An Oracle of Repackaged Android Applications

Daniel E. Krutz, Justin Peterson, and XXXXX
Rochester Institute of Technology
1 Lomb Memorial Drive
Rochester, NY 14623, USA
{dxkvse, jmp3833, XXXXX}@rit.edu

ABSTRACT

The flexibility offered by the Android platform has contributed to it being the most popular in the world. Unfortunately, malevolent actors often reverse engineer legitimate Android applications (apps), insert malicious code, repackage them, and then upload them for an unknowing user to download. Studies have shown that approximately 5-13% of apps on third party markets have been repackaged from the official Android market.

A wide range of powerful repackage detection tools and techniques have been developed. Unfortunately, when evaluating these techniques they are often compared against custom data sets of repackaged and non-repackaged apps, which is not only time consuming redundant work, but may be inaccurate as well due to any biases in created their oracle.

We present an oracle of 18[check] repackaged Android apps which was created through the combination of existing repackaged apps and self created repackage apps. Future researchers may use this oracle when evaluating their own Android repackage techniques and tools. Our complete data set and further information about the project may be found on the project website: http://www.se.rit.edu/~dkrutz/RAA/[create site]

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Maintaining software, Reusability, Software Evolution

Keywords

Mobile Application Development, Android Applications, Software Engineering [update categories, keywords and terms]

1. INTRODUCTION

Android has grown to be the most popular mobile platform in the world today, largely to the flexibility and openness that is offered to both developers and users. Not only can Android run on a wide range of devices, but its apps may be attained with relative ease from a variety of sources other than the GooglePlay store. This is in contrast to its primary competitor, Apple which forces all non-jailbroken iPhones to download apps only through the Apple App store. This ability for Android users to download apps from numerous sources is a double edged sword for its users. While they are given this extra freedom, the apps which come from these 3rd party locations often contain viruses, or are repackaged apps. Zhou et al. [4] found that 5-13% of all apps in third party markets are repackaged versions which were initially attained from GooglePlay.

Android application (.apk) files may be reverse engineered with relative ease using tools such as dex2jar[1] and jd-cmd[2] allowing an actor to slightly alter the source code with malicious information, repackage it, and upload it to the website under the guise of it being the true version of the app. An unsuspecting user will then install the app, and likely not recognize that it is a fraudulent copy since it continues to act as the legitimate version and they are unable to see any malicious functionality.

A wide variety of tools have been created to detect these repackaged applications. Detecting these fraudulent versions can be a problematic task since finding such small alterations can be difficult, and legitimate versions of an app are frequently released with only minor alterations making it tough to separate legitimate, and non-legitimate apps from one another. [cite]

When evaluating new or existing Android repackage detection techniques or tools, a custom data set is often used. Creating such a data set may not only be time consuming, but inaccurate as well. Unfortunately, a bias often exists between the tool being evaluated and the data set it is being evaluated against. The data set may be partially created using the evaluated technique, or project authors may skew the results to make their technique look more favorable then at is.

In order to assist with the evaluation of Android repackage detection techniques, we have created a public dataset comprised of two main parts: A compilation of existing repackaged applications taken existing malware sites, and one we created by reverse engineering existing apps and then

[1]https://code.google.com/p/dex2jar/
[2]https://github.com/kwart/jd-cmd

repackaging them with malicious information. All modified apk files and a complete listing of malicious information may be found on the project website[3]. To our knowledge, no other attempts have been made to create such an oracle.

A secondary benefit of our work is a comparison and evaluation of several leading repackage detection tools. Our complete dataset and further information about the project may be found on the project website: http://www.se.rit.edu/~dkrutz/RAA/[create site]

2. REPACKAGED ANDROID APPS

There are generally considered two types of Android app repackaging [3]. The first is when the attacker repackages an app under their own name and changes things such as advertisements for monetary gain. The second type is where attackers alter an existing app, inserting dangerous source code which can perform malicious activities such as steal user information, or send unwanted SMS messages. Zhou and Jiang [5] found that 86% of 1260 malware samples were repackaged from non-malicious apps.

The most common strategy used by malicious developers to distribute segments of mobile malware is to add infected code to an already existing reputable Android application in a process known as repackaging. This process is performed by a developer to break down an Android executable file into either assembly language or java code segments. The malware is then inserted and the application is then re-compiled and assembled into an Android executable to upload to an Android application market. The repackaging process allows a developer to trick end users to download infected applications since the users believe the application comes from a trusted source.

There are many common tools that can be used in the repackaging process. The most commonly used tool is dex2jar which can break down apk files to both Java jarfiles and dalvik bytecode in the form of .smali assembler instructions. The .smali format compiles down to dalvik bytecode which is executed on the Android device. Once the applications are broken down, a developer can either use a tool such as jd-gui[4] to open the Java jar file and modify source code, or they may edit the assembly language directly in order to introduce exploits into application memory. The same tool can then be used to repackage the jar file into an Android executable which matches the original, and the infected app can then be uploaded to application markets.

The first line of defense to protect users is the "signature" that is applied to each app when it is uploaded to application markets. This signature, composed of a hashed certificate, is a stored value that is unique to each application developer. A simple check is done to determine if this signature is compromised through the repackaging process whenever an application is uploaded to more reputable Android application markets. If an infected app is able to bypass this restriction, then it is up to a suite of malware detection tools to discover whether or not an application has been repackaged. Google's "Bouncer" tool has reduced the amount of malware infected applications in the Google Play store by up to 40%[cite], but is not effective in all situations and is not an open source technology.

[3]http://www.se.rit.edu/~dkrutz/RAA/
[4]http://jd.benow.ca/

3. DATASET CONSTRUCTION

We created our data set using two primary sources: One of existing malware taken from other sources and one with self created malware. We felt that this diversity was important for ensuring a dataset that would not only be realistic, but also cover any many different types of repackaged apps as possible.

3.1 Existing Repackaged Apps

We selected 10 repackaged apps from the DroidBench [5]. Although there are 50 samples, we decided to use 10 since some of the samples in the original test suite were redundant and tested the same strains of malware[JustinL fill in why else].

3.2 Self-Created Malware

We created XXX examples of self created repackaged Android apps for several reasons.This data differs from DroidBench since we used actual malware strains from the wild and implemented them into our examples of repackaged Android apps. This differs from Droidbench since their examples were created by the creators of Droidbench and wrapped into the apk. The other sample set from the Contagio mobile malware dump[6] included apps that were more like those found in the wild, with strains of malware hidden in popular applications that have been disguised and repackaged. [confusing? Clarify this?]

4. DATA

4.1 Data Format

An example of the format of our data is shown in Table ??.

4.2 Website

5. TOOL COMPARISION

While not a primary objective of this study, we compared three leading repackage detection tools against each another using our oracle. The tools we evaluated were AnaDroid [2], FlowDroid[7], and Dexter[8]. The results of this analysis is shown in Table 2, while more data from the analysis may be found on our project website. This report is only a preliminary examination of the tools demonstrating the benefits of our oracle and not should be considered an in-depth analysis of the tools. We evaluated the tools against one another measuring their analysis time, precision, recall, and accuracy.

The raw data generated by each tool as well as the apk files that were used in the analysis can be found on the project website, along with complete results.

[Make sure these are in line with what existing research says]

6. THREATS TO VALIDITY

[5]http://sseblog.ec-spride.de/tools/droidbench/
[6]http://contagiominidump.blogspot.com/
[7]http://sseblog.ec-spride.de/tools/flowdroid/
[8]http://dexter.dexlabs.org/

Table 1: Format of Example Results

APK Name	Description	FileName	Location	Start Line	End Line	# of java files
APK #1	SMS Exploit	File1	onCreate()	25	27	1
APK #2	System monitoring spyware	File2	InstallTask()	43	48	1
APK #3	Undercover SMS sending tool	File3	getIMEI()	18	26	3

Table 2: Comparison of Detection Tools

	AnaDroid	FlowDroid	Dexter
Run Time(s)	172.68	10.69	52.05
Precision	1	0.82	0.82
Recall	0.43	0.76	0.38
Accuracy	1	0.94	0.94

7. RELATED WORK

The works that are most similar to this study include those by Huang et al. [1] and Zhou and Wu [4]. Huang's study provides a potential solution to detect how easily an Android application could have portions of its Dalvik bytecode obfuscated. If an application's components may be easily obfuscated, then it is straightforward for a developer to repackage the application and introduce a strain of malware without being detected. This study suggests that through simple code obfuscation algorithms, malicious applications may be able to avoid detection and that the use of dynamic analysis techniques although costly may be the optimal solution to detect the greatest number of Android malware strains. This study analyzes fuzzy hashing and data dependency graph detection techniques and suggests a generalized framework to determine an Android application's resilience to obfuscation, rather than a tool's effectiveness at detecting a repackaged application. Zhou and Wu's publication relies on fuzzy hashing of application signatures of clean and potentially infected applications to determine if an application is repackaged.

There are several other well known Android malware repositories that contain large amounts of malware samples. DroidBench, the Contagio mobile malware mini dump, and the Malware Genome Project[9] are just a few, and have been used in a large number of publications. Our benchmark differs in that it only contains repackaged Android applications, while the others all contain a mixture of purely malicious and repackaged applications. In many cases, the malicious application examples haven't been repackaged, and are simply repackaged applications. **[Justin: Can you verify this?]** This differs from our oracle which contains only repackaged applications. **[check this.]**

8. CONCLUSION

We have created a public oracle that may be used to evaluate new and existing techniques for detecting repackaged Android applications. We also used our oracle to conduct an initial evaluation on three leading detection tools, AnaDroid, FlowDroid, and Dexter comparing their run time, precision, recall, accuracy and rate of discovery. Our complete dataset and further information about the project may be found on the project website: http://www.se.rit.edu/~dkrutz/RAA/

[9]http://www.malgenomeproject.org/

9. REFERENCES

[1] H. Huang, S. Zhu, P. Liu, and D. Wu. A framework for evaluating mobile app repackaging detection algorithms. In *Trust and Trustworthy Computing*, pages 169–186. Springer, 2013.

[2] S. Liang, M. Might, and D. V. Horn. Anadroid: Malware analysis of android with user-supplied predicates. *CoRR*, abs/1311.4198, 2013.

[3] F. Zhang, H. Huang, S. Zhu, D. Wu, and P. Liu. Viewdroid: Towards obfuscation-resilient mobile application repackaging detection. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless &#38; Mobile Networks*, WiSec '14, pages 25–36, New York, NY, USA, 2014. ACM.

[4] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012.

[5] 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.