Future Generation Computer Systems 80 (2018) 537–545

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Reprint of "You can't touch this: Consumer-centric android application repackaging detection"[☆]

Iakovos Gurulian[*], Konstantinos Markantonakis, Lorenzo Cavallaro, Keith Mayes

Information Security Group, Royal Holloway, University of London, Egham Hill, Egham, Surrey TW20 0EX, United Kingdom

HIGHLIGHTS

- We propose an application store agnostic repackaging detection method.
- Detection based on elements that an attacker is reluctant to significantly alter.
- 91% detection rate on real repackaged applications.
- Detection of repackaged applications that clone original application's name and icon.
- Detection of repackaged applications that only clone the application name and icon.

ARTICLE INFO

Article history:
Available online 14 November 2017

Keywords:
Android
Application repackaging
User-centric security
User privacy
Effectiveness analysis
Electronic fraud

ABSTRACT

Application repackaging is a widely used method for malware distribution, revenue stealing and piracy. Repackaged applications are modified versions of original applications, that can potentially target large audiences based on the original application's popularity. In this paper, we propose an approach for detecting repackaged applications. Our approach takes advantage of the attacker's reluctance to significantly alter the elements that characterise an application without notably impacting the application's distribution. These elements include the application's name and icon. The detection is initiated from the client side, prior to an application's installation, making it application store agnostic. Our experimental results show that detection based on our algorithm is effective and efficient.

1. Introduction

One of the major challenges for a malicious user is to get a malicious application distributed to a substantial population of genuine users. On the Android platform, 86% of all malware distribution relies on *repackaged applications* [1]. In the context of this paper, we define repackaged applications as applications that impersonate a genuine application, by slight modifications/variations to the genuine application's artwork and/or changes to its source code in a way that the repackaged application looks and/or feels like the genuine application. The main objective of a repackaged application is to mimic a genuine application so it can target novice users that gravitate towards the popularity/functionality of the genuine application. In this definition, we do not include applications that infringe the potential intellectual property of the original application and present themselves as unique/different applications.

Repackaged applications are a serious threat and are part of the OWASP's *Top Ten Mobile Risks* for 2014 [2], posing the first and only threat in the list related to malware distribution. Many different methods for detection of repackaged applications have been proposed, but most of them rely on the application stores to perform the detection [3–10] (Section 2.2).

In this paper we propose a method for detecting repackaged applications, initiated on the client side, prior to an application's installation. The target application is being checked against a database of legitimate applications, hosted by a trusted third party (TTP). If it does not exist in the database, string and image similarity algorithms are used to detect original applications with similar name and icon pairs (Section 3). These two elements characterise an application prior to its installation. An attacker who wants to increase the spreading rate of a repackaged application by taking advantage of the already established popularity of an original application might not be likely to alter these elements.

DOI of original article: http://dx.doi.org/10.1016/j.future.2016.05.021.

☆ This article is a reprint of a previously published article. A publishers' error resulted in this article appearing in the wrong issue. The article is reprinted here for the reader's convenience and for the continuity of the special issue. For citation purposes, please use the original publication details; I. Gurulian et al. / Future Generation Computer Systems 65 (2016) 1–9.

* Corresponding author.
E-mail address: Iakovos.Gurulian.2014@live.rhul.ac.uk (I. Gurulian).

http://dx.doi.org/10.1016/j.future.2017.11.011
0167-739X/© 2017 Published by Elsevier B.V.

Our experimental results have shown that the proposed mechanism is both effective and efficient in detecting repackaged applications (Section 5). Furthermore, in comparison to existing methods, our proposal scored high on a set of predefined criteria (Section 5.3).

The main contributions of this work are:

- A method for detecting repackaged applications, based on elements that an attacker cannot significantly alter without substantially minimising the attack potential. The method was designed to be fast and application store agnostic, so that the detection process can be initiated from the client side, prior to an installation.
- We were capable of detecting applications that only copy the name and the icon of an original application in order to trick the users into installing them. This is a known technique that attackers use [11], but to our knowledge, we are the first to effectively detect such applications.

2. Application repackaging

Android applications come in `.apk` containers (basically `.zip` files). These containers include the application's bytecode, resources and libraries, as well as a folder (named `META-INF`) that holds the signature(s), generated by the developer, on different elements of the respective application. An attacker that repackages an application can modify its bytecode, alter its resources and libraries, and then remove the `META-INF` folder and sign the application with his/her own key.

Each Android application has a package name that is used by the operating system as the differentiator factor between applications. If an application that is about to be installed shares the same package name with an already installed one, Android will perceive it as an update attempt on the last. The update process cannot continue, unless the signatures of the two applications match.

2.1. Threats to the Android ecosystem

The threats posed by application repackaging can be separated into those concerning the application developer and those concerning the user.

2.1.1. Threats to the developer

Developers are given the opportunity to gain financial profit through their applications. Repackaged applications pose a threat to them in various ways.

- *Unauthorised redistribution*: An attacker can redistribute an application, signed with his/her own signature and possibly sell it in an application store, without the original developer's permission.
- *Advertisements*: Potential revenue from an application might be redirected to the attacker, as he/she might have altered the developer's account with his/her account, thus receiving the advertisement revenue.
- *Cracking (Piracy)*: Application repackaging might distribute pirated copies of a genuine paid application, by bypassing any implemented verification and validation mechanisms.

2.1.2. Threats to the user

Repackaged applications that aim to harm the user can achieve their goal in two different ways.

- *Trojan horse*: A repackaged application could act as a Trojan horse. Malicious code can be implanted in the original application, capable of intruding the platform's security and the user's privacy. For example, repackaging an application that requires permissions to access the Internet and the device's storage can allow an attacker to steal the user's images.
- *Denial of upgrade*: A repackaged application will not be updated by a genuine application's update. Therefore, the user will not be able to upgrade, once a repackaged application has been installed.

2.2. Related works

AndroGuard was proposed by Desnos and Gueguen [3]. Their proposal is based on Control Flow Graphs, used to measure the similarity between applications.

Crussell et al. proposed *DNADroid* that uses Program Dependency Graphs to detect similar applications [4]. The authors claim that their system produces a low false positive rate. However, they accept that advanced obfuscation techniques can go undetected. To increase the system's performance, they only compare the application in question against applications with similar names.

Zhou et al. used Fuzzy Hashing for repackaging detection [5]. This method is based on generating a hash of the application by breaking it down into small chunks and combining their hashes. They also remove string operands that can easily be altered from the instructions prior to hashing, in order to prevent common obfuscation techniques. They have created an application called *DroidMOSS*. They state that although their system is very robust, detection may fail if big chunks of code have been added to the original application.

Another solution, proposed by Hanna et al., introduces the idea of Feature Hashing for the detection of similar applications [6]. For this purpose, a tool called *Juxtapp* was created, that according to the authors is resilient to some amount of obfuscation.

Zhauniarovich et al. [7] proposed detection of repackaged applications based on the contents of the `.apk` file. Their method showed results similar to those of techniques that involve code analysis. SHA1 is used for the comparison of all the files between different applications. A repackaged application with slight changes on many files might go undetected. The authors propose a combination of their method with code analysis in order to overcome this issue.

A framework capable of measuring the obfuscation resilience of algorithms used in repackaging detection programs was also proposed in 2013 by Huang et al. [12]. Other researchers have also proposed methods for detection of repackaged applications as well, or have investigated the subject [8–10].

Due to the computational complexity of the methods discussed above, they are only meaningful on an application store level. For this reason, Zhou et al. [13] proposed client side initiated repackaging detection, with *AppInk*. Their method uses application watermarking in order to confirm the authenticity of an application. In order to achieve that, a few additional steps have to be taken by the developer in order to embed the watermark.

3. Proposed solution

We propose a repackaging detection technique that takes advantage of the attacker's reluctance to significantly alter elements that characterise an application, without substantially minimising the attack vector. The data that characterises an