Using Concolic Analysis to Detect Repackaged Android Applications

Daniel E. Krutz and Samuel A. Malachowsky
Software Engineering Department
Rochester Institute of Technology
1 Lomb Memorial Drive
Rochester, NY 14623
{dxkvse, samvse}@rit.edu

ABSTRACT

The Android platform has emerged as a market leader largely due to its flexibility, running on a diverse set of hardware and allowing users to freely install applications from a wide variety of sources. Unfortunately, a significant portion of Android applications are repackaged versions of legitimate applications, often containing malware. Excepting small variations in the source code, they often precisely mimic their legitimate counterparts, making detection of these malicious applications very difficult for end users, researchers, and protection systems. *[Sam says: Change "protection systems" to "app stores"?]*

We propose a new technique for discovering repackaged Android applications based on Concolic analysis. This static analysis process will form a powerful Android repackaging detection technique since it only traverses the functional aspects of the application, and is not affected by the syntax of the application's code. In this paper, we demonstrate how concolic analysis can be used to detect repackaged Android applications and lay the foundation for future research. *[Dan says: Really work on this.. Maybe shorten it a bit]*

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; *[Sam says: Are these 1998 or 2012? Security and privacy - Software and application security]*

Keywords

Android, Malware, Repackaged Applications

1. INTRODUCTION

The Android platform has gained widespread popularity, largely due its flexibly, including the ability to download applications (*apps*) from a wide range of locations and the ability to view the source code of the application *[Dan says: find a better benefit]* Unfortunately, this openness often leads to severe security vulnerabilities. Since anyone may extract the source code of the application file (*.apk*), malicious developers often download a legitimate version of an app, reverse engineer the application to view its source code using a simple process with a tool such as dex2jar[1], inject malicious code in the file, and then upload the malicious version of the application for users to unsuspectingly download [5]. While GooglePlay employs various protection techniques to varying degrees of success against these repackaged applications [2], 3rd party application sites such as AppksAPK[2] and F-Droid[3] offer essentially no protection. Previous research shows that about 5%-13% of applications in third party markets have been repackaged [13]. Furthermore, a recent study found 86% of malware samples were re-packaged Android applications, which indicates the formidability of this attack method [14].

Repackaged Android applications are often very difficult for users to discover since they may appear to be functionally equivalent to the legitimate applications, and may be used for years with no sign of malicious actions. Researchers often have a hard time detecting the difference between these applications since they are so functionally equivalent, with often very little variations in the source code between legitimate, and malicious applications. Current techniques for detecting repackaged Android applications include techniques based on dependency graphs [4], fuzzy hash matching [13] and user interface based approaches [12].

In the following work, we propose a new technique for detecting repackaged Android application based on concolic analysis, an approach which we are not aware has been previously attempted. A concolic analysis based approach has the advantage of only examining the functional nature of the, and is not affected by common obfuscation attempts [8], such as altering of naming conventions or other syntactical changes.

2. CONCOLIC ANALYSIS

Concolic analysis combines concrete and symbolic values in order to traverse all possible paths (up to a given length) of an application. Concolic Analysis is not affected by factors such as naming conventions, syntax and comments. Traditionally, concolic analysis has been used for software testing [10], code clone detection [8] and vulnerability detection [3].

[1]https://code.google.com/p/dex2jar/
[2]http://www.appsapk.com/
[3]https://f-droid.org/

SAC'15 April 13-17, 2015, Salamanca, Spain.

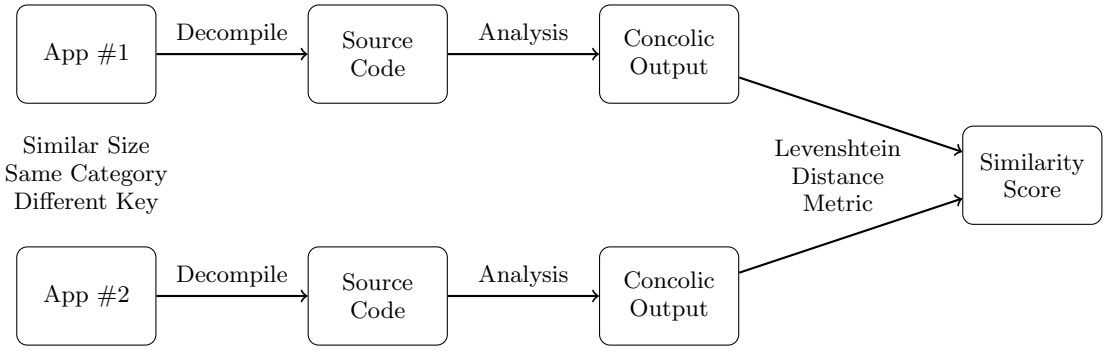

Figure 1: Comparision Process to Determine Repackaged App Candidates

We show example concolic output in Listing **??**, where constant variable types are represented generically by "CONST" while the variable type integer is represented by a generic tag "SYMINT." Though not present above, other variable types are represented in a similar fashion in concolic output such as this. Actual variable names do not appear anywhere in the output and are irrelevant to the concolic analysis technique. We more complete example of concolic analysis and its output may be found on our project website.**[Add this data to the website - also add a link to the site as well?]**

Since concolic analysis is not affected by syntax or comments, identically traversed paths are indications of duplicate functionality, and is therefore functionally equivalent code [7,8]. Very large amounts of duplicated code in Android applications, which have been signed by different developers, is an indication of a potentially repackaged application .

[9]

[Show how similar malware can be detected as well.] [Show a brief example of CA?– I do not feel I am describing CA well enough here]

3. PROPOSED DETECTION TECHNIQUE

[Come up with name?] Our technique will utilize a modified version of Concolic Code Clone Detection (CCCD) which we proposed in a previous work [8]. CCCD detects duplicate functionality by first performing concolic analysis on the target source code and then separating the output at the method level and finally comparing methods against each other using the Levenshtein distance metric. A high similarity score is an indication of redundant functionality, and potentially duplicate or very similar source code. One of the biggest differences between CCCD and our proposed technique is that we will be unable to use CREST [4] as its concolic testing engine since it is only compatible with C. We will use an existing concolic analysis tool such as Java Path Finder(JPF)Java Path Finder (JPF) [11], or will adapt other existing such as that method [1].

Similar to previous research [13], we assume that repackaged applications are the same app type with the legitimate version and do not differ substantially in the size of the application due to the small amount of altered code, and we also assume that signing keys are not leaked so that repacked applications will have not been signed with the same key as legitimate versions. Based on these assumptions, we will only compare applications which have been signed by different keys, have a similar application size*[Dan says: mention specific value]*, and are the same app type. This will assist in significantly limiting the amount of comparisons that need to be conducted.

The proposed concolic analysis comparison process is shown in Figure 1.

[Make this look better] *[Dan says: Is this a good way to show the process?]*

As a proof of concept, we conducted several small analysis of Android applications using a modified version of JPF. We chose JPF since it is highly configurable, and has been extensively used in previous research [6, 11].

4. CONCLUSION & FUTURE WORK

Several barriers need to be overcome in order to adapt a concolic based approach such as this to detecting repackaged Android applications. The first is that Android applications lack a main method and implement parts of the Android SDK API, making it very difficult to run existing concolic analysis tools on Android applications since they often require main methods [1]. In order to address this issue, we will either alter an existing concolic analysis tool such as JPF, or employ an approach similar to that of Anand et al. [1] who use an approach based on concolic testing and event sequences.

The next phase of our research will be to create a more robust detection tool to conduct a more widespread analysis on the capabilities of concolic analysis in detecting repackaged Android applications. The initial step will be to create a concolic analysis based tool to generate the necessary output for comparison. We will likely follow a process presented by Anand *et al.* [1], but are exploring other possibilities as well, such as modifying JPF.

Once the tool has been completed, we will conduct our analysis in several phases. We will first evaluate our tool in a controlled environment comparing known repackaged applications as identified by sources such as Conagio Mobile[5] and the Malware Genome Project[6] to their legitimate counterparts. We will also compare non-repackaged applications in order to ensure that our technique exceeds ac-

[4]https://code.google.com/p/crest/

[5]http://contagiominidump.blogspot.com

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

ceptable levels of precision and recall. We will next examine Android applications from sources such as GooglePlay and AppsAPK to discover repackaged applications which may have been previously undiscovered and compare our findings to previously proposed techniques for detection repackaged applications [4, 12, 13].

[Dan says: Add to this?] [Dan says: Should we mention that name of our tool? - I am not sure what the name will be yet?]

5. REFERENCES

[1] S. Anand, M. Naik, M. J. Harrold, and H. Yang. Automated concolic testing of smartphone apps. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 59:1–59:11, New York, NY, USA, 2012. ACM.

[2] T. Armendariz. Virus and computer safety concerns. http://antivirus.about.com/od/wirelessthreats/a/Is-Google-Play-Safe.htm.

[3] B. Chen, Q. Zeng, and W. Wang. Crashmaker: An improved binary concolic testing tool for vulnerability detection. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1257–1263, New York, NY, USA, 2014. ACM.

[4] K. Chen, P. Liu, and Y. Zhang. Achieving accuracy and scalability simultaneously in detecting application clones on android markets. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 175–186, New York, NY, USA, 2014. ACM.

[5] C. Gibler, H. Chen, R. Stevens, H. Zang, J. Crussell, and H. Choi. Adrob: Examining the landscape and impact of android application plagiarism.

[6] T. Kalibera, P. Parizek, M. Malohlava, and M. Schoeberl. Exhaustive testing of safety critical java. In *Proceedings of the 8th International Workshop on Java Technologies for Real-Time and Embedded Systems*, JTRES '10, pages 164–174, New York, NY, USA, 2010. ACM.

[7] D. E. Krutz and F. J. Mitropoulos. Code clone discovery based on concolic analysis. 2013.

[8] D. E. Krutz and E. Shihab. Cccd: Concolic code clone detection. In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 489–490. IEEE, 2013.

[9] Z. Qin, Z. Yang, Y. Di, Q. Zhang, X. Zhang, and Z. Zhang. Detecting repackaged android applications. In *Computer Engineering and Networking*, pages 1099–1107. Springer, 2014.

[10] K. Sen, D. Marinov, and G. Agha. Cute: A concolic unit testing engine for c. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 263–272, New York, NY, USA, 2005. ACM.

[11] W. Visser, C. S. Păsăreanu, and S. Khurshid. Test input generation with java pathfinder. *SIGSOFT Softw. Eng. Notes*, 29(4):97–107, July 2004.

[12] 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 & Mobile Networks*, WiSec '14, pages 25–36, New York, NY, USA, 2014. ACM.

[13] 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*, CODASPY '12, pages 317–326, New York, NY, USA, 2012. ACM.

[14] 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.