Securing applications from commit to execution for Android apps and Jar files

James Tapsell

28th September 2017

Year: 2017/18
Module: Final Year Project (IY3821)
Supervisor: Dr Jorge Blasco Alis

1 Abstract

A lot of applications are now signed to prevent tampering with them, this usually consists of a created executable that is signed by the company or group releasing some software, allowing the end user or their system to verify the source of the software. The signature usually covers the executable code, but also the meta-data, and the resource files, allowing prompts to show who signed a piece of software, and stopping manipulating the resource files to for example swap out error messages for security issues with benign looking messages.

For some classes of software (device drivers on windows for example), this is a mandatory step, as 64 bit editions need settings changes to allow the use of unsigned drivers[1]. This was introduced to try to prevent malware from getting deep into the system, as once it is at such a layer not only does it have the ability to take over the system, but also hide itself from any software looking for it.

Unfortunately this is just moving the insertion of malicious payloads to before the signature, such as the CCLeaner[2] attack, which exploited exactly this loophole (This has not been confirmed to be how the attack took place, but analysis by Cisco[3] point to this, and no other explanation has been provided). This kind of attack generally works by injecting malicious code into the repositories used to build the executables that are then signed for release holding their payload.

Trying to defend against this attack is difficult, but there are some things that can help. One example of this is that git allows for signed commits[4], which can help to prove the author of the commit. This does introduce the issue of validating all commits are signed though, and the storage of keys for developers, although yubikey style devices [5] or smartcards can help reduce the difficulty of this.

This project looks to create a proof-of-concept defence against such an attack with the following features:

1. Developers should not need to install new software on their own devices if possible (although this may be necessary in some cases, this should be avoided if possible).

2. The releases are signed in such a way that end user devices can be sure who committed and released an application, and there should be no breaks in the chain of custody of the source or artefacts created from it.

3. The releases should not be able to be tampered with in an undetectable way.

4. It should not be possible to include an entity (person, group or company) in the list of people who released an application without their permission.

This project does not attempt to solve the issue of rogue developers, not for malicious commits signed by a developer, as these are beyond the scope. It also does not deal with key management for developers, as such tools already exist.

As I would like to gain more experience using Kotlin, and JGit is written in Java, I will write the majority of the code for this project in Kotlin, which is cross compatible with Java (Java code can access all methods and fields of Kotlin code and vice versa, and they can run together in the same JVM. This is possible as Kotlin compiles to Java bytecode, rather than having its own format. Kotlin is also a first class language on Android[6], so the apps that are needed can be developed in it too, without needing to be translated to Java.

By having code that can run on both computers and Android devices, I can make simple versions of the systems, package them as JARs and test them locally, and once they work I can integrate the logic into an Android app, as an APK file. By using this 2 stage pattern I can test locally, which is more convenient than needing to use a device for testing, and also as a computer can have more resources I can run tests that are more exhaustive in less time.

2 Reasons for choosing this project

A proof of concept of a defence
With a system like this be put in place, then a whole class of attacks which be removed, as code injection would need to take place before commits, and as commits should be checked at the pull request stage, this should mean that injected code will be found. The only ways would be:

1. To inject code and hope it passes review
2. To make a pull request still have to pass review
3. To try to manipulate the build system, which should be detectable, and would require an exploit to get on to the server first
4. To try to make a colliding release, but this would be incredibly difficult.

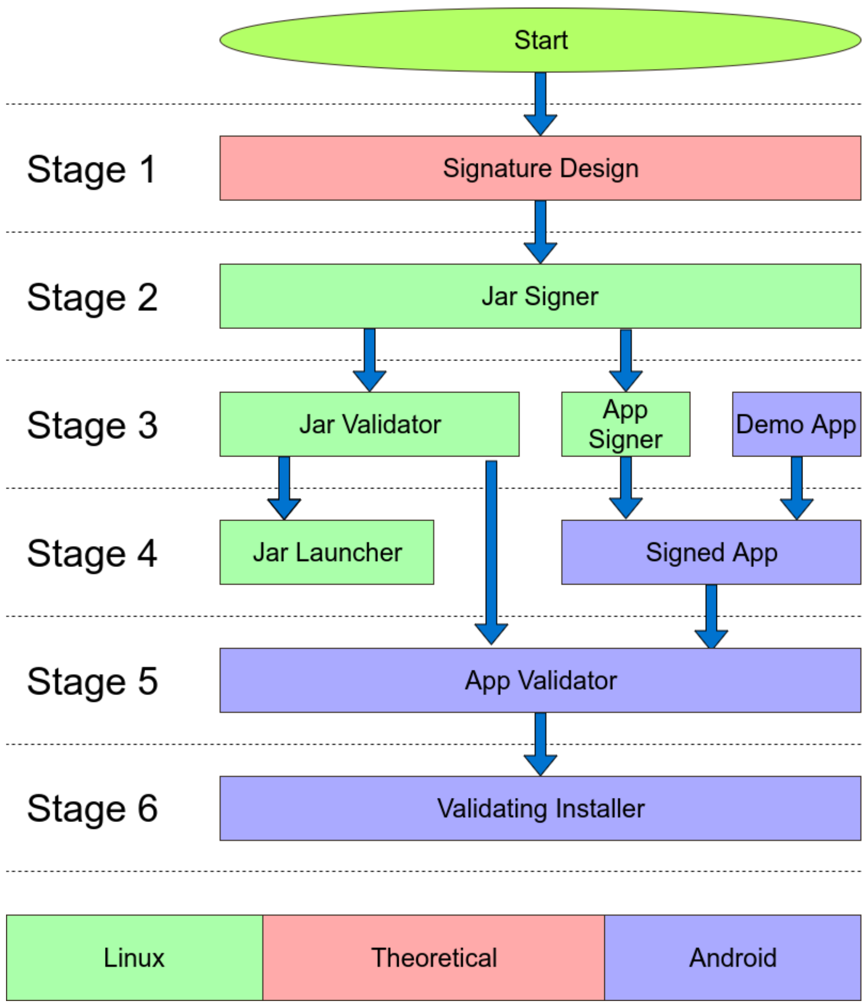
This also helps in the event that a key is lost, because at the moment apps cannot be updated on Android if the release key is lost, the only way to move forward is to change the app ID and to sign it with a different key, which requires the user to manually uninstall the original app and manually install the new app, but if the keys are made from a set of keys then loss of one or more shares should be less of a problem, unless the majority of shares are lost.

A chance to work on build automation
I want to learn more about how to automate builds and make them reproducible.

A chance to work with Kotlin
I would like to learn Kotlin, and how to testing using Kotlin.

A chance to work with cyptography
I feel my understanding of this field could use work, and this will be a good chance to improve this.

3 Milestones



Signature design
A base of how the signatures will be calculated and stored, this has to be completed first in order for the other parts of the project to be done, as they all will use the signature format.

Jar Signer
A tool that takes a working repo and signs it with a given key or set of keys. This will be the first test of the Signature design, so it needs the signature design completed first.

Jar Validator
A tool that validates a jar file and shows the details of the keys that wrote and released it.

Jar Launcher
A tool that launches a jar file after asking a user if they trust the developers and/or releasers.

Demo App
A basic app with all commits signed.

App Signer
Signs an app and creates a signed release.

Signed App
An app signed which proves the developers and/or releasers.

App Validator
Checks if an app is valid and shows the signature details.

Validating Installer
An app which replaces the android installer, and shows the signature details before the user installs.

4 Planning and Time-scales

The deliverables are split into 6 stages: 1-6, each of which should take around 2 weeks, which gives a few weeks extra at the end to allow for any delays caused.

Stage 1
2nd Oct to 13th Oct

Stage 2
16th Oct to 27th Oct

Stage 3
30th Oct to 10th Nov

Stage 4
13th Nov to 26th Nov

Stage 5
27th Nov to 10th Dec

Stage 6
8th Jan to 19th Jan

After these stages there should be time to clean up the written code, and to finalise the reports and other materials before the hand in date.

The handbook deadlines are included below for completeness:

Project Plan
23:59 on Friday, 29 September, 2017

Interim Programs and Reports
23:59 on Friday, 1 December 2017

Interim Review Viva
Monday, 4 – Friday, 8 December, 2017

Full Unit Draft report ready (send by email to supervisor)
Friday, 16 February, 2018

Full Unit Final Programs and Report
14:00 on Friday, 23th March, 2018

Project Demo Days
Summer Term, 2018

5 Risks and Mitigations

The signatures may not be possible - Unlikely - Highly important
This seems unlikely, even if it is not part of the JRE it should be possible with libraries, this is much more likely to be an issue on Android, but is still unlikely.

Cannot create an app - Unlikely - Highly important
This seems unlikely, there are basic tutorials online, so even if I cannot create it on my own I can use a template, or even just clone an open-source app and sign it with demo keys.

Cannot get signature details from git - Possible - Fatal
This would be a show stopping, JGit does not expose signature details by default, but this still leaves 2 ways to get around this issue:

1. Patch JGit to support this
   This would be a lot of work, but it would mean I would have a contribution to a big project and also help future developers working on similar tasks.

2. Write a wrapper around the git CLI to do this
   This would be less work, but would not perform as fast as using JGit, if it was invoked on each commit individually. This would also be more fragile, as changes to the git CLI would break it, and it would be less portable, as it would rely on git being installed.

   I have tested this mitigation and it seems to work, although it is quite complex, using annotations to simplify linking git pretty output to fields in a data class.

Using insecure algorithms - Likely - Highly important (Mitigable)
Unfortunately git commit hashes are based on SHA1, which is known to be broken[7], although it is thought that git should be resistant for longer by Linus Torvalds.[8], this is likely to mean that the root trust of a commit may be broken already, and will be at a future point.

A temporary workaround would be to sign a secondary hash, using a more secure algorithm, but this would require developers to run non-standard software on their machines, leading to an increased attack surface.

A longer term solution would be to make the verification system able to have its algorithm changed, possibly with the option of having multiple acceptable algorithms available, so older apps can still be ran, although this would increase the attack surface more.

Bibliography

[1] "Driver signing policy." https://docs.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-policy–windows-vista-and-later-, 2017.
This lists the signing policy for windows drivers, which shows it is mandatory on recent versions of windows using secure boot, which is a mandatory part of the windows hardware specs[9].

[2] A. Greenberg, "Software has a serious supply-chain security problem." https://www.wired.com/story/ccleaner-malware-supply-chain-software-security/, 2017.
This was a good example of malicious code being injected before signatures, getting around the current protections.

[3] L. Tung, "Hackers hid malware in ccleaner pc tool for nearly a month." http://www.zdnet.com/article/hackers-hid-malware-in-ccleaner-pc-tool-for-nearly-a-month/, 2017.
This is second source about the CCleaner hack.

[4] "7.4 git tools - signing your work." https://git-scm.com/book/en/v2/Git-Tools-Signing-Your-Work, 2017.
This shows that git supports signed commits, proving their author or commiter.

[5] "Git signing." https://developers.yubico.com/PGP/Git_signing.html, 2017.
This shows yubikeys can be used to store keys used for signing commits, proving that they were created by the person that was expected, this helps to simplify managing keys, and excluding an attack against the yubikey keeps the key safe even when inserted into malicious machines.

[6] M. Shafirov, "Kotlin on android. now official." https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/, 2017.
This shows that Kotlin is now a first class language on android, which allows it to be used for this project, although before it would have required additions to the toolchain it is now fully supported.

[7] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov, "Shattered." https://shattered.io/, 2017.
SHAttered attack, demonstrating a pair of colliding PDFs.

[8] L. Torvalds, "Re: Sha1 collisions found." https://marc.info/?l=git&m=148787047422954, 2017.
Linus Torvalds' thoughts on the effects of the SHA1 collion on git.

[9] "Hardware compatibility specification for systems for windows 10, version 1607." https://docs.microsoft.com/en-gb/windows-hardware/design/compatibility/systems#systemfundamentalsfirmwareuefisecureboot,

2017.
This shows that secure boot is a mandatory part of the requirements for windows 10.