Resolving Gatekeeper Problems

This thread has been locked by a moderator.

1.2k

This post is part of a cluster of posts related to the trusted execution system. If you found your way here directly, I recommend that you start at the top.

Share and Enjoy

—

Quinn "The Eskimo!" @ Developer Technical Support @ Apple

```
let myEmail = "eskimo" + "1" + "@" + "apple.com"
```

Resolving Gatekeeper Problems

Gatekeeper strives to ensure that only trusted software runs on a user's Mac. It's important that your code pass Gatekeeper. If not, you're likely to lose a lot of customers, and your users' hard-won trust.

There are four common Gatekeeper problems:

- App blocked by a dangling load command path
- Broken code signature
- Lack of notarisation
- Command-line tool blocked by Gatekeeper

The first problem is *by far* the most common. For the details, see Resolving Gatekeeper Problems Caused by Dangling Load Command Paths.

For general information about Gatekeeper, read Apple > Developer > Signing Mac Software with Developer ID and Apple > Support > Safely open apps on your Mac.

IMPORTANT This post focuses on Developer ID-signed code. Gatekeeper should not block App Store apps. If an app downloaded from the App Store fails to run, it's likely to be some other trusted execution issue. For more about this, read Resolving Trusted Execution Problems.

Verify Your Signature

A good first step in any Gatekeeper investigation is to verify that your code is signed correctly. Use the `codesign` tool for this:

```
% codesign -v -vvv --strict --deep MyApp.app
```

The `-vvv` options increase verbosity to the point where `codesign` will give you useful diagnostics. For example:

```
% codesign -v -vvv --strict --deep "Munged.app"
Munged.app: a sealed resource is missing or invalid
file added: …/Munged.app/Contents/Resources/names/Adam.txt
file modified: …/Munged.app/Contents/Resources/names/Morgan.txt
file missing: …/Munged.app/Contents/Resources/names/Rhonda.txt
```

This app was changed after it was signed in three different ways:

- `Adam.txt` was added.
- `Morgan.txt` was modified.
- `Rhonda.txt` was removed.

You might see some results that make no sense. For example:

1. Start with an app with a valid code signature:

   ```
   % codesign -v -vvv --strict --deep "NotNormal.app"
   NotNormal.app: valid on disk
   NotNormal.app: satisfies its Designated Requirement
   ```

2. Use the Finder to create a zip archive (File > Compress).
3. Use the Finder to unpack that archive.
4. Check the code signature of the unpacked file:

   ```
   % codesign -v -vvv --strict --deep "NotNormal 2.app"
   NotNormal 2.app: a sealed resource is missing or invalid
   file added: …/NotNormal 2.app/Contents/Resources/names/Zoë Schrödinger.txt
   file missing: …/NotNormal 2.app/Contents/Resources/names/Zoë Schrödinger.txt
   ```

There are two things to note here. First, just compressing and decompressing the app broke its code signature. Weird! Second, look at the error messages. It seems that the `Zoë Schrödinger.txt` file is was both added and removed. Weirder!

To see what's going on here you have to look at a hex dump of the file name:

```
% ls "NotNormal.app/Contents/Resources/names" | xxd
00000000: 5a6f c3ab 2053 6368 726f cc88 6469 6e67  Zo.. Schro..ding
00000010: 6572 2e74 7874 0a                        er.txt.
% ls "NotNormal 2.app/Contents/Resources/names" | xxd
00000000: 5a6f 65cc 8820 5363 6872 6fcc 8864 696e  Zoe.. Schro..din
00000010: 6765 722e 7478 740a                      ger.txt.
```

The names are not the same! The app started out with the `ë` in precomposed form and the `ö` in decomposed form. Compressing and decompressing the app converted the `ë` to its decomposed form, and that change broke the code signature.

Programs that deal with Unicode are expected to ignore differences in normalisation. Sadly, Apple's code signing implementation missed that memo (r. 68829319). For more details see this post but the executive summary is that it's best to stick to ASCII when naming files in a bundle.

Identify a Notarisation Problem

Gatekeeper requires that your app be notarised. If not, it will block the execution of your app with a generic, user-level message. If you find your app blocked by Gatekeeper, check if this is a notarisation issue by looking in the system log for an entry like this:

```
type: info
time: 2022-05-11 14:57:21.812176 -0700
process: syspolicyd
subsystem: com.apple.syspolicy
category: default
message: ticket not available: 2/2/8b7410713591e6c79ea98f0132136f0faa55d22a
```

Note If the ticket details show as `<private>`, enable private data in the system log. For information on how to do that, see Recording Private Data in the System Log. For general information about the system log, see Your Friend the System Log.

The long hex number is the code directory hash, or cdhash, of the offending code. In this example, it's the cdhash of the app itself:

```
% codesign -d -vvv /Applications/NotNotarised.app
…
CDHash=8b7410713591e6c79ea98f0132136f0faa55d22a
…
```

However, in some cases it may be the cdhash of some library referenced by the app.

For more information about cdhashes, see TN3126 Inside Code Signing: Hashes.

Resolve a Notarisation Problem

The obvious cause of this problem is that you haven't notarised your app. For information on how to do that, see Notarizing macOS Software Before Distribution.

If you have notarised your app and yet you still see this problem, something more subtle is happening. For example, your app might reference a dynamic library that wasn't seen by the notary service.

To investigate this:

1. Fetch the notary log for your app. For advice on that, see Fetching the Notary Log.
2. Confirm that the notary log matches the app you installed. Look in the notary log for the `sha256` property. Its value is a SHA-256 hash of the file received by the notary service. Check that this matches the SHA-256 hash of the file you used to install your app. If not, see *Hash Mismatch*, below.
3. Search the notary log for the cdhash value from the Gatekeeper log message.

If the notary log doesn't contain that cdhash, that code wasn't included in the notarised ticket. It's possible that you failed to submit the code to the notary service, that it was switched out with a different version after you notarised your app, that it was package in some way that the notary service couldn't see it, or that something went wrong within the notary service.

Hash Mismatch

If you stapled your notarised ticket to the file used to install your app then the hashes in step 2 of the previous section won't match. What to do depends on the file type:

- If the file used to install your app was a zip archive (`.zip`), you definitely have the wrong file. Zip archives don't support stapling.
- If the file used to install your app was a signed disk image (`.dmg`), compare the disk image's cdhash with the cdhash for the disk image in the notary log. If those match, you know you're working with the same disk image.

  To dump a disk image's cdhash, run the `codesign` tool as follows:

  ```
  % codesign -d -vvv DISK_IMAGE
  …
  CDHash=d963af703ac2e54af6609e9ad309abee7b66fae2
  …
  ```

  Replace `DISK_IMAGE` with the path to your disk image.
- If the file used to install your app was a disk image but it wasn't signed, switch to a signed disk image. It's generally a better option.
- If the file used to install your app was an installer package (`.pkg`), there's no good way to know if this is the correct package. In this case, modify your notarisation workflow to retain a copy of the file before it was modified by `stapler`.

Tool Blocked by Gatekeeper

If your product includes a command-line tool, you might notice this behaviour:

- When you double click the tool in Finder, it's blocked by Gatekeeper.
- When you run the tool from within Terminal, it works.

This is a known bug in macOS (r. 58097824). The issue is that, when you double click a tool in the Finder, it doesn't run Gatekeeper's standard *execution* logic. Rather, the Finder passes the tool to Terminal as a document and that opens a window (and associated shell) in which to run that document. This triggers Gatekeeper's *document* logic, and that logic always blocks the tool.

There are two ways around this:

- Embed your tool in an application. If the user runs the application first, Gatekeeper runs its normal application check. If the user allows the app to run, Gatekeeper records that decision and applies it to the app and any code within the app, including your tool.
- Install your tool using an installer package. When the user goes to install the package, Gatekeeper checks it. Assuming that check passes, Gatekeeper does no further checks on the content it installed.

Revision History

- **2022-05-20** Added the *Verify Your Signature* section. Made other minor editorial changes.

Gatekeeper Code Signing Notarization

Reply

Posted 1 year ago by eskimo

Add a Comment