various malware detection techniques on mobile platform. In *International Journal of Computer Applications*, 2016.

[16] Y. Jing, G.-J. Ahn, Z. Zhao, and H. Hu. Riskmon: Continuous and automated risk assessment of mobile applications. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 99–110, New York, NY, USA, 2014. ACM.

[17] X. Li, J. Liu, Y. Huo, R. Zhang, and Y. Yao. An android malware detection method based on androidmanifest file. In *2016 4th International Conference on Cloud Computing and Intelligence Systems*, CCIS '16. IEEE, 2016.

[18] F. Mohsen, H. Abdelhaq, H. Bisgin, A. Jolly, and M. Szczepanski. Countering intrusiveness using new security-centric ranking algorithm built on top of elasticsearch. In *17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications / 12th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2018, New York, NY, USA, August 1-3, 2018*, pages 1048–1057, 2018.

[19] F. Mohsen and M. Shehab. Hardening the oauth-webview implementations in android applications by re-factoring the chromium library. In *2nd IEEE International Conference on Collaboration and Internet Computing, CIC 2016, Pittsburgh, PA, USA, November 1-3, 2016*, pages 196–205, 2016.

[20] F. Mohsen and M. Shehab. The listening patterns to system events by benign and malicious android apps. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 546–553, Nov 2016.

[21] A. Mylonas, M. Theoharidou, and D. Gritzalis. Assessing privacy risks in android: A user-centric approach. In *RISK@ICTSS*, 2013.

[22] M. Nauman, S. Khan, and X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '10, pages 328–332, New York, NY, USA, 2010. ACM.

[23] 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 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 241–252, New York, NY, USA, 2012. ACM.

[24] X. Ping, W. Xiaofeng, N. Wenjia, Z. Tianqing, and L. Gang. Android malware detection with contrasting permission patterns. In *China Cummmunications*. IEEE, 2014.

[25] N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, SSYM'03, Berkeley, CA, USA, 2003. USENIX Association.

[26] U. R, S. F, P. D, , and L. A. In threat evolution in q3 2017, 2017.

[27] S. Rosen, Z. Qian, and Z. M. Mao. Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users. In *Third ACM Conference on Data and Application Security and Privacy, CODASPY'13, San Antonio, TX, USA, February 18-20, 2013*, pages 221–232, 2013.

[28] D. C. Ryo Sato and S. Goto. Detecting android malware by analyzing manifest files. In *Proceedings of the Asia-Pacific Advanced Network 2013*, APAN '13, pages 23–31, 2013.

[29] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli. Madam: Effective and efficient behavior-based android malware detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, PP(99):1–1, 2016.

[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] Statista. Android Permission System. "https://www.statista.com/statistics/266210/", February 2018.

[32] Statista. Number of available applications in the Google Play Store from December 2009 to December 2018. "https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/", January 2019.

[33] E. Struse, J. Seifert, S. Üllenbeck, E. Rukzio, and C. Wolf. *PermissionWatcher: Creating User Awareness of Application Permissions in Mobile Systems*, pages 65–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[34] G. K. Stuart McClure, Joel Scambray. *Hacking Exposed 7: Network Security Secrets and Solutions 7th Edition*. McGraw-Hill Education, 2012.

[35] V. F. Taylor and I. Martinovic. Securank: Starving permission-hungry apps using contextual permission analysis. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '16, pages 43–52, New York, NY, USA, 2016. ACM.

[36] H. Wang, H. Li, L. Li, Y. Guo, and G. Xu. Why are android apps removed from google play?: A large-scale empirical study. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 231–242, New York, NY, USA, 2018. ACM.

[37] Y. Wang, J. Zheng, C. Sun, and S. Mukkamala. Quantitative security risk assessment of android permissions and applications. In *Proceedings of the 27th International Conference on Data and Applications Security and Privacy XXVII*, DBSec'13, pages 226–241, Berlin, Heidelberg, 2013. Springer-Verlag.

[38] Wikipedia. Bonferroni correction, 2020.

[39] Wikipedia. Friedman test, 2020.

[40] Wikipedia. Wilcoxon signed-rank test, 2020.

[41] XIANGYU-JU. Android malware detection through permission and package. In *Proceedings of the 2014 International Conference on Wavelet Analysis and Pattern Recognition*, ICWAPR '14. IEEE, 2014.

[42] L. Xie and J. pierre Seifert. S.: pbmds: a behavior-based malware detection system for cellphone devices. In *In: 3rd ACM Conference on Wireless Network Security (WiSec 2010*, pages 37–48. ACM, 2010.

[43] H. Zhu, H. Xiong, Y. Ge, and E. Chen. Mobile app recommendations with security and privacy awareness. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 951–960, New York, NY, USA, 2014. ACM.

[44] S. Zou, J. Zhang, and X. Lin. An effective behavior-based android malware detection system. *Sec. and Commun. Netw.*, 8(12):2079–2089, Aug. 2015.