Développement Apple à l'Ircam

Choses à savoir sur les certificats, le codesigning, la notarisation et autres gros mots…

Le développement sous Apple (Mac OS ou iOS) ressemble de plus en plus à un parcours du combattant dans un univers passablement paranoïaque. Puisse ce petit guide vous éclairer dans les situations les plus courantes…

⚠ *En naviguant sur le site developer d'Apple, il vous sera fréquemment demandé des mots de passe de type "identification à deux facteurs" envoyés à la volée par SMS… Ayez votre téléphone portable à portée de main...*

Certificats

Pour développer sous Mac (le plus souvent avec Xcode), vous avez sans doute besoin d'un certificat lié à votre compte Apple Ircam, en tant membre de la "team"

INST RECHER COORD ACOUST MUSICALE

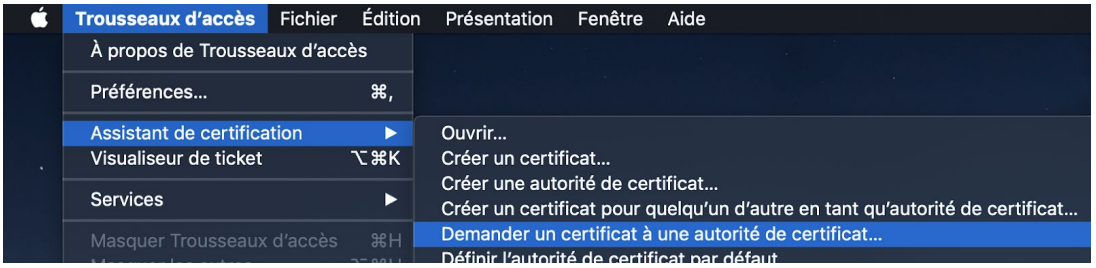
Il y a principalement deux situations :

Développement sans distribution

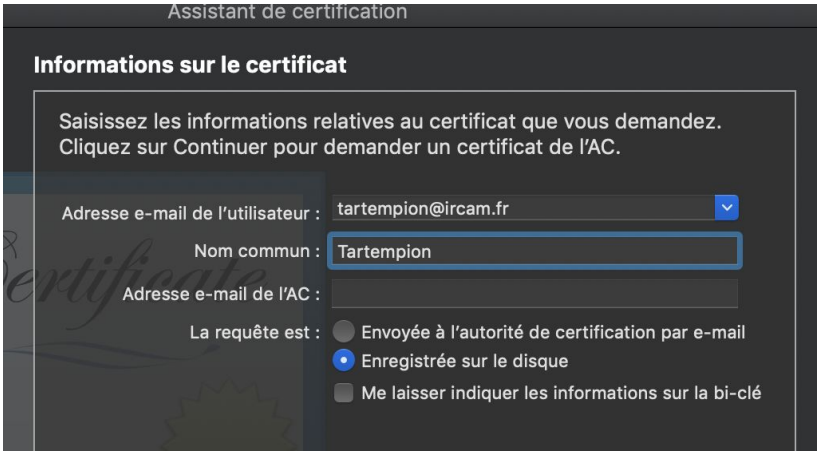
*C'est quand vous développez dans votre coin des bundles (apps ou librairies) ou des outils en ligne de commande, **sans les distribuer**, par exemple dans un but de recherche ou de prototypage*.
Dans ce cas vous pouvez vous-même créer un certificat personnel directement à partir de l'interface web d'Apple, parmi ces trois types :

1) Apple Development (toutes cibles, mais ne marche qu'avec Xcode 11 et ultérieur !)
2) iOS App Development (iOS uniquement)
3) Mac Development (Mac OS uniquement)

Comment faire ? Pour créer un certificat, vous devez uploader un "CSR" (Certificat Signing Request) généré depuis votre Keychain :



Il faut saisir votre adresse email et choisir "enregistrer sur le disque" :



Puis uploadez le CSR sur la page web comme demandé.

Enfin, téléchargez le certificat généré et installez-le dans la keychain.

💡 *S'il s'agit de développement macOS et qu'il ne sort pas de votre machine, vous n'avez même pas besoin de tout ça et pouvez ne rien signer du tout.* Mais ce n'est pas le cas pour pour iOS !

Développement avec distribution

C'est la situation a priori la plus délicate, donc quand vous développez et *distribuez* (typiquement via le Forum) des logiciels, librairies ou outils ligne de commande. Selon le cas, il vous faut un certificat de type :

1) **Developer ID Application** (nécessairement associé à une clef privée dans la keychain). Il vous permet de signer des **executables** (app ou librairies) ou des **.dmg** à partir de Xcode ou via la ligne de commande `codesign`.
2) **Developer ID Installer** (également associé à une clef privée). Il vous permet de signer des **packages d'installation** de type .pkg
3) **iPhone Distribution** (associé à une clef *et* à un profil) qui va vous permettre de signer et de publier des apps sur l'App Store. *Voir ce cas en détail plus bas.*

Comment faire ? Vous ne pouvez pas générer vous-même[1] ce type de certificats, et il faut donc en faire la demande à system@ircam.fr (**Olivier Labat**) en justifiant brièvement l'usage que vous en aurez : par exemple projet, nom de l'application, contexte de recherche etc.

La nouveauté est que ces 3 certificats, créés par Olivier, doivent être à présent partagés par tous les développeurs de l'Ircam. Avant, chacun essayait de se débattre dans son coin...

Vous allez recevoir par email un fichier sécurisé de type ".p12" (le mot de passe attenant vous sera aussi communiqué) qui, une bonne fois pour toutes, déploiera dans votre keychain le ou les bons certificats munis de leur clef privée respective.

- *Récupérez le fichier **.p12** de l'email et double-cliquez dessus : la keychain s'ouvre.*
- *Saisissez le mot de passe demandé. Vous devriez alors voir dans votre keychain de nouveaux certificats parmi les suivants :*
  - Developer ID Application: INST RECHER COORD ACOUST MUSICALE (3BD2P55TR2)
  - Developer ID Installer: INST RECHER COORD ACOUST MUSICALE (3BD2P55TR2)
  - iPhone Distribution: INST RECHER COORD ACOUST MUSICALE (3BD2P55TR2)

⚠ *Parfois la keychain n'affiche pas immédiatement tel certificat récemment installé. On peut rafraîchir en quittant puis relançant la keychain.*

Les certificats servent à "code signer" les executables et les .dmg, à "product signer" les .pkg, afin de préparer tous ces éléments à la "notarisation".

Certains pourront peut-être s'émouvoir de ce que les certificats soient signés par l'administrateur de la team (Olivier) et non par soi-même. Il faut savoir qu'en définitive la signature d'un fichier porte l'identifiant de la team (3BD2P55TR2) et non un nom propre.

[1] Si toutefois vous êtes "administrateur" sur le compte Apple Ircam, vous le pouvez techniquement, mais ce n'est pas du tout recommandé ! (il y a une limite incompressible de 5 certificats par type et par team…)

Profiles (iOS)

Cette section sera étoffée au fil du temps... et des galères rencontrées !

Dans le contexte de l'Ircam (et à ce jour), la notion de *profile* n'a de sens que pour les apps iOS, car en effet aucun logiciel pour mac OS n'est pour l'instant distribué sur le Mac Store ou n'utilise de services Apple spéciaux (de type Push etc.)

A minima un profile iOS associe deux choses :

- Un certificat de type **iPhone Distribution** (voir plus haut)
- Un **identifier** qui doit s'accorder avec l'identifiant (au sens Xcode) de votre app iOS. Cet *identifier* peut être plus ou moins générique (*wildcard*), correspondant tantôt à un seul produit ou tantôt au contraire à une gamme plus ou moins large…

La notion d' "identifier"

Si votre application s'appelle **Truc** et que vous faites partie de l'équipe **Bidule**, son identifiant (au sens Xcode) pourra être par exemple `com.ircam.bidule.truc`, et *l'identifier* pourra être `com.ircam.*`, ou plus précisément `com.ircam.bidule.*` ou encore `com.ircam.bidule.truc`. Les produits Truc1 et Truc2 de l'équipe Bidule pourront être développés et distribués selon le même profile associé à l'identifier générique (*wildcard*) `com.ircam.bidule.*`, ou plus générique encore `com.ircam.*`

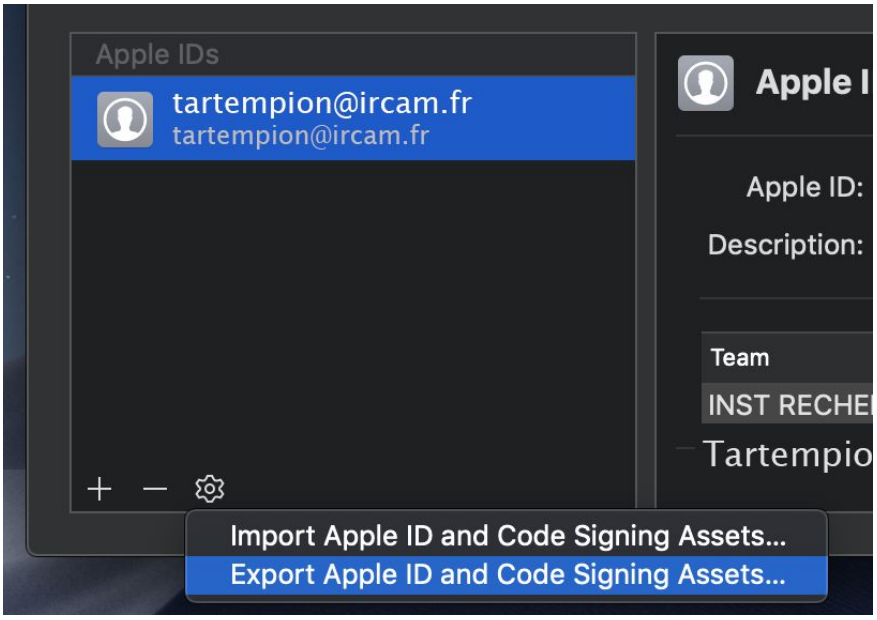
Si vous démarrez un projet iOS destiné à la distribution, il vaut mieux contacter Olivier Labat pour la mise au point du profile le mieux adapté.

⚠ *Il faut aussi que vos machines de test (iPad, iPhone…) soient référencées et associées au compte Team Ircam. Pour cela vous devez envoyer le UUID d'une machine à Olivier Labat[2] qui se chargera de l'ajouter en tant que device de développement.*

Changement de machine de développement

(ou déployement de la même config sur plusieurs machines…)
Il y a une procédure fort commode dans Xcode qui permet d'exporter toutes les données utiles d'un compte Apple. Cela suppose évidemment que vous ayez votre identifiant Apple présent en tant que compte dans Xcode.
Allez dans **`Preferences > Account :`**

[2] En tant que copier-coller (de iTunes par exemple) et non pas comme copie d'écran !



Et là vous pouvez exporter votre compte sous la forme d'un fichier unique `.developerprofile` sécurisé par mot de passe. Vous pouvez alors l'importer sur n'importe quelle machine via Xcode.

Code signing

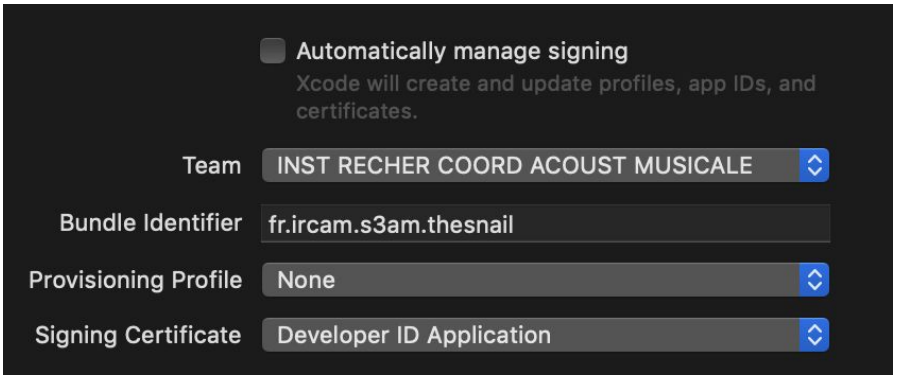
Le *code signing* est un mode de signature sécurisée qui utilise un certificat **Developer ID Application** et réservée...

- aux executables en ligne de commande
- aux bundles, qu'ils soient des applications ou des librairies (ex : objets Max)
- aux **.dmg**

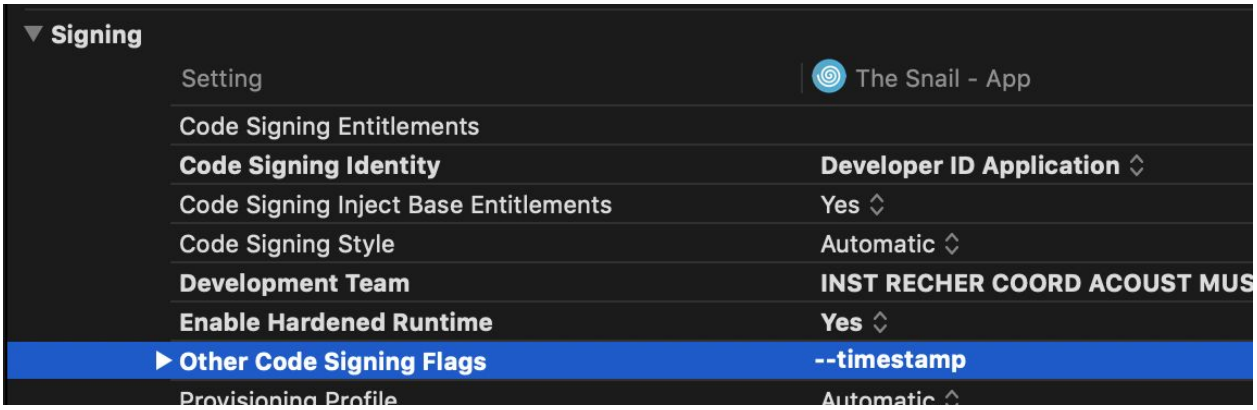
Le code signing peut se faire de deux façons :

Code signing dans Xcode

Les options du code signing sont regroupées dans les paramètres du projet.
On doit décocher l'option "Automatically Manage Signing" et parvenir à peu près à ceci :

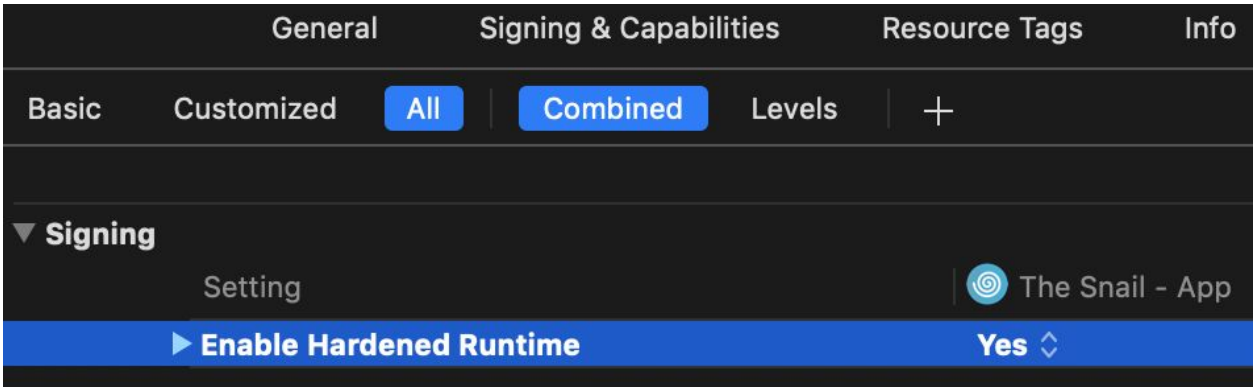


⚠ *Il faut ajouter le flag important `--timestamp` qui par défaut est désactivé par XCode !*



L'option Hardened Runtime

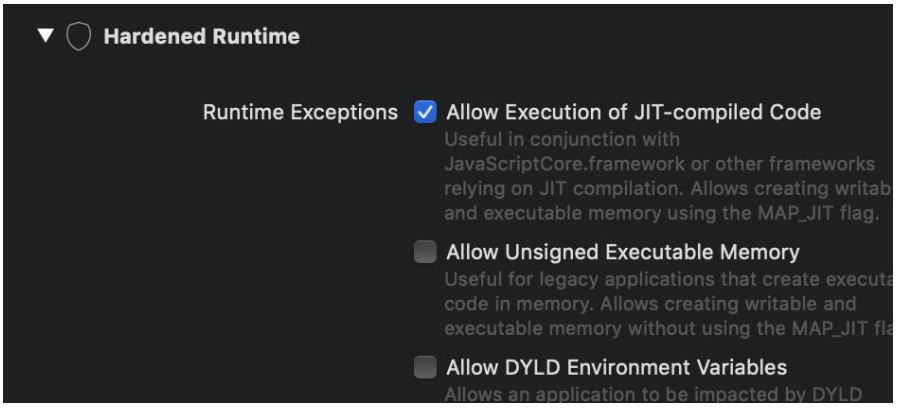
Par dessus le marché, vous devez activer dans Xcode l'option Hardened Runtime *pour tous vos exectutables standalone* de votre projet :



⚠ *Sans cette option, la notarisation échouera !*

Entitlements

Les *entitlements* sont des droits accordés à un exécutable au moment de la signature. Par exemple accéder au micro, faire du code Just-in-Time etc. L'activation de "Hardened Runtime" ouvre un certain nombre de ces droits :



Voir cette page et encore celle-ci pour plus de détails.

NB : si un exécutable charge des librairies, l'usage de celles-ci sera en principe limité par les entitlements de l'exécutable.

Code signing en ligne de commande

On utilise l'outil `codesign` en ligne de commande. C'est assez simple :

```
codesign --deep --timestamp -s "Developer ID Application: INST RECHER COORD ACOUST MUSICALE" /path/Truc.app
```

Options utiles de l'outil codesign

`--force` : force la signature même si l'objet est déjà signé.

`--deep` : permet de signer récursivement, par exemple un bundle.

`--options=runtime` : ceci correspond à l'option hardened runtime de Xcode. Si on utilise cette option, il faut aussi ajouter un fichier adhoc de type **.entitlements** et l'option :
`--entitlements truc.entitlements`
Le fichier aura par exemple cette tête :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>com.apple.security.cs.allow-jit</key>
    <true/>
    <key>com.apple.security.cs.disable-library-validation</key>
    <true/>
</dict>
</plist>
```

Vérification de la signature

Pour obtenir les informations au niveau le plus verbeux possible :

```
codesign -dvvvv --deep --strict --no-legacy-signing Truc.app
```

Vérification des 'entitlements'

```
codesign -d --entitlements :- Truc.app
```

Production d'un .dmg

Comme on l'a vu, les .dmg doivent être également signés. Rappelons la ligne de commande qui permet la fabrication d'un .dmg à partir d'un répertoire :

```
hdiutil create /path/MyArchive.dmg -fs HFS+ -srcfolder /path/SourceFolder -ov
```

⚠ *Notez l'importante option* **`-fs HFS+`** *qui assure la compatibiité pour macOS avant 10.13 !*

Product signing

Le *product signing* sert essentiellement à signer les packages de type **.pkg**
Le product signing ne se fait qu'en ligne de commande, avec l'outil `productsign` et un certificat **Developer ID Installer** :

```
productsign --timestamp --sign "Developer ID Installer: INST RECHER COORD
ACOUST MUSICALE" /path/Truc.pkg /path/signed/Truc.pkg
```

Pour vérifier la signature :
```
pkgutil --check-signature /path/signed/Truc.pkg
```

Notarisation

La *notarisation* est une couche de sécurité supplémentaire qui est aujourd'hui inévitable dans macOS 10.15, alias Catalina.
Ne peuvent être notarisés que les fichiers de type **.dmg**, **.pkg** ou **.zip**[3].

NB : Il faut que tous les éléments d'une archive à notariser aient été préalablement et correctement codesignés...

"Mot de passe pour application"

Afin de notariser, vous devez d'abord générer un **mot de passe pour application** en vous connectant avec votre compte apple Ircam sur :
https://appleid.apple.com

Et ensuite sur la page, allez sur :

MOTS DE PASSE POUR APPLICATION
Générer un mot de passe...

Et là après avoir entré un "titre", un mot de passe de la forme **`abcd-wxyz-efgh-ijkl`** apparaît dans un simple popup...

Copiez ce mot de passe et conservez-le quelque part bien à l'abri…

NB : si vous vous plantez, vous pouvez toujours le régénérer...

[3] Cependant les .zip ne sont pas acceptés par l'outil `stapler`. Apple recommande alors de "stapler" chaque élément du .zip puis de refaire un .zip pour la distribution. Hmm… où est l'intérêt ?..

La ligne de commande pour notariser

La notarisation se fait uniquement par ligne de commande. C'est une requête avec upload du fichier à notariser vers un serveur d'Apple qui va en effectuer la vérification et renvoyer un identifiant unique, une sorte de "ticket".

```
xcrun altool --notarize-app -f /path/Truc.dmg --primary-bundle-id [bundle id] -u [mon appleID] -p [mon password pour app]
```

- `bundle id` peut être un peu ce qu'on veut (en ASCII), pas besoin que ça corresponde à l'identifiant de l'application. Par exemple : **`ircam.notarization`**
- `mon appleID` est votre apple ID de type **`tartempion@ircam.fr`**
- `mon password pour app` est le mot de passe [généré plus haut](#) de type **`abcd-wxyz-efgh-ijkl`**

Une fois la notarisation lancée, il faut attendre sagement la réponse. Aucune "barre de progression" n'existe, et donc pour les gros fichiers ça peut prendre un certain temps… Si l'upload s'est bien passé, on reçoit ceci :

```
No errors uploading 'Truc.dmg'.
RequestUUID = f68d10e1-3fbb-4afb-8865-2d6f13a5e086
```

Mais ça ne veut pas dire que ça a marché, ça serait bien trop facile ! Il faut interroger le ticket pour savoir ce qu'il en est.

Voir le résultat de la notarisation

```
xcrun altool --notarization-info f68d10e1-3fbb-4afb-8865-2d6f13a5e086 -u tartempion@ircam.fr -p abcd-wxyz-efgh-ijkl
```

⚠ *Parfois le log n'est pas immédiatement disponible… Il faut alors attendre une dizaine de seconde et renouveler la demande d'info...*

Si ça a raté, on obtiendra par exemple :

```
No errors getting notarization info.

            Date: 2020-05-26 20:04:37 +0000
            Hash: f4e943d98f10290088a59d4c3758f26ec6b1981327ca760ec9d3ad25a7aae495
      LogFileURL: https://osxapps-ssl.itunes.apple.com/itunes-assets/Enigma113/v4/06/91/8a/06918a8d-f2f6-ef4d-6e6b-f040c12c24d1/developer_log.json?accessKey=1590718517_4572752107532115053_8sGD3WAtRCUgjBDujqKG4IrIu7IBOuZ5J9esnf1VPgvtpYZGTMIXIVTlJm9Uo1KA%2BxjrOk0cXxHLqkkdXkPSRe3kGmJVTLE6AGVilZGI5KDEkp5Sify1qvATF8UUwvyRw3RJ67oDE328pQBDb9eKenMrIGXOj1SYg7OFsvIsULA%3D
```

```
   RequestUUID: f68d10e1-3fbb-4afb-8865-2d6f13a5e086
      Status: invalid
   Status Code: 2
Status Message: Package Invalid
```

Consulter le log de notarisation

Pour consulter le log, il suffit de copier l'url à rallonge dans un navigateur et on obtient un fichier json, plus ou moins détaillé selon la complexité du fichier uploadé. Ce log est d'une grande utilité pour corriger un à un les problèmes :


```
{
  "logFormatVersion": 1,
  "jobId": "f68d10e1-3fbb-4afb-8865-2d6f13a5e086",
  "status": "Invalid",
  "statusSummary": "Archive contains critical validation errors",
  "statusCode": 4000,
  "archiveFilename": "Truc.dmg",
  "uploadDate": "2020-05-26T20:04:37Z",
  "sha256":
"f4e943d98f10290088a59d4c3758f26ec6b1981327ca760ec9d3ad25a7aae495",
  "ticketContents": null,
  "issues": [
      {
      "severity": "error",
      "code": null,
      "path": "Truc.dmg/truc",
      "message": "The executable does not have the hardened runtime
enabled.",
      "docUrl": null,
      "architecture": "x86_64"
      }
  ]
}
```


Par tâtonnements successifs (et chaque correction doit évidemment faire l'objet d'une nouvelle requête en notarisation !) et si on est chanceux, on finira par obtenir :

```
No errors getting notarization info.

            Date: 2020-05-26 20:41:37 +0000
            Hash:
294adc197719448af1661ebe51d4715a566bb81aa1dbcda03c756f5734855fed
      LogFileURL:
https://osxapps-ssl.itunes.apple.com/itunes-assets/Enigma113/v4/73/11/87/7311
87d6-e69d-4624-dbeb-a8446323644f/developer_log.json?accessKey=1590722530_2286
```

```
640616520590209_eTsuWUMqxXFqv4YX31pzEWi5QP%2FTr2KgeiDLzcvucV15T%2Bep6KgkBXpe2
%2F1GeQoW%2BXCMJTCICyMx%2Bc7dAcXknGHv1Hf1uFycaFz2lCxdgRE%2FdPyc9Sndb8odypMGp1
Tv7eioWlc8zdD%2Bo7Ql1jEmkyw6WlDzTc3q2%2BN2t3kNNCc%3D
   RequestUUID: 152db8c7-963f-4e4d-a8a9-1991935ad108
      Status: success
   Status Code: 0
Status Message: Package Approved
```

NB : on obtient également une confirmation par email.

“Tamponner” la notarisation sur le fichier

Pour achever le processus, on doit utiliser l’outil `xcrun stapler` :

```
xcrun stapler staple /path/Truc.dmg
```

Et normalement on obtient :

```
Processing: /path/Truc.dmg
The staple and validate action worked!
```

Et le fichier d’installation est enfin prêt à être distribué 🙂

Vérifier le fichier .dmg vis-à-vis de Gatekeeper

Cette ligne de commande (avec connexion internet !) permet de simuler la vérification de Gatekeeper :

```
spctl --assess --ignore-cache --type open --context context:primary-signature
--verbose /path/Truc.dmg
```

Si tout se passe bien, on obtient :

```
/path/Truc.dmg: accepted
source=Developer ID
```

💡 Pour en savoir plus sur la notarisation sur le site d’Apple.

Quarantaine

La notion de mise en *quarantaine* d’un fichier a été introduite dans macOS 10.5, donc ça ne date pas d’hier...
La quarantaine est un attribut ajouté par le système pour signaler qu’un fichier est potentiellement suspect. L’attribut s’appelle `com.apple.quarantine` et est géré par l’outil ligne de commande `xattr`.
Pour savoir si un fichier est flagué quarantaine, il suffit de taper la ligne de commande :

```
xattr /path/to/my-external.mxo
```

S’il ne l’est pas, la réponse est vide, et s’il l’est on obtient :

`com.apple.quarantine`

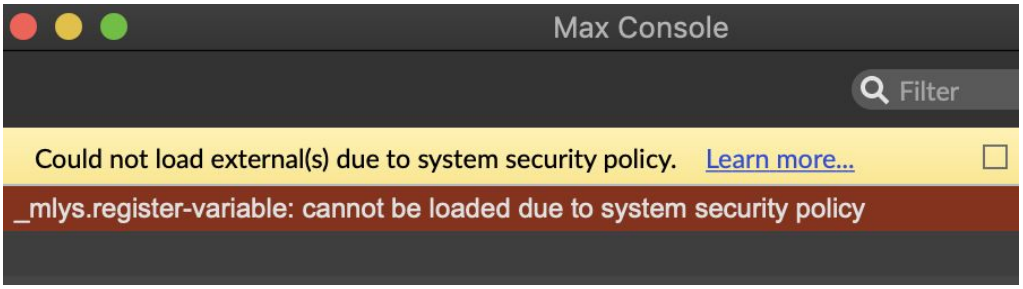
Tout fichier téléchargé via un navigateur ou l'email est flagué quarantaine. Le fait qu'il soit correctement signé et notarisé n'y change rien.

NB : les fichiers téléchargés par ftp échappent généralement à la quarantaine.

💡 Un package d'installation de type **.pkg**, bien que mis en quarantaine par le fait du téléchargement, *n'aura pas ses éléments installés mis en quarantaine*.

Problème spécifique à Max 8

Max 8.1.3 fait du zèle avec les système de sécurité d'Apple : tout *external* flagué quarantaine est rejeté au chargement, même s'il est signé et notarisé comme il faut (cette condition étant d'ailleurs nécessaire dans Catalina) :



Ce problème peut être contourné manuellement par un script qui enlèvera le flag "quarantine", comme ceci :

```
sudo xattr -r -d com.apple.quarantine /path/to/my-external.mxo
```

Passer à Max 8.1.4 (et ultérieur)

Max **8.1.4** a un système de vérification signature/quarantaine plus fin et moins bloquant et il est donc recommandé d'utiliser cette version.

Plateforme ARM

Voici un nouveau sujet bien épineux... On se souvient de la transition des processeurs PowerPC vers Intel (avec Rosetta notamment), on se souvient de l'abandon progressif du 32bit.

Ce qui arrive aujourd'hui, c'est le passage d'Intel vers **ARM**, pour tous les processeurs équipant les machines Apple.

Le macOS à venir Big Sur, "mac OS 11", intègre Rosetta 2 pour donc pouvoir faire tourner des apps Intel sur de futures machines ARM. Il existe en bêta.

Xcode 12 peut dors et déjà compiler du code universal Intel+ARM.

NB: Xcode 12 tourne aussi sur macOS 10.15 Catalina!

Un DTK ARM (machine de test) existe peut être demandé à Apple moyennant une location (il sera bon d'en avoir un mutualisé à l'Ircam…)

Enfin, il y a pas mal d'informations sur ARM+Apple dans cette page wiki et ici ou là.