Understanding the Relationship Between Quality and Security: A Large-Scale Analysis of Android Applications

Anthony Peruma and Daniel E. Krutz
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY, USA
{axp6201,dxvse}@rit.edu

ABSTRACT

Android applications (apps) are not immune to the problems which also plague conventional software including security vulnerabilities, quality defects, permission misuse, and numerous other issues. Many developers even intentionally create vulnerable or malicious apps (malware) for often highly lucrative purposes. We need to better understand current trends in app quality and security to create higher quality software, and more effectively battle malware. To gather this critical information, we collected and reverse engineered 70,785 Android apps from the Google Play store, along with 1,420 malicious apps from other sources. Each app was analyzed using several static analysis tools to record a variety of information about each of them including requested permissions, size (LOC), possible defects and permission misuse. Our findings conclude that: 1) app categories substantially differ in terms of permissions misuse; 2) at an aggregate level, there is no significant correlation between an app's quality and security; 3) that malware typically requests more permissions and suffers from several quality-related metrics in comparison to benign apps; 4) that malware and benign apps are growing annually both in terms of LOC and requested permissions. We also present an easy to use, robust dataset for the community to replicate or extend this study.

CCS CONCEPTS

• **Security and privacy** → **Software security engineering**; • **Software and its engineering** → *Software notations and tools*;

KEYWORDS

Mobile Security, Mobile Malware, Reverse Engineering, Android Development

ACM Reference Format:
Anthony Peruma and Daniel E. Krutz. 2018. Understanding the Relationship Between Quality and Security: A Large-Scale Analysis of Android Applications. In *SEAD'18: SEAD'18:IEEE/ACM 1st International Workshop on Security Awareness from Design to Deployment , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194707.3194711

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SEAD'18, May 27, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5727-2/18/05…$15.00
https://doi.org/10.1145/3194707.3194711

1 INTRODUCTION

Android is the world's most popular mobile operating system with over 1.8 million applications (apps) available from Google Play alone [2]. To create better apps, we need to better understand the current trends in development, and what some of the most profound issues from a security and quality perspective are. A few areas which should be examined include permissions misuse, security vulnerabilities, defects, and adherence to coding standards. In recent academic studies, static analysis tools have been used as one method of measuring the quality and security of mobile software [7, 12, 13, 26]. *An empirical analysis of a large body of malicious and benign Android applications over time can, therefore, provide insights into these apps, and their trends.*

The goal of this work is to compare and contrast between benign and malicious Android apps from various perspectives. We collected and reverse engineered 70,785 Android applications (APK files) in 41 different categories from Google Play, along with 1,420 malware samples from the Contagio Mobile Mini Dump [1] and the Malware Genome Project [30]. Each app was analyzed using five static analysis tools: Stowaway [7], Androrisk[1], CheckStyle[2], Jlint[3], and APKParser[4]. The comparison criteria include structural measures such as LOC, the rate of potential defects, adherence to coding standards, the rate of permissions misuse, and security risk level. We have also created a publicly available dataset and robust website which may be used by researchers, developers, students, and general Android users to understand these apps better. Our work is guided by the following research questions:

RQ1: *How do app categories compare in terms of several security metrics?* We found that app categories significantly differ in terms of size, and other permissions based security metrics. Some of the evaluated areas include number of requested permissions, and permission gap.

RQ2: *Is there a correlation between quality & security?* We found only a weak association between the quality and security of apps in the several evaluated comparisons.

RQ3: *How do benign & malicious Android apps compare in terms of several security based metrics?* Our results indicate that malicious apps typically contain more over-permissions, and total requested permissions, while Google Play apps have a more significant rate of under-permissions and Androrisk score.

RQ4: *Is there an evolution pattern that differs malicious apps from benign apps over time?* We found that both malicious and benign

[1] https://github.com/androguard/androguard
[2] http://checkstyle.sourceforge.net/
[3] http://jlint.sourceforge.net/
[4] https://github.com/joakime/android-apk-parser

apps are annually growing in terms of both LOC and requested permissions.

The rest of the paper is organized as follows: Section 2 discusses related works, while Section 3 describes the Android permissions model. Section 4 provides details of how we collected the apps and conducted our static analysis on them. Section 5 discusses the results of our research questions and analyzes our findings. Section 6 presents information regarding our public dataset. Section 7 discusses limitations of our research and future work to be conducted, while Section 8 concludes our study.

2 RELATED WORK

There have been several studies which analyzed mobile apps on a large scale. Sarma *et al.* evaluated several large data sets, including one with 158,062 Android apps in order to gauge the risk of installing the app, with some of the results broken down by category. However, this work did not analyze the apps using the range of static analysis tools which we used. Viennot *et al.* [27] developed a tool called 'PlayDrone' which they used to examine the source code of over 1,100,000 free Android apps. While the authors studied a very large number of apps, they mostly only used existing information which could be gathered from Google Play and only examined features such as library usage and duplicated code. They did not study areas such as security risk levels, quality attributes, or the permission gap, which were a part of our analysis. Felt *et al.* [7] described some common developer errors found using their tool Stowaway, including confusing permission names, the use of depreciated permissions, and errors due to copying and pasting existing code. In another work, Felt *et al.* [8] very briefly described some inclinations they had for why developers gave too many permissions to applications, but this was primarily based on assumptions and not necessarily data.

Grace *et al.* [10] conducted work on permissions probing, which is when a 3rd party component attempts to use a permission in the hope that the attached app has requested it from the user. If the attached app has requested a permission, then the component will also have access to that permission as well. This is often done to collect, and transmit potentially sensitive information which should not be normally available to the 3rd party component. They found that more than half of all ad libraries try to probe for open permissions. This could often be the cause of an under-permission in an app since the ad library will try to use a permission which the developer did not request.

Stevens *et al.* [23] analyzed 10,000 free Android apps and found a strong sub-linear relationship between the popularity of a permission and the frequency of its misuse. They found that developers were more likely to misuse a permission when they did not understand it, and that the popularity of a permission is strongly associated with its misuse. A powerful method of avoiding permission misuse is through developer education and community support. Krutz *et al.* [12] created a public dataset of over 1,100 Android apps from the F-Droid[5] repository. This research analyzed a much smaller number of apps than our study and focused more on the life cycle of the apps and how each iteration of the app evolved with every version control commit.

[5]https://f-droid.org/

There are several other websites which gather metrics about Android apps. One of the most popular is AppAnnie[6] which collects Android apps and performs several types of analysis on each of them including downloads of the app over time and advertising analytics. However, no known services perform the same types of static analysis and comparisons on apps that we do.

A substantial amount of research has been conducted on Android malware ranging from various detection techniques [9, 18, 24] to reconstructing malware behavior [19]. Zhou *et al.* [30] analyzed a large number of malware samples and characterized them in several areas including installation techniques, activation methods, and nature of malicious payloads. Unfortunately, the newest of these malware samples was only from 2011, and did not compare malicious and benign apps against one another, or use the range of static analysis tools as in this study.

3 ANDROID PERMISSIONS

Android apps operate under a permission-based system where an app requires specific permissions to carry out specific functionality. 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 application actually uses [7]. In this study, we use the term *over-permission* to describe a permission setting that grants more than what a developer needs for the task. Likewise, an *under-permission* is a setting for which the app could fail because it was not given the proper permissions. If not properly handled, an application could throw a *SecurityException* when it attempts to perform an operation which it does not have permission to conduct.

Over-permissions are considered security risks since they do not adhere to the *principle of least privilege* and unnecessarily increase an app's attack surface. Under-permissions are considered quality risks, with a possible indication of permission probing. The primary difference between requested permissions and over-permissions 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. While primarily a quality concern, under-permissions can represent a possible security concern as well. One example of this is when permissions are, unknowingly to the developer, misused in a variety of ways by 3rd party libraries or even by associated ad networks which may collect and transmit potentially sensitive user data [10]. Under-permissions essentially leave a door open for other system operations, malicious or not, to use.

4 COLLECTION & STATIC ANALYSIS

We analyzed apps collected from Google Play and several malware sources using a variety of different tools. The results of this analysis have been stored in a publicly accessible database located on our project website http://darwin.rit.edu. An overview of the collection and analysis process is shown in Figure 1.

4.1 Step 1: APK files collection

Android APK files were retrieved from Google Play with a custom-built collector, which uses *Scrapy*[7] as a foundation. To limit the

[6]https://www.appannie.com
[7]http://scrapy.org

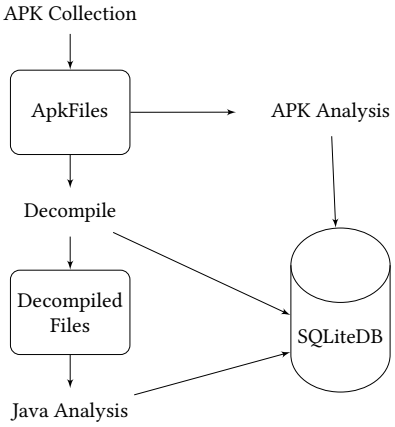

Figure 1: APK Analysis Process

impact of seldom-downloaded applications, we divided of our results into two groups: applications with at least 10,000 downloads, and those with less than 10,000 downloads. Of the 70,785 apps downloaded, 31,234 had at least 10,000 downloads. The creation date of the collected apps ranged from 2011, to late 2015. We assume that these apps from Google Play are not malicious, and will often refer to them as benign apps when comparing them with malware. Although malware has been found on Google Play, we do not believe that this amount will be enough to significantly affect our findings. We only used free apps in our analysis. The malware was collected from two well-known sources, the Contagio Mobile Mini Dump [1] and the Malware Genome Project [30]. The Contagio Mobile Mini Dump has been collecting malware affecting many platforms, including Android, for several years. We used a total of 1,420 malware examples from 49 malware families in our analysis.

4.2 Step 2. Static analysis tools Execution

After reverse engineering the apps using a process established in previous works [13, 29], the next phase was to analyze them for a variety of security and quality metrics. In addition to using the tools below, we also recorded other metrics about each application including total lines of code, number of Java files, application version, target SDK, and minimum SDK.

Stowaway: Reports the under-permissions and over-permissions of an application. Similar to previous work [23], we made slight modifications to Stowaway to accommodate our process and stay current with updated Android permissions.

Androrisk: Determines the security risk level of an app by evaluating several criteria. The first 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 rights 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.

APKParser: Reads various information from Android APK files including the version, intents, and requested permissions.

CheckStyle: Measures how well developers adhere to coding standards such as annotation usage, size violations, and empty block checks. We recorded the total number of violations of these standards. Default application settings for Android were used in our analysis. While adherence to coding standards may seem to be an overly fastidious thing to measure, compliance to coding standards in software development can enhance team communication, reduce program errors and improve code quality [14, 15].

Jlint: Discovers bugs, inconsistencies, and synchronization problems by conducting a data flow analysis and building lock graphs. We recorded the total number of discovered possible defects. This tool was selected over FindBugs since it was able to analyze the applications much faster, while still providing accurate results [20].

5 EVALUATION

5.1 RQ1: How do app categories compare in terms of several security metrics?

Android apps are separated into different groups known as categories, some of which include 'Communication', 'Education', 'Lifestyle', and 'Tools'. These different app groups all have different focuses, likely different target audiences, and very often different developers. We sought to better understand how apps from different categories compare in terms of several metrics including the use of permissions, the rate of permissions misuse, and app size. Table 1 shows the top 10 collected categories, and an aggregate of all collected apps from Google Play (not just from the top categories) along with malicious apps.

We found a large variation in the rate of over-permissions for each category with 'Casual' and 'Education' apps having the fewest with 2.2 per app and 'Communication' having the largest with 5.2 for each app. 'Personalization' apps had the fewest number of under-permissions (2.5), while 'Casual' apps had the highest rate with 4 per app. The 'Books & Reference' category requested the fewest permissions per app, while 'Communication' apps requested the most permissions with an average of 14.6. 'Lifestyle' apps were narrowly the largest with an average 166,114 LOC per app, while 'Tools' apps were the smallest with 106,131 LOC.

Analysis: These findings are important for several reasons. The results show a wide disparity among the various categories in terms of app size and security concerns. Developers may use this information to devote more resources to specific categories in the identified permissions based issues, while for researchers it demonstrates the wide disparity in terms of size, requested permissions and the permission gap between various categories. Over-permissions are a possible security vulnerability, so app categories with more over-permissions should be given more focus from security researchers from both the perspective of why they occur, to what their negative implications may be for these categories. With the introduction of Android 6.0, future work should be done to examine the rate of change in over-permissions in the different categories with this new permissions model. Also, since over-permissions are often caused by lack of developer understanding with the abused permissions [23], these categories with the most over-permissions require the most developer education.

Table 1: Comparison of App Groups

Group	LOC	Over-Permissions	Under-Permissions	Requested Permissions
Arcade	152,563	2.3	3.8	6.6
Books & Reference	123,600	2.6	2.7	5.4
Casual	150,396	2.2	4	6.5
Communication	166,083	5.2	3.6	14.6
Education	131,615	2.2	3.3	5.8
Entertainment	138,229	2.6	2.8	6.3
Lifestyle	166,114	2.9	3	8.2
Music & Audio	146,211	2.5	2.7	6.7
Personalization	113,539	3.8	2.5	8.3
Puzzle	142,998	2	3.8	5.6
Tools	106,131	4.1	2.9	8.4
Avg Google Play	158,689	3.0	3.3	7.8
Malware	32,373	6	1.9	12.4

5.2 RQ2: Is there a correlation between quality & security?

When creating software, there are several primary concerns of developers, two of which include the security and the quality of the application. We next sought to determine if there was a frequent correlation between the security of an application, and its quality. We measured quality by adherence to coding standards (Checkstyle) and detected possible bugs (Jlint), while security was measured by the Androrisk score and number of detected over-permissions. Both Jlint and Checkstyle values were normalized by the app's LOC to limit the effects of an app's size on the results. We used the Spearman rho and Kendall correlation metrics to determine if there was a correlation between quality and security. Using these correlation metrics, we evaluated the relationship strength of quality (Coding Standards Defects & Jlint Errors) and security (Androrisk & over-permissions). The results of this analysis are shown in Table 2.

Table 2: Correlation Metrics for Quality & Security

Area		Correlation	
Security	**Quality**	**Spearman**	**Kendall**
Over-Permission	Jlint	.18249	.13764
Over-Permission	CSDefect	.12584	.10128
Androrisk	CSDefect	.14305	.11481
Androrisk	Jlint	.38502	.29424

Analysis: Based on our analysis, we found only a weak association between the quality and security of applications in the several evaluated comparisons. These findings are somewhat surprising since previous research has discussed the importance of adhering to coding standards as a way for developers to avoid mistakes and improve overall application security [6, 11]. Although we are unable to declare a definitive reason for this, there are several possible causes for this weak correlation. One is that developers may make permissions related mistakes for a variety of reasons including confusing permission names, the use of depreciated permissions, and errors due to copying and pasting existing code [7]. The cause of these types of mistakes are likely very different than the factors leading to other developer mistakes such as defects, or even a lack of adherence to coding standards. Our findings demonstrate the need for further research and analysis in this area.

5.3 RQ3: How do benign & malicious Android apps compare in terms of several security based metrics?

In 2013 alone, there were at least 1,800 new mobile malware families, with over 96% of malware targeting Android [3]. In order to better understand Android malware, we compared the static analysis results from apps collected from the Google Play store against 1,417 malware examples taken from the Contagio Mobile Mini Dump [1] and the Malware Genome Project [30]. We evaluated the malware samples against those collected from Google Play in a variety of areas including Androrisk score, under & over-permissions, and the requested app permissions. The results of this analysis are shown at the bottom of Table 1.

Table 3: MWU Results for Malicious & Benign Apps

	Greater In		
Area	**Malware**	**GP**	**P-value**
Over-permissions/App	✓		2.7002e-142
Under-permissions/App		✓	3.0972e-30
Permissions/App	✓		3.0872e-121
Androrisk		✓	2.1298e-32

To test the statistical significance of our results, we used Stowaway, Androrisk, and APKParser to check if the values of permission and risk-based metrics are different in malicious and benign apps. Our null hypothesis is that there is no difference in the distribution of the security metrics between the malicious and benign apps. Our alternate hypothesis is that malicious and benign apps have different distributions for each of the security and quality-related metrics. 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 malicious apps indeed have higher or lower values for each of the evaluated metrics. In our analysis, we used an α-value of .05 to determine if the null hypothesis can be rejected or not. As shown Table 3, the results of this analysis further validate our findings displayed in Table 1.

We next compared the requested permissions for all malicious and benign apps using APKparser. The primary difference between requested permissions and over-permissions 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. We compared the top ten rates of requested permissions for the malicious apps against those recorded from Google Play. The results of this comparison are shown in Table 4. The Android Malware Repository[8] used a different malware oracle and found similar permissions results to what we discovered, which provides confidence to our findings. Unfortunately, they do not appear to have analyzed any apps since 2012.

Table 4: Permissions Usage

Permission	Malware %	G-Play %
INTERNET	98	94
READ_PHONE_STATE	93	45
ACCESS_NETWORK_STATE	81	84
WRITE_EXTERNAL_STORAGE	68	62
ACCESS_WIFI_STATE	61	36
READ_SMS	60	4
RECEIVE_BOOT_COMPLETED	56	1
WRITE_SMS	49	2
SEND_SMS	45	4
RECEIVE_SMS	42	5

Analysis: Table 3 indicates that malicious apps typically contain more over-permissions, and total permissions, while Google Play apps have a larger rate of under-permissions and Androrisk score. Under-permissions are a quality concern since an app will likely crash if it requests a permission which it was not granted. We also found that malicious apps have fewer under-permissions per app compared to apps from Google Play. An explanation for this could merely be that the permissions in malicious apps are more often being used, unfortunately for disruptive activities. We also found that malicious apps request more permissions than benign apps with an average of 12.4 permissions compared with 7.8 permissions for those collected Google Play. Malware is more likely to request these additional permissions to perform malicious activities [21].

The knowledge of what permissions have a higher prevalence in malware has several possible uses. Apps which request these permissions could be noted to have a higher probability that they are malicious. Since many benign apps have legitimate uses for these permissions, they cannot be the sole indicator, but can be a sign of a possibly malicious application. Understanding what permissions malware requests may be useful for understanding how current and future malware spreads and affects devices.

[8]https://sites.google.com/site/androidmalrepo/permission-stats

Table 5: Evolution of Google Play Apps By Year

Year	LOC	Opriv	UPriv	Requested Permissions
2011	25,549	2.1	2.3	3.9
2012	38,183	2.3	2.4	4.8
2013	83,445	2.3	2.9	5.9
2014	206,903	3.3	3.3	9.2
2015	248,802	3.6	5.3	10

5.4 RQ4: How are apps changing over time?

We next examined how apps are evolving on an annual basis. We first grouped the collected Google Play apps into the year they were created by using the *Date Published* value for the app from Google Play. We next found the average size (LOC), under & over-permissions, and requested permission for all apps in each age range. The results are shown in Table 5. We found that benign apps are growing in terms of size (LOC), and in the number of requested permissions. Unfortunately, the amount of under & over-permissions are growing as well.

We compared how Android malware and Google Play apps are evolving on an annual basis in terms of size (LOC) and requested permissions. We separated the apps collected from the Contagio Mobile Mini Dump into separate groups based on the years they were created (2012, 2013, 2014, and 2015). These years were selected as they were there four years that provided the largest amount of data. The results of this analysis are shown in Figure 2 and Figure 3.

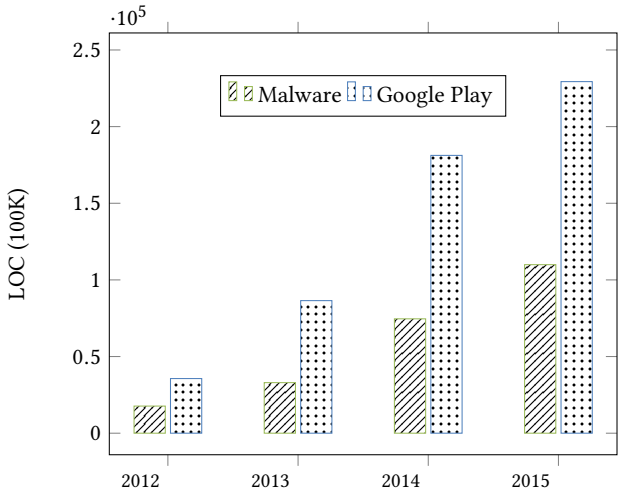

Figure 2: Evolution of Apps - LOC

Analysis: We found that malicious apps are growing on an annual basis in both LOC and requested permissions. An interesting discovery is that although both are consistently growing, Google Play apps are larger in terms of LOC, but are smaller in the number of requested permissions. Our findings have several important implications for both general app development, and malware research. For developers of benign apps, indications are that apps will

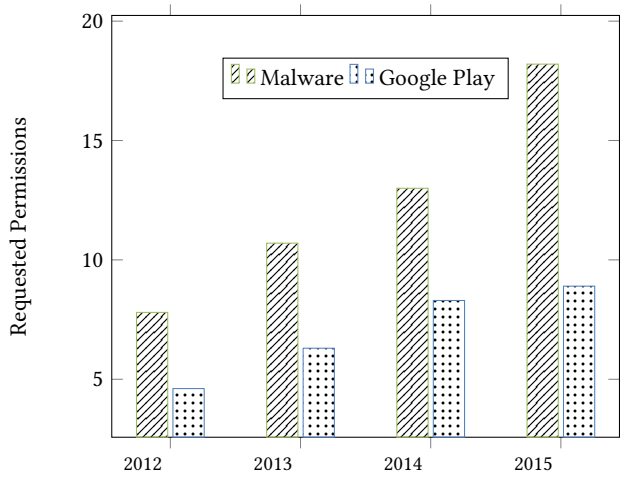

Figure 3: Evolution of Apps - Permissions

continue to grow in terms of LOC and in requested permissions. This is significant for several reasons. The first is that with more permissions, developers will need to be better educated about the permissions they use and diligent to ensure that they are correctly using them. This will be further complicated by Google's move to a new permission structure in Android 6.0 which will allow apps to request permissions at runtime instead of only upon installation of the app.

Benign Apps are becoming larger in terms of LOC, which likely means that teamwork will be increasingly more important since more developers will be needed to create an maintain the apps. With more developers being involved in the app, fundamental software engineering skills such as communication, documentation, diverse team roles, testing and adhering to coding standards will become increasingly important [17]. On average, malware is smaller in terms of LOC in comparison to benign apps, but is growing and requesting more permissions on an annual basis. There are a variety of current malware detection techniques including API tracing [28], behavioral based techniques [4, 22], permission-based systems [25], and signature-based approaches [9]. Larger software will make many of these processes more difficult and time-consuming, leading to the need for different and more efficient processes.

Table 6: Malware Permission Growth By Year

	% Occuring			
Permission	**2012**	**2013**	**2014**	**2015**
RECEIVE_BOOT_COMPLETED	55	52	71	86
READ_CONTACTS	36	55	41	68
SYSTEM_ALERT_WINDOW	1	16	27	61
WAKE_LOCK	34	19	41	54
GET_TASKS	17	25	43	39

We next chose to examine the usage rate in several of the permission requests which were growing at the most substantial rate between 2012 and 2015 in malware. The results of this are shown in Table 6. These results indicate that malware is using these permissions much more frequently now, than even a few years ago. For instance, the usage of `SYSTEM_ALERT_WINDOW` has gone from appearing in 1% of malware in 2012, to 61% in 2015. This permission allows an app to create new windows on the Android UI, and may often be used by malware to display a message in an attempt to convey misinformation to the user. Security researchers can analyze these trends to see how malware is evolving over time and the different types of attacks they are attempting to perform using these extra permissions. This information could prove to be invaluable for both detecting and countering the negative effects of malware.

6 PUBLIC DATASET & WEBSITE

Dataset: Our dataset is available from our publicly accessible GitHub repo[9], which includes the scripts used for collecting apps and invoking the static analysis tools. The SQLite database with our complete results is updated on a regular basis from our collection and analysis software. The goal of this dataset is to allow future researchers to both learn from and expand upon our work. This dataset contains the information collected from Google Play, and the results of our static analysis tools. The raw data used in our analysis is available in three SQLite databases from our public GitHub repository, with one each for the malware data from Contagio and the Malware Genome projects, and a third for the collected information from Google Play. Unfortunately, the actual malicious APK files may only be obtained from the Contagio and Genome websites due to usage agreements.
Website: Our project website (**http://darwin.rit.edu**) contains information about our project, links to our GitHub repository, and a robust reporting tool which will allow users to create their own datasets from over 70,000 analyzed applications. New apps will be added on a regular basis as they are collected and analyzed from Google Play.

7 LIMITATIONS & FUTURE WORK

While static analysis tools have demonstrated their value in numerous previous works [7, 16], it is unreasonable to expect that any tool will ever be flawless and that no static analysis tool is perfect and generally inherently contains limitations [5]. Although Stowaway is a powerful static analysis tool which has been used in previous research [7, 16], it does suffer from drawbacks. Stowaway's own authors state that the tool only achieves 85% code coverage [7], so the under & over-permissions reported by this tool are imperfect. Additionally, any reported vulnerabilities or defects by a static analysis tool should be deemed as *possible* vulnerabilities or defects, not necessarily actual ones. There is also the possibility of issues in the reverse engineering process of the apps, but we are confident in the process due to its demonstrated effectiveness and accuracy in existing research [12, 13], and due to our manual verification of some of the apps.

We only analyzed apps from Google Play and not other sources such as the Amazon Appstore[10] or F-Droid, which would have led to more varied application origins. However, we feel the diversity

[9] https://github.com/DroidDarwin
[10] http://www.amazon.com/mobile-apps/b?node=2350149011

of our apps was already quite robust since we collected 70,785 applications from 41 categories. We also only examined free applications in our research due to cost constraints. Thus, the measurements comparison of apps is not representative of the entire Google Play market. Our results only apply to an evaluation of free apps, not paid apps.

We analyzed 1,420 malicious Android apps in a variety of areas. Although this represents a substantial number of malicious apps, it obviously represents only a minor portion of all Android malware. Attaining Android malware samples is a difficult task since they are often difficult to identify, and are often not widely publicized or shared for a variety of reasons, including the fear that this may only lead to them spreading.

While we have demonstrated profound results through the collection of over 70,000 Android apps, future work may be conducted in several key areas. An interesting topic would be to analyze how apps evolve over time through the examination of numerous released versions of the same app, and not through the aggregate values of apps as we have done. Android 6.0 received a massive permissions overhaul and work may be done to see how this new release affects how developers use permissions. Naturally more apps can always be examined, and with new apps being released on a daily basis the process is never-ending.

8 CONCLUSION

We described our collection and analysis process used to examine over 70,000 malicious and benign Android apps from several security and quality perspectives. Some of the primary discoveries include significant differences in permissions usage across app categories and malware, a comparison of the top over-permissions in benign and malicious apps, the discovery of only a weak correlation between an app's quality and security, a comparison benign and malicious apps in terms of several security based metrics, and that both benign and malicious apps are growing on an annual basis. Our work has laid the foundation for important future research in analyzing both quality and security issues in Android apps, along with providing an invaluable dataset for other researchers to use in their own studies.

REFERENCES

[1] Contagio mobile. http://contagiominidump.blogspot.com.

[2] Number of apps available in leading app stores as of november 2015. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

[3] T. Armendariz. Fortinet's fortiguard labs reports 96.5% of all mobile malware tracked is android based, symbian is distant second at 3.45blackberry, palmos, and windows together represent less than 1 http://www.fortinet.com/press_releases/2014/fortiguard-quarterly-labs-reports.html.

[4] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 15–26. ACM, 2011.

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

[6] David and M. Kleidermacher. Using coding standards to improve software quality and security. http://www.embedded.com/design/safety-and-security/4418986/Using-coding-standards-to-improve-software-quality-and-security.

[7] 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.

[8] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

[9] Y. Feng, S. Anand, I. Dillig, and A. Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 576–587, New York, NY, USA, 2014. ACM.

[10] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WISEC '12, pages 101–112, New York, NY, USA, 2012. ACM.

[11] D. Kleidermacher and M. Kleidermacher. *Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development.* Newnes, Newton, MA, USA, 1st edition, 2012.

[12] D. E. Krutz, M. Mirakhorli, M. S. A., A. Ruiz, J. Peterson, A. Filipski, and J. Smith. A dataset of open-source android applications. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. ACM, 2015.

[13] S.-H. Lee and S.-H. Jin. Warning system for detecting malicious applications on android system. In *International Journal of Computer and Communication Engineering*, 2013.

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

[15] 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.

[16] 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.

[17] R. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, Inc., New York, NY, USA, 7 edition, 2010.

[18] V. Rastogi, Y. Chen, and X. Jiang. Droidchameleon: Evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 329–334, New York, NY, USA, 2013. ACM.

[19] A. Reina, A. Fattori, and L. Cavallaro. A system call-centric analysis and stimulation technique to automatically reconstruct android malware behaviors. *EuroSec, April*, 2013.

[20] N. Rutar, C. B. Almazan, and J. S. Foster. A comparison of bug finding tools for java. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 245–256. IEEE, 2004.

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

[22] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[23] 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, 2013.

[24] M.-Y. Su and W.-C. Chang. Permission-based malware detection mechanisms for smart phones. In *Information Networking (ICOIN), 2014 International Conference on*, pages 449–452, Feb 2014.

[25] K. A. Talha, D. I. Alper, and C. Aydin. Apk auditor: Permission-based android malware detection system. *Digital Investigation*, 13:1–14, 2015.

[26] T. Vidas, N. Christin, and L. F. Cranor. Curbing android permission creep. In *In W2SP*, 2011.

[27] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. *SIGMETRICS Perform. Eval. Rev.*, 42(1):221–233, June 2014.

[28] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69, Aug 2012.

[29] S. Yerima, S. Sezer, and G. McWilliams. Analysis of bayesian classification-based approaches for android malware detection. *Information Security, IET*, 8(1):25–36, Jan 2014.

[30] Y. Zhou and X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109, Washington, DC, USA, 2012. IEEE Computer Society.