Concolic Threat Analysis: Detecting Repackaged Android Applications

XXX X XXX
xxxxx
xxxx, xx, xxx
xxxxx@xxx.xxx

ABSTRACT

The Android platform has emerged as a market leader largely due to its flexibility, ability to run on a diverse set of hardware, and its flexibility in allowing users to freely install applications from a wide variety of sources. Unfortunately, a significant portion of Android applications (*apps*) 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 app store protection systems.

In this paper, we propose a new process called Concolic Threat Analysis (CTA). This static analysis process will form a powerful Android repackaging detection technique as it only traverses the functional aspects of the application, and is not affected by the syntax of the application's code. Furthermore, we demonstrate how concolic analysis can be used to detect repackaged Android applications (potentially identifying malware threats) and lay the foundation for future research.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection;

Keywords

Android, Malware, Repackaged Applications

1. INTRODUCTION

The Android platform has gained widespread popularity largely due to its flexibly: its open source nature, the ability to download apps from a wide range of sources, and its ability to operate on a wide variety of mobile devices. Unfortunately, this openness can also lead to severe security vulnerabilities. Anyone may extract the source code of the application file (*.apk*), giving malicious developers the ability to download a legitimate version of an app, view its source code using a simple reverse engineering process (or a tool such as dex2jar[1]), inject malicious code, and then upload the visually similar modified version for users to unwittingly download [5]. While GooglePlay, the largest Android app store, employs various protection techniques against these repackaged apps [2], 3rd party app sites such as AppksAPK[2] and F-Droid[3] offer essentially no protection. Previous research shows that about 5%-13% of apps in third party markets have been repackaged [10], and a recent study [11] found that 86% of malware samples were re-packaged Android apps, which is an indication of the formidability of this attack method.

Repackaged Android apps are often very difficult for users to discover, as they may appear to be visually and functionally equivalent to the legitimate app, and may be used for years with no sign of malicious functionality. Researchers often have a hard time detecting the difference between these apps since they are so functionally similar, often with very little source code variation between legitimate and malicious apps. Current techniques for detecting repackaged Android apps include those based on dependency graphs [4], fuzzy hash matching [10], and user interface based approaches [9].

In the following section, we propose a new technique for detecting repackaged Android apps called Concolic Threat Analysis (CTA). A concolic analysis based approach such as this has the advantage of only examining the functional nature of the app and is not affected by common obfuscation attempts, such as altering of naming conventions, insertion of whitespace, or other syntactical changes [7].

2. CONCOLIC THREAT ANALYSIS (CTA)

Concolic analysis combines concrete and symbolic values in order to traverse all possible paths (up to a given length) of an app. Traditionally it has been used for software testing [8], code clone detection [7], and vulnerability recognition [3]. Since concolic analysis is not affected by syntax or comments, identically traversed paths are indications of duplicate functionality, and therefore functionally equivalent code [6, 7]. These traversed paths are expressed in the form of *concolic output* which represents the execution path tree and typically displays the utilized path conditions and representative input variables; the precise nature of the output is dependent on the selected concolic analysis tool. Very large amounts of duplicated code in Android apps which have been signed by different developers is an indication of a potentially repackaged app.

Concolic Threat Analysis (CTA) will utilize a process similar to Concolic Code Clone Detection (CCCD), which was proposed in a previous work [7]. CCCD detects duplicate functionality by first performing concolic analysis on the target source code, then

[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.

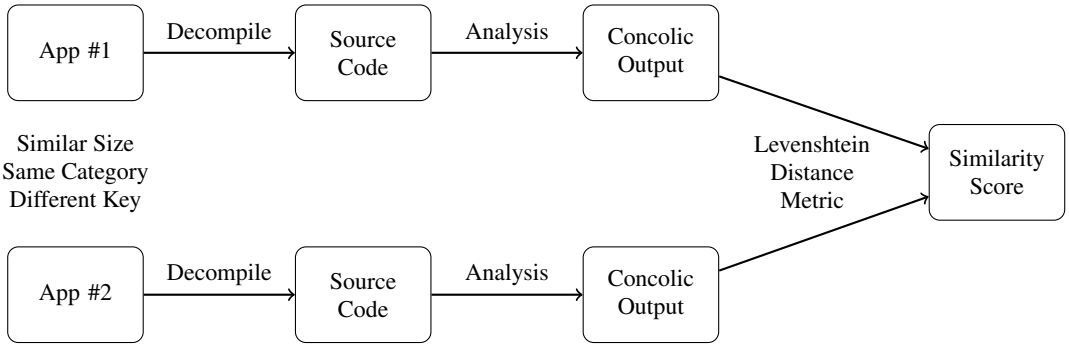


Figure 1: Comparision Process to Determine Repackaged App Candidates

separating the output at the method level, and finally comparing methods against each other using a string comparison metric, likely 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 a concolic analysis engine since it is only compatible with C. We will use another tool such as Java Path Finder (JPF)[5], jCUTE[6], or will adapt other existing techniques such as the concolic analysis method developed by Anand *et al.* [1]. Sample concolic output of JPF and jCUTE are shown in Figure 2.

```
PC#=3
CONST_3>a_1_SYMINT[2]&&
CONST_2<=a_1_SYMINT[2]&&
CONST_1<=a_1_SYMINT[2]
PC#=2
CONST_2>a_1_SYMINT[1]&&
CONST_1<=a_1_SYMINT[1]
```

(a) Java Path Finder

```
jwtc.chess.Move: boolean isOO(int)
jwtc.chess.Move: boolean isOOO(int)
java.lang.StringBuilder: void <init>()
Method setOptionExtraOptions value  read
Method setOptionLogLevel value 3 read 3
Method setOptionPrintOutput value true
```

(b) jCUTE

Figure 2: Example concolic output

Similar to previous research [10], we assume that repackaged apps are of the same type or category as the legitimate version, that the size of the two do not differ substantially (due to the small amount of altered code), and that they have not been signed with the same key (assuming that original keys have not leaked). Only comparing apps of similar size, in the same category, and with different keys will assist in significantly limiting the amount of comparisons that need to be conducted. The process which CTA will use to find repackaged Android applications is shown in is shown in Figure 1.

As a proof of concept, we built a sample application based on public Android code and a known malware example. This application a was a clone of Android Chess[7] injected with a common premium SMS exploit[8], focusing on common activities that were not directly tied to the activity model. Due to the limitations and requirements of jCUTE, the easy-to-implement analysis tool selected for the prototype, Android interactions were mocked out and a wrapper to call specific functions was created. Running jCUTE on this sample app provided text output that demonstrated the flow of calls to naked functions. We manually compared the concolic output produced when running jCUTE on both the malicious and original versions of the application; a sample is shown in Figure 3. While both use concolic analysis, the output of jCUTE and JPF are very different in that JPF shows utilized concrete and symbolic values, and jCUTE only displays the traversed method signatures. Because of this, we will likely choose to use JPF for our final implementation.

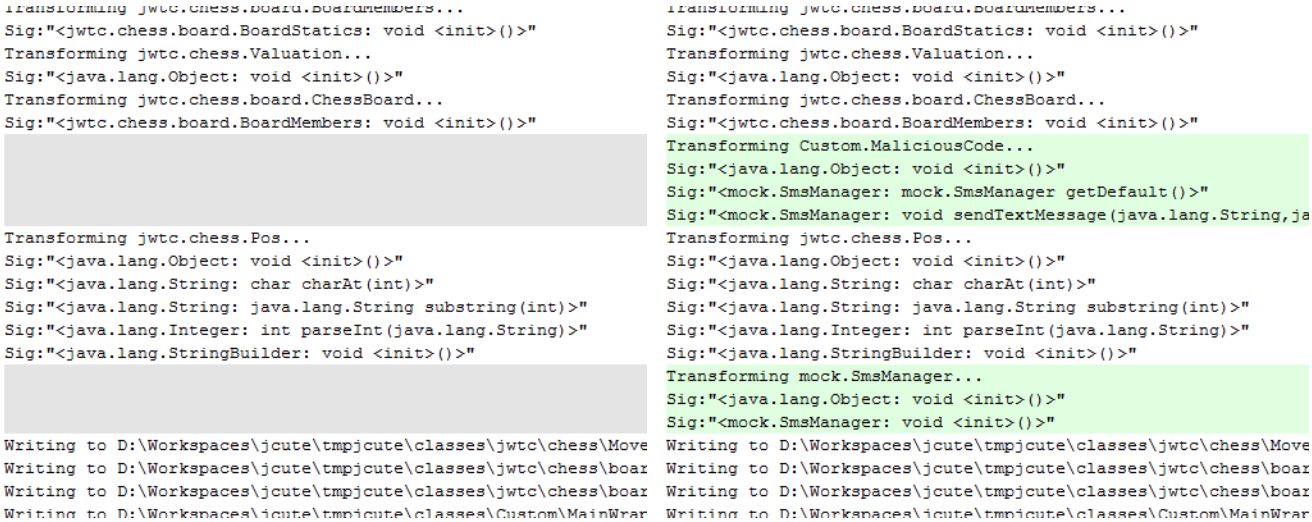
This *diff* of the concolic output demonstrates the substantive differences between the two versions of the application and how a concolic based technique can be used to find variations in Android application versions - instances where an application has been repackaged and could be malicious. Complete results, existing project source code, and further project information may be found on our project website[9].

3. FUTURE WORK & CONCLUSION

Several barriers must be overcome in this approach; the largest is related to the fact that Android applications lack a main method and only implement parts of the Android SDK API. The absence of a main method makes it very difficult to run existing concolic analysis tools, because main methods are often a requirement [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.

Once the tool has been completed, we will conduct our analysis in several phases. First, we will evaluate our tool in a controlled environment comparing known repackaged applications (identified by sources such as Conagio Mobile[10] and the Malware Genome Project[11]) to their legitimate counterparts. Next, we will compare non-repackaged applications in order to ensure that our technique exceeds acceptable levels of precision and recall. Finally, we will

[4]https://code.google.com/p/crest/
[5]http://babelfish.arc.nasa.gov/trac/jpf/wiki
[6]http://osl.cs.illinois.edu/software/jcute/
[7]https://github.com/jcarolus/android-chess
[8]https://www.webroot.com/shared/pdf/Android-Malware-Exposed.pdf
[9]site removed to keep paper anonymous
[10]http://contagiominidump.blogspot.com
[11]http://www.malgenomeproject.org



(a) Original

(b) Repackaged malicious

Figure 3: Diff of concolic output of modified application versions

examine Android applications from sources such as GooglePlay and AppsAPK to discover repackaged applications which may have been previously undiscovered, comparing our findings to previously proposed techniques for detecting repackaged applications [4, 9, 10].

In this work, we outlined the problem of repackaged Android applications and described a possible solution for detecting them using concolic analysis. We also demonstrated a protoytpe which was able to detect malicious code in a small, controlled environment. Finally, we described our future work and discussed some challenges which will need to be overcome in the next phases of development.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.