The Multi-Generation Repackaging Hypothesis

Li Li, Tegawendé F. Bissyandé, Alexandre Bartel, Jacques Klein, Yves Le Traon
Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
{li.li, tegawende.bissyande, alexandre.bartel, jacques.klein, yves.letraon}@uni.lu

***Abstract*—App repackaging is a common threat in the Android ecosystem. To face this threat, the literature now includes a large body of work proposing approaches for identifying repackaged apps. Unfortunately, although most research involves pairwise similarity comparison to distinguish repackaged apps from their "original" counterparts, no work has considered the threat to validity of not being able to discover the true original apps. We provide in this paper preliminary insights of an investigation into the Multi-Generation Repackaging Hypothesis: is the original in a repackaging process the outcome of a previous repackaging process? Leveraging the Androzoo dataset of over 5 million Android apps, we validate this hypothesis in the wild, calling upon the community to take this threat into account in new solutions for repackaged app detection.**

I. Introduction

The process of assembling/disassembling Android app code and resource files into/from an installable package is accessible to all users and developers via public tools. This situation has made Android app packages vulnerable to repackaging attacks where malware writers and software pirates build on existing apps. Indeed, a simple way of constructing a malicious app consists in obtaining the code of an existing (preferably popular) app, injecting in it a malicious payload, and repackaging the whole into a new app that would be advertised as being equivalent to the original one [1]. Early studies on Android malware have thus shown that repackaging is common, with over 80% of some malware datasets being built through repackaging [2].

The repackaging threat is exacerbated by the emergence of alternative markets performing limited sanity checks on the uploaded apps. Fortunately, state-of-the-art works have put a lot of effort in proposing new approaches for detecting repackaged apps. Existing techniques perform pairwise similarity comparisons [3], [4], build on unsupervised learning [5] and supervised learning [6], leverage runtime monitoring [7], or focus on symptoms identification [8], [9], [**?**] to detect repackaged apps. Recently, some researchers [10] have acknowledged the difficulty of identifying the original app from a given repackaging pair. Nevertheless, to the best of our knowledge, no research work has considered the hypothesis that the true original app may not even be the one identified by pairwise similarity comparison.

The multi-generation repackaging (MGRep) hypothesis considers the possibility that an original app identified in a repackaging pair is actually a repackaged app from a prior repackaging generation. Fig. 1 illustrates a theoretical example of a multi-generation repackaging process involving three app cases. As time goes, apps get updated (e.g., v_1 is updated from v_0) with some versions being repackaged into new apps (e.g., from v_1 to v_2 and from v_2 to v_3). In this work, we define a MGRep as a continuous repackaging process where at least two generations (from v_1 to v_2 and then v_3) are involved. Our goal in this work is to check whether the MGRep hypothesis is validated or not.

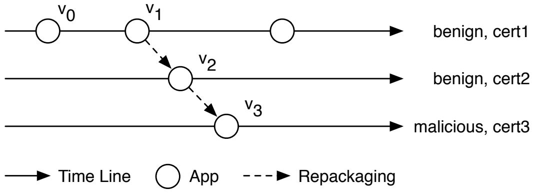

Fig. 1. Example of Multi-Generation Repackaging.

If the hypothesis of MGrep is validated, the state-of-the-art on repackaged app detection should be re-evaluated. For instance, let us consider a supervised learning-based approach for differentiating repackaged apps from non-repackaged ones as an example. Given that app v_2 is simultaneously a repackaged app and an original app in a repackaging pair, its features constitute noise in training a classifier, as illustrated in Fig. 2. The same confusion may also arise when considering approaches based on other techniques such as pairwise similarity computation.

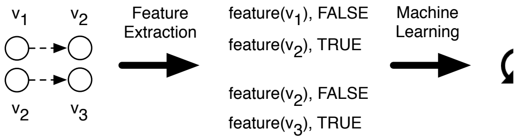

Fig. 2. The Hidden Problem Induced by Multi-Generation Repackaging for Machine Learning based Analysis.

II. Experimental Design

Fig. 3 illustrates the working process of our experiment, which is completed in four steps.

1) **Clustering.** The objective of the first step is to reduce the search space (i.e., the number of apps that must be considered for further analysis) in a simple and efficient manner. Thus, we regroup apps into families by considering app package names as features for clustering. The insight behinds this step is that apps in a same family are more probable to be repackaging each another.

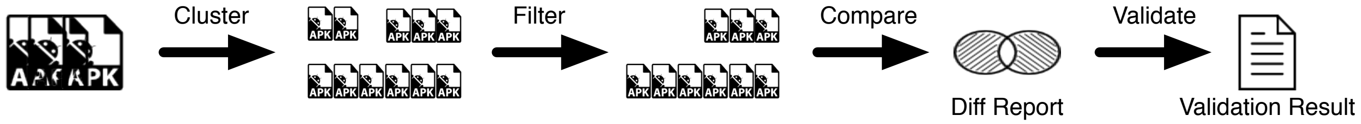

Fig. 3. The working process of our experiment.

2) **Filtering.** In the second step, we further reduce the number of apps to be analyzed by filtering out families which are least appealing for verifying the MGRep hypothesis. To that end, we consider the number of certificates used to signed apps in a family cluster. We do not consider a family if all of its apps signed by less than three certificates[1]. We consider the changes of apps sharing a same certificate as a self-updating process, and thus not relevant to repackaging.
3) **Comparison.** In this step, for each selected family, we perform pairwise similarity analysis for all its apps, attempting to compute the *diff* code between every possible pair of apps. The *diff* code will then be used in the next step for verifying our Multi-Generation Repackaging Hypothesis. Theoretically, given a selected family with n apps inside, we need to perform $\binom{n}{2}$ pairwise comparisons. Considering that the pairwise comparison is relatively expensive, we need to further filter out some pairs that are obviously irrelevant in the context of our experiments. Towards addressing this need, we drop such candidate pairs (let us denote a pair of apps as ($v_1 \to v_2$)) when 1) v_1 is created after v_2. Indeed, it is impossible to repackage an app which does not yet exist. 2) v_1 is apparently more recognized as malicious than v_2, i.e., more anti-virus products flag v_1 as malicious compared to v_2. Indeed, we consider repackaging as a process of injecting extra (probably malicious) payload to further narrow down the number of candidate pairs.
4) **Validation.** Based on all the *diff* code we have obtained, our validation works as follows: Given two overlapping pairs of apps, say (v_1,v_2) and (v_2, v_3), if the Multi-Generation Repackaging Hypothesis is valid for these pairs, the *diff* code of (v_1, v_3) should constrain to Formula 1, where the *diff* code of (v_1, v_3) should equal to the union of the *diff* code of (v_1, v_2) and the *diff* code of (v_2, v_3) (i.e., the so-called *transitive law*). More specially, we compute the *diff* code in the method level for *changed*, *added* and *deleted* methods. We consider the MGRep hypothesis is validated as long as Formula 2 (i.e., a more concrete version of Formula 1) is satisfied.

$$diff(v1, v2) \cup diff(v2, v3) = diff(v1, v3) \tag{1}$$

$$\begin{aligned} changed(v1, v2) \cup changed(v2, v3) &= changed(v1, v3) \\ added(v1, v2) \cup added(v2, v3) &= added(v1, v3) \\ deleted(v1, v2) \cup deleted(v2, v3) &= deleted(v1, v3) \end{aligned} \tag{2}$$

A. Prototype Implementation

Our experiments are carried out via a set of shell scripts calling upon an in-house Java program built on top of the Soot analysis framework [11] for implementing a reliable pairwise similarity comparison between apps [4]. This program performs at the level of the Jimple default intermediate representation (IR) in Soot. Jimple is a popular IR that has been recurrently adopted by many state-of-the-art static analysis approaches [12], [13], [14].

III. Results

Our clustering step has found in the Androzoo dataset of over 5 million apps, 372,120 families that contain at least three app versions (suggesting potential MGRep scenarios inside each family). The largest family, *com.slideme.sam.manager*, contains 20,712 apps. Due to time and resource constraints, it is expensive (and likely unnecessary) to analyze all families in our search for multi-generation repackaged apps, we select 31 families (from the top 500 families) that contain apps with at least three different certificates for further validating.

Among those selected families, we were able to identify two cases satisfying the constraint of Formula 2, demonstrating the existence of MGRep and the validation of our MGRep hypothesis. This validation will significantly impact on the state-of-the-art on repackaged app detection and related research directions. The community is thus called upon to react on this reality for better evaluating future approaches and for reflecting on the state-of-the-art.

Unfortunately, only 6.5% (two out of 31) selected families have revealed symptoms of MGRep, showing that MGRep is not a common phenomenon in Android apps. However, since the objective of this work is not to precisely detect all repackaged apps, but to focus on a sufficient number of reliable repackaging pairs that are likely to lead to the validation of the multi-generation repackaging hypothesis, we have applied a number of restrictions in this work to reduce the search space of our investigation. Such restrictions may have been too restrictive, failing to hit all the possible MGRep instances. Therefore, as our future work, we plan to revisit this hypothesis with a more systematized approach and a bigger dataset to pinpoint the severity of MGRep in the Android ecosystem.

IV. Acknowledgments

This work was supported by the Fonds National de la Recherche (FNR), Luxembourg, under the project AndroMap C13/IS/5921289 and Recommend C15/IS/10449467.

[1] Three (3) is the minimal number of apps from different developers that can be involved in a Multi-Generation Repackaging process.

References

[1] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics & Security*, 2017.

[2] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 95–109. IEEE, 2012.

[3] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *The 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, 2016.

[4] Li Li, Tegawendé F Bissyandé, and Jacques Klein. Simidroid: Identifying and explaining similarities in android apps. In *Technical Report*, 2017.

[5] Yuru Shao, Xiapu Luo, Chenxiong Qian, Pengfei Zhu, and Lei Zhang. Towards a scalable resource-driven approach for detecting repackaged android applications. In *ACSAC*, 2014.

[6] Ke Tian, Danfeng Daphne Yao, Barbara G Ryder, and Gang Tan. Analysis of code heterogeneity for high-precision classification of repackaged malware. In *MoST*.

[7] Alessandro Aldini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. Detection of repackaged mobile applications through a collaborative approach. *CCPE*, 2015.

[8] Hugo Gonzalez, Andi A Kadir, Natalia Stakhanova, Abdullah J Alzahrani, and Ali A Ghorbani. Exploring reverse engineering symptoms in android apps. In *Proceedings of the Eighth European Workshop on System Security*, page 7. ACM, 2015.

[9] Li Li, Daoyuan Li, Tegawendé F Bissyandé, David Lo, Jacques Klein, and Yves Le Traon. Ungrafting malicious code from piggybacked android apps. *Technical Report*, 2016.

[10] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Haipeng Cai, David Lo, and Yves Le Traon. Automatically locating malicious packages in piggybacked android apps. In *Technical Report*, 2017.

[11] Haoyu Wang, Yao Guo, Ziang Ma, and Xiangqun Chen. Wukong: A scalable and accurate two-phase approach to android app clone detection. In *ISSTA*, 2015.

[12] Patrick Lam, Eric Bodden, Ondrej Lhoták, and Laurie Hendren. The soot framework for java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, 2011.

[13] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick Mcdaniel. IccTA: Detecting Inter-Component Privacy Leaks in Android Apps. In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2015)*, 2015.

[14] Li Li, Tegawendé F Bissyandé, Damien Octeau, and Jacques Klein. Droidra: Taming reflection to support whole-program analysis of android apps. In *The 2016 International Symposium on Software Testing and Analysis (ISSTA 2016)*, 2016.

[15] Li Li, Tegawendé François D Assise Bissyande, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Yves Le Traon. Static analysis of android apps: A systematic literature review. Technical report, SnT, 2016.