The Pros and Cons of Stapling

This thread has been locked by a moderator.

16

I regularly run into folks who are confused about stapling. This is my attempt to clear up that confusion.

If you have questions or comments about this post, please start a new thread and tag it with *Notarization* so that I see it.

Share and Enjoy
—
Quinn “The Eskimo!” @ Developer Technical Support @ Apple
`let myEmail = "eskimo" + "1" + "@" + "apple.com"`

The Pros and Cons of Stapling

Apple’s notarisation system supports the concept of *stapling*. This attaches a notarised ticket to your product so that it’s immediately available to the trusted execution system. Stapling is an optional part of notarisation. It’s generally a good idea to do it, but it’s not absolutely required.

Background

To explain this I need to explain how notarisation works. The best way to do this is in terms of the notary log. When you notarise a product, you get back a notary log.

Note For a more background on the notary service, see Notarisation Fundamentals. For info on how to get the notary log, see Fetching the Notary Log.

On successful notarisation, that log contains a `ticketContents` array:

```
"ticketContents": [
  {
    "path": "Test808631564.zip/Test808631564.app",
    "digestAlgorithm": "SHA-256",
    "cdhash": "8a89083d248e059b46b26e3562cb1fda0723efca",
    "arch": "x86_64"
  },
  {
    "path": "Test808631564.zip/Test808631564.app",
    "digestAlgorithm": "SHA-256",
    "cdhash": "8fa406444f8b7b8417d6d72921ae67b06a50102f",
    "arch": "arm64"
  },
  {
    "path": "Test808631564.zip/Test808631564.app/Contents/Frameworks/libswiftAppKit.dylib",
    "digestAlgorithm": "SHA-256",
    "cdhash": "b2a04c6274506082fc66b4a25bee875aad0ed3a0",
    "arch": "x86_64"
  },
  {
    "path": "Test808631564.zip/Test808631564.app/Contents/Frameworks/libswiftAppKit.dylib",
    "digestAlgorithm": "SHA-256",
    "cdhash": "b2a04c6274506082fc66b4a25bee875aad0ed3a0",
    "arch": "x86_64"
  },
  …
],
```

This lists, unsurprisingly, the ticket contents (-: It has an entry for each notarisable item in your submission. For code items, there’s a separate entry for each architecture. The `path`, `digestAlgorithm`, and `arch` properties are essentially comments. What really matters is the `cdhash` property.

The term *cdhash* stands for *code directory hash*. It’s a hash that uniquely identifies your code. If you’d like to know more about what this actually means, see TN3126 Inside Code Signing: Hashes.

When you notarise your product, the notary service issues a ticket. The format of this ticket is not documented but you can think of it as a set of cdhash values, all signed by the notary service. When the notary service signs the ticket, it’s saying “I’ve checked the code identified by these cdhash values and they are OK to run.”

When the user runs your product on their Mac, the trusted execution system checks the cdhash of every code item that gets loaded [1]. It looks for a ticket to cover that code item. If it finds one, it allows the code to load. If not, it blocks the code from loading.

[1] This is a *very* high-level summary. The exact circumstances under which the trusted execution system checks for cdhashe is… shall we say… complex.

Ticket Availability

When the notary service generates a ticket, it uploads it to Apple’s servers. The ticket is *always* available there.

You may also staple the ticket to your product, which includes a copy of your ticket in the product. When the user runs your product for the first time, the system checks the ticket for validity and, if it’s valid, adds its contents to a local database of cdhashes of code that’s allowed to load. Later on, when the trusted execution system checks your code as its loading, it finds the cdhashes there and allows the code to load.

If the trusted execution system doesn’t find the code’s cdhash in its local database, it reaches out to Apple servers to find a ticket for that cdhash. If it finds one, it ingests that ticket into its local database and allows the code to load.

Why Staple?

Given the above background, it should now be clear why stapling is important: It allows the user to run your product for the first time when the Mac is offline. For example:

1. The user downloads an app but doesn’t run it.
2. They then take their Mac offline.
3. And run the app.

If the app has a ticket stapled to it, the app will launch. OTOH, if there’s no stapled ticket, the trusted execution system will block the launch. That’s because:

- It wasn’t able to find the app’s cdhash in the local database.
- Its attempt to fetch the ticket from Apple’s servers failed because the Mac is offline.

Note This is why I included step 3 in my Testing a Notarised Product instructions.

These scenarios are relatively rare but they do crop up from time to time, which is why stapling a ticket to your product is best practice.

However, it’s not absolutely required. Most Macs spend most time online, so it’s likely that the trusted execution system will be able to download the ticket from Apple’s servers. And if that works, the only downside is a slight delay on first launch.

There is, however, one somewhat common scenario where stapling really matters, namely, isolated networks. Some Macs are deployed in high security environments where they *never* have access to the Internet. In that case, stapling is really important [1].

[1] There are ways for the site admin to get around this — by repackaging your product and then notarising and stapling that — but it’s best if you don’t make them jump through those hoops.

Notarization

Reply

Posted 1 week ago by eskimo

Add a Comment