Darwin: A Static Analysis Dataset of Malicious and Benign Android Apps

Nuthan Munaiah, Casey Klimkowsky, Shannon McRae, Adam Blaine,
Samuel A. Malachowsky, Cesar Perez, and Daniel E. Krutz
{nm6061,cek3403,smt9020,amb8805,samvse,cap7879,dxkvse}@rit.edu
Rochester Institute of Technology, Rochester, NY, USA

ABSTRACT

The Android platform comprises the vast majority of the mobile market. Unfortunately, Android apps are not immune to issues that plague conventional software including security vulnerabilities, bugs, and permission-based problems. In order to address these issues, we need a better understanding of the apps we use everyday. Over the course of more than a year, we collected and reverse engineered 64,868 Android apps from the Google Play store as well as 1,669 malware samples collected from several sources. Each app was analyzed using several static analysis tools to collect a variety of quality and security related information. The apps spanned 41 different categories, and constituted a total of 576,174 permissions, 39,780 unique signing keys and 125,159 over-permissions. We present the dataset of these apps, and a sample set of analytics, on our website—`http://darwin.rit.edu`—with the option of downloading the dataset for offline evaluation.

CCS Concepts

•**Human-centered computing** → *Mobile computing;*

Keywords

mobile computing, software quality, mobile security

1. INTRODUCTION

Android applications (apps) suffer from the same problems that plague conventional software: security vulnerabilities, defects, adherence to coding standards, and numerous other issues. Additionally, malicious developers create malware for the purpose of exploiting users or devices for profit. There are innumerable areas of Android research which examine both malicious and benign apps in a variety of quality, security, and evolutionary contexts. However, collecting this information is a difficult or time-consuming endeavor. To overcome this limitation, we have created a statistical dataset to assist researchers in examining Android apps in a variety of security and quality contexts. *Our goal was to create an information set which could be used in a wide variety of mobile research including security, quality and evolution analysis.* Our dataset contains information about 64,868 reverse engineered apps from Google Play and 1,669 apps collected from known malware sources.

In this paper we describe (i) the data collection and analysis process, (ii) an easy to use web application to share project information and provide access to raw data in our SQLite database, and (iii) the groundwork for future research by exploring a variety of vital areas of future research. Due to usage agreements, we are unable to publicly share the APK files of the apps we examined. If researchers would like access to the APK files, they should contact the authors of the paper.

WAMA'16, November 14, 2016, Seattle, WA, USA
© 2016 ACM. 978-1-4503-4398-5/16/11...$15.00
http://dx.doi.org/10.1145/2993259.2993264

2. RELATED WORK

There have been many studies which analyzed mobile apps on a large scale. Sarma et al. [10] evaluated several large datasets, including one with 158,062 Android apps 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 presented in this paper. Viennot *et al.* [11] developed a tool called PlayDrone which they used to examine the source code of over 1,100,000 free Android apps. Unfortunately, they largely used 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 quality, security vulnerability levels, and over-permissions, which were a part of our analysis. Felt et al. described some common developer errors found using their tool, Stowaway, including confusing permission names, the use of deprecated permissions, and errors due to copying and pasting existing code [3]. Krutz et al. [5] created a public dataset of over 1,100 Android apps from the F-Droid[1] repository and 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.

There are several other websites which gather metrics about Android apps. AppAnnie[2] and Koodous[3], collect Android apps and perform 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.

[1] `https://f-droid.org/`
[2] `https://www.appannie.com`
[3] `https://koodous.com/`

VirusTotal[4] is a service which is able to analyze files and URLs for viruses. Darwin does not analyze Android apps to determine if they are viruses, it checks for vulnerabilities in apps whose assumed intention is to be non-malicious. Several works have created malware repositories containing malicious application (apk) files for download, including the Contagio Mobile Mini Dump[5] and the Malware Genome Project[6].

3. DATASET CONSTRUCTION

Our dataset was built by collecting apps and analyzing them using several well-known security and quality static analysis tools. Our paramount process concern was accuracy; problems with even a small percentage of the apps or at any stage in the collection or reverse engineering process could have had devastating effect on the accuracy of our work. To achieve an appropriate level of quality, we relied on unit testing, manual verification, and frequent test runs. Because of this, the optimization and testing phases took the largest share of development time for the project.

3.1 Collect APK files

Our initial step was to collect app meta data information and APK files from Google Play. We constructed a custom-built data collection system, which uses *Scrapy*[7] as a foundation. Our goal was to collect a diverse set of apps, so apps were randomly selected and collected by Scrapy from Google Play and did not target specific apps, or specific versions of apps. Our data collection process took place over the course of more than a year.

Malware samples were collected from the Contagio Mobile Mini Dump and the Malware Genome Project. The Contagio Mobile Mini Dump has been collecting malware affecting many platforms, including Android, for several years. In this study, 160 malware examples from the Contagio Mobile mini dump were used. The Malware Genome Project began in 2010 and has collected a substantial number of mobile malware. For our analysis, we used examples from 49 malware families.

3.2 Static Analysis

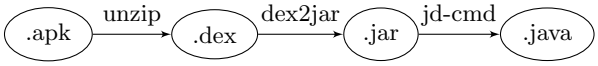

Figure 1: APK Decompilation Process

We employed a previously established process [6] to reverse engineer the collected apps. The decompilation process is shown in Figure 1. The APK file of each app was first unzipped and then two open source tools (described below) were used to retrieve the source code.

- **dex2jar[8]:** Converts .dex file into a .jar file. The `jar` command is then used to extract .class files from .jar.
- **jd-cmd[9]:** Converts .class files to .java source files.

The Java source files were then subject to static analysis to collect a variety of security and quality metrics. The static analysis tools used were:

Stowaway [3]: The *principle of least privilege* states that each app should request the minimum number of permissions that it needs to function. Requesting more permissions than required creates unnecessary security vulnerabilities [9]. Android operates under a permissions-based system where apps must be granted specific functionality before they may be used. Some of these include access to the camera, contacts, microphone, and location data.

Over-permissions are considered security risks, and under-permissions are considered quality risks. The primary difference between requested permissions and over-permissions is the consideration of whether the app actually needs them or not. Under- and over-permissions of an app, reported by Stowaway, were recorded in our data. Modifications were made to the existing version of Stowaway to accommodate our process and stay current with updated Android permissions.

AndroRisk[10]: AndroRisk reports the risk indicator of an application concerning potential malware and determines the security risk level of an application by examining several criteria such as the presence of permissions which are deemed to be more dangerous (i.e. access to the internet, SMS messages, or payment systems) and the presence of generally more dangerous functionality in the app (i.e. a shared library, use of cryptographic functions, the reflection API). We recorded the reported risk level for each APK file.

CheckStyle[11]: This tool 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 a trite aspect to measure, compliance to coding standards in software development can enhance team communication, reduce program errors, and improve code quality [7].

Jlint[12]: This examines Java code to find bugs, inconsistencies, and synchronization problems by conducting a data flow analysis and building lock graphs. We recorded the total number of discovered bugs per app. This tool was selected over FindBugs[13] for its ability to analyze the applications much faster while providing accurate results [8].

APKParser[14]: A tool designed to read various information from Android APK files including the version, intents, and permissions. We used the output from this tool to determine the application version, minimum SDK, and target SDK.

Keytool[15]: A key and certificate management utility which we use to determine various signing information including owner and issuer information and the MD5, SHA1, and SHA256 signing keys.

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 and AndroRisk were able to analyze the raw APK files, while CheckStyle, Jlint, and Nicad required the APK files to be decompiled. All results were saved to a SQLite database, which is publicly available on the project website.

[4] `https://www.virustotal.com/`
[5] `http://contagiominidump.blogspot.com`
[6] `http://www.malgenomeproject.org/`
[7] `http://scrapy.org`
[8] `https://code.google.com/p/dex2jar/`
[9] `https://github.com/kwart/jd-cmd`
[10] `https://code.google.com/p/androguard`
[11] `http://checkstyle.sourceforge.net`
[12] `http://jlint.sourceforge.net`
[13] `http://findbugs.sourceforge.net`
[14] `https://github.com/joakime/android-apk-parser`
[15] `https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html`

3.3 Challenges

There were several significant challenges which had to be overcome in our collection and analysis process. Although using Scrapy was immensely beneficial, collecting such a large number of APK files for over a year was not a trivial process. There were several cases where our collection tool needed to be modified due to alterations to Google Play. These required changes would occur intermittently and would often disrupt our collection process.

One of the biggest challenges we needed to overcome was analysis time. Although we optimized our scripts to make the analysis as fast as possible, the reverse engineering and analysis process for an app took from 2 to 10 minutes to complete. This meant that we were limited in the amount of apps that we could analyze on a daily basis. Frustratingly, for an reasons unknown, we were unable to reverse engineer a small number (less than 0.5%) of collected apps. Our hypothesis is that a library or obfuscated portion of code created problems for our analysis, but unfortunately we were unable to determine a root cause for the problem.

4. ANALYTICS AND DATA SHARING

We have shared all of our project results on our project website: `http://darwin.rit.edu`. Our goal is to provide a robust, and easy to use mechanism for other researchers and interested parties. Android users may search for particular apps on the website to view a variety of quality and security related metrics (as well as comparing different versions). A researcher may utilize the more advanced features of the website and download the entire dataset for their own analysis.

All data is available in three SQLite databases—one for Google Play and one for each of the two malware sources. A database schema is provided on the project website to assist others in understanding our dataset. Unfortunately we could not make the .apk files collected from Google Play available due to both size restrictions (the total collected APK files exceeded 680 GB) and possible copyright infringement. Additionally, we could not make the malware available due to usage agreements.

4.1 Exploring the Dataset

Table 1 provides some high-level statistics of the dataset including the total number of categories, unique signing keys, total permissions, total under-permissions, and total over-permissions.

Table 1: Overview of Collected Data

Total	Google Play	Malware
Apps	64,868	1,669
Unique Category	41	n/a
Unique Signing Keys	39,592	188
Requested Permissions	558,216	17,958
Intents	232,645	3,331
Over-Permissions	125,159	7,288
Under-Permissions	228,475	2,222

While our primary goal was not to target specific versions of apps, we did collect numerous versions of the same app. Table 2 displays the number of analyzed apps and their version counts. Analyzing multiple app versions can be extremely useful in understanding the evolution of quality and security attributes.

Table 2: Collected App Version Counts

Version	2+	3+	4+	5+	10+
Count	6,546	1,853	823	421	41

Using a custom built analysis tool, we collected each app's requested permissions from the reverse engineered *AndroidManifest.xml* file. Table 3 displays the five most requested permissions from Google Play apps, along with the number collected from malware.

Table 3: Top Permission Counts

Permission	Google Play	Malware
`INTERNET`	73,484	943
`ACCESS_NETWORK_STATE`	62,494	821
`WRITE_EXTERNAL_STORAGE`	43,904	618
`READ_PHONE_STATE`	31,345	890
`WAKE_LOCK`	26,144	316

We collected the number of apps signed using the same developer key for both the Google Play and malware apps. These values are shown in Table 4. A signing key is used to verify the origin of the app; only the developer holds the proper key used to sign a created app.

Table 4: Apps Signed Using Same MD5 Key

App Count	Google Play	Malware
10+	12,981	576
25+	7,703	269
50+	5,608	66
100+	3,992	-
250+	2,916	-
500+	2,629	-

4.2 Analytical Results

The project website contains pre-built reports and information pages which may be used to view aggregated or individual app data. This includes some pre-built reports in .csv format, some of which include: all reported over-permissions for each app, requested permissions for each app, and all reported static analysis metrics. The site also contains several pre-built graphical representations of the data.

Users may explore our data set in two ways: (1) search for the results of individual apps as shown in Figure 2 or (2) explore the data by writing their own queries against the dataset using an interface on the website as shown in Figure 3. Users may also choose to download our entire raw dataset as a SQLite file.

4.3 Enabled Research

This dataset provides a wide range of benefits for Android users, researchers, and app developers and we present some potential usage scenarios for dataset.

Facilitate research on Google Play apps. A goal of our work is to allow others to extend upon our research. Since we collected a variety of information from Google Play and from static analysis tools, comparisons could be done against the user ratings of the apps from a variety of quality and security-related metrics. Permissions data (558,216 total collected permissions) could be used to provide insight in numerous areas including the tendencies of permissions use and the popularity of various permissions.

We collected several forms of signing information about each app which could also prove useful for future researchers.

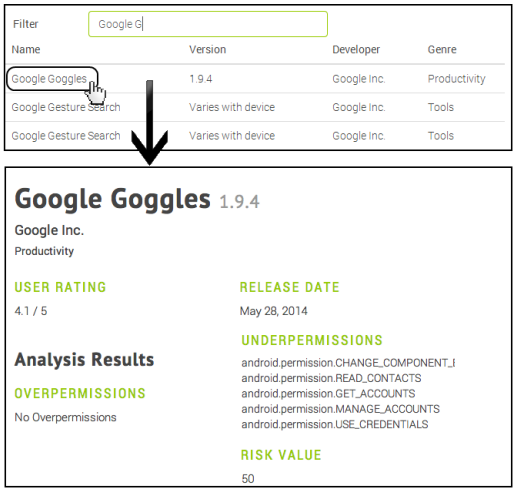

Figure 2: Example App Search

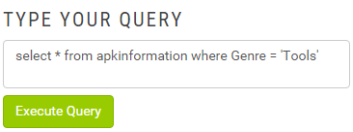

Figure 3: Webpage Search Query

Collecting and validating previous app versions is difficult, and while we are unable to provide the actual .apk file, we are able to provide the signing key—which may be used to verify the authenticity of other apps. Although developers often use different app and developer names, one way to identify the creator of an app is through the MD5 key. Individual developers may use the same MD5 value to sign multiple apps, which is a reliable method of identifying the actual creator of an app. Additionally, we collected other signing information about the apps including various owner and issuer values and the SHA1 and SHA256 values.

Facilitate research on Malware. To better detect and defend against malware, we need to understand more about it: how it is created, evolves, and its common characteristics. Including data from benign and malicious apps will enable researchers to study these apps in a variety of ways including how malware is evolving, signing information, app quality, and requested permissions.

5. LIMITATIONS AND FUTURE WORK

There are several limitations to our dataset and possible improvements. We only examined free apps, thus excluding a significant population (paid apps) in our analysis. Although our reverse engineering process has been demonstrated to be highly effective in previous research [2], all such processes contain possible flaws which could lead to imperfections in the analysis process. We used Stowaway to analyze apps for permission misuse, however new tools such as PScout [1] could have been used to conduct this analysis.

Along with the rating of an app, user reviews are an effective way of measuring a user's perception of an app [4]. Future work may be done to collect the associated user reviews and include these results in the dataset as well. Our goal was to collect a wide variety of apps and did not target specific apps for collection. Future work may be done to target specific apps to ensure that numerous versions of each app are collected. We did not target specific apps or versions for this study since our goal was to collect a diverse set of apps as possible. Analyzing numerous versions of the same app can provide valuable insight into the evolution of the app from a security, and quality perspective.

6. CONCLUSION

We created a valuable, publicly accessible dataset by collecting and analyzing 64,868 apps from Google Play and various malware sources. This dataset is beneficial to developers, researchers, and Android users in not only understanding existing apps, but in how apps are developed, evolve, and are maintained. The collected data is publicly available on the project website: `http://darwin.rit.edu`.

7. REFERENCES

[1] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. PScout: Analyzing the Android Permission Specification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 217–228. ACM, 2012.

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

[3] 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.

[4] H. Khalid, M. Nagappan, and A. E. Hassan. Examining the Relationship between FindBugs Warnings and App Ratings. *IEEE Software*, 33(4):34–39, July 2016.

[5] 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 *Proceedings of the 12th Working Conference on Mining Software Repositories*. ACM, 2015.

[6] 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.

[7] X. Li and C. Prasad. Effectively Teaching Coding Standards in Programming. In *Proceedings of the 6th Conference on Information Technology Education*, pages 239–244, New York, NY, USA, 2005. ACM.

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

[9] J. H. Saltzer and M. D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[10] 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.

[11] N. Viennot, E. Garcia, and J. Nieh. A Measurement Study of Google Play. *SIGMETRICS Perform. Eval. Rev.*, 42(1):221–233, June 2014.