Verifying the integrity of open source Android apps[1]

Authors
Michael Macnair, MSc (Royal Holloway, 2014)
Keith Mayes, ISG, Royal Holloway

Overview

The published source code of an open source Android application may have been audited, but what if the application downloaded from the Play Store was not built from this source code?

This article introduces AppIntegrity[2], a service that enables researchers to easily verify that each version of an app that is published on the Play Store was actually built from its published source code.

The problem

Android has become the most popular platform for mobile devices such as smart phones. Users often grant Android applications permission to access their personal information such as contacts or current location. In doing so, users trust the application not to abuse the access they have to that information. Sometimes this trust is earned by the application developer releasing the source code under an open source licence, enabling it to be freely audited. However, even if the source code is reviewed and found to be trustworthy, that is not sufficient to establish that the installed app itself is trustworthy.

For an Android application to be installed it must first be compiled into a binary form (an 'APK' file) and then published, typically on the Play Store. Unfortunately, there is no assurance of correspondence between this binary file and the source code that the application was built from. This means that whilst the source code for an open source application may be available for review, the user has no guarantees that the application they download from the Play Store was actually built from the published source code.

This gap allows malicious developers, or attackers who have obtained the developer's Play Store credentials, to insert malware into an application and potentially remain undetected. The modified app could be used to harvest personal information or send spam, for example.

---

[1] This article is to be published online by Computer Weekly as part of the 2015 Royal Holloway info security thesis series. The full MSc thesis is published on the ISG's website.

[2] The AppIntegrity logo: Git Logo by Jason Long is licensed under the Creative Commons Attribution 3.0 Unported License. Android robot is modified from work created and shared by Google and is licensed under the Creative Commons 3.0 Attribution License. The AppIntegrity git-robot logo by Michael Macnair is licensed under the Creative Commons Attribution 4.0 International License.

The AppIntegrity solution

The AppIntegrity service helps bridge the gap between known source code and opaque binaries on the Play Store. It does this by automatically downloading application binaries and sources, building the sources and comparing the resulting applications. The results are presented on a website, allowing reviewers to see the extent of the differences between the published application and the version built from source by the independent AppIntegrity service.

AppIntegrity serves two main purposes:

- encouraging and helping application developers to produce applications with a so-called 'reproducible' or 'deterministic' build, that produces the same output no matter who builds it;
- providing a check against subverted binary versions of open source applications appearing in the Play Store.

The service monitors the Play Store and downloads any new versions of the target applications that are uploaded.

Once the new app has been downloaded, the version of source code that corresponds to the version number of the new app is identified and downloaded from the project's public repository. The source is then built in a virtual machine that is provisioned specifically for the purposes of building the application in question. This enables the right dependencies and compiler versions to be used for the build, and also simplifies contributing to the service.

Anatomy of an Android app

Android applications are distributed as APK files, which are zip files containing:

- AndroidManifest.xml (app metadata)
- classes.dex (Dalvik bytecode)
- resources.arsc (constants, resource mapping)
- res/ (other resources)
- assets/ (arbitrary files)
- lib/ (native code shared objects)
- META-INF/ (signature data)

Once built, the Play Store APK file can be compared to the from-source APK file. This starts with unzipping each APK, then comparing each element and storing the output of the comparison for viewing on the website. Textual files (such as the application manifest) are compared using a standard diff tool; the application bytecode is automatically reverse engineered into both human-readable instructions and Java source code which are then textually compared; the compiled Android resources are also reverse engineered into a textual form for comparison; other binary files such as images are simply compared for equality. One element of the APKs that should always be different is the signature: the from-source copy of the APK will be unsigned, or signed using a dummy key, whereas the Play Store version of the APK will be signed using the production key (which should not be kept in the public source code repository!).

Whilst the public AppIntegrity website is focussed on open source Android applications, vendors of proprietary Android applications could host their own private instance of AppIntegrity. This could be used to monitor their own applications for reproducibility, or for compromised binaries being uploaded to the Play Store.

Analysing secure communications apps

During its development AppIntegrity was used to check the integrity of four popular open source secure communications applications published on the Play Store: RedPhone, ChatSecure, TextSecure and Telegram. Each of these applications exhibited differences between the version on the Play Store and the version that was built from the published source, however the differences found between the published sources and binaries were all benign.

The review highlighted some poor release practices: determining the version of the source code that corresponds to the published binaries proved to be non-trivial in some cases, with some releases of Telegram having no corresponding version of source code in their repository.

Other variations ranged from different binary representations of the same floating point number, to differing Java compilers, accidentally included or omitted files, and deliberate source code changes to identify the source of bug reports.

In addition to the secure communications apps, the Guardian Project's Lil' Debi app was checked, as this is the first Android application with a self-proclaimed reproducible build. The contents of the application on the Play Store were indeed found to be identical to the version built from source, proving that it is feasible for developers to reach the goal of a reproducible build.

Concluding remarks

AppIntegrity has shown itself to be a useful tool in providing assurance that installed apps are indeed based on published source code and have not been maliciously modified. Whilst the primary focus is on protecting the users of apps, AppIntegrity is also of benefit to developers in helping to produce reproducible builds of their apps.

Availability

The idea for this service evolved from research work carried out by Michael Macnair at Royal Holloway University of London. It is under active development and contributors are welcomed. The free AppIntegrity service is available at https://appintegrity.info and the source code is available on GitHub under the Apache 2.0 licence.

An Android word cloud, highlighting key words from the full report

Biographies

Michael Macnair is a Security Engineer working for Thales e-Security in the UK. In this role he guides development teams on secure product development, designs security mechanisms and performs in-house vulnerability assessments. Michael holds an MSc in Information Security and an MEng in Computer Science.

Keith Mayes B.Sc. Ph.D. CEng FIET A.Inst.ISP, has spent much of his life working in/with industry, yet is also an active researcher/author with 100+ publications. He is Director of the ISG Smart Card Centre at Royal Holloway University of London and of Crisp Telecom Limited. He has worked in hardware/software development, DSP and sensors, standardisation, mobile communications, smart cards/RFIDs, embedded systems, systems modelling plus diverse aspects of information security. Current interests include (but are not restricted to) mobile comms and trusted execution technologies, NFC, Smart cards/RFIDs, transport ticketing systems, automotive security, m-commerce, attacks and attack resistant system implementations.