Reproducible Builds for Android Apps

Andreas Itzchak Rehberg
IzzyOnDroid

Who am I?

Andreas Itzchak Rehberg (aka Izzy)

Oracle DBA at `$DAYJOB`

Founder of IzzyOnDroid

Overview

1. What are RB, and what are they good for?
2. How do we approach RB at IzzyOnDroid/F-Droid?
3. What are the challenges?
4. What are the most frequent sources of failed RBs?
5. What should Android App Developers be aware of / take care for?
6. Q&A

What are RB, and what are they good for?

- third parties make sure independently that your software hasn’t been altered, increasing safety, reliability, and trust.
- make sure that developers’ code always works the same way, which makes the software more consistent and trustworthy.
- detect unauthorized changes to the build process
- ensure software complies with licenses and industry standards by proving that binaries match their source code.
- protect your software against being compromised
- increase confidence/trust (builds verified independently)
- confirm ownership in case of signing-key loss

How do we approach RB at IzzyOnDroid / F-Droid?

IzzyOnDroid	F-Droid
RB on separate track	RB integrated with regular builds
failed RB does not prevent updates from being published	failed RB means failed build (no update shipped)
using rbtlog	using fdroidserver
using only FOSS tooling	using only FOSS tooling
Podman containers, different images (Debian bullseye/bookworm/trixie, Ubuntu jammy/noble)	Vagrant with Debian bookworm (main builder) / Podman (verification builder)
multiple builders, including independents (not operated by IzzyOnDroid)	(currently) only F-Droid's own builders
independently verifying developers' builds	independently verifying developers' builds for apps established as RB (main builder), automatically checking for apps that *could* be established as RB (verification builder)
(always shipping the APKs built & signed by the resp. dev)	(shipping the APKs built & signed by the resp. dev and F-Droid when established as RB, else only signed by F-Droid)

What are the challenges?

- Easy to handle: Java, Kotlin. In most cases, only the JDK version must match here.
- More komplex: apps with native libraries (Flutter, NodeJS/React, Go, Rust …)
  - build paths embedded into native libs
  - paths from Windows builds cannot be matched on Linux (drive letters)
  - different versions of build SDKs produce different output, so versions must be matched as well

What are the most frequent sources of failed RBs?

- “first basic rule” violated: dirty builds, different commit
- switching between building on Windows/Linux (line endings)
- introducing (more or less complex) enforced signing (see: Signing Config)
- building locally with different setups than specified
- embedded build timestamps
- using non-LTS JDKs (see: JDK versions)

What should Android App Developers be aware of / take care for?

For this, we've established some RB Hints for Developers, e.g.

- Aforementioned "first basic rule" (to always make release builds from a clean tree at the tagged commit)
- Avoid build-time related things (embedded timestamps, BuildIDs, etc)
- For release builds, avoid "canaries" and other non-LTS build tools (e.g. JDK)
- Avoid already mentioned things like switching between Linux and Windows, deviating from your own settings (local Gradle/Flutter not matching what your project states), etc.

Related Links

- Reproducible Builds Documentation at IzzyOnDroid, incl. Debug failed RBs
- Our RB framework can easily be set up using rbuilder_setup (5 minutes and some disk space) – and we welcome additional builders
- Reproducible-Builds.org
- F-Droid's Reproducible Builds documentation
- Our clients (Droid-ify, Neo Store) show the RB status of apps

Q&A

Now you’re welcome to ask

And to visit us at our stand: UD2-02

Thank you!

Thank you all for

- Your interest in the topic
- Listening to my talk
- Your support

nlnet FOUNDATION

NGI MOBIFREE

DONATE TO OUR COLLECTIVE