International Journal of Computer
(IJC)
ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

Framework for Vetting and Identifying Emulated Android Mobile Apps

Mr. Faustine Moogi Soroncho*[a], Dr. Wilson Cheruiyot[b], Dr. Stephen Kimani[c]

[a]*School of Computer Science and Information Technology*
[b]*Jomo Kenyatta University of Agriculture and Technology*
[a]*Email: msoroncho@gmail.com*
[b]*Email: wilchery68@gmail.com*
[c]*Email: stephenkimani@googlemail.com*

Abstract

Mobile apps emulation is increasing becoming serious threat. Criminals target popular apps that are used to carry sensitive user information like financial usernames and passwords. Criminals can download and repackage these apps with malicious codes which will help in stealing user information or send annoying adverts to user that will benefit the criminals. This paper focuses on XDroid Moss, an enhanced vetting framework to detect the emulated apps.

Keywords: Emulation; Repackaging; Mobile Apps; Similarity Levels; XDroid Moss.

1. Introduction

Mobile application that started as simple games and for communication, have been improved and now other apps have been developed to handle sensitive information such as financial and authentication details including usernames and passwords [1]. It on this back ground that cyber criminals are targeting the mobile apps since they tend to follow where money is [2]. Some of the techniques they are using involves developing an application with the same name as a genuine app or modifying the genuine app and repackaging the app with some adware or malware that are intended to help them gain some money [3]. This can be as simple as advertising for other companies to as stealing some information from your device and selling it or using it maliciously.

---

* Corresponding author.

Android mobiles users have grown tremendously [4]. This has influenced the growth of android mobile application development leading to the availability of many mobile applications on Google Play Store and other App store either for free or at a cost. Google has also made it easier for developers to upload their applications for free to Google Play after paying one time registration fee of just 25$ [5]. This makes it easier for criminals to emulate or repackage mobiles apps that will work on their favor.

According to the recent study [6], there are mainly two motivations for app repackaging. First, dishonest developers may repackage others' apps under their own names or embed different advertisements, and then republish them to the app market to earn monetary profit. Second, malware writers modify a popular app by inserting some malicious payload into the original program. The purpose is to take over mobile devices, steal users' private information, send premium SMS text messages stealthily, or purchase apps without users' awareness. They also leverage the popularity of the original apps to increase the propagation of the malicious one. Both types of repackaging are severe threats to the app markets.

A survey that was done on 50 free applications that were available on google play, showed 77% of the applications had a fake version [2]. Among the fake apps, more than 50% were deemed malicious. According to Symphony Luo [2], fake apps are more likely to be high-risk apps or malware rather than just mere harmless copycats. As of April 2014, of the 890,482 sample fake apps discovered from various sources, 59,185 were detected as aggressive adware and 394,263 were detected as malware.

However, as the commercial motivation grows, nothing prevents plagiarizers and repackagers from using code obfuscation techniques to evade detection [7]. Moreover, since users can download applications from both official market Google Play and third party markets in different countries, the repackaging problem can appear both inter- and intra- market, which increases the scale and challenge for repackaging detection[8].

2. Proposed XDroid MOSS framework

EXtended Droid Moss (XDroid Moss) seeks to identify the original and the emulated app among the duplicate apps. The XDroid MOSS system has three main processes to effectively determine the emulated and the original app. The three processes involves the Extraction of the author ID, Checking for Similarity of the code base, and Comparing the posting date and app size.

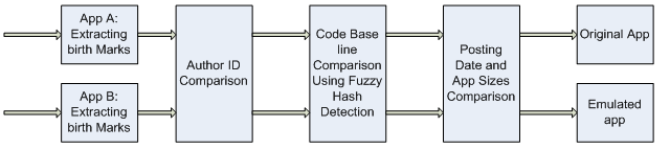

Figure 1: XDroid MOSS Framework

As show in the in, the framework will at the end suggest the emulated app and the original app bases on the posting date and the app sizes.

3. Processing Analysis

XDroid MOSS has four main steps Extraction of Birth Marks, Author ID Comparison, Code Base Line Comparison, and Apps Size and posting date comparison. In this section, we look at how the framework works in details explaining every step.

3.1 Extraction of Birth Mark Features

The first stage is extracting the birth mark features including Author ID, Code Baseline, App size, and Posting date. This is done to the pair of similar apps. These apps will be drawn from both Android official Play Store and other third parties including Amazon.

3.2 Author ID Comparison

When Android apps are developed, they must be signed by the authors. This is done through a process assigning the apps "Signing Keys". Since the author signing keys cannot be leaked [9], it means each app must have unique Signing Keys for each author. Therefore if two similar apps have different Signing Keys then they belong to different author and one of them is likely to be an emulated app. XDroid MOSS extracts the signing keys from the two app and compare them. If they're the same then the app was signed by one author and therefore the like hood repackaging is minimal. But if they are different then, there is like hood of emulated app. To confirm this, the two apps are subjected to the next step, Fuzzy Hashing Based Detection.

3.3 Fuzzy Hashing Based Detection

In this step the code base of the two apps, are subjected to FHBD algorithm to determine if they have any similarities. The first step being sub dividing the code bases for each app into smaller segments and then generating the signature for each segment. This unique signature is used to create the final signature for the app. Finally the final signatures sometimes referred as finger prints are compared and their similarity score recorded. If the two app have an acceptable similarity, then according to Droid MOSS they are considered as repackaged apps [10]. If they are proved to be repacked pair, then the last step involves determine the emulated and the original apps.

3.4 Determining the Emulated and the original app

Since most of the emulated apps are posted after the original apps have gained popularity [9], then the initial posting date of the two emulated apps are likely to be later than the posting the date of the original. That is why in this research, the pair of apps under investigations were subjected to the step of comparing their initial date they were posted to the market. Another last comparison that was done is testing the size of the apps. The main objective of repackaging apps is to add code that will either help advertising or redirecting the user to their sites [2]. The additional code may not increase the size significantly but high chance of an emulated app will have a big size. To mark the app as emulated, XDroid MOSS uses the following algorithm.

```
IF PD_A > PD_B and Size_A > Size_B THEN
     High chance of A being emulated app
IF PD_A > PD_B and Size_A <= Size_B THEN
     Average chances of A being Emulated app
IF PD_B > PD_A and Size_B > Size_A THEN
     High chance of B being emulated app
IF PD_B > PD_A and Size_B <= Size_A THEN
     Average chances of B being emulated app
```

Figure 2: XDroid Algorithm

4. Results

The purpose of this study was to design a framework that will be used to vet and evaluate mobile apps to determine the original and the fake or emulated apps. In this section presents the research result and discussions on the performance of XDroid Moss; the framework that was developed in this study to help identify emulate apps in the online mobile app stores.

Thirty pairs of apps suspected to be emulated were downloaded and each pair was analyzed separately and results recorded. For each pair, the .apk files were extracted and AndroidManifest.xml source file retrieved. The two files for the respective apps were then compare for similarity using the XDroid Moss algorithm. XDroid Moss checked for similarity for each line in the first app against every line of statement in the second app. The results shown in the percentage form 0% to 100% were varying for each pair. In this study, the apps were grouped into four levels depending on their level of similarity as follows.

- **The Lower Level**. This was the lowest level where XDroid Moss returned the similarity percentage that was less than 25% from each pair.
- **The Lower Middle level**. This was the second level for the pair of apps whose results range was between 26% and 50%. These were considered as the mildly emulated apps.
- **The Upper middle level**. This was the third level for the pair of apps whose results range was between 51% and 75%. They were considered as the substantially emulated apps
- **The top level.** This is the highest level and their similarity result is above 75%. They were considered to be highly emulated apps.

Table 1: Tabulated results for the 30 pairs and their levels

Levels	No. of pairs	Percentage	Remarks
Top Level	21	70%	Identified as the emulated apps
Upper middle level	5	16.67%	Substantially emulated
Lower Middle level	3	10%	Mild emulation
Lower Level	1	3.33%	No emulation

From the table above, XDroid Moss labelled 70% of the pairs as highly emulated. A clear indication of how big the app emulation problem is. Having proved the existence of app emulation, the next step is to determine the emulated and the original app through the use of size and release date.

4.1 Size and Release Date Comparison

As mentioned in the earlier chapters of this paper, criminals will emulate popular apps that are already existing in the market. This means the emulated app release date is later compared to the date of the original app was released. The criminals also tend to add some lines of malicious codes that will definitely add more bytes making the emulated apps to have a bigger size compared to original app. It is for this reason, the researcher decided to consider the release date and the size to determine the emulated app and original app.

The 29 apps were grouped into two categories, the first app to be released in each pair and the second app that was released later. The next step was to check the percentage of pairs where the first app was smaller than the second app. It was observed that 89.66% of the pairs, the first app was smaller in size compared to the second app. The chat below summarizes this information.

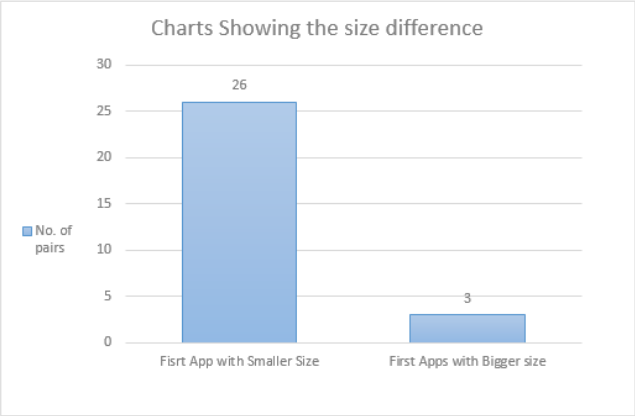

Figure 3: Chat Showing the First –Second App Size difference

From the Figure 4.1 above, the study indicate that the apps that were released after tend to have a bigger size compared to the apps that were released earlier.

4.2 Similarity-Size Difference

At this level, the researcher wanted to check the relationship between similarity levels and size-difference - determining the relationship between the size-difference and the level of emulation. The 29 pairs were analyzed and it was observed that the smaller the size-difference the more similar they are. Meaning that the pairs with the highest emulation tend to have a smaller size- difference as shown in the graph below.

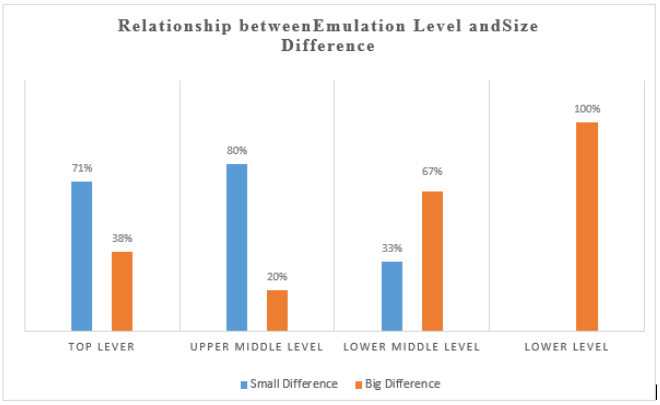

Figure 4: Emulation level vs size difference

From the chart above, it is observed that the majority of top and upper middle emulated apps have a small size-difference compared to lower and lower middle level apps that tend to have a big size-difference.

5. Conclusion

In order to maintain the secure and reliable mobile app usage, there is need to implement frameworks that will help protect users, their information and devices in general. Online stores cannot be left with the responsibility of vetting and controlling the emulations. Users of this apps and the organizations like banks that sponsor the development of this apps, need to be part of the vetting process.

6. Recommendation for further research areas.

Other researchers should focus on how visible it will be to come up with copy rights for app ideas. To find out if it will be possible to have international patent rights that can used to prevent other people from coming up with apps with the same functionalities as the existing apps.

Acknowledgement

A special thanks to Dr. Wilson Cheruiyot, and Dr. Stephen Kimani, my supervisors who were more than generous with their expertise and precious their time, encouraging, and most of all patience throughout the entire process. I wish to also thank Samiah Millicent and Zipporah Kerubo for your encouragement and guidance.

References

[1] Statista. (2015). Number of apps available in leading app stores as of July 2015. Retrieved from http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/ on Jan 2016.

[2] Symphony Luo and Peter Yan. (2014). Fake Apps Feigning Legitimacy. Mobile Threat Research Team

published on A Trend Micro Research Paper

[3] S. Li. Juxtapp. (2012). A scalable system for detecting code reuse among android applications.

[4] Gartner. (2011) Gartner says android to command nearly half of worldwide smartphone operating system market by year-end 2012. Retrieved from http://www.gartner.com/it/page.jsp?id=1622614 on Jan 2016

[5] Play. (2016). Google Play Developer Console retrieved form https://play.google.com/apps/publish/signup on Jan 2016

[6] Heqing Huang, Sencun Zhu, Peng Liu, and Dinghao Wu. (2013). A Framework for Evaluating Mobile App Repackaging Detection Algorithms

[7] You and K. Yim. (2010). Malware obfuscation techniques: A brief survey. In In Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications,

[8] J. Crussell, C. Gibler, and H. Chen. (2012). Attack of the clones: Detecting cloned applications on android markets.

[9] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. (2012). Detecting repackaged smartphone applications in third-party android marketplaces.

[10] Zhihong Zeng, Tianhong Fang, Shishir Shah and Ioannis A. Kakadiaris. (2013). Local Feature Hashing for Face Recognition

[11] Chao Liu, Chen Chen, Jiawei Han. (2006). Detection of Software Plagiarism by Program Dependence Graph Analysis,

[12] Dalvik virtual machine: code and documentation accessed from http://code.google.com/p/dalvik on Dec 2015.

[13] Smali/Baksmali. Accessed from http://code.google.com/p/smali/ on Dec 2015

[14] David J. Eck (2006). Introduction to Programming Using Java Version 5.0,

[15]Wala.(n.d). Welcome to the T.J. Watson Libraries for Analysis (WALA) http://wala.sourceforge.net/wiki/index.php/Main_Page accessed on Jan 2016