Noname manuscript No.
(will be inserted by the editor)

Exposed! A Case Study on the Vulnerability-Proneness of Google Play Apps

Andrea Di Sorbo* and
Sebastiano Panichella

Received: date / Accepted: date

Abstract Mobile applications are used for accomplishing everyday life activities, such as shopping, banking, and social communications. To leverage the features of mobile apps, users often need to share sensitive information. However, recent research demonstrated that most of such apps present critical security and privacy defects. In this context, we define as *vulnerability-proneness* the risk level(s) that users meet in downloading specific apps, to better understand whether (1) users select apps with lower risk levels and if (2) vulnerability-proneness of an app might affect its success. We use as proxy to measure such risk level the *"number of different types of potential security issues exhibited by the app"*. We conjecture that the vulnerability-proneness levels may vary based on (i) the types of data handled by the app, and (ii) the operations for which the app is supposed to be used. Hence, we investigate how the vulnerability-proneness of apps varies when observing (i) different app categories, and (ii) apps with different success levels. Finally, to increase the awareness of both users and developers on the vulnerability-proneness of apps, we evaluate the extent to which contextual information provided by the app market can be exploited to estimate the vulnerability-proneness levels of mobile apps. Results of our study show that apps in the Medical category exhibit the lowest levels of vulnerability-proneness. Besides, while no strong relations between vulnerability-proneness and average rating are observed, apps with a higher number of downloads tend to have higher vulnerability-proneness levels, but lower vulnerability-proneness density. Fi-

* Corresponding author.

Andrea Di Sorbo
Department of Engineering, University of Sannio, Benevento, Italy.
E-mail: disorbo@unisannio.it

Sebastiano Panichella
Zurich University of Applied Science, Switzerland
E-mail: panc@zhaw.ch

nally, we found that apps' contextual information can be used to predict, in the early stages, the vulnerability-proneness levels of mobile apps.

Keywords Mobile Applications, Vulnerability-proneness, Empirical Study

1 Introduction

Nowadays, mobile applications are widely adopted in most of our everyday activities. About 2.8 and 2.2 million apps are currently available for download on Google Play Store and Apple App Store, respectively[1], with worldwide users relying on mobile applications to deal with a lot of professional and personal tasks, such as shopping, banking, social communication, and events organization [85]. However, as reported by Gartner, a well-known market research organization, several popular apps typically process a lot of sensitive data (*e.g.,*, user's location, lists of contacts, personal photos, etc.), providing often *a little or no security assurances* [33] due to either inappropriate implementations or poor design choices [28].

Previous studies demonstrated that the users' perception, satisfaction, and judgments of mobile apps is influenced by many factors related to app development, such as (i) the change- and fault-proneness of adopted APIs [10], (ii) the complexity of users interfaces [78], (iii) the app size [84], and (iv) the amount of promotional images on the web-store pages (that could represent a proxy for estimating the amount of different functionalities provided by the app) [84]. Complementary to what reported also by Gartner, recent research [5,80], conducted on popular apps handling sensitive user data (*i.e.,* Financial and Health apps), demonstrated that the vast majority of mobile apps tend to suffer from security and/or privacy issues.

In general, consumers are less likely to rely on products experiencing privacy flaws [2]. However, from a practical point of view it is not clear the extent to which (i) the users are aware of the risks they tackle while using insecure mobile applications [61] and (ii) the extent to which *vulnerability risk level(s)* of a mobile app can hinder its success. We argue that users tend to be unaware of the risks concerning downloading and using apps of different categories, and this assumption is also based on the fact that mobile apps generally receive few user reviews related to security and privacy concerns [36, 61]. Moreover, app details shared in app stores rarely provide (or do not provide) information about the *"security risks associated to the usage of certain apps"* [61]. In this context, we define as *vulnerability-proneness* the risk level(s) that users meet in downloading specific apps. Thus, we consider, in this paper, as proxy to measure the `vulnerability-proneness` of an app, ***"the set of the different types of potential security defects exhibited by the app"***. We argue that higher *vulnerability-proneness* levels may result in a higher probability (or potential) of an app of being attacked, as a larger attack surface (*i.e.,* the set of different ways in which an adversary might cause damage [56]) is

[1] `https://buildfire.com/app-statistics`

exposed. This means that, vulnerability-proneness measures the range of different opportunities malicious users can attempt to exploit for triggering an attack. More specifically, rather than summarizing the proportion of flawed code units, vulnerability-proneness is more oriented at modeling the attack surface potential hackers have, by only quantifying vulnerabilities of different natures that could be likely exploited. For instance, given a specific vulnerability type, V_t, vulnerability-proneness does not report how many times vulnerabilities of the V_t type were found in the app's code, but it only accounts that the app is vulnerable to the specific kind of weakness V_t. In this paper, we propose to answer the following and rising questions concerning the vulnerability-proneness of app that, to the best of our knowledge, no prior work investigated:

Do users actually select apps exhibiting low security risks? What are the existing (if any) relationships among (in)security issues and app success (i.e., rating and downloads)?

Mobile app vulnerabilities have been explored from various research perspectives. For example, research literature proposed (i) comprehensive analyses of specific vulnerability types [18,20,99], (ii) approaches and tools for automatically detecting security flaws [52, 67], (iii) enhancements to current security models [46,94], and (vi) investigations on app vulnerabilities evolution [32,81]. Nevertheless, little attention has been paid to better understand how and to what extent security risks can affect the success of mobile apps. To fill this gap, this paper answers the aforementioned general questions, by proposing the following contributions:

- we first study (i) the most widespread vulnerability warnings exhibited by about 1,000 apps mined from the official Google app market, and (ii) the diffusion of specific vulnerability warning types over different app categories;
- we investigate the extent to which the vulnerability-proneness influences app success. Previous work [84] demonstrated that successful apps have more dependence on libraries. As third party code may represent the main carrier of app vulnerabilities [32], we aim at understanding whether apps with higher success are more or less prone to security flaws;
- we evaluate the extent to which an app's contextual information (*e.g.,* app metadata, such as app category, permissions required, etc.) can be used for automatically identifying, in early stages, its vulnerability-proneness, with the benefit of estimating the security and privacy concern levels, without the need of downloading or inspecting the actual app executables or sources.

Our investigation is conducted according to the overarching hypothesis that user perception of app (in)security levels may vary based on (i) the types of data handled by the app, as well as (ii) the app usage scenario. For instance, users may be more concerned to use an insecure app to access their bank account rather than sharing their position when using a flawed navigator app. We argue that these factors strictly depend on the actual app category, as apps

belonging to different categories deal with different users' data and present different user diffusion/popularity levels.

Paper structure. The paper proceeds as follows: Section 2 analyzes the related literature, while Section 3 discusses the empirical study design. Section 4 presents the main findings achieved, while the threats that could affect our results are discussed in Section 5. Finally, Section 6 draws the conclusions and outlines future research directions.

2 Related Work

In this section, we focus the discussion on two main research fields from the literature: (i) studies exploring vulnerabilities in mobile apps, and (ii) studies investigating the factors that could influence the app success.

2.1 Vulnerabilities in mobile apps

In the mobile domain, software vulnerabilities have been extensively studied in prior research. Most of these studies were aimed at (i) identifying new types of flaws that could expose mobile users to security risks, and (ii) proposing approaches or tools for detecting such kinds of flaws. As result, a non-exhaustive list of some of these issues targeted (i) inter-application communication vulnerabilities [18,30,40], (ii) misuses of SSL/TLS protocols [28,77], (iii) component hijacking vulnerabilities [53], (iv) security risks related to file browsing [93], (v) WebView vulnerabilities [19], (v) security flaws related to input validation [16], and the major types of privacy policy violations [75].

Besides, several empirical investigations concerning different aspects related to the vulnerabilities occurring in mobile apps and systems have been carried out. In this context, Jimenez *et al.* [43] studied Android vulnerabilities reported in the National Vulnerability Database (NVD) between 2008 and 2014 and characterized the corresponding fixes. Linares Vásquez *et al.* [87] conducted a large-scale empirical study on Android OS-related vulnerabilities aimed at (i) characterizing the types of Android-related vulnerabilities, (ii) exploring the layers and subsystems from the Android OS affected by vulnerabilities, and (iii) investigating the survivability of vulnerabilities, as the number of days between the vulnerability introduction and its fixing. Similarly, Thomas *et al.* [82] explored API vulnerabilities in Android and quantified the rate at which these flaws are fixed on real devices. In a subsequent work, Thomas *et al.* [83] studied the different players in the Android ecosystem who must collaborate to provide updates aimed at fixing security flaws and showed that the bottleneck is represented by the manufacturers, who often fail to provide updates to fix critical vulnerabilities. More recently, Watanabe *et al.* [92] used automated vulnerability scanners to conduct a large-scale study aimed at understanding whether app vulnerabilities depend on software libraries and revealed that approximately 70% of vulnerabilities of free apps and 50% of

vulnerabilities of paid apps stem from software libraries, particularly from third-party libraries. Also Gao *et al.* [32] relied on vulnerability-finding tools and observed for the first time certain security flaws (*e.g.,* vulnerability reintroduction) that are strictly related to the app's evolutionary perspective.

Similarly to prior research we use a state-of-the-art tool for detecting potential vulnerabilities in Android applications concerning, among others, (i) SSL-related issues, (ii) sandbox mechanism, and (iii) code injection.

2.2 Factors affecting app success

Several studies have investigated the relationships between the rating of an app and different app characteristics [10, 17, 22, 38, 39, 48, 69, 78, 84, 86]. For example, Harman *et al.* [39] examined the existing relations between description, download count, and average user rating of BlackBerry apps, while Tian *et al.* [84] have demonstrated that marketing effort, app size and the target SDK version of Android apps are the most relevant factors impacting their success. Complementary to these work, recent studies investigated the reasons behind the removal of Android apps from Google Play [91], and demonstrated that several security and privacy approaches do not take advantage of the whole range of features of *an appified ecosystem*, which could be exploited to design mechanisms less disruptively [1], as the one proposed in our paper.

Some studies explored the types of software libraries used in mobile apps that can impact user satisfaction. In this context, Linares Vásquez *et al.* [86] and Bavota *et al.* [10] demonstrated that change- and fault-proneness of APIs used in apps could represent a threat to mobile app success. Other researchers [69] have analyzed the impact of ad libraries on app ratings. On the one hand, they found no evidence that the number of ad libraries in apps is related to their rating. On the other hand, they demonstrated that the integration of *certain* ad libraries can negatively affect app rating.

Further research efforts focused on analyzing features specifically related to app development and source code quality. Corral and Fonza [22] showed that source code metrics only marginally relate to market success. Taba *et al.* [78] explored the relationships between the complexity of user interfaces used in Android apps and the user-perceived quality, measured by the number of downloads and app rating. They found that simpler user interfaces may result in a better perceived quality by users.

A close work to ours is the one by Chia *et al.* [17], who demonstrated that community ratings used in app markets are not reliable indicators of privacy risks of an app. In particular, they quantified privacy risks in terms of permissions requested by an app and found that popular Android applications typically tend to request more permissions than new apps, while no negative relations have been found between the rating of an app and the number of permissions requested.

Consistently to these previous studies, we estimate the *app success* in terms of app average rating and the number of app downloads. In addition, as done in

previous work [17], we also focus on investigating whether average app rating is reflected in the level of security and privacy risks of apps. Differently from prior work, we (i) define the *vulnerability-proneness* of an app as the number of different types of potential security flaws (and, thus, that attackers could exploit) exposed by the app, (ii) study the different *vulnerability-proneness* levels exposed by apps belonging to different categories, (iii) explore the eventual relationships that could exist between the *vulnerability-proneness* of an app and its success, and, finally, (iv) investigate the extent to which an app's contextual information could be used to predict its *vulnerability-proneness* level. Concerning the last point, Tao *et al.* [79] recently proposed to identify security issues for mobile applications based on user review summarization. Differently from this work, we use more structured app store metadata to achieve our goal. App store metadata is information the users typically consult before downloading and installing an app, providing information about app functionalities or permissions that the app requires. We believe that this information could be useful for estimating security risks related to the download and usage of an app. However, for future work, we envision the usage of both sources of information to improve the accuracy of our approach.

3 Study Design

The *goal* of this study is two-fold: (i) to assess the vulnerability-proneness of mobile apps belonging to different categories from the official Google Play app store, as well as (ii) to estimate the extent to which vulnerability-proneness can affect app success. To pursue this goal, we formulated three research questions:

- ***RQ*$_1$*: Which are the different vulnerabilities exhibited by Google market apps belonging to different app categories?*** The first research question has the purpose of studying (i) whether different categories of apps expose different types of vulnerability warnings, and (ii) if the vulnerability-proneness levels of apps generally vary according to app categories. We conjecture that the types of vulnerabilities exposed by an app can be somehow related to the functionalities the app provides. Indeed, (i) many vulnerabilities stem from software libraries [92], and (ii) developers can use similar APIs for implementing similar app functionalities [88]. Thus we use the standard Google Play app categories to verify our conjecture, as these categories could be indicative of the functionalities provided by a software application [89].
- ***RQ*$_2$*: Does the vulnerability-proneness of Google market apps affect app success?*** This research question aims at investigating if Google Play apps having lower market success (*i.e.,* lower ratings and/or lower number of downloads) are more *vulnerability-prone*. The underlying hypothesis is that a major amount of vulnerabilities may increase the probability of attacks and user experiencing attacks would provide low ratings or negative posts/feedback to discourage other users from installing the app, thus impacting the apps popularity.

Table 1 Number of apps in each Play Store category

Category	# apps	Avg. Rating	Raters
Entertainment	106	2.0 – 4.7	808 – 3,593,139
Communication	67	2.1 – 4.6	1308 – 5,741,755
Social	61	2.1 – 4.8	843 – 14,229,757
Education	58	1.5 – 4.9	618 – 1,367,308
Books & Reference	57	1.9 – 4.8	566 – 1,564,635
Lifestyle	53	1.7 – 4.8	1,351 – 3,662,229
Health & Fitness	52	2.6 – 4.9	735 – 566,949
Music & Audio	49	3.5 – 4.8	5,016 – 2,076,002
Shopping	49	2.4 – 4.8	761 – 6,865,377
Finance	46	2.1 – 4.9	1,143 – 597,024
Travel & Local	46	1.8 – 4.7	799 – 772,401
Weather	45	1.8 – 4.7	816 – 230,389
News & Magazines	45	2.1 – 4.7	874 – 1,317,066
Sports	42	2.2 – 4.8	954 – 147,370
Business	38	2.5 – 4.8	829 – 113,094
Comics	34	1.9 – 4.7	605 – 183,772
Medical	33	2.2 – 4.7	552 – 28,079
Food & Drink	32	1.7 – 4.8	817 – 3,703,496
Maps & Navigation	25	1.3 – 4.8	1,123 – 967,872
Dating	18	2.4 – 4.2	1,363 – 71,727
Video Players & Editors	18	2.9 – 4.7	4,368 – 371,801
Auto & Vehicles	17	1.8 – 4.8	1,713 – 61,181
Parenting	11	1.3 – 4.8	1,320 – 83,881
Overall	**1,002**	**1.3 – 4.9**	**552 – 14,229,757**

– ***RQ$_3$: Is it possible to predict the level of vulnerability-proneness of an app by using the app's contextual information?*** This research question is aimed at understanding whether (or not) the contextual information that users examine before installing an app is useful (or not) for estimating its level of vulnerability-proneness.

In the next paragraphs, we present the data collection, as well as the analyses performed to answer our research questions.

3.1 Context selection and Dataset construction

To answer our research questions, we collected a dataset comprising 1,002 mobile applications downloaded from the Google Play market. We focused our attention on free apps for practical reasons (paid apps would clearly require a fee) and decided to target Google Play as users of devices running Android OS generally choose their apps from this app store [42] that represents the official Android market. It is worth pointing out that in several prior studies [13, 54, 76, 96] investigating Android vulnerabilities and security defects, the number of samples analyzed is similar to ours.

The RICO mobile apps dataset & Sampling strategy. To guarantee a non-biased selection of the data ,we relied on information collected in the RICO mobile apps dataset [24], which contains metadata information of more than 9.7k Android apps spanning 27 different app categories. Although there are other (and larger) datasets of apps (e.g., Androzoo [6]), we decided to consider the RICO dataset for 3 main motivations:

- real crowd-workers successfully interacted with each app contained in this dataset; this reduces the likelihood of considering toy apps [24];
- about 30% of the apps in this dataset have been removed from the Google Play; this allows to reduce the likelihood of considering apps that have been already judged as low-quality, risky, or malicious by the app market itself [91];
- all the apps in this dataset were released in 2017, at the latest. As vulnerabilities may survive a long time after the first release of an app [32], apps with years of development behind them are more likely to implement well-established processes for mitigating security threats.

Since several apps (*i.e.,* about 30%) appearing in the RICO dataset are no longer available for download on the Google Play store, we selected a statistically significant sample from this large dataset, to ensure a reliable and fair representativeness of the data collection (i.e., a confidence level of 95% and a confidence interval less than 3). In particular, we performed a stratified random sampling of apps present in the RICO dataset, by selecting apps belonging to different app categories. The number of apps downloaded for each category, along with the ranges of average star ratings assigned to these apps are shown in Table 1. The categories considered in our dataset are a subset of all Google Play categories.

To have a sample with more balanced data concerning the app ratings, we applied the following steps to select the apps to include in our dataset:

1. We scraped Google Play to collect the updated average rating values associated with the apps in the RICO dataset that are still available on Google Play.
2. For each app category C_i, we computed the median of average ratings values, M_i, assigned to apps of the RICO dataset belonging to C_i.
3. For each app category C_i, a half of the C_i apps included in our sample have an average rating value below or equal to M_i, and the remaining half of the C_i apps in our sample have an average rating value above M_i.

For instance, for the *Medical* category we selected 33 different apps (as reported in Table 1), 16 of such apps have an average rating value below or equal to 4.1, while the remaining apps have an average rating value higher than 4.1. It is worth noticing that in our sample we only considered apps having at least 500 raters, this to reduce the risk that our results may depend on very subjective or small amount of ratings. Consistently to previous work [4], we adopted this threshold (*i.e.,* 500 raters) to target popular apps.

According to recent statistics[2] there are 2.87 million apps available for download on the Google Play Store. Thus, our dataset is also a representative sample (*i.e.,* a confidence level of 99% and a margin of error of 4.075%) of all apps present on the Google app market.

Data Collection. Once selected the apps to download, all APK files were downloaded from Google Play. In particular, we implemented a web crawler

[2] `https://buildfire.com/app-statistics` - accessed on February 2021.

able to automatically scrape Google Play with the aim of gathering updated information about the apps from the store and collecting the APK files of the selected apps.

Along with the APKs we collected all the information provided by the Google Play store related to each app. More specifically, we gathered the following metadata: (i) the app's name, (ii) the app's description, (iii) the app's Play store category, (iv) the app's size, (v) the photos/screenshots appearing in the app's market webpage, (vi) the app's average rating, (vii) the number of users who rated the app, (viii) range of the number of installs, (ix) the list of permissions the app requires, and (x) monetization-related information (*i.e.,* whether the app contains ads and/or offers in-app purchases). All the APK files, and the related metadata were collected during December 2019.

The downloaded apps were scanned through AndroBugs[3], with the aim of detecting potential security defects related to each of the considered apps. Among the several vulnerability scanning tools available, we selected AndroBugs as, differently from other vulnerability scanners, it tries to simulate the operation of an app considering the paths or means through which such weaknesses would be exploited, instead of just scanning the code for weak spots [23]. Besides checking for many known common vulnerabilities in the Android apps, Androbugs also inspects the code against (i) missing best practices, and (ii) dangerous shell commands (e.g., `su`). In particular, AndroBugs is a state-of-the-art well-known vulnerability scanner for mobile applications that was first presented at the BlackHat security conference, and has been used in several previous studies investigating vulnerabilities occurring in mobile apps [32, 92]. This static detection tool was also successfully used to find vulnerabilities and other critical security issues in Android apps developed by several big players, such as: Facebook, eBay, Twitter, etc. [32].

Listing 1 Excerpt from an AndroBugs report

```
[Critical] <Command> Runtime Command Checking:
           This app is using critical function 'Runtime.getRuntime().exec
               ("...")'.
           Please confirm these following code secions are not harmful:
               => Lcom/milkmangames/extensions/android/coremobile/c;->d()Z
                   (0x10) --->
                   Ljava/lang/Runtime;->exec(Ljava/lang/String;)Ljava/lang
                       /Process;
```

All the reports generated by AndroBugs have been parsed to enumerate, for each app, the detected vulnerabilities. Listing 1 illustrates an excerpt from an AndroBugs report. To each detected vulnerability a severity score (*e.g.,* `Critical` in Listing 1) and a type (*e.g., Runtime Command Checking* in Listing 1) is assigned. It is worth noticing that we only collected vulnerabilities which are marked as `Critical`, `Warning`, or `Notice` by AndroBugs, as vulnerabilities with a `Info` categorization indicate that the specific issue was not found on the specific apk [59]. Some of the potential security weaknesses that the AndroBugs static analyzer is able to detect are related to: (i) SSL connections,

[3] `https://github.com/AndroBugs/AndroBugs_Framework`

implementation and certificate validation, (ii) WebView- and Fragment-related vulnerabilities, (iii) implicit intents, (iv) data storage, (v) KeyStore protection, (vi) Android Manifest settings, and (vii) entry points for command injection. The full list of vulnerabilities checked is reported in our online appendix[4].

Replication package. We make available in our replication package[5] all the raw data used to answer our research research questions.

3.2 Research method

To answer our $\mathbf{RQ}_1$ we compared the vulnerability warnings detected in the apps belonging to the different Google Play store's categories. In particular, to characterize the vulnerability-proneness level of an app category, we group all the apps in our sampled dataset (see Section 3.1) that belong to that specific category and compute the vulnerability-proneness levels related of these apps. Then, to establish whether there are categories exhibiting higher/lower vulnerability-proneness levels, we compared the vulnerability proneness levels associated with each category. Besides, to check whether the observed differences are statistically significant, we make use of the Kruskal-Wallis test [50] and subsequent Mann-Whitney pairwise comparison [21] (with the Holm's p-value correction procedure [41] and $\alpha = 0.05$), for identifying the specific pairs whose differences have statistical evidence. The Kruskal-Wallis test is a rank-based nonparametric test that is used to determine if statistically significant differences between more than two groups of an independent variable can be observed. However, since this test does not allow identifying the specific groups for which the differences have statistical evidence, we perform a posthoc analysis (using Mann-Whitney pairwise comparison) to determine which levels of the independent variable differ from each other level. To counteract the problem of multiple comparisons and reduce the probability of obtaining Type 1 errors (false positives), we use the Holm's correction procedure, which allows adjusting the rejection criteria for each of the individual hypotheses tested.

In particular, we tested the following null hypothesis:

H_{01}: There are no significant differences between the vulnerability-proneness levels of apps belonging to different Play store's categories.

We also estimated the magnitude of the differences with statistical significance, through the Cliff's delta (or d) effect-size measure [37]. Following the guidelines in [37], we interpret the effect-size as: *small* for $|d| < 0.33$, *medium* for $0.33 \leq |d| < 0.474$, and *large* for $|d| \geq 0.474$. For graphically visualizing the distributions, we make use of boxplots. Moreover, to complement this quantitative analysis, we qualitatively investigated the different types of vulnerability warnings encountered in apps belonging to categories exhibiting higher/lower vulnerability-proneness levels.

[4] `https://github.com/adisorbo/vulnerability_proneness/wiki/Vulnerability-Types`

[5] `https://github.com/adisorbo/vulnerability_proneness`

To answer **RQ**$_2$, we compared the vulnerability-proneness levels of apps having different *app success*, using a methodology similar to the one adopted by Linares Vásquez *et al.* [86]. The *dependent variable* for this research question is represented by the success of the considered apps. For estimating app success, as in previous studies, we make use of two proxy values—*i.e.,* the average rating and the number of downloads—as they could be easily gathered from the Google Play store webpage. The *independent variable* is the number of potential security defects exhibited by the different apps.

Concerning the rating, we clustered the apps in four different groups based on their average rating values. To achieve this goal and assign each app to one of the groups,we computed the quartiles of the distribution of average rating values assigned to the apps. In particular, given r_a the average user rating, the four sets are: (i) group1 = [apps having $r_a \leq$ *1st Quart.*]; (ii) group2 = [apps having *1st Quart.* $< r_a \leq$ *Median*]; (iii) group3 = [apps having *Median* $< r_a \leq$ *3rd Quart.*]; and (iv) group4 = [apps having $r_a >$ *3rd Quart.*]. Similarly, we clustered the apps in three different groups based on their install class, as Google Play does not provide the exact number of downloads for apps. In particular, the three sets are: (i) set1 = [apps having up to $1,000,000$ downloads] (*i.e., up to 1M*); (ii) set2 = [apps having from $1,000,001$ to $50,000,000$ downloads] (*i.e., up to 50M*); and (iii) set3 = [apps having more than $50,000,000$ downloads] (*i.e., more than 50M*). We tested the following null hypothesis:

H_{02}: There are no significant differences between the vulnerability-proneness levels of apps belonging to different success groups.

More specifically, we compared the distributions of the vulnerability-proneness levels associated with the different rating groups, and the vulnerability-proneness levels associated with the different download groups, through the Kruskal-Wallis test and subsequent Mann-Whitney pairwise comparison (with the Holm's p-value correction procedure and $\alpha = 0.05$). In addition, we estimated the magnitude of the statistically significant differences, through the Cliff's delta. To investigate whether specific results could be observed among the different app categories, we repeated the analysis for each of the app categories in our dataset.

We observed that the apps in our dataset strongly vary in size (*i.e.,* from apps having less than 10 classes to apps having tens of thousands of classes). To account for any potential bias that the difference of app sizes may introduce, beyond evaluating the number of cumulative warnings obtained for each app, for both RQ$_1$ and RQ$_2$ we also proceeded to a normalized analysis. Specifically, for each app, we divided the number of vulnerability warnings signaled by AndroBugs by the number of classes. This allows us to estimate the *vulnerability-proneness density* of each app.

To answer **RQ**$_3$ we used app-related information collected from the Google Play store to (i) train a set of machine learning (ML) classifiers and (ii) evaluate the extent to which they were able to identify the vulnerability proneness levels of apps. In particular, to have a balanced dataset, after computing the

Table 2 Factors potentially affecting the vulnerability-proneness of an app

Aspect	Feature	Description	Rationale
Behavior	Name	App Name	App name and its natural language description can be used as proxy for the advertised behavior of an app [35]. Similarly, app categories can be used to predict the common features found within software of a particular category [89]. Similar features may lead to similar vulnerabilities.
	Description	App Description	
	Play Store Category	Category of the app.	
Richness of functionalities	Description Length	Number of words appearing in the description of the app on its Play store page.	The number of words in the description and the number of photos appearing in the app's webpage can be used as proxy for estimating the number of functionalities the app provides [84]. Similarly, larger apk size might indicate more features or more complete functionalities [84] and more functionalities increase the probability of vulnerabilities [92].
	Photos	Number of images shown on the app store webpage	
	APK Size	Binary size of the APK file (measured in MB)	
Success	Average Rating	Average user rating as reported in the Google Play store	Popular apps may exhibit more vulnerabilities [92]. Similarly, high-rated apps may present higher levels of security risks than low-rated apps [51].
	Number of Raters	Number of users who have rated the app	
	Number of Installs	Amount of times the app has been installed	
Permissions	Number of Permissions	Number of permissions required by the application as declared in the app's Play Store webpage	Each app must declare upfront what permissions it requires. Overprivileged applications increase the impact of vulnerabilities [29]. Similarly apps can use dangerous permissions combinations [97].
	Permissions List	List of permissions required by the application as declared in the app's Play Sore webpage.	
Monetization	Contains Ads	Boolean value to indicate if the app contains advertisement.	Ad libraries [92] as well as In-App payment services [95] may represent sources of vulnerabilties.
	In App Purchase	Boolean value to indicate if the app offers in-app purchasing.	

median value, M_{vulns}, of the distribution of the vulnerability-proneness values associated with the apps in our dataset, we assigned to each app the label (i) *low*, if its vulnerability-proneness level was lower (or equal) than M_{vulns}, or (ii) *high*, if its vulnerability-proneness level was higher than M_{vulns}. These labels represented our ground truth for computing the prediction performance. By relying on the Weka tool[6], three different ML algorithms—namely the Naive Bayes, J48, and Random Forest— were trained to predict if the vulnerability-proneness level of an app should be marked as either *low* or *high*. The selection of these specific algorithms is not random, as they have been successfully adopted in previous work concerning defect prediction tasks [34, 47].

Extraction and pre-processing of ML features. To train such ML algorithms, we considered 13 features provided by the app market concerning five different aspects that might be correlated with the vulnerability-proneness of apps. The meaning and rationale of each selected feature is described in Table 2. It is worth noticing that all the features reported in Table 2 can be gathered by simply scraping Google Play. Most of these features are of (i) numeric (*i.e.,* `Description Length`, `Photos`, `APK Size`, `Average Rating`, `Number of Raters` and `Number of Permissions`), (ii) nominal (*i.e.,* `Play Store Category` and `Number of Installs`), or (iii) boolean (*i.e.,* `Contains Ads`, `In App Purchase`, and each specific `Permission` in the list) types. Note that, while the relationships between the individual features in Table 2 and security risks have been extensively studied in prior research [29, 51, 92, 95, 97], we leverage findings from these previous studies for selecting significant features to train machine learning algorithms.

[6] `https://www.cs.waikato.ac.nz/ml/weka/`

Textual features. However, `Name` and `Description` features involve text contents that must be properly preprocessed to be used as features to train the ML models. Previous work demonstrated that natural language texts could be profitably treated (through well-known text analysis techniques aimed at extracting relevant textual features from them) and used for issue/vulnerability classification (or prediction) tasks [8, 27, 45, 71]. In our context, for each considered app, the `Name` and the `Description` features have been concatenated. The resulting strings have been then used as an information base to build a textual corpus (i.e., bag of words composing the concatenated text elements), that was pre-processed by applying stop-word removal (using the English Standard Stop-word list) and stemming (*i.e.,* English Snowball Stemmer) [9]. The usage of a stemming approach is not random, since it was successfully leveraged to reduce the number of text features for the ML problems in the mobile context [64]. In future work we plan to investigate the effect of using lemmatization approaches in our results. As result of this process, a set of textual features (*i.e.,* words) with relevance values (i.e., the frequency of the words in the app textual corpus) is associated to each app. In addition to these text features, we also computed, for each app, the *n-grams*— with $[n \in [2-4]]$— appearing in the aforementioned textual corpus to preserve word locality information. The rational of using *n-grams* is due to the fact that app functionalities are usually described by groups of 2,3 or 4 words rather than single words [44]. Thus, each resulting text feature (*i.e.,* word or n-gram) is weighted using the *tf-idf* weighting score [9].

Static analysis features. Previous work [72] demonstrated that code metrics can be used to predict Android vulnerabilities. Thus, we also used widely-known static analysis tools to extract a set of code-related metrics, for two main reasons:

- to compare our results with a baseline approach and verify whether machine learning models trained with information extracted from the app store could achieve similar results to the ones obtained by measuring code-related features. If similar results would be achieved with the two approaches, using app store metrics as a proxy for estimating the vulnerability-proneness level of an app would be a clear advantage since there would be no need to download and inspect the app's code;
- to better understand whether app-related details extracted from the Google store could provide complementary information to the one provided by code-related metrics. In particular, we aim at verifying if the details provided by the app store could be leveraged for improving the results achieved by approaches exclusively based on code-related information.

In particular, as third-party libraries often represent carriers for app vulnerabilities [32], we used `LibRadar` [55] to extract the number of third-party libraries used by each application in our dataset. In addition, we used the `apk-parser` tool[7] for extracting the following features from each `apk` files:

[7] `https://github.com/hsiafan/apk-parser`

(i) the minimum API Level required for the application to run (`min_sdk`), (ii) the specific API Level that the application targets (`target_sdk`), (iii) the number of classes (`classes`), (iv) the number of packages (`packages`), (v) the number of interfaces (`interfaces`), (vi) the number of *annotated* classes (`annotations`), and (vii) the number of *public* classes (`public_classes`). While the numbers of classes of different types determine the size of an app, which plays an important role in determining the security of the app (*i.e.,* more classes mean a larger attack surface [3]), the API Levels indicators are strictly connected with the app security [60].

Grouping of features. For convenience, the different extracted and pre-processed features have been grouped as follows:

- *Market metrics:* comprising app market features without considering the app `Description`, the app `Name` and the resulting textual features.
- *Textual features:* comprising all the words and n-grams derived from the text processing steps described above.
- *Static analysis metrics:* comprising all the code related metrics (*i.e.,* `libraries, min_sdk, target_sdk, classes, interfaces, annotations, public_classes`)

Training of the ML models. We trained the ML algorithms using different combinations of these groups of features, namely (i) *Market metrics + Textual features*, (ii) *Market metrics*, (iii) *Static analysis metrics*, (iv) *Market metrics + Textual features + Static analysis metrics*, and (v) *Market metrics + Static analysis metrics*. As the goal of $\mathbf{RQ}_3$ is to understand the extent to which contextual information provided in the app store is useful (or not) for estimating app vulnerability-proneness, note that we only considered the combinations of features involving the *Market metrics* and a baseline (i.e., *Static analysis metrics*) for comparison purposes.

To alleviate concerns related to overfitting and selection bias, all the ML experiments were carried-out by using the 10-fold cross-validation strategy, while the classification/prediction performance achieved by the different ML models were evaluated through widely-known metrics in the information retrieval field: *precision, recall and F-measure* [9].

4 Results

This section discusses the results of RQ1-3 (see Section 3).

4.1 RQ_1: Which are the different vulnerabilities exhibited by Google market apps belonging to different app categories?

All the considered apps exhibit at least one vulnerability warning. The high numbers of potential vulnerabilities identified were quite expected, as vulnerability warnings considered in our study are of different severity levels,

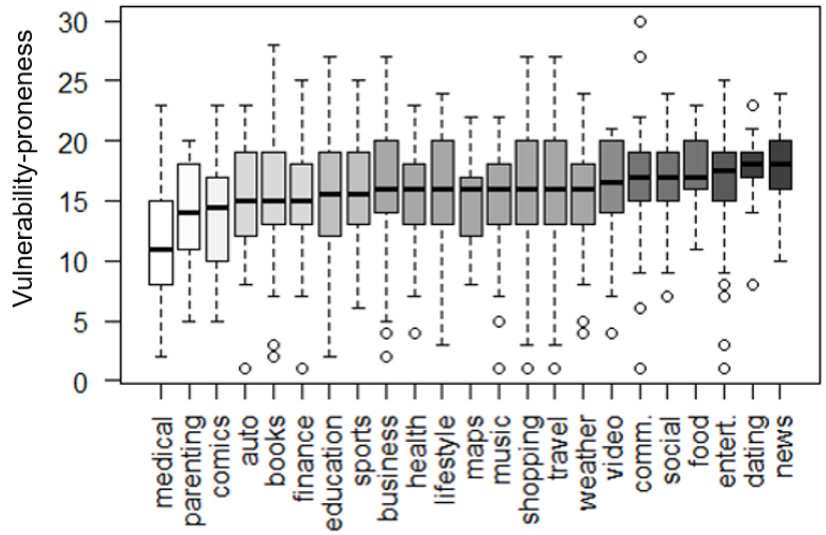

Fig. 1 Distributions of detected vulnerabilities for the apps in the different Play Store's categories.

and some of them (the ones with lower severity levels) could be deliberately neglected by developers [71]. Mobile apps significantly differ from more traditional software systems [58], as Android apps are poorly tested and also test cases have low code coverage [49, 66]. This could depend on the fact that the majority of app developers do not use test automation frameworks [49] and practices [98], while security testing is often neglected [7]. Besides, app developers pose more attention to functional bugs and security bugs get fixed slower [11], also because fixing vulnerabilities requires specialized knowledge that is not widespread among developers [14].

As illustrated in Figure 1, apps in the *Medical* category are the ones exhibiting the lowest level of vulnerability-proneness. Since attackers often make use of automated tools to identify likely exploitable attack vectors [90], we can observe that *Medical* apps are less risky than other types of apps, thanks to a less wide variety of warnings to known vulnerabilities. More specifically, the differences in terms of vulnerability-proneness levels between the different app categories are statistically significant (the Kruskal-Wallis test returned $p = 8.068 * 10^{-5}$). The subsequent Mann-Whitney pairwise comparisons highlighted that the vulnerability-proneness of apps in the *Medical* category significantly differs from apps in the *Communication, Entertainment, Food & Drink, News & Magazines,* and *Social* categories with *large* effect-size. From these first results, we could observe that, on a positive side, there seems to be special attention towards security and privacy concerns in the development of apps in the *Medical* category, that often deal with very sensitive information (*e.g.,* disease status, medications, etc.). However, from the security and pri-

Table 3 Critical vulnerabilities found in apps belonging to different categories

Vulnerability	Overall	Communication	Entertainment	Food & Drink	Medical	News & Magazines	Social
SSL Connection Checking	93.61%	100.00%	97.17%	100.00%	84.85%	100.00%	96.72%
WebView RCE Vulnerability Checking	69.06%	73.13%	86.79%	81.25%	36.36%	86.67%	73.77%
Implicit Service Checking	41.92%	52.24%	49.06%	46.88%	15.15%	35.56%	54.10%
App Sandbox Permission Checking	27.84%	17.91%	27.36%	37.50%	24.24%	31.11%	27.87%
SSL Certificate Verification Checking	20.06%	16.42%	18.87%	21.88%	6.06%	40.00%	14.75%
KeyStore Protection Checking	19.96%	13.43%	9.43%	31.25%	9.09%	35.56%	19.67%
Runtime Command Checking	19.06%	34.33%	13.21%	21.88%	12.12%	31.11%	9.84%
Fragment Vulnerability Checking	15.77%	22.39%	11.32%	12.50%	3.03%	13.33%	8.20%
AndroidManifest ContentProvider Exported Checking	15.57%	19.40%	16.98%	18.75%	9.09%	24.44%	14.75%
SSL Implementation Checking (Verifying Host Name in Custom Classes)	14.77%	11.94%	17.92%	9.38%	12.12%	28.89%	21.31%
SSL Implementation Checking (Verifying Host Name in Fields)	11.38%	5.97%	8.49%	12.50%	9.09%	26.67%	11.48%

vacy perspective, the same attention can not be noticed in the development of apps of the *Finance* or *Shopping* categories, which can also handle very sensitive information like bank accounts (*e.g.,* the *Finance* category comprises several mobile banking apps).

To have a clearer picture of the potential security defects occurring in the investigated app categories, for brevity's sake, in the following, we focus our analysis on the categories whose vulnerability-proneness levels exhibited statistically significant differences with large effect size. In particular, in Table 3, for the most recurrent vulnerability warnings marked as *Critical* (on the rows) and the app categories exhibiting statistically significant differences in the vulnerability-proneness levels (on the columns), the percentages of likely vulnerable apps are reported. Comparing such percentages, we found that:

- 938 out of 1002 (*i.e.,* 93.61%) total apps in our dataset connect to URLs that are not under SSL, thus such communications are insecure by construction. In particular, 100% of considered apps belonging to the *Communication, Food & Drink,* and *News & Magazines* categories suffer from this vulnerability warning that is quite common also in apps handling very sensitive data, as those belonging to the *Medical* category (*i.e.,* 84.85% of apps in this category exhibit this vulnerability).
- 692 (*i.e.,* 69.06%) apps in our dataset make use of the `addJavascriptInterface` method, that can be used to allow JavaScript to control the application, in devices running older Android versions. More specifically, more than 80% of apps belonging to the *Entertainment, Food & Drink,* and *News & Magazines* categories suffer from this vulnerability, while only 36.36% of apps in the *Medical* category exhibited this security flaw. We conjecture that such diversified results are strongly connected with the actual testing practices that mobile developers perform on the considered apps. Indeed, as reported in previous work [57], mobile apps tend to have less (and in some cases no) test cases compared to other types of applications, despite their potential higher level of popularity among

users. It is worth noticing that, as AndroBugs detects the `minSdk` (*i.e.,* the minimum API Level on which the application is able to run), it only reports this vulnerability as critical when the `addJavascriptInterface` method can likely execute.
- 420 (*i.e.,* 41.92%) of considered apps use implicit intents to start services, which is very risky because the responding service can not be identified. While this vulnerability is rare in apps of the *Medical* category (*i.e.,* 15.15% of apps in this category are vulnerable), it has been found to be quite frequent in apps falling in the *Social* category (*i.e.,* 54.10% of *Social* apps in our dataset suffer from this vulnerability).
- 279 (*i.e.,* 27.84%) of considered apps perform insecure data storage by creating world-readable or world-writeable files. More than 30% of apps belonging to *Food & Drink* and *News & Magazines* categories exhibit this vulnerability.
- 201 (*i.e.,* 20.06%) of considered apps do not check the validity of SSL Certificate, allowing self-signed, expired or mismatch CN certificates for SSL connection. 40% of *News & Magazines* apps considered in our dataset result vulnerable to this flaw, while in only 6.06% of *Medical* apps we encountered this security defect.
- 200 (*i.e.,* 19.96%) of considered apps do not protect KeyStore properly as they seem to use byte array or hard-coded certificate info to do SSL pinning. 35.56% of *News & Magazines* apps encompassed in our dataset exhibit this security hole, while only about 9% of *Medical* apps result vulnerable to this flaw.
- 191 (*i.e.,* 19.06%) of considered apps use the critical function `Runtime.getRuntime().exec(...)`, allowing attackers to inject arbitrary system commands. 34.33% of *Communication* apps and 31.11% of *News & Magazines* apps in our dataset exhibit this vulnerability, while it has been found in less than 13% of apps belonging to the *Social* and *Medical* categories.
- 158 (*i.e.,* 15.77%) of considered apps have been found to be vulnerable to fragment injection, making it possible to access sensitive information that should not be accessible by the application itself on devices running older versions of Android. This vulnerability has been discovered in 22.39% of apps belonging to the *Communication* category while only about 3% of *Medical* apps present this security flaw.
- 156 (*i.e.,* 15.57%) of considered apps use `exported` Content providers, making it possible to any other app on the device to access it. More than 24% of apps belonging to the *News & Magazines* category suffer from this type of vulnerability, while less than 10% of apps belonging to the *Medical* category exhibited this security flaw.
- 148 (*i.e.,* 14.77%) of considered apps allow Self-defined `HOSTNAME VERIFIER` to accept all Common Names(CN), making it possible that malicious users perform Man-In-The-Middle (MITM) attacks. In particular, about 30% of apps belonging to the *News & Magazines* category exhibit this vulnerability.

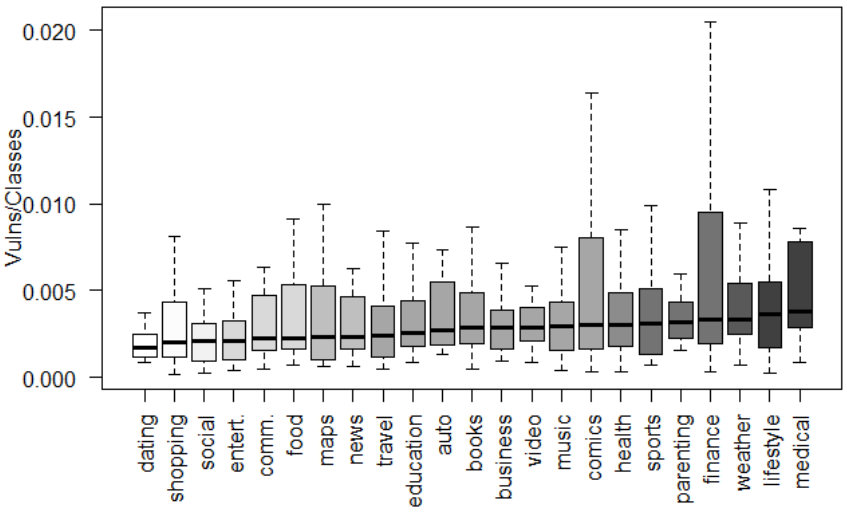

Fig. 2 Distributions of vulnerability-proneness density for the apps in the different Play Store's categories.

– 114 (*i.e.,* 11.38%) of considered apps do not check the validation of the CN(Common Name) of the SSL certificate and this could allow attackers to perform MITM attacks. More than 26% of the apps belonging to the *News & Magazines* category suffer from this vulnerability.

In summary, apps in the *News & Magazines* category tend to be more exposed than the other apps to vulnerabilities that could be likely exploited for performing Man-In-the-Middle attacks (*i.e., SSL Connection Checking, SSL Certificate Verification Checking,* and *SSL Implementation Checking*). In addition, *News & Magazines* apps tend to be quite prone also to vulnerabilities (i) that may cause injection attacks (*i.e.,* vulnerabilities related to `WebView` and `Runtime command`), and (ii) that may allow unauthorized access to sensitive data (*i.e.,* `KeyStore` protection vulnerability). It is worth noticing that also apps in the *Communication* category are (i) more prone than others to vulnerability warnings related to injection attacks (*i.e.,* `Runtime command`, and `Fragment` vulnerabilities), and (ii) make more frequent use of implicit `Intents`, as apps in the *Social* category. Finally, the most recurrent vulnerability warnings found in the applications of the *Medical* category are related to the connection to URLs that are not under SSL, and the usage of vulnerable `WebView`, while any other critical warning has been found in less than 16% of apps belonging to this category.

Surprisingly, looking at Figure 2 we can easily observe that, while *Medical* apps exhibit the lowest levels of vulnerability-proneness (probably due to

the reduced sizes of this kind of apps that decrease the probability of releasing vulnerable code) apps in this category are also the ones exhibiting the highest vulnerability-proneness density. Indeed, the differences in terms of vulnerability-proneness density between the different app categories are statistically significant (the Kruskal-Wallis test returned $p = 1.748 * 10^{-6}$). Specifically, the *Medical* apps significantly differ from apps in the *Entertainment, Social,* and *Dating* categories with *large* effect size, while *Weather* apps significantly differ from apps in the *Social* and *Entertainment* category with *medium* effect size. We can conclude that although *Medical* apps exhibit the highest vulnerability-proneness density levels, they tend to be less prone to potentially critical vulnerabilities than other kinds of apps, as evidenced in our qualitative analysis.

RQ_1 summary: *Almost all apps likely connect to URLs that are not under SSL. Although Medical apps exhibit the highest levels of vulnerability-proneness density, they tend to be less prone to potentially critical vulnerabilities. Among all apps, applications in the News & Magazines category tend to be the most exposed to vulnerabilities that could be exploited for performing Man-In-the-Middle attacks.*

4.2 RQ_2: Does the vulnerability-proneness of Google market apps affect app success?

As illustrated in Figure 3, in terms of vulnerability-proneness levels, no significant differences could be observed between apps having different rating scores. Indeed, the Kruskal-Wallis test returned $p = 0.3704$.

To investigate more in-depth whether users of apps in specific categories could be more concerned about security and privacy aspects, as stated in Section 3.2, we repeated the analysis for each specific app category. The results of such an analysis partially confirm our conjecture that the vulnerability-proneness of an app does not generally affect user ratings. Indeed, for none of the considered categories, we observed statistically significant differences between the vulnerability-proneness levels of apps with different ratings. This last result was quite unexpected, as some of the studied categories encompass apps handling very sensitive data (*e.g., Medical, Finance, Shopping, etc.*).

As done in RQ_1, we also conducted a normalized analysis to better understand if any relations can be observed between vulnerability-proneness density and the app rating. As shown in Figure 4, apps having a lower average rating tend to have a higher vulnerability-proneness density. More specifically, the Kruskal-Wallis test highlighted that the the differences in terms of vulnerability-proneness density among apps having different rating are statistically significant (*i.e.,* the Kruskal-Wallis test returned $p = 1.175 * 10^{-5}$), while the post-hoc Mann-Whitney pairwise comparisons revealed that the differences are statistically significant for the following pairs: (i) apps with $r < 3.6$ and apps with $r > 4.4$, (ii) apps with $r < 3.6$ and apps with $4.1 < r \leq 4.4$,

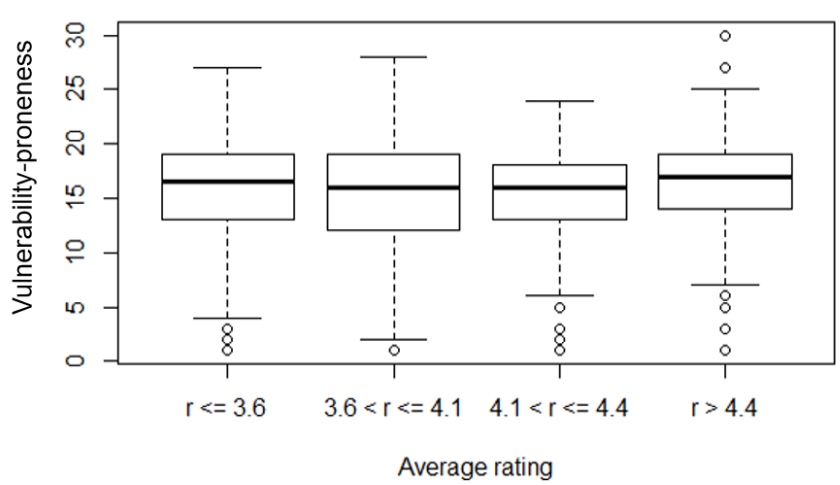

Fig. 3 Distributions of detected vulnerabilities in apps belonging to different rating groups.

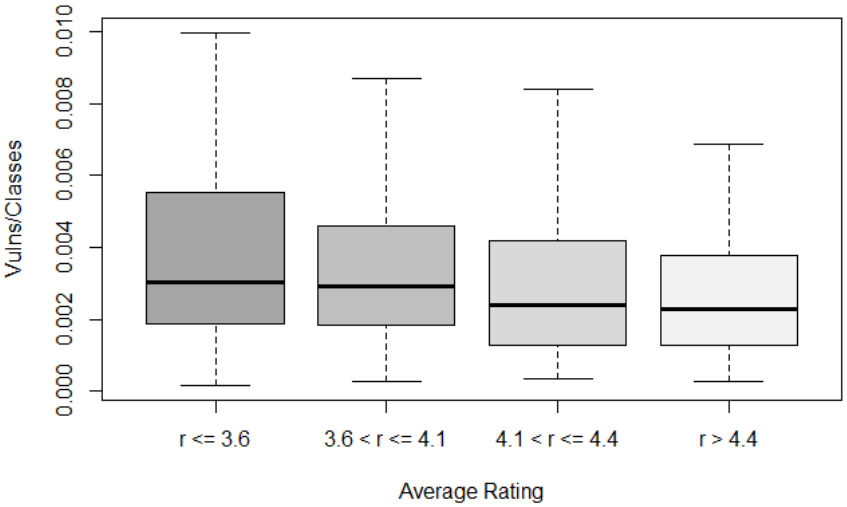

Fig. 4 Distributions of vulnerability-proneness density in apps belonging to different rating groups.

(ii) apps with $3.6 < r \leq 4.1$ and apps with $r > 4.4$, and (iv) apps with $3.6 < r \leq 4.1$ and apps with $4.1 < r \leq 4.4$. However, such differences exhibit *negligible* to *small* effect sizes (*i.e.,* Cliff's d ranges from 0.13 to 0.23).

Though no significant relationships could be found between the vulnerability-proneness levels of apps and app ratings, differences with *negligible* or *small* effect sizes can be observed between the vulnerability-proneness density of apps with different ratings. Thus, we can argue that a higher vulnerability-proneness density may lead users to give lower ratings.

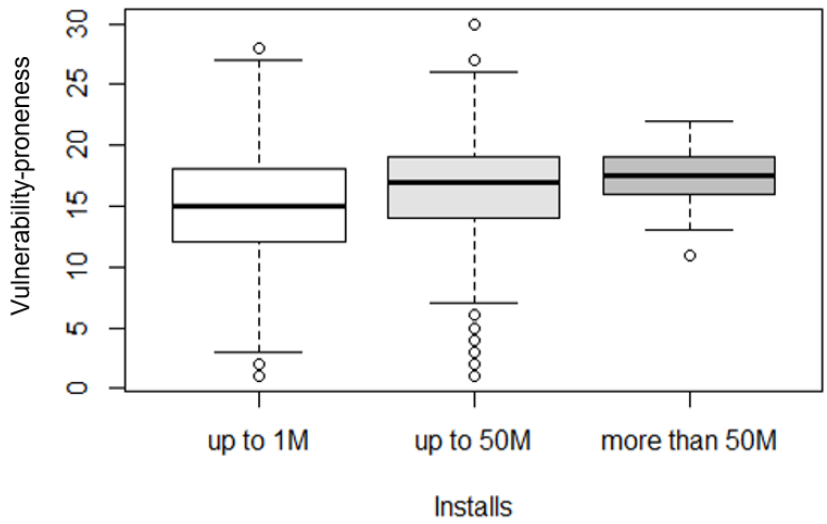

Fig. 5 Distributions of detected vulnerabilities in apps belonging to different install groups.

For better understanding if security and privacy concerns are taken into consideration by users when selecting apps to install, as reported in Section 3.2, we also measured the vulnerability-proneness levels of apps having different number of installs. As shown in Figure 5, more *popular* apps tend to exhibit higher levels of vulnerability-proneness. More specifically, the Kruskal-Wallis test returned $p = 1.966 * 10^{-8}$ and the posthoc Mann-Whitney pairwise comparisons revealed that the vulnerability-proneness levels of apps belonging to the *up to 1M* group are different with statistical evidence from those of apps belonging to both *up to 50M* and *more than 50M* groups with *small* and *medium* effect-sizes, respectively.

However, this result could be due to the fact that more popular apps likely have larger sizes [22]. Indeed, more meaningful differences are observed when looking at the vulnerability-proneness density of the apps with different levels of installs, as shown in Figure 6. In this case, the Kruskal-Wallis test returned $p < 2.2 * 10^{-16}$, while the post-hoc Mann-Whitney pairwise comparisons revealed that the differences are statistically significant for all the pairs with *medium* to *large* effect sizes (*i.e.,* Cliff's d varies from 0.39 to 0.89).

The conjunction of these results leads us to conclude that, while the vulnerability-proneness of apps does not seem to significantly impact their success, more popular apps tend to exhibit a lower vulnerability-proneness density. Despite app markets do not provide enough information to help users carefully selecting higher quality apps [15, 25], Google Play Protect[8] can lead users more towards installing apps with lower vulnerability-proneness density. Besides, the lower vulnerability-proneness density of apps with higher ratings and installs could also depend on the fact that this kind of apps is usually

[8] `https://www.android.com/safety/`

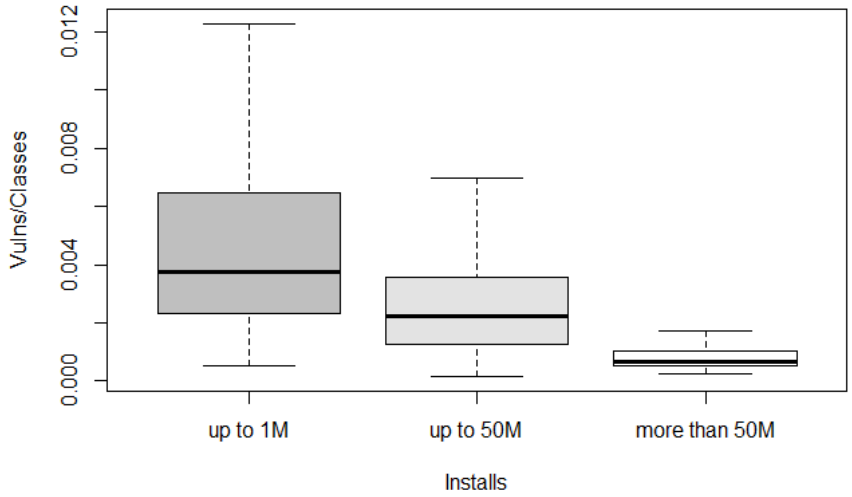

Fig. 6 Distributions of vulnerability-proneness density in apps belonging to different install groups.

developed (and maintained) by more experienced developers [12], also considering that larger apps tend to be tested more [74].

We argue that, for achieving the adoption of higher security standards in app development, the app markets could also provide details about the security risks to which users would be exposed, along with the information about an app's functionalities. In this context, to promote their apps, developers would be forced to avoid known vulnerabilities and common security flaws, as user choices could also depend on the security aspects.

RQ_2 summary: *The vulnerability-proneness of an app can only marginally affect its success. No strong relations are found between the vulnerability-proneness levels of apps and ratings. Apps with higher numbers of downloads tend to exhibit higher vulnerability-proneness levels, but a lower vulnerability-proneness density.*

4.3 RQ_3: Is it possible to predict the level of vulnerability-proneness of an app by using the app's contextual information?

Table 4 reports the classification performance achieved by the considered ML algorithms (on the columns) trained with the different combinations of features (on the rows) and the ten-fold cross-validation strategy (see Section 3.2). Looking at Table 4, the first result that we can observe is that Random Forest is the best performing algorithm. Indeed, in each experiment, it achieved the best F-measure results. While the Naive Bayes algorithm obtained the less

Table 4 Precision (P), Recall (R) and F-measure (F1) obtained by the ML algorithms trained with the different combinations of features.

Features	J48			Random Forest			Naive Bayes		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Market metrics + Textual features	0.619	0.620	0.619	0.660	0.660	0.658	0.581	0.577	0.576
Market metrics	0.671	0.667	0.666	0.730	0.728	0.728	0.647	0.647	0.645
Static analysis metrics	**0.726**	**0.712**	**0.709**	0.716	0.714	0.714	0.660	0.652	0.643
Market metrics + Textual features + Static analysis metrics	0.664	0.665	0.664	0.723	0.720	0.719	0.592	0.590	0.590
Market metrics + Static analysis metrics	0.691	0.687	0.686	**0.760**	**0.751**	**0.751**	**0.660**	**0.657**	**0.652**

accurate classification results in all experiments, in only one case (*i.e.,* using static analysis features) the J48 ML technique produced similar outcomes to the ones achieved by the Random Forest technique. For brevity's sake, in the following, we focus on discussing the results achieved by the best performing model, *i.e.,* Random Forest, in the various experiments.

On the one hand, by only using app market information (*i.e., Market metrics* in Table 4), the Random Forest algorithm achieved precision, recall and F-measure values of about 73%. This means that in about 3 out of 4 cases the classification algorithm based on these features can properly identify apps with either *low* or *high* vulnerability-proneness levels. This is a promising result if we consider that when the same algorithm is trained by using code-related features (*i.e., Static analysis metrics* in Table 4) slightly lower performance is achieved. On the other hand, the information provided by Google Play is not sufficient to make very accurate predictions. For this reason, the app markets should provide additional information useful to warn users of possible security and privacy risks, with the explicit aim of improving their awareness about security concerns.

Interestingly, when both app market information and code-related features are jointly used (*i.e., Market metrics + Static analysis metrics* in Table 4), higher precision, recall, and F-measure values are obtained. This means that app market metrics provide complementary information to the one related to code. Therefore, they could be adopted for improving strategies exclusively based on code-related characteristics. Surprisingly, textual features seem to introduce noise that degrades the classification results. Indeed, in both experiments in which textual contents have been employed (*i.e., Market metrics + Textual features* and *Market metrics + Textual features + Static analysis metrics* in Table 4) lower performance is achieved. We argue that this result is mainly due to the fact that topics discussed in app name and description is not strictly related to the vulnerability proneness of an app.

To more-in-depth understand the specific app market information that could drive the classification, we computed the information gain [68] for each of the features in the *Market metrics* group and ranked these features based on their scores. In Table 5, the top 15 ranked features, along with the related information gain scores are reported. Looking at Table 5, we can observe that the app size is the feature with the highest score. Besides, permissions-

Table 5 App market metrics with the 15 highest information gain scores

Score	Feature
0.1308	Size
0.0923	Permissions
0.0830	Other_receive data from Internet
0.0709	Other_prevent device from sleeping
0.0413	Raters
0.0408	Other_control vibration
0.0394	Photos/Media/Files_modify or delete the contents of your USB storage
0.0388	Storage_modify or delete the contents of your USB storage
0.0384	Category
0.0378	Photos/Media/Files_read the contents of your USB storage
0.0378	Other_run at startup
0.0371	Storage_read the contents of your USB storage
0.0322	Installs
0.0300	Other_view network connections
0.0299	Wi-Fi connection information_view Wi-Fi connections

related aspects such as (i) the number of permission, and (ii) the presence of specific permissions (*e.g.,* mostly related to storage and networking operations) required by the app represent relevant information for deciding the vulnerability-proneness level of an app. Finally, as further confirmation of what we found in RQ_1 (Section 4.1) and RQ_2 (Section 4.2), both the *Category* of an app and its popularity indicators (*i.e., Raters* and *Installs*) provide valuable information for identifying the risk levels. Since the `Size` feature represents the best predictor, the resulting model might be biased towards this metric (*i.e.,* the predictions of vulnerability-proneness levels could be mainly based on this feature). Thus, to estimate the extent to which the performance of the ML models is driven by the `Size` metric, we repeated the ML experiments (in which ML algorithms are fed with *Market metrics*) by not considering this feature and observed degradations in precision, recall, and F-measure values lower than 5%. This allows us to conclude that the eventual bias introduced by the `Size` metric is only marginal.

RQ_3 summary: *In about 3 out of 4 cases, an approach based on the app's contextual information and Random Forest algorithm can properly identify apps with either low or high vulnerability-proneness levels. App market metrics provide complementary information to the one related to code, while text-related features introduce noise that degrades the classification results.*

5 Threats to Validity

Threats to construct validity concern the relationship between theory and observation. The most important threat that could affect the results of our study is related to possible imprecision/incompleteness in identifying the vulnerabilities and, thus, the vulnerability-proneness levels of apps. It is worth noticing

that we rely on a state-of-the-art tool (*i.e.,* AndroBugs) for identifying well-known security flaws. As this is a static analysis tool, it may yield false positives. In addition, in our RQ_2 we used the average rating as an indicator of the success of an app. However, user ratings can be subjective and imprecise. To alleviate such an issue, we (i) only considered apps rated by a reliable number of users (*i.e., Raters* > 500), and (ii) complemented the analysis by also considering the number of installs as an alternative way to measure app success. We believe that, with a smaller number of ratings, there is a higher risk that our results may depend on the subjectiveness of the ratings. Indeed, considering only a few tens of ratings, extremely positive or negative ratings given by certain groups of users may have too much impact on the average score [10]. For this reason, the average rating value of an app that is rated by at least 500 users is more likely to take into account the opinions of more heterogeneous users and, thus, better reflects the actual rating. Our study does not consider the temporal dimension associated with the user rating and the number of installs, and this could represent a threat that can affect the validity of our findings. Unfortunately, Google Play does not allow extracting the user ratings and number of installs referring to a specific app version, and only average or cumulative counts are provided. However, prior research [70] demonstrated that average user rating is quite resilient to version-rating changes, especially for apps having higher numbers of raters. Finally, our measurements on vulnerability proneness are based on the assumption that users need to be aware of the security level of all apps. However, the usage of specific vulnerability-prone apps could be unavoidable for users when real alternatives are not available to replace popular apps (e.g., Facebook, Whatsapp, etc.). This means that it is unclear weather users having more information about the level of vulnerabilities of popular apps can push them to not install or use them. This is something we plan to investigate for future work.

Threats to conclusion validity concern the relationship between treatment and outcome. Appropriate, non-parametric statistical procedures have been adopted to draw our conclusions, since the variables of interest were not well-modeled by normal distributions (as verified through the Shapiro-Wilk normality test [73]). More specifically, we used the Kruskal-Wallis test and post hoc Mann-Whitney pairwise comparisons for investigating the statistically significant differences. Moreover, the magnitude of the observed differences is quantified by using Cliff's delta effect size measure. For coping with multiple comparisons, the Holm's correction procedure has been adopted to adjust p-values.

Threats to internal validity concern factors that can affect our results. A possible source of bias might be related to the thresholds we used when analyzing the data and presenting our results. In particular, to answer our RQ_2, we clustered the apps into four levels of rating and three levels of installs. Similarly, to answer our RQ_3, we grouped the apps into two vulnerability-proneness level groups. The thresholds to define the average rating categories and the vulnerability-proneness categories were based on the descriptive statistics indicators (*i.e.,* quartiles) of the related distributions for the 1,002 considered

apps, while the apps have been assigned to the different levels of installs based on the downloads category assigned by the Google Play store (see Section 3.2). Different choices might lead to different results. Nevertheless, we obtained similar results when considering different (i) rating —*e.g., low*, encompassing the apps with an average rating lower than the first quartile, *medium* comprising the apps with an average rating value between the first and the third quartile, and *high*, consisting of the apps with an average rating higher than the third quartile– and (ii) downloads groups. To verify weather the adopted independent features (or variables) could correlate among themselves we computed the Kendall correlation between all the pairs of (numeric) features considered in our study. As result, no correlations with either large or medium effect size are observed ($\tau_B \leq 0.27$ for all the pairs of features).

Threats to external validity concern the generalization of the findings. Our analysis involves about 1,000 Android apps sampled from a dataset not specifically designed for security purposes. Thus, it is unclear if our results may generalize to further apps or apps developed for other mobile platforms. However, as discussed in Section 3.1, our data collection is a statistically significant sample of a dataset consisting of about 10,000 apps. While our results are in line with previous investigations demonstrating that also very popular apps suffer from insecure communication vulnerabilities [31,65,81], it is worth pointing out that some of these vulnerabilities could be non-exploitable (*i.e.,* appearing in dead or legacy code), or present in third-party libraries [92]. To estimate the trustworthiness of our results, as well as the actual exploitability of the signaled vulnerabilities, we randomly selected 20 apps in our dataset. We reverse-engineered (by using `dex2jar`[9]) such apps and inspected their source code to understand whether the vulnerable code portions reported by AndroBugs and marked as *Critical* were actually reachable. As a result of this analysis, we observed that less than 5% of the signaled critical vulnerabilities appeared in (statically detectable) unreachable code. Besides, we performed a dynamic analysis on the same 20 selected apps, following the approach used in previous work [23]. Indeed, to conduct such an analysis, we used the Charles Proxy[10] and tried to perform man-in-the-middle (MITM) attacks, by testing the SSL certificate validation, SSL certificate checking, and that all important URLs are SSL protected. Specifically, we opened each target application and tried different functionalities of the application while intercepting the packets on the Charles proxy. The attack succeeded if we were able to read (or decode) the intercepted packets. Despite for all the 20 analyzed apps AndroBugs reported issues with SSL certificates, for four of them (i.e., 20%), we were unable to trigger the attack. Furthermore, we only considered free apps and it is unclear if our conclusions are also valid for paid apps, as more rigorous processes for avoiding security flaws could be adopted when developing paid apps. To further increase the generalizability and reliability of our results, in the future, we plan to replicate our study at a larger scale considering apps

[9] `https://github.com/pxb1988/dex2jar`
[10] `https://www.charlesproxy.com/`

from other datasets [62] and investigating if our findings are still valid when considering paid apps.

6 Conclusions and Future Work

Users typically share sensitive information to use mobile apps for accomplishing a lot of everyday life activities. However, recent research demonstrated that the majority of these apps suffer from critical security defects.

In this paper, we defined the concept of vulnerability-proneness of an app and investigated if different vulnerability warnings and vulnerability-proneness levels are observed in apps belonging to different Google Play's categories (RQ_1). Moreover, we also explored the extent to which vulnerability-proneness of mobile apps can affect the overall app success, measured in terms of average app rating and number of app downloads (RQ_2). Finally, we proposed to use app contextual information to predict the vulnerability-proneness level of an app (RQ_3).

The results of our study demonstrated that most of the considered apps exhibit at least one critical vulnerability warning. On a positive side, (i) apps in the *Medical* category, that usually handle very sensitive information, tend to be less prone to potentially critical vulnerabilities, while (ii) there is not the same attention on security warnings in apps belonging to other categories that also deal with sensitive data (*e.g.,* Finance, Shopping, etc.). Our work also proved that, while no strong relations could be observed between the vulnerability-proneness of an app and its average rating, apps with a higher number of downloads tend to have higher vulnerability-proneness levels, but a lower vulnerability-proneness density. Finally, we also showed how an app's contextual information can be used to predict, in the early stages, its vulnerability-proneness level.

Our work can have important implications for *app markets, developers* and *users*. Indeed, on the one hand, the proposed classification could help *users* selecting apps that exhibit lower risk levels. On the other hand, the *app markets* could integrate approaches similar to ours, to provide timely and relevant information about security and privacy risks to users *before* installing an app. We believe that the usage of such mechanisms can stimulate *developers* (interested in promoting their apps) to apply the best security practices and carefully avoid known security flaws.

As future work, we plan to study the vulnerability-proneness of apps developed for different platforms (*e.g.,* iOS). In this context, it could be interesting to investigate if the same apps exhibit different vulnerability-proneness levels when considering different versions (*e.g.,* Android vs. iOS). Moreover, further information from the app store (*e.g.,* interactive elements, developer-related information, user review comments, etc.) [64] could be collected to improve the classification results. Very important, as future work, is also to empirically investigate the extent to which users are unaware of the level of vulnerability-proneness of apps as well as the extent to which our approach can actually help

them understanding more about the potential risks of downloading or using an app. In this context, we plan to (i) leverage summarization [26, 63] techniques to generate reports on the vulnerability-proneness levels of mobile apps, and (ii) use such reports to support users in making decisions on downloading (or using) specific apps. Moreover, we plan to explore the extent to which an app's contextual information is useful to predict the presence or absence of specific security flaws.

Acknowledgment

We gratefully thank Prof. Dr. Harald Gall, Dean of the Faculty of Business, Economics, and Informatics of the University of Zurich and director of the Software Evolution and Architecture Lab, for supporting this research, making the lab facilities available to the development of this research project. We also thank Prof. Gall for the qualitative feedback on the direction of this work and the ongoing collaboration in close-related research projects. Finally, we thank the anonymous reviewers and the editors for the constructive and relevant feedback on our study. Their openness to dialogue has been fundamental to improve the manuscript. Sebastiano Panichella gratefully acknowledges the Horizon 2020 (EU Commission) support for the project *COSMOS* (DevOps for Complex Cyber-physical Systems), Project No. 957254-COSMOS).

References

1. Acar, Y., Backes, M., Bugiel, S., Fahl, S., McDaniel, P.D., Smith, M.: Sok: Lessons learned from android security research for appified software platforms. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016, pp. 433–451. IEEE Computer Society (2016). DOI 10.1109/SP.2016.33
2. Afroz, S., Islam, A.C., Santell, J., Chapin, A., Greenstadt, R.: How privacy flaws affect consumer perception. In: Workshop on Socio-Technical Aspects in Security and Trust, pp. 10–17 (2013). DOI 10.1109/STAST.2013.13
3. Alenezi, M., Almomani, I.: Empirical analysis of static code metrics for predicting risk scores in android applications. In: 5th International Symposium on Data Mining Applications, pp. 84–94. Springer (2018)
4. Ali, M., Joorabchi, M.E., Mesbah, A.: Same app, different app stores: A comparative study. In: 4th IEEE/ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft@ICSE 2017, Buenos Aires, Argentina, May 22-23, 2017, pp. 79–90 (2017). DOI 10.1109/MOBILESoft.2017.3
5. Aliasgari, M., Black, M., Yadav, N.: Security vulnerabilities in mobile health applications. In: Conference on Application, Information and Network Security, pp. 21–26 (2018). DOI 10.1109/AINS.2018.8631464
6. Allix, K., Bissyandé, T.F., Klein, J., Traon, Y.L.: Androzoo: collecting millions of android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016, pp. 468–471 (2016). DOI 10.1145/2901739.2903508
7. Amin, A., Eldessouki, A., Magdy, M.T., Abdeen, N., Hindy, H., Hegazy, I.: Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. Inf. **10**(10), 326 (2019). DOI 10.3390/info10100326

8. Antoniol, G., Ayari, K., Penta, M.D., Khomh, F., Guéhéneuc, Y.: Is it a bug or an enhancement?: a text-based approach to classify change requests. In: Proceedings of Centre for Advanced Studies on Collaborative Research, p. 23 (2008). DOI 10.1145/1463788.1463819
9. Baeza-Yates, R., Ribeiro-Neto, B., et al.: Modern information retrieval, vol. 463. ACM press New York (1999)
10. Bavota, G., Vásquez, M.L., Bernal-Cárdenas, C.E., Penta, M.D., Oliveto, R., Poshyvanyk, D.: The impact of API change- and fault-proneness on the user ratings of android apps. IEEE Trans. Software Eng. **41**(4), 384–407 (2015). DOI 10.1109/TSE.2014.2367027
11. Bhattacharya, P., Ulanova, L., Neamtiu, I., Koduru, S.C.: An empirical analysis of bug reports and bug fixing in open source android apps. In: 17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013, pp. 133–143 (2013). DOI 10.1109/CSMR.2013.23
12. Businge, J., Openja, M., Kavaler, D., Bainomugisha, E., Khomh, F., Filkov, V.: Studying android app popularity by cross-linking github and google play store. In: 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019, pp. 287–297 (2019). DOI 10.1109/SANER.2019.8667998
13. Cai, Y., Tang, Y., Li, H., Yu, L., Zhou, H., Luo, X., He, L., Su, P.: Resource race attacks on android. In: 27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020, pp. 47–58 (2020). DOI 10.1109/SANER48275.2020.9054863
14. Canfora, G., Di Sorbo, A., Forootani, S., Pirozzi, A., Visaggio, C.A.: Investigating the vulnerability fixing process in oss projects: Peculiarities and challenges. Computers & Security **99**, 102067 (2020)
15. Canfora, G., Di Sorbo, A., Mercaldo, F., Visaggio, C.A.: Exploring mobile user experience through code quality metrics. In: Product-Focused Software Process Improvement - 17th International Conference, Proceedings, pp. 705–712 (2016). DOI 10.1007/978-3-319-49094-6_59
16. Cao, C., Gao, N., Liu, P., Xiang, J.: Towards analyzing the input validation vulnerabilities associated with android system services. In: Annual Computer Security Applications Conference, pp. 361–370 (2015). DOI 10.1145/2818000.2818033
17. Chia, P.H., Yamamoto, Y., Asokan, N.: Is this app safe?: a large scale study on application permissions and risk signals. In: Proceedings of the World Wide Web Conference, pp. 311–320 (2012). DOI 10.1145/2187836.2187879
18. Chin, E., Felt, A.P., Greenwood, K., Wagner, D.A.: Analyzing inter-application communication in android. In: International Conference on Mobile Systems, pp. 239–252 (2011). DOI 10.1145/1999995.2000018
19. Chin, E., Wagner, D.A.: Bifocals: Analyzing webview vulnerabilities in android applications. In: Information Security Applications - International Workshop, WISA, pp. 138–159 (2013). DOI 10.1007/978-3-319-05149-9_9
20. Clark, J., van Oorschot, P.C.: Sok: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In: Symposium on Security and Privacy, pp. 511–525 (2013). DOI 10.1109/SP.2013.41
21. Conover, W.: Practical nonparametric statistics. Wiley series in probability and statistics: Applied probability and statistics. Wiley (1998)
22. Corral, L., Fronza, I.: Better code for better apps: A study on source code quality and market success of android applications. In: International Conference on Mobile Software Engineering and Systems, MOBILESoft, pp. 22–32 (2015). DOI 10.1109/MobileSoft.2015.10
23. Darvish, H., Husain, M.I.: Security analysis of mobile money applications on android. In: IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018, pp. 3072–3078 (2018). DOI 10.1109/BigData.2018.8622115
24. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: Annual ACM Symposium on User Interface Software and Technology, pp. 845–854 (2017). DOI 10.1145/3126594.3126651

25. Di Sorbo, A., Grano, G., Visaggio, C.A., Panichella, S.: Investigating the criticality of user-reported issues through their relations with app rating. Journal of Software: Evolution and Process **33**(3), e2316 (2021). DOI 10.1002/smr.2316
26. Di Sorbo, A., Panichella, S., Alexandru, C.V., Shimagaki, J., Visaggio, C.A., Canfora, G., Gall, H.C.: What would users change in my app? summarizing app reviews for recommending software changes. In: T. Zimmermann, J. Cleland-Huang, Z. Su (eds.) Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016, pp. 499–510. ACM (2016). DOI 10.1145/2950290.2950299
27. Di Sorbo, A., Panichella, S., Visaggio, C.A., Di Penta, M., Canfora, G., Gall, H.C.: Exploiting natural language structures in software informal documentation. IEEE Transactions on Software Engineering pp. 1–1 (2019). DOI 10.1109/TSE.2019.2930519
28. Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., Freisleben, B.: Why eve and mallory love android: an analysis of android SSL (in)security. In: Conference on Computer and Communications Security, pp. 50–61 (2012). DOI 10.1145/2382196.2382205
29. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.A.: Android permissions demystified. In: ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011, pp. 627–638 (2011). DOI 10.1145/2046707.2046779
30. Felt, A.P., Wang, H.J., Moshchuk, A., Hanna, S., Chin, E.: Permission re-delegation: Attacks and defenses. In: USENIX Security Symposium (2011)
31. Gajrani, J., Tripathi, M., Laxmi, V., Somani, G., Zemmari, A., Gaur, M.S.: Vulvet: Vetting of vulnerabilities in android apps to thwart exploitation. Digital Threats: Research and Practice **1**(2), 1–25 (2020)
32. Gao, J., Li, L., Kong, P., Bissyandé, T.F., Klein, J.: Understanding the evolution of android app vulnerabilities. IEEE Transactions on Reliability pp. 1–19 (2019). DOI 10.1109/TR.2019.2956690
33. Gartner: Gartner Says More than 75 Percent of Mobile Applications will Fail Basic Security Tests Through 2015. `https://tinyurl.com/uavh5nq` (2015). Online; accessed 20 January 2020
34. Giger, E., D'Ambros, M., Pinzger, M., Gall, H.C.: Method-level bug prediction. In: International Symposium on Empirical Software Engineering and Measurement, pp. 171–180 (2012). DOI 10.1145/2372251.2372285
35. Gorla, A., Tavecchia, I., Gross, F., Zeller, A.: Checking app behavior against app descriptions. In: International Conference on Software Engineering, pp. 1025–1035 (2014). DOI 10.1145/2568225.2568276
36. Grano, G., Di Sorbo, A., Mercaldo, F., Visaggio, C.A., Canfora, G., Panichella, S.: Android apps and user feedback: a dataset for software evolution and quality improvement. In: Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, WAMA@ESEC/SIGSOFT FSE 2017, Paderborn, Germany, September 5, 2017, pp. 8–11 (2017). DOI 10.1145/3121264.3121266
37. Grissom, R.J., Kim, J.J.: Effect sizes for research: A broad practical approach, 2nd edition edn. Lawrence Earlbaum Associates (2005)
38. Guerrouj, L., Azad, S., Rigby, P.C.: The influence of app churn on app success and stackoverflow discussions. In: International Conference on Software Analysis, Evolution, and Reengineering, pp. 321–330 (2015). DOI 10.1109/SANER.2015.7081842
39. Harman, M., Jia, Y., Zhang, Y.: App store mining and analysis: MSR for app stores. In: Working Conference of Mining Software Repositories, pp. 108–111 (2012). DOI 10.1109/MSR.2012.6224306
40. Hay, R., Tripp, O., Pistoia, M.: Dynamic detection of inter-application communication vulnerabilities in android. In: International Symposium on Software Testing and Analysis, pp. 118–128 (2015). DOI 10.1145/2771783.2771800
41. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics **6**(2), 65–70 (1979)
42. Islam, M.R.: Numeric rating of apps on google play store by sentiment analysis on user reviews. In: International Conference on Electrical Engineering and Information & Communication Technology, pp. 1–4. IEEE (2014)

43. Jimenez, M., Papadakis, M., Bissyandé, T.F., Klein, J.: Profiling android vulnerabilities. In: International Conference on Software Quality, Reliability and Security, pp. 222–229 (2016). DOI 10.1109/QRS.2016.34
44. Johann, T., Stanik, C., B., A.M.A., Maalej, W.: SAFE: A simple approach for feature extraction from app descriptions and app reviews. In: International Requirements Engineering Conference, pp. 21–30 (2017). DOI 10.1109/RE.2017.71
45. Kallis, R., Di Sorbo, A., Canfora, G., Panichella, S.: Ticket tagger: Machine learning driven issue classification. In: 2019 IEEE International Conference on Software Maintenance and Evolution, pp. 406–409 (2019). DOI 10.1109/ICSME.2019.00070
46. Kantola, D., Chin, E., He, W., Wagner, D.A.: Reducing attack surfaces for intra-application communication in android. In: Workshop on Security and Privacy in Smartphones and Mobile Devices, Co-located with CCS 2012, pp. 69–80 (2012). DOI 10.1145/2381934.2381948
47. Kaur, A., Kaur, I.: Empirical evaluation of machine learning algorithms for fault prediction. Lecture Notes on Software Engineering **2**(2), 176 (2014)
48. Khalid, H., Nagappan, M., Hassan, A.E.: Examining the relationship between findbugs warnings and app ratings. IEEE Software **33**(4), 34–39 (2016). DOI 10.1109/MS.2015.29
49. Kochhar, P.S., Thung, F., Nagappan, N., Zimmermann, T., Lo, D.: Understanding the test automation culture of app developers. In: 8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015, pp. 1–10 (2015). DOI 10.1109/ICST.2015.7102609
50. Kruskal, W.H., Wallis, W.A.: Use of ranks in one-criterion variance analysis. Journal of the American statistical Association **47**(260), 583–621 (1952)
51. Krutz, D.E., Munaiah, N., Meneely, A., Malachowsky, S.A.: Examining the relationship between security metrics and user ratings of mobile apps: a case study. In: Proceedings of the International Workshop on App Market Analytics, pp. 8–14 (2016). DOI 10.1145/2993259.2993260
52. Li, L., Bartel, A., Bissyandé, T.F., Klein, J., Le Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P.: Iccta: Detecting inter-component privacy leaks in android apps. In: IEEE International Conference on Software Engineering, vol. 1, pp. 280–291 (2015). DOI 10.1109/ICSE.2015.48
53. Lu, L., Li, Z., Wu, Z., Lee, W., Jiang, G.: CHEX: statically vetting android apps for component hijacking vulnerabilities. In: the ACM Conference on Computer and Communications Security, pp. 229–240 (2012). DOI 10.1145/2382196.2382223
54. Lyu, Y., Gui, J., Wan, M., Halfond, W.G.J.: An empirical study of local database usage in android applications. In: 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017, pp. 444–455 (2017). DOI 10.1109/ICSME.2017.75
55. Ma, Z., Wang, H., Guo, Y., Chen, X.: Libradar: fast and accurate detection of third-party libraries in android apps. In: International Conference on Software Engineering, Companion Volume, pp. 653–656 (2016). DOI 10.1145/2889160.2889178
56. Manadhata, P.K., Wing, J.M.: An attack surface metric. IEEE Trans. Software Eng. **37**(3), 371–386 (2011). DOI 10.1109/TSE.2010.60
57. Minelli, R., Lanza, M.: Software analytics for mobile applications–insights lessons learned. In: 2013 17th European Conference on Software Maintenance and Reengineering, pp. 144–153 (2013). DOI 10.1109/CSMR.2013.24
58. Minelli, R., Lanza, M.: Software analytics for mobile applications-insights & lessons learned. In: 17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013, pp. 144–153 (2013). DOI 10.1109/CSMR.2013.24
59. Montealegre, C., Njuguna, C.R., Malik, M.I., Hannay, P., McAteer, I.N.: Security vulnerabilities in android applications. In: Australian Information Security Management Conference, pp. 14–28. Security Research Institute, Edith Cowan University (2018). DOI 10.25958/5c5274d466691
60. Mutchler, P., Safaei, Y., Doupé, A., Mitchell, J.C.: Target fragmentation in android apps. In: 2016 IEEE Security and Privacy Workshops, SP Workshops 2016, San Jose, CA, USA, May 22-26, 2016, pp. 204–213 (2016). DOI 10.1109/SPW.2016.31

61. Nguyen, D., Derr, E., Backes, M., Bugiel, S.: Short text, large effect: Measuring the impact of user reviews on android app security & privacy. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019, pp. 555–569 (2019). DOI 10.1109/SP.2019.00012
62. Oltrogge, M., Huaman, N., Amft, S., Acar, Y., Backes, M., Fahl, S.: Why eve and mallory still love android: Revisiting tls (in) security in android applications. In: 30th USENIX Security Symposium (USENIX Security 21) (2021)
63. Panichella, S.: Summarization techniques for code, change, testing, and user feedback (invited paper). In: C. Artho, R. Ramler (eds.) 2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, VST@SANER 2018, Campobasso, Italy, March 20, 2018, pp. 1–5. IEEE (2018). DOI 10.1109/VST.2018.8327148
64. Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C.A., Canfora, G., Gall, H.C.: How can i improve my app? classifying user reviews for software maintenance and evolution. In: R. Koschke, J. Krinke, M.P. Robillard (eds.) 2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015, pp. 281–290. IEEE Computer Society (2015). DOI 10.1109/ICSM.2015.7332474
65. Papageorgiou, A., Strigkos, M., Politou, E.A., Alepis, E., Solanas, A., Patsakis, C.: Security and privacy analysis of mobile health applications: The alarming state of practice. IEEE Access **6**, 9390–9403 (2018). DOI 10.1109/ACCESS.2018.2799522
66. Pecorelli, F., Catolino, G., Ferrucci, F., Lucia, A.D., Palomba, F.: Testing of mobile applications in the wild: A large-scale empirical study on android apps. In: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020, pp. 296–307 (2020). DOI 10.1145/3387904.3389256
67. Qian, C., Luo, X., Le, Y., Gu, G.: Vulhunter: Toward discovering vulnerabilities in android applications. IEEE Micro **35**(1), 44–53 (2015). DOI 10.1109/MM.2015.25
68. Quinlan, J.R.: Induction of decision trees. Machine learning **1**(1), 81–106 (1986)
69. Ruiz, I.J.M., Nagappan, M., Adams, B., Berger, T., Dienst, S., Hassan, A.E.: Impact of ad libraries on ratings of android mobile apps. IEEE Software **31**(6), 86–92 (2014). DOI 10.1109/MS.2014.79
70. Ruiz, I.J.M., Nagappan, M., Adams, B., Berger, T., Dienst, S., Hassan, A.E.: Examining the rating system used in mobile-app stores. IEEE Softw. **33**(6), 86–92 (2016). DOI 10.1109/MS.2015.56
71. Russo, E.R., Di Sorbo, A., Visaggio, C.A., Canfora, G.: Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. Journal of Systems and Software **156**, 84–99 (2019). DOI 10.1016/j.jss.2019.06.001
72. Scandariato, R., Walden, J.: Predicting vulnerable classes in an android application. In: International Workshop on Security Measurements and Metrics, MetriSec '12, p. 11–16. Association for Computing Machinery (2012). DOI 10.1145/2372225.2372231
73. Shapiro, S.S., Wilk, M.B.: An analysis of variance test for normality (complete samples). Biometrika **52**(3/4), 591–611 (1965)
74. Silva, D.B., Eler, M.M., Durelli, V.H.S., Endo, A.T.: Characterizing mobile apps from a source and test code viewpoint. Inf. Softw. Technol. **101**, 32–50 (2018). DOI 10.1016/j.infsof.2018.05.006
75. Slavin, R., Wang, X., Hosseini, M.B., Hester, J., Krishnan, R., Bhatia, J., Breaux, T.D., Niu, J.: Toward a framework for detecting privacy policy violations in android application code. In: L.K. Dillon, W. Visser, L. Williams (eds.) International Conference on Software Engineering, pp. 25–36. ACM (2016). DOI 10.1145/2884781.2884855
76. Song, W., Huang, Q., Huang, J.: Understanding javascript vulnerabilities in large real-world android applications. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2018)
77. Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., Khan, L.: Smv-hunter: Large scale, automated detection of SSL/TLS man-in-the-middle vulnerabilities in android apps. In: 21st Annual Network and Distributed System Security Symposium (2014)
78. Taba, S.E.S., Keivanloo, I., Zou, Y., Ng, J.W., Ng, T.: An exploratory study on the relation between user interface complexity and the perceived quality. In: Web Engineering, International Conference, pp. 370–379 (2014). DOI 10.1007/978-3-319-08245-5_22

79. Tao, C., Guo, H., Huang, Z.: Identifying security issues for mobile applications based on user review summarization. Information and Software Technology **122**, 106290 (2020). DOI https://doi.org/10.1016/j.infsof.2020.106290
80. Taylor, V.F., Martinovic, I.: Short paper: A longitudinal study of financial apps in the google play store. In: Financial Cryptography and Data Security - International Conference, pp. 302–309 (2017). DOI 10.1007/978-3-319-70972-7_16
81. Taylor, V.F., Martinovic, I.: To update or not to update: Insights from a two-year study of android app evolution. In: ACM on Asia Conference on Computer and Communications Security, pp. 45–57 (2017). DOI 10.1145/3052973.3052990
82. Thomas, D.R., Beresford, A.R., Coudray, T., Sutcliffe, T., Taylor, A.: The lifetime of android API vulnerabilities: Case study on the javascript-to-java interface. In: Security Protocols XXIII - 23rd International Workshop, pp. 126–138 (2015). DOI 10.1007/978-3-319-26096-9_13
83. Thomas, D.R., Beresford, A.R., Rice, A.C.: Security metrics for the android ecosystem. In: Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 87–98 (2015). DOI 10.1145/2808117.2808118
84. Tian, Y., Nagappan, M., Lo, D., Hassan, A.E.: What are the characteristics of high-rated apps? A case study on free android applications. In: International Conference on Software Maintenance and Evolution, pp. 301–310 (2015). DOI 10.1109/ICSM.2015.7332476
85. Tien, C., Huang, T., Huang, T., Chung, W., Kuo, S.: MAS: mobile-apps assessment and analysis system. In: International Conference on Dependable Systems and Networks Workshops, pp. 145–148 (2017). DOI 10.1109/DSN-W.2017.17
86. Vásquez, M.L., Bavota, G., Bernal-Cárdenas, C., Penta, M.D., Oliveto, R., Poshyvanyk, D.: API change and fault proneness: a threat to the success of android apps. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 477–487 (2013). DOI 10.1145/2491411.2491428
87. Vásquez, M.L., Bavota, G., Escobar-Velasquez, C.: An empirical study on android-related vulnerabilities. In: Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017, pp. 2–13 (2017). DOI 10.1109/MSR.2017.60
88. Vásquez, M.L., Holtzhauer, A., Poshyvanyk, D.: On automatically detecting similar android apps. In: 24th IEEE International Conference on Program Comprehension, ICPC 2016, Austin, TX, USA, May 16-17, 2016, pp. 1–10 (2016). DOI 10.1109/ICPC.2016.7503721
89. Vásquez, M.L., McMillan, C., Poshyvanyk, D., Grechanik, M.: On using machine learning to automatically classify software applications into domain categories. Empirical Software Engineering **19**(3), 582–618 (2014). DOI 10.1007/s10664-012-9230-z
90. Votipka, D., Stevens, R., Redmiles, E.M., Hu, J., Mazurek, M.L.: Hackers vs. testers: A comparison of software vulnerability discovery processes. In: 2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA, pp. 374–391 (2018). DOI 10.1109/SP.2018.00003
91. Wang, H., Li, H., Li, L., Guo, Y., Xu, G.: Why are android apps removed from google play?: a large-scale empirical study. In: A. Zaidman, Y. Kamei, E. Hill (eds.) Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018, pp. 231–242. ACM (2018). DOI 10.1145/3196398.3196412
92. Watanabe, T., Akiyama, M., Kanei, F., Shioji, E., Takata, Y., Sun, B., Ishii, Y., Shibahara, T., Yagi, T., Mori, T.: Understanding the origins of mobile app vulnerabilities: a large-scale measurement study of free and paid apps. In: International Conference on Mining Software Repositories, pp. 14–24 (2017). DOI 10.1109/MSR.2017.23
93. Wu, D., Chang, R.K.C.: Analyzing android browser apps for file: // vulnerabilities. In: Information Security - International Conference, pp. 345–363 (2014). DOI 10.1007/978-3-319-13257-0_20
94. Xu, M., Song, C., Ji, Y., Shih, M., Lu, K., Zheng, C., Duan, R., Jang, Y., Lee, B., Qian, C., Lee, S., Kim, T.: Toward engineering a secure android ecosystem: A survey of existing techniques. ACM Comput. Surv. **49**(2), 38:1–38:47 (2016). DOI 10.1145/2963145

95. Yang, W., Zhang, Y., Li, J., Liu, H., Wang, Q., Zhang, Y., Gu, D.: Show me the money! finding flawed implementations of third-party in-app payment in android apps. In: Annual Network and Distributed System Security Symposium (2017)
96. Yang, Z., Yang, M., Zhang, Y., Gu, G., Ning, P., Wang, X.S.: Appintent: analyzing sensitive data transmission in android for privacy leakage detection. In: 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013, pp. 1043–1054 (2013). DOI 10.1145/2508859.2516676
97. Yeom, C., Won, Y.: Vulnerability evaluation method through correlation analysis of android applications. Sustainability **11**(23) (2019). DOI 10.3390/su11236637
98. Zampetti, F., Di Sorbo, A., Visaggio, C.A., Canfora, G., Di Penta, M.: Demystifying the adoption of behavior-driven development in open source projects. Inf. Softw. Technol. **123**, 106311 (2020). DOI 10.1016/j.infsof.2020.106311
99. Zhou, Y., Jiang, X.: Detecting passive content leaks and pollution in android applications. In: Annual Network and Distributed System Security Symposium (2013)