What exactly is a provisioning profile?

This thread has been locked by a moderator.

3.2k

IMPORTANT This post is now retired in favour of TN3125 Inside Code Signing: Provisioning Profiles. I'm leaving the original post here just for the record, but you should consider the official documentation authoritative.

I regularly help folks with code signing issues and have learnt that a lot of people are confused by the whole concept of a provisioning profile. This is my attempt to clear that up (-:

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

What exactly is a provisioning profile?

Apple platforms will not run arbitrary third-party code [1]. All execution of third-party code must be specifically authorised by Apple. This is done using a provisioning profile [2], which describes five requirements:

- Who is allowed to sign code?
- What apps [3] are they allowed to sign?
- Where can those apps run?
- When can those apps run?
- How are those apps entitled?

When the developer web site creates a profile [4] it cryptographically signs it. When you run an app on a device, the device checks this signature to determine if the profile is valid and, if so, checks that all of the app's requirements are met by that profile.

[1] Except for macOS, although provisioning profiles are still relevant on macOS, as we'll see later.

[2] There is an interesting edge case here. When you submit your app to the App Store, it re-signs your app as part of the distribution process. Before doing that, it checks that the app is signed and provisioned correctly. That ingestion check means that each individual iOS device does not need to perform further security checks, so the final app does not have a provisioning profile. However, this third-party code was still, technically, authorised by a profile, albeit during the App Store ingestion process.

[3] In this document I'm using the term *app* to refer to a main executable packaged in a bundle structure. So everything I say about *apps* also applies to app extensions, App Clips (iOS), system extensions (macOS), and XPC Services (macOS).

[4] Either directly, using the web site itself, or indirectly, using tools like Xcode.

Unpack a Profile

A profile is a property list wrapped within a CMS signature [1]. To view the original property list, remove the CMS wrapper using the `security` tool:

```
% security cms -D -i Profile_Explainer_iOS_Dev.mobileprovision -o Profile_Explainer_iOS_Dev.plist
% cat Profile_Explainer_iOS_Dev.plist
…
<dict>
  … lots of properties …
</dict>
</plist>
```

IMPORTANT The exact format of a provisioning profile is not documented and could change at any time. I discuss these details here because I think it's useful to understand how things work under the covers. I recommend against building a product based on these details; if you do build such a product be prepared to update it as the Apple development story evolves.

[1] Cryptographic Message Syntax, as defined by RFC 5652.

The Who

Every profile has a `DeveloperCertificates` property holding the certificates of each developer who is allowed to sign code that's covered by the profile. For example:

```
% cat Profile_Explainer_iOS_Dev.plist
…
<dict>
  …
  <key>DeveloperCertificates</key>
  <array>
    <data>MIIFxDCC…tOLyk4==</data>
    … more certificates …
  </array>
  …
</dict>
</plist>
```

Use `PlistBuddy` to extract a specific certificate:

```
% /usr/libexec/PlistBuddy -c "Print :DeveloperCertificates:0" Profile_Explainer_iOS_Dev.plist > cert0.cer
% certtool d cert0.cer
Serial Number     : 7E FF 9C 91 BB EB D8 AB 42 B1 52 35 64 F9 0F 72
Issuer Name       :
    Common Name   : Apple Worldwide Developer Relations Certification Authority
…
Subject Name      :
…
    Common Name   : Apple Development: Quinn Quinn (7XFU7D52S4)
…
Not Before        : 09:15:23 Apr 21, 2021
Not After         : 09:15:22 Apr 21, 2022
…
```

The What

Most profiles apply to a single App ID. This is encoded in the `Entitlements` > `application-identifier` [1] property:

```
% cat Profile_Explainer_iOS_Dev.plist
…
<dict>
  …
  <key>Entitlements</key>
  <dict>
    …
    <key>application-identifier</key>
    <string>SKMME9E2Y8.com.example.apple-samplecode.ProfileExplainer</string>
    …
  </dict>
  …
</dict>
</plist>
```

This property holds an App ID, composed of an App ID prefix (`SKMME9E2Y8` in this example) and a bundle ID (`com.example.apple-samplecode.ProfileExplainer`).

It is also possible to create a profile for a wildcard App ID:

```
% security cms -D -i Profile_Explainer_Wild_iOS_Dev.mobileprovision -o Profile_Explainer_Wild_iOS_Dev.plist
% cat Profile_Explainer_Wild_iOS_Dev.plist
<dict>
  …
  <key>Entitlements</key>
  <dict>
    …
    <key>application-identifier</key>
    <string>SKMME9E2Y8.com.example.apple-samplecode.*</string>
    …
  </dict>
  …
</dict>
</plist>
```

This profile is valid for any App ID starting with `SKMME9E2Y8.com.example.apple-samplecode.`.

[1] On macOS this is `Entitlements` > `com.apple.application-identifier`.

The Where

Most profiles apply to a specific list of devices. This is encoded in the `ProvisionedDevices` property:

```
% cat Profile_Explainer_iOS_Dev.plist
…
<dict>
  …
  <key>ProvisionedDevices</key>
  <array>
    <string>00008030-00154452…</string>
  </array>
  …
</dict>
</plist>
```

App Store distribution profiles have no `ProvisionedDevices` property because you can't run a distribution-signed app locally.

Developer ID and In-House (Enterprise) distribution profiles have the `ProvisionsAllDevices` property, indicating that they apply to all devices.

The When

Every profile has an `ExpirationDate` property which limits how long the profile remains valid. For example:

```
% cat Profile_Explainer_iOS_Dev.plist
…
<dict>
  …
  <key>ExpirationDate</key>
  <date>2022-07-23T14:30:34Z</date>
  …
</dict>
</plist>
```

Developer ID profiles do not expire and thus their expiration date is far in the future.

The How

Every profile has an `Entitlements` property which authorises the app to use specific entitlements. For example:

```
% cat Profile_Explainer_iOS_Dev.plist
…
<key>Entitlements</key>
<dict>
  <key>application-identifier</key>
  <string>SKMME9E2Y8.com.example.apple-samplecode.ProfileExplainer</string>
  <key>keychain-access-groups</key>
  <array>
    <string>SKMME9E2Y8.*</string>
    <string>com.apple.token</string>
  </array>
  <key>get-task-allow</key>
  <true/>
  <key>com.apple.developer.team-identifier</key>
  <string>SKMME9E2Y8</string>
</dict>
```

The entitlements in the profile act as an allowlist. This is not the same as the entitlements claimed by the app. To actually claim an entitlement you must include the entitlement in the app's code signature.

Every entitlement claimed by the app must be in the profile's allowlist [1] but the reverse is not true. It's fine for the allowlist to include entitlements that the app does not claim.

The wildcard syntax only make sense in a profile's allowlist. In the above example, `SKMME9E2Y8.*` means that the app can claim any keychain access group starting with `SKMME9E2Y8`. Wildcards do not make sense in the app's code signature.

To dump the entitlements claimed by the the app, use `codesign` with the `--entitlements` argument:

```
% codesign -d --entitlements :- ProfileExplainer.app
…
<dict>
  <key>application-identifier</key>
  <string>SKMME9E2Y8.com.example.apple-samplecode.ProfileExplainer</string>
  <key>keychain-access-groups</key>
  <array>
    <string>SKMME9E2Y8.com.example.apple-samplecode.ProfileExplainer</string>
  </array>
  <key>get-task-allow</key>
  <true/>
  <key>com.apple.developer.team-identifier</key>
  <string>SKMME9E2Y8</string>
</dict>
</plist>
```

As you can see, every entitlement claimed here is allowed by the profile, and thus the app should run. Note that the `keychain-access-groups` value, `SKMME9E2Y8.com.example.apple-samplecode.ProfileExplainer`, starts with `SKMME9E2Y8.*` and thus is allowed by the wildcard.

[1] Except on macOS, as discussed in the next section.

Entitlements on the Mac

A macOS app can claim certain entitlements without them being authorised by a provisioning profile. These unrestricted entitlements include:

- `com.apple.security.get-task-allow` [1]
- `com.apple.security.application-groups` [2]
- Those used to enable and configure the App Sandbox
- Those used to configure the Hardened Runtime

Other entitlements must be allowed by a provisioning profile, just like on iOS. This is an important security feature on macOS. For example, the fact that the `keychain-access-groups` entitlement must be allowlisted by a profile means that other developers can't impersonate your app in order to steal its keychain items.

[1] On iOS this is named `get-task-allow` and, as with all entitlements on iOS, must be allowlisted by the profile.

[2] App Groups work differently on macOS and iOS; see this post for the gory details.

Profile Location

Historically it was common to install provisioning profiles on the device as a whole (in Settings on iOS or System Preferences > Profiles on the Mac). That's still possible, but standard practice is to embed the profile within the app itself:

- iOS expects to find the profile at `MyApp.app/embedded.mobileprovision`.
- macOS expects to find the profile at `MyApp.app/Contents/embedded.provisionprofile`.

Note The platforms use different file name extensions for these profiles (`.mobileprovision` on iOS, `.provisionprofile` on macOS)

App Store apps do not contain an embedded provisioning profile because the App Store checks that the app is signed and provisioned correctly as part of its app ingestion process.

Provisioning Profiles Code Signing

Reply

Posted 1 year ago by eskimo

Add a Comment