Checks on executable code in Catalina and Big Sur: a first draft

Published: 2020-11-16 · Archived: 2026-04-05 17:48:08 UTC

All eyes have turned to the checks that recent versions of macOS make when preparing to load and run executable code. To date, as far as I can see, there are disparate accounts of what have been considered certificate and notarization checks, and a few marginal notes about changes when running native code on Apple Silicon Macs. What I attempt in this article is a coherent account of how macOS checks executable code before it's loaded and run, in macOS 10.15 and 11.0. This is my first draft, so please bear with my errors and help me correct them to build a more accurate picture. You can see my external sources listed at the end, to which you can add a succession of my own articles here.

These processes don't just occur when you open an app, but also when any other executable code is loaded, for example when using a command tool, or when running code needs to load an executable from another source such as a dylib. I'm not going to consider the loading of code contained within the protected System volume, or that signed by Apple, and that run in sandboxed apps supplied from the App Store may also differ in parts. I'm also going to ignore the handling of code for Intel architecture on Apple Silicon Macs, which involves its translation by Rosetta 2.

Code signing

There's an important difference between loading natively executable code on Intel and Apple Silicon Macs: the former allows code to be completely unsigned, but the latter refuses to load ARM code which isn't signed at all. However, that requirement is enforced not to establish any chain of trust from its signing certificate (using an ad hoc certificate is sufficient), but so that the cdhashes created during the signing process are available.

Has this code been run before?

The first question to be answered is whether this copy of macOS has run this code before. To determine that, if the code is signed its hash is looked up in the cdhashes created during the signing process. If there are no cdhashes (because the code is unsigned), then macOS has to compute its hash on the fly. That takes time, particularly if this is a large app, so Apple Silicon Macs attain superior performance by refusing to compute the hash of native code, so block its loading. The code signing requirement on Apple Silicon Macs is therefore to eliminate the first run delays which many have noticed on Intel Macs running Catalina. Although a minor nuisance for some users, this should hasten app launch and execution.

Armed with the hash for the code to be run, this is looked up in the local security database, which establishes whether that code has been run before. If it has, checks move on to examine the code signature.

If this is the first time that code with that hash has been run, the hash is added to the local security database, and that hash is sent to Apple via a connection to api.apple-cloudkit.com. This has been thought to relate to notarization, but I don't think has anything to do with that. At the least, Apple collects hashes as they're encountered, and appears to look them up in a database of known malware (see below).

Is it signed using a developer certificate?

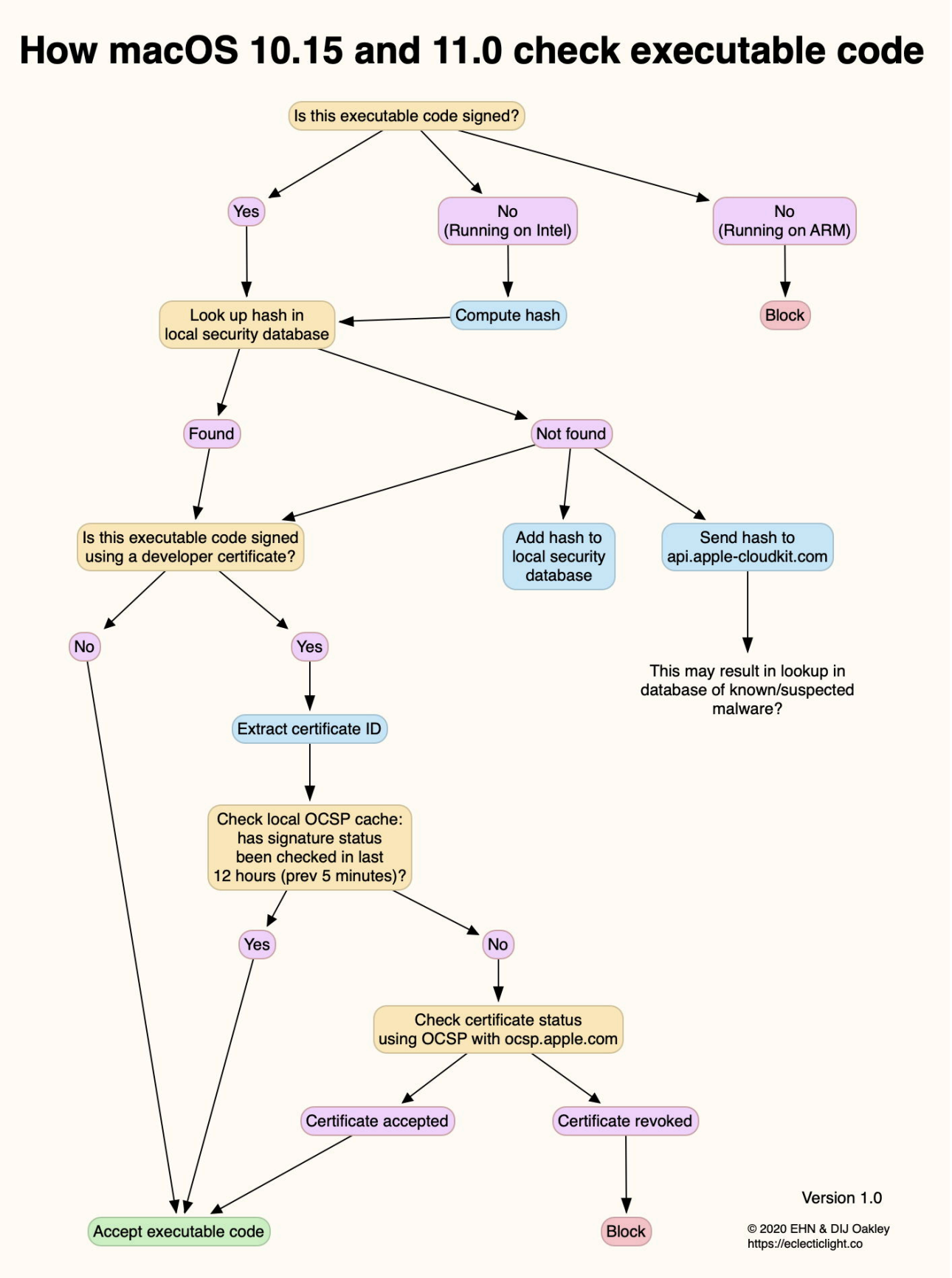
Code which hasn't been signed using a developer certificate now can't be checked any further, except by routine checks in XProtect to see if it is known malware and should be blocked. Otherwise, it is now loaded and run.

Where a developer certificate has been used to sign that code, the second question to be answered is whether that certificate is still valid. Before macOS 10.14, this appears to have been handled locally by lookup in the 'Gatekeeper' database. However, that was only updated periodically, so revoked signatures could still be accepted for a couple of weeks after they were known to have been compromised. Apple appears to have changed this system over a year ago, and at least in Catalina and later, now checks the validity of certificates online with its own servers at ocsp.apple.com.

This process uses the well-known OCSP protocol, in which the Mac queries the status of the developer certificate through that certificate's ID. To prepare for the check, macOS therefore extracts that ID. Because online checking takes time, this isn't performed on signatures which have been checked recently: before Apple's server problems last week, results were cached for a period of 5 minutes; this appears to have been extended to 12 hours now, so should only occur the first time that you run an app each day.

If the local OCSP cache doesn't contain a sufficiently recent result for that certificate ID, macOS sends an OCSP query to establish the status of that certificate. If that reports the certificate should be accepted, the executable code is given a clean bill of health, loaded and run. If the OCSP service reports that certificate has been revoked, then the code is blocked.

I have drafted a first proposal of how these processes integrate, shown here:



You can download this as a PDF from here: SignatureCheck1015a

How can you block transfer of information?

Checking whether executable code can be run is a fundamental part of your Mac's security. Apple can and does revoke certificates because they're being abused by malicious software. If you were to prevent OCSP checks on signatures, you could end up running known malware whose certificate has already been revoked. Interfering with these checks thus has serious consequences, and shouldn't be considered unless you have a really compelling reason, and have assessed the security risk of the consequences.

For a very few Mac users, such as journalists who work in hostile environments where state security services may be monitoring them, the OCSP exchange and transmission of new hashes could conceivably be used against them. There are solutions which should be able to mitigate against that. For instance, provided that an app has already been run and its cdhashes entered into the local security database, no repeated copies of those hashes should be sent to api.apple-cloudkit.com.

Muting the OCSP query is just as simple. It can't occur when the code isn't signed, so stripping an app's signature will ensure that no query can be sent. However, unsigned code can't be run on Apple Silicon Macs, and on both architectures its integrity can't be checked either. For specific apps whose use might be sensitive, one approach is to strip any developer signature and replace it with an ad hoc signature. That isn't technically difficult at all using `codesign`, and used only on specific apps shouldn't increase security risk significantly.

How is it changing?

Exceptionally, Apple has responded strongly to recent concerns about online checks being made on executable code (see link below). This confirms that the hash lookup performed with api.apple-cloudkit.com does involve verifying "if an app contains known malware", and certificate revocation checks with ocsp.apple.com as described above.

For those concerned with protecting their privacy, Apple makes it clear that "these security checks have never included the user's Apple ID or the identity of their device", and that it has stopped logging IP addresses. Furthermore, it states that certificate revocation checks will change in the next year to feature:

- "a new encrypted protocol";
- "strong protections against" [OCSP] "server failure";
- "a new preference for users to opt out of these security protections", which presumably means both hash lookup and certificate revocation checks.

I will report on those changes as and when they occur.

Key external sources:

Apple's updated account of 'Gatekeeper' checks, including details of intended changes,
Jeff Johnson on OCSP checks,
Jeff Johnson on hash reporting,
Jacopo Jannone has an excellent account of the OCSP checks.
