Examining the Relationship between Security Metrics and User Ratings of Mobile Apps: A Case Study

XXXXXXXXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX, XX, XXX
XXXXXX@XXXXX.XXX

ABSTRACT

The success or failure of a mobile application ('app') is largely determined by user ratings. Users expect apps to continually provide new features while maintaining quality, or the ratings drop. However, apps must also be secure. But is there a historical trade-off between security and ratings? Or are app store ratings a more all-encompassing measure of product maturity? We collected and compared several security related metrics from 798 random Android apps in the Google Play store with a user rating of less than 3 against 861 apps with a user rating of 3 or greater.

We found that while lower-rated apps typically request more permissions, higher rated apps have a larger permissions gap which creates security and quality issues. We also found that lower rated apps have a higher risk vulnerability score based on the results of a static analysis tool used to access risk vulnerability. Conversely, an underprivilege is when an app does not request enough permissions.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection;

Keywords

Android, User Ratings, Security

1. INTRODUCTION

Mobile applications ('apps') are the software that runs on mobile devices such as smartphones and tablets. These apps are often distributed through a centralized online store. Google Play[1] is the most popular app store for Android, with over 1.5 billion downloads every month[2]. Apps are a major part of mobile consumer technology and have changed the computing experience of our modern digital society, allowing users to perform a variety of tasks not previously possible in a portable environment.

[1]https://play.google.com/store
[2]http://developer.android.com/about/

SAC'16, April 4-8, 2016, Pisa, Italy.

http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx ...$15.00.

The success of a mobile app is largely determined by user ratings from a digital storefront ('app store'). Users expect apps to continually provide new features, otherwise poor app store reviews and low ratings can be expected [16]. Developers must frequently update their app's dependencies to keep up with the rapid progress of mobile technology [29]. Most importantly, apps must be secure since they are a crucial entry point into our digital lives.

At a glance, one may assume that the challenge of security and customer satisfaction are trade-offs, since if developers focus on functionality and quality to keep ratings up, they may not pay the necessary attention to security. New security-inspired features may also be perceived by users as cumbersome, leading to lower ratings. Even a vulnerability in a dependency can be detrimental to users, yet developers may not have the resources to thoroughly inspect a third-party framework for security concerns. Experts have even warned that security trade-offs with other properties such as usability and performance are considered universal [1].

But is this trade-off historically true in the case of mobile apps? Empirically, do mobile apps with higher ratings have more potential security risks? Or do app store ratings represent a more all-encompassing measure of customer experience that indicates a maturity in all of the properties of an app, with security being just one aspect? These questions motivated us to empirically examine the relationship between user ratings and security. To measure potential security risks, we use automated static analysis tools specifically tailored to the Android platform. While far from a comprehensive security audit, the static analysis tools provide a broad and consistent measure of basic security flaws that might plague Android apps. To measure user rating, we extracted the user ratings of more than 1,600 Android apps from the Google Play app store.

The objective of this study is to investigate the the relationship between potential security risks and customer satisfaction by empirically evaluating Android apps with static analysis tools. Specifically, our research question is: *Are user ratings correlated with low potential security risks and security permissions, or do apps with higher ratings have more security risks?*

We found that while lower rated apps requested more permissions, higher rated apps did not adhere to the principle of least privilege and have a larger permission gap. Based on the results from a static analysis risk assessment tool, lower rated apps generally have a higher risk score, although only 3 of the 9 evaluated risk assessment areas were higher in low rated apps.

The rest of this paper is organized as follows: Section 2 discusses related works, and Section 3 provides an overview of the Android permission structure. Section 4 presents our study's design including the data collection process and description of static analysis tools. Section 5 provides an overview of our selected analysis algorithm and how we used it in our project. Section 6 presents our

study's findings, while Section 7 discusses limitations and future work to be conducted. Our paper is concluded in Section 8.

2. RELATED WORK

There has been a substantial amount of previous research analyzing the effects of permissions on the user's perception of the app. Felt et al. [12] performed usability studies with App users over the internet and in a laboratory setting. The work found that only 17% of users paid attention to permissions when installing apps. Lin et al. [20] conducted a study analyzing user's comfort levels when apps requested permissions which the users did not understand why the apps needed them. They found that users generally felt uncomfortable and may even delete applications when they did not understand why it requested a permission they deemed unnecessary.

Egelman et al. [10] found that approximately 25% of users were typically willing to pay a premium in order to use the same application, but with fewer permissions, while about 80% of users would be willing allow their apps more permissions to receive targeted advertisements if it would save them .99 cents on the purchase of the app. Stevens et al. [26] found that certain permissions that are not popular among developers were frequently misused, possibly due to a lack of documentation. Wijesekera et al. [32] conducted an analysis using 36 users studying how comfortable they were with the requested permissions of various apps. They found that 80% of participants would have preferred to prevent one permissions request upon app installations and that approximately 33% of all app requests were invasive and would have preferred to block them.

Wei et al. [31] analyzed third party and pre-installed Android apps and found that a high number of apps violated the principle of least privilege and that apps tend to add more privileges with each released version. While these works are profound, none perform a direct evaluation of security risks and user ratings as we have done on such a significant scale.

App ratings have demonstrated their importance in other areas of research as well. Harman et al. [13] found a strong correlation between the rating and the number of app downloads. Linares-Vasquez et al. [21] found that the fault-proneness of the APIs used by the apps negatively impacts their user ratings. Khalid et al. [15] examined 10,000 apps using FindBugs and discovered that warnings such as 'Bad Practice', 'Internationalization', and 'Performance'categories are typically found in lower rated apps. Their primary discovery was that app developers could use static analysis tools, such as FindBugs, to repair issues before users complained about these problems. Even though we also use ratings as an evaluation metric, we additionally look at permission and security risks. Vasquez et al. [22] studied over 7,000 Android apps to determine how the fault proneness of APIs contributed to an application's lack of success. They found that APIs used by highly rated apps are much less fault prone than APIs used by low-rated apps.

There has also been a substantial amount of work regarding the risks of overprivileges in Android apps. Felt et al. [11] discussed the dangers of overprivileged apps including unnecessary permissions warnings and exposure to various bugs and vulnerabilities. The study also examined 940 Android apps and found that about 33% of of them were overprivileged.

3. ANDROID PERMISSION STRUCTURE

Android developers operate under a permission-based system where apps must be granted access to various areas of functionality before they may be used. When an Android app is created, developers must explicitly declare in advance which permissions the app will require [11], such as the ability to write to the calendar, send SMS messages, or access the GPS. If an app attempts to perform an operation for which it does not have permission, a *SecurityException* is thrown. These security settings are stored in the *AndroidManifest.xml* file and include a wide range of permissions, some of which are `READ_CONTACTS`, `WRITE_SETTINGS` and `INTERNET`.

When installing the app, the user is asked to accept or reject these requested permissions. Once installed, the developer cannot remotely modify the permissions without releasing a new version of the app for installation [25]. Unfortunately, developers often request more permissions than they actually need, as there is no built in verification system to ensure that they are only requesting the permissions their app actually uses [11].

Inside of the AndroidManifest.xml file, permissions may be granted four protection levels[3]:

1. **Normal:** A lower-risk level that only grants apps access to isolated application-level features. Presents minimal risk to apps, user or system. Apps are automatically granted this type of setting at installation, without asking for the user's explicit approval. However, the user does have the option to review these permissions before installing an app.

2. **Dangerous:** A higher-risk permission that provides the app with control over the device or access to private user data. Due to the risk posed by these permissions, they may not be automatically granted by the system and the user must confirm these permissions before they are used.

3. **Signature:** Granted only if the app is signed with the same certificate as the app that declared the permission. The permissions are automatically granted by the system without notifying the user if the certificates match.

4. **signatureOrSystem:** Granted by the system to only apps that are signed with he same certificate as the app that declared the permission or are in the Android system image. This level is typically not recommended for use except in very specific situations.

In this study, we use the term *overprivilege* to describe a permission which is granted to an app which it does not need. Granting extra privileges creates unnecessary security vulnerabilities by allowing malware to abuse these unused permissions, even in benign apps. These extra privileges also increase the app's attack surface [5, 8]. Previous research has found that Android developers often mistakenly add unnecessary privileges in a counterproductive and futile attempt to make the app work properly, or due to confusion over the permission name they add it incorrectly believing its functionality is necessary for their app [11].

An *underprivilege* is a setting for which the app could fail because it was not given the proper permissions. Overprivileges are considered security risks and underprivileges are considered quality risks. The primary difference between requested permissions and overprivileges is that requested permissions are merely those that the app asks to use, and does not take into consideration if the app actually needs them or not.

Android's permission structure will be changing with the new API release. 'Android M' implements a new permission model where the user will no longer need to grant all permissions whenever they install or upgrade the app. The app instead requests permissions as they are needed and the user is provided a dialog box

[3]http://developer.android.com/guide/topics/manifest/permission-element.html

to accept or reject the permission. Additionally, this new structure will allow developers to ensure that they are in fact requesting all the necessary permissions using the *checkSelfPermission()* function. [2].

4. STUDY DESIGN

We first collected a variety of apps from Google Play using a modified collection tool and then analyzed them using two well known Android static analysis tools. An overview of our collection and analysis process is shown in Figure 1. We will next describe or data collection, selection and analysis process.

Figure 1: App Collection and Analysis Process

4.1 Data Collection

We collected apps from the Google Play store with a custom-built collector, which uses *Scrapy*[4] as a foundation. We chose to pull from Google Play since it is the most popular source of Android apps[5] and was able to provide other app related information such as the developer, version, genre, user rating, and number of downloads. We downloaded 1,000 apps with a user rating of at least 3, and were able to download 833 apps with a user rating of less than 3. Locating apps with a user rating of less than 3 was much more difficult than finding apps with a user rating of at least 3 [23]. In order to collect each app's requested permissions, lines of code (LOC) and number of Java files, we decompiled the Android application (apk) files using dex2jar[6] and jd-cmd[7], repeating a similar reverse engineering process as used in previous works [3,6,17].

4.2 Data Selection

We removed all apps with less than 1,000 downloads to limit the effects that rarely used apps would have on our study. This left us with 798 apps with a rating of less than 3, and 861 apps with a user rating of at least 3.

4.3 Static analysis tools

The next phase was to analyze the apps for potential security risks, and permissions issues. We used two open source static analysis tools in our study: Stowaway [11] and AndroRisk[8]. Stowaway evaluates the app for permission gaps, while AndroRisk determines the risk vulnerability level.

We selected Stowaway for determining the permission gap in apps since it is able to state the permissions that were causing it to be overprivileged and underprivileged, while using a static analysis based approach that did not require it to be ran on an Android device or through an emulator. Stowaway has also demonstrated its effectiveness in existing research [11, 24, 27]. Permlyzer [33], a more modern permission detection tool, was not used since its authors have not made it available for download. We use Stowaway to extract the number of overprivileges and underprivileges that are present in each app. This tool is comprised of two parts - API calls made by the app are determined using a static analysis tool and the permissions needed for each API are determined using a permissions map.

AndroRisk determines the security risk level of an application by examining several criteria. The first set is the presence of permissions which are deemed to be more dangerous. These include the ability to access the internet, manipulate SMS messages or the ability to make a payment. The second is the presence of more dangerous sets of functionality in the app including a shared library, use of cryptographic functions, and the presence of the reflection API.

We chose AndroRisk for several reasons. The first is that the Androguard library, which is it a part of, has already been used in a variety of existing research [4, 9, 30]. AndroRisk is also a freely available, open source tool which will allow others to replicate our findings. Finally, as a static analysis based vulnerability detection tool, AndroRisk was quickly able to effectively determine the risk level of a large number of downloaded apps.

4.4 Data Analysis

The data collected from the static analysis tools along with user ratings data was analyzed using standard libraries, such as the *stats* library in R.

5. APPROACH

In this section, we discuss the motivation, approach, and findings for our research question. A more detailed discussion of our results is then provided. Before we present the results to our research questions we present some basic information about the apps used in the study.

We have two sets of apps in our case study - 798 apps with a rating lower than 3 stars (low rating apps) and 861 apps with a rating greater than or equal to 3 stars (high rating apps). We use the one tailed Mann Whitney U (MWU) test for the hypothesis testing since it is non-parametric and we can find out if the low-rated apps indeed have higher or lower values for each of the security metrics. The MWU test compares two population means that originate from the same population set and is used to determine if two population means are equal. A p value which is smaller than the significance level implies that the null hypothesis can be rejected. A p value which is equal to or greater signifies that the null hypothesis cannot be rejected. In our analysis, we used a significance level of .05 to determine if we have enough data to make a decision if the null hypothesis should be rejected.

We first checked to see if there was a correlation between the size of the app and the evaluated security areas by using the Spearman rho and Kendall tau coefficient metrics. Each of these correlation metrics evaluate the relationship strength between two values. This is an important step since we found that more popular apps were generally larger.

When using the Spearman rho and Kendall correlation metrics, the correlation coefficient will vary between -1 and +1. As the value of the coefficient approaches ± 1, there is more of a monotonic, or perfect degree of association between the evaluated values. The relationship between the evaluated values becomes weaker as the correlation coefficient approaches 0. Based on our analysis, we did find a reasonably moderate correlation between the amount of underprivileges and AndroRisk score and the size of the app, but did not find a significant correlation between the number of overprivileges in an app, or the number of permissions an app requested. Complete results are shown in Table 1.

We next addressed our research question: **Do apps with lower ratings have more security risks?**

We apply Stowaway and AndroRisk to the two sets of apps in

[4]http://scrapy.org
[5]http://www.onepf.org/appstores/
[6]https://code.google.com/p/dex2jar/
[7]https://github.com/kwart/jd-cmd
[8]https://code.google.com/p/androguard/

Table 1: Correlation Metrics for App Size

Area	Spearman	Kendall
OPriv	.29	.22
UPriv	.6	.45
Permissions	-.04	-.03
AndroRisk	.50	.38

our case study - 798 apps with a rating lower than 3 stars (low rating apps) and 861 apps with a rating greater than or equal to 3 stars (high rating apps). We get the distribution for each of the permission and security based metrics from each set of apps. In order to answer our research question we checked if the values of permission and risk based metrics are different in high and low rating apps taken as two distinct groups. Our null hypothesis is that there is no difference in the distribution of the several evaluated metrics between the low and high-rated apps. Our alternate hypothesis is that low and high-rated apps have different distributions for each of the security related metrics.

6. EVALUATION

We present the results of comparing the distributions (hypothesis tests) of the various security risk based metrics from the two static analysis tools Stowaway and AndroRisk in Tables 2 and 3.

Table 3: MWU Permission Results for Low & High-rated apps

	Greater In	
Value	**Low**	**High**
# of Permissions in App	✓	
# of Overprivileges		✓
# of Underprivileges		✓

We performed a MWU on the number of requested permissions, and permission gap for both low and high-rated apps. In Table 3, we can see that low-rated apps indeed ask for more permissions than high-rated apps. However, the overprivilege and underprivilege rates are greater in high-rated apps. These results imply that even though low-rated apps request more permissions, they actually use a higher percentage of them. High-rated apps are asking for permissions that they do not even use, and this is quite dangerous. The high-rated apps also have a larger incidence of underprivileges. This implies that components of these apps are trying to perform activities which they do not have permissions to do. Such a problem not only indicates reliability issues, but also indicates that developers of the high-rated apps might be trying to get more information than they initially ask for. Due to the strong permissions setup in Android, such an attempt will fail and an exception will be being thrown.

There are several other possible reasons why higher-rated apps have a greater permission gap. Users will probably be unaware of this gap and may only very sporadically encounter possible crashes caused by underprivileged apps. When problems do arise, they may assume that these are issues with the Android OS, and not with the app itself. Users are generally unaware of what Android's permissions actually mean, and coupled with a lack of internal knowledge of the source code, may simply believe that the permission is actually required when it actually represents a security vulnerability [19]. Additionally, a large portion of users do not even pay attention to an app's permissions when installing it [12]. Studies have shown that users are generally wary of apps that request large numbers of privileges [32]. This is likely a reason why apps with more permissions generally have a lower rating.

From Table 2, we can see that the overall risk score (FuzzyRisk) in low-rated apps is greater than in high-rated apps. However, when we look at the breakdown in the individual risk metrics, only three of the nine metrics are higher in low-rated apps as compared to high-rated apps: reflection API use (Dex_Reflection), access of internet (Perm_Internet), and access to private data (Perm_Dangerous). High-rated apps have a larger risk of manipulating SMS messages as well as loading .dex files dynamically. High-rated apps also more likely to ask for the permission to process payments - combined with the risk of loading dex files dynamically, a new executable could possibly take money out of the user's account as well. The combination of the risk of access to private data and the ability to access to internet, is largely the reason why the overall risk score is high even though only three of the nine risk metrics are higher in low-rated apps.

6.1 Coding Standards & App Defects

We performed a secondary analysis evaluating the variations in coding standards mistakes, and discovered defects in the apps using CheckStyle[9] and JLint[10], two popular static analysis tools.

Table 4: MWU Analysis of Coding Standards Mistakes & Errors

	Greater In	
Value	**Low**	**High**
Coding Standards/LOC	✓	
JLint Errors/LOC	✓	

Adhering to coding standards is important for software developers as it has been shown to enhance team communication, reduce program errors and improve code quality [18,19]. We measured the number of coding standards defects for each app using CheckStyle and then divided the number of coding standards reported for the app by number of LOC to normalize the results by the size of the application. We then analyzed these results using the same MWU formula we used for our security analysis. We found that higher rated apps suffer from fewer coding standards defects per LOC in relation to low-rated apps. The results of our analysis are shown in Table 4.

Although more work should be conducted in this area, there are several possible reasons for this. Developers that spend more time ensuring that they are adhering to coding standards will likely spend more time testing their app to ensure it is of high quality. Adhering to coding standards make it quicker and easier to release new versions, features and bug fixes [18, 19], which could increase user satisfaction in the app.

We then performed a similar analysis using JLint, a powerful defect detection tool. We found that lower-rated apps had a higher number of JLint defects per LOC than their higher-rated counterparts. Although this analysis was conducted in a much smaller scale, our results correlate to those found by Khalid et al. [15], who used a different tool to achieve their results.

7. LIMITATIONS & FUTURE WORK

While we feel our findings to be profound, they are not without their limitations. Although Google Play is the largest Android market place, it is not the only source for Android apps. Alternatives include the Amazon app store, F-Droid[11] and a multitude of other sources. Other studies may choose to include apps from these other

[9] http://checkstyle.sourceforge.net
[10] http://jlint.sourceforge.net
[11] https://f-droid.org/

Table 2: AndroRisk MWU Metrics For Low & High Rated Apps

		Greater In	
Value	**Description**	**Low**	**High**
Dex_Native	Presence of loading a shared library		✓
Dex_Dynamic	Presence of loading dynamically a new dex file		✓
Dex_Crypto	Presence of crypto functions		✓
Dex_Reflection	Presence of the reflection API	✓	
Perm_Money	Presence of permissions which can result to a payment		✓
Perm_Internet	Presence of permissions which can access to internet	✓	
Perm_SMS	Presence of permissions which can manipulate sms		✓
Perm_Dangerous	A higher-risk permission that would give a requesting application access to private user data or control over the device that can negatively impact the user	✓	
Perm_Signature	A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission		✓
FuzzyRisk	Overall Risk Score of App (A smaller score is better)	✓	

sources. Additionally, we chose apps at random and only selected a total of 1,659 apps, which is a small minority of the over 1.5 million Android apps available. However, given that this is a random sample we believe that it is representative of the Android application population.

While static analysis tools have demonstrated their value in numerous previous works [11, 24], it is unreasonable to expect that any tool will ever be flawless and that no static analysis tool is perfect and they generally inherently contain limitations [7]. Although Stowaway is a powerful static analysis tool which has been used in previous research [14, 24, 28], it does suffer from drawbacks. Stowaway's own authors state that the tool only achieves 85% code coverage [11], so the over & under privileges reported by this tool are imperfect. Additionally, any reported vulnerabilities by a static analysis tool should be deemed as *possible* vulnerabilities, not actual vulnerabilities. The only way of identifying actual vulnerabilities is through manual analysis and verification.

Identifying possible vulnerabilities or security risks is extremely difficult, and like any static analysis tool, AndroRisk is only capable of making educated observations about the risk level of an app and that more substantial risk assessments will require a far more substantial level of analysis, which will likely include a manual investigation of the app. Due to the large number of examined apps in our study, this thorough level of analysis was not practical. Even with almost certain imperfections, we believe that AndroRisk was a good selection due to its ability to quickly analyze apps and its use in existing research [17].

We have presenting interesting findings about the relationship between an Android app's user ratings and its security. However, there is still a large amount of future work which can be conducted on this topic. Our study evaluated 1,659 Android apps, which represents only a very minor portion of all existing Android apps. Future work may be done to expand on our data set and analyze a more substantial number of apps. We chose to use Stowaway and AndroRisk as the major static analysis tools in our study, but new and powerful assessment tools are constantly being created and may be applied to our work.

We used the Mann Whitney U, Spearman, and Kendall metrics in our research to determine variations in high and low-rated apps. Future work could be done to see if the Pearson correlation could be used to form a predictive model with our collected data.

In our evaluation, we only measured the user ratings of apps. An interesting study would be examine the comments left by users to see if they are complaining more about security or permissions in apps which have more vulnerabilities, more permissions, or more overprivileges. Work could also be done to ensure that all apps are compared against each other in groups based upon functionality, size, complexity, or other differentiating metrics. However, we are confident in our results due to the magnitude and app diversity of our study. As with previous work [23], we unfortunately found that collecting low-rated apps with at least 1,000 downloads was difficult. Finally, breaking down the results into more groups would have created smaller sample sizes, thus possibly hurting the confidence of our results.

8. CONCLUSION

Summary: The goal of our research was to determine if low-rated apps suffered from more possible security vulnerabilities than high-rated apps. We statically analyzed more than 1,600 Android apps (both high-rating and low-rating apps) for security threats primarily using two static analysis tools. We evaluated apps for over & underprivileges using Stowaway, and AndroRisk for a more general security risk assessment.

Findings: Low-rated apps tend to have more permissions, and a higher AndroRisk score, while high-rated apps suffered from more overprivileges and underprivileges.

Acknowledgements

Author and funding acknowledgments hidden for review anonymity.

9. REFERENCES

[1] *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[2] Android m permissions. https://developer.android.com/preview/features/runtime-permissions.html, 2015.

[3] A. Apvrille. Android reverse engineering tools, 2012.

[4] A. Atzeni, T. Su, M. Baltatu, R. D'Alessandro, and G. Pessiva. How dangerous is your android app?: An evaluation methodology. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MOBIQUITOUS '14, pages 130–139, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[5] A. Bartel, J. Klein, Y. Le Traon, and M. Monperrus. Automatically securing permission-based software by reducing the attack surface: An application to android. In

Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012, pages 274–277, New York, NY, USA, 2012. ACM.

[6] T. K. Chawla and A. Kajala. Transfiguring of an android app using reverse engineering. 2014.

[7] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, (6):76–79, 2004.

[8] L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Privilege escalation attacks on android. In *Proceedings of the 13th International Conference on Information Security*, ISC'10, pages 346–360, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*, CCS '13, pages 73–84, New York, NY, USA, 2013. ACM.

[10] S. Egelman, A. P. Felt, and D. Wagner. Choice architecture and smartphone privacy: There's a price for that. In *In Workshop on the Economics of Information Security (WEIS)*, 2012.

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

[12] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

[13] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: Msr for app stores. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 108–111, June 2012.

[14] J. Jeon, K. K. Micinski, J. A. Vaughan, N. Reddy, Y. Zhu, J. S. Foster, and T. Millstein. Dr. android and mr. hide: Fine-grained security policies on unmodified android. 2011.

[15] H. Khalid, M. Nagappan, and A. E. Hassan. Examining the relationship between findbugs warnings and end user ratings: A case study on 10,000 android apps. In *IEEE Software Journal*, 2014.

[16] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan. What do mobile app users complain about? a study on free ios apps. *IEEE Software*, 99(PrePrints):1, 2014.

[17] D. E. Krutz, M. Mirakhorli, S. A. Malachowsky, A. Ruiz, J. Peterson, A. Filipski, and J. Smith. A dataset of open-source android applications. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 522–525. IEEE, 2015.

[18] X. Li. Using peer review to assess coding standards-a case study. In *Frontiers in education conference, 36th annual*, pages 9–14. IEEE, 2006.

[19] X. Li and C. Prasad. Effectively teaching coding standards in programming. In *Proceedings of the 6th Conference on Information Technology Education*, SIGITE '05, pages 239–244, New York, NY, USA, 2005. ACM.

[20] J. Lin, S. Amini, J. I. Hong, N. Sadeh, J. Lindqvist, and J. Zhang. Expectation and purpose: Understanding users' mental models of mobile app privacy through crowdsourcing. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 501–510, New York, NY, USA, 2012. ACM.

[21] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 477–487, New York, NY, USA, 2013. ACM.

[22] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. Api change and fault proneness: a threat to the success of android apps. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*, pages 477–487. ACM, 2013.

[23] I. J. Mojica Ruiz. Large-scale empirical studies of mobile apps. 2013.

[24] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner. Addroid: Privilege separation for applications and advertisers in android. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '12, pages 71–72, New York, NY, USA, 2012. ACM.

[25] K. Shaerpour, A. Dehghantanha, and R. Mahmod. Trends in android malware detection. *Journal of Digital Forensics, Security & Law*, 8(3), 2013.

[26] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen. Asking for (and about) permissions used by android apps. In *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, pages 31–40, May 2013.

[27] R. Stevens, J. Ganz, V. Filkov, P. Devanbu, and H. Chen. Asking for (and about) permissions used by android apps. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 31–40, Piscataway, NJ, USA, 2013. IEEE Press.

[28] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen. Investigating user privacy in android ad libraries.

[29] M. D. Syer, M. Nagappan, A. E. Hassan, and B. Adams. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source android apps. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 283–297, Riverton, NJ, USA, 2013. IBM Corp.

[30] T. Vidas, J. Tan, J. Nahata, C. L. Tan, N. Christin, and P. Tague. A5: Automated analysis of adversarial android applications. In *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, SPSM '14, pages 39–50, New York, NY, USA, 2014. ACM.

[31] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 31–40, New York, NY, USA, 2012. ACM.

[32] P. Wijesekera, A. Baokar, A. Hosseini, S. Egelman, D. Wagner, and K. Beznosov. Android permissions remystified: A field study on contextual integrity. *arXiv preprint arXiv:1504.03747*, 2015.

[33] W. Xu, F. Zhang, and S. Zhu. Permlyzer: Analyzing permission usage in android applications. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 400–410, 2013.