#WWDC19

All About Notarization

Robert Kendall-Kuppe, Security Engineering and Architecture
Garrett Jacobson, Security Engineering and Architecture

What is notarization?

Application requirements

Workflows

What is notarization?

Application requirements

Workflows

Notarization

Notarization

Identify and block malicious software prior to distribution

Notarization

Identify and block malicious software prior to distribution

Extension of the Developer ID program

Notarization

Identify and block malicious software prior to distribution

Extension of the Developer ID program

Developers control signing and distribution

Notarization

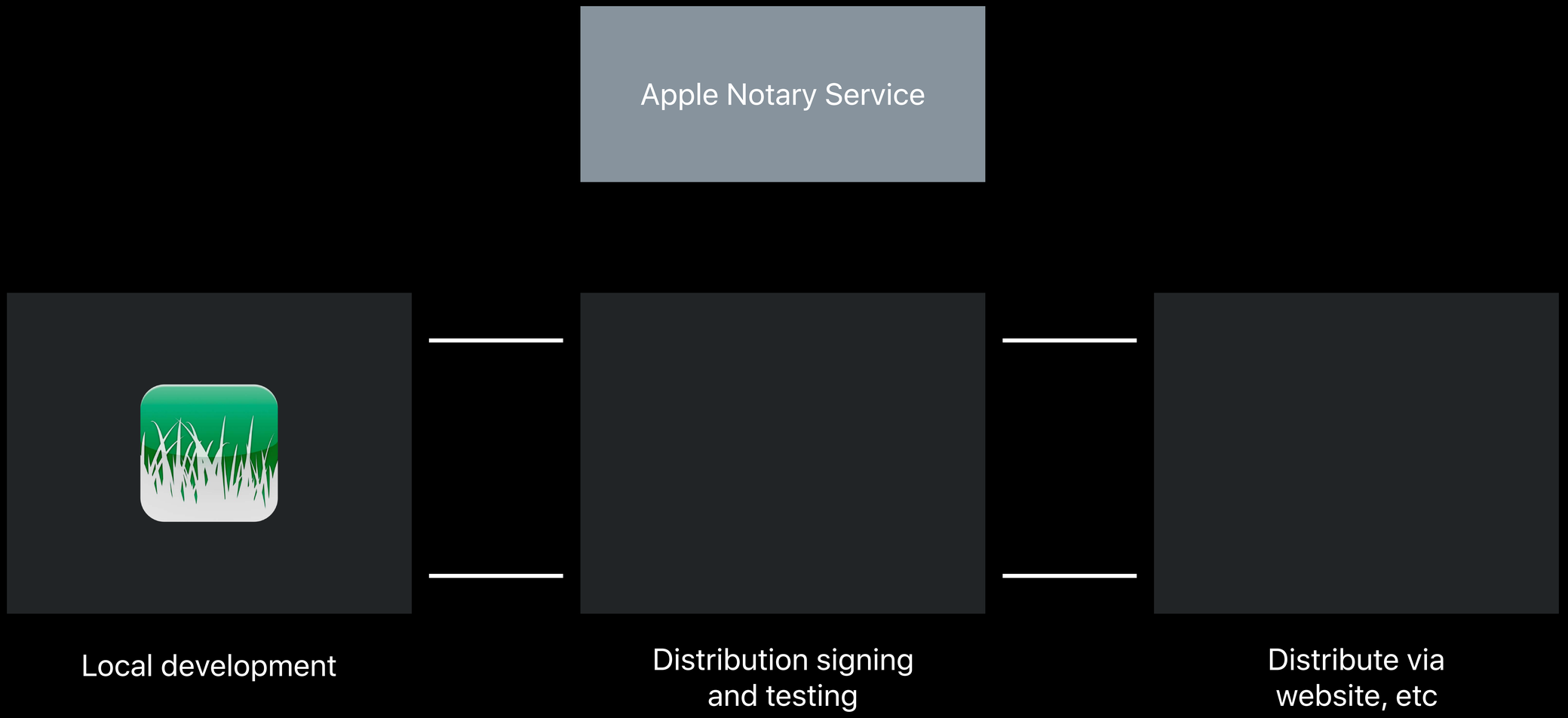
Identify and block malicious software prior to distribution

Extension of the Developer ID program

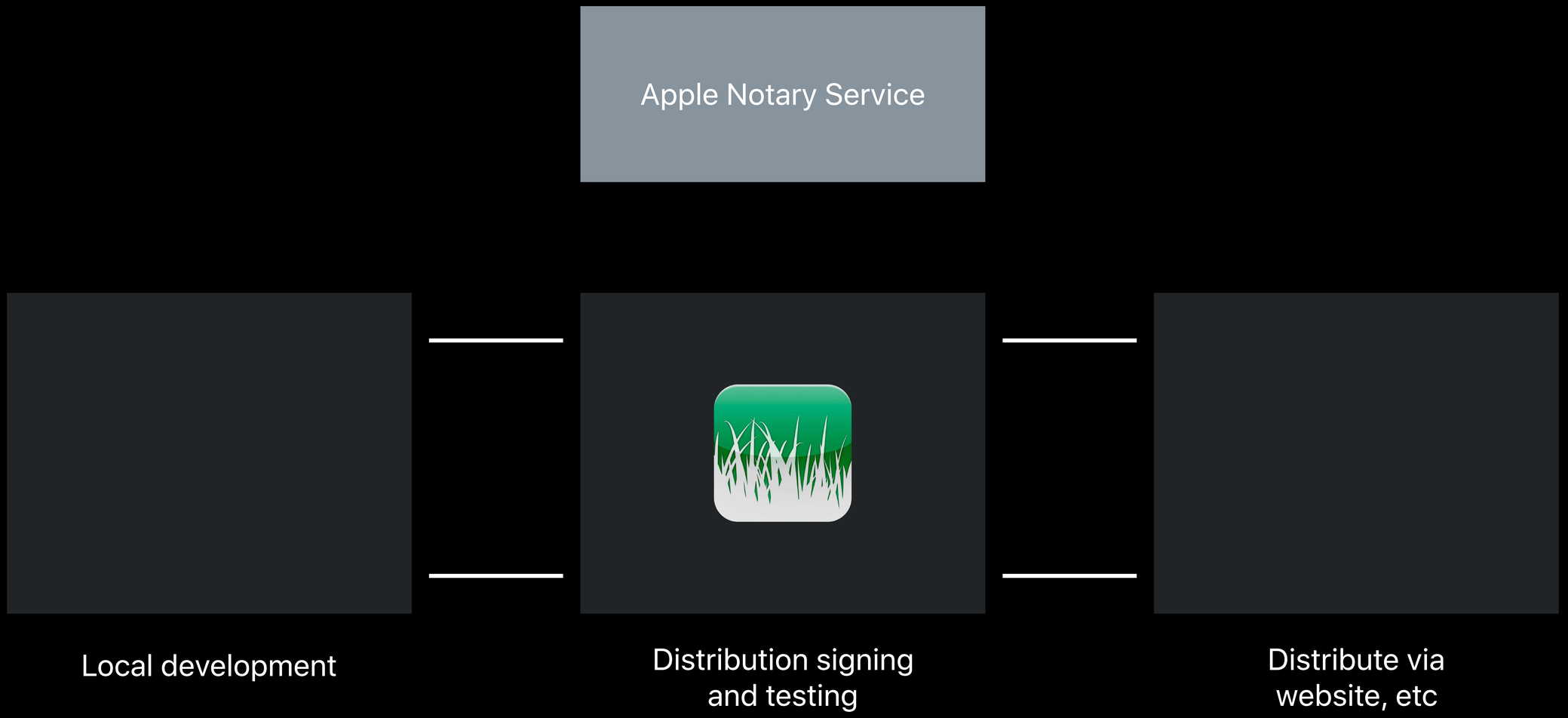
Developers control signing and distribution

Notary service performs automated security checks

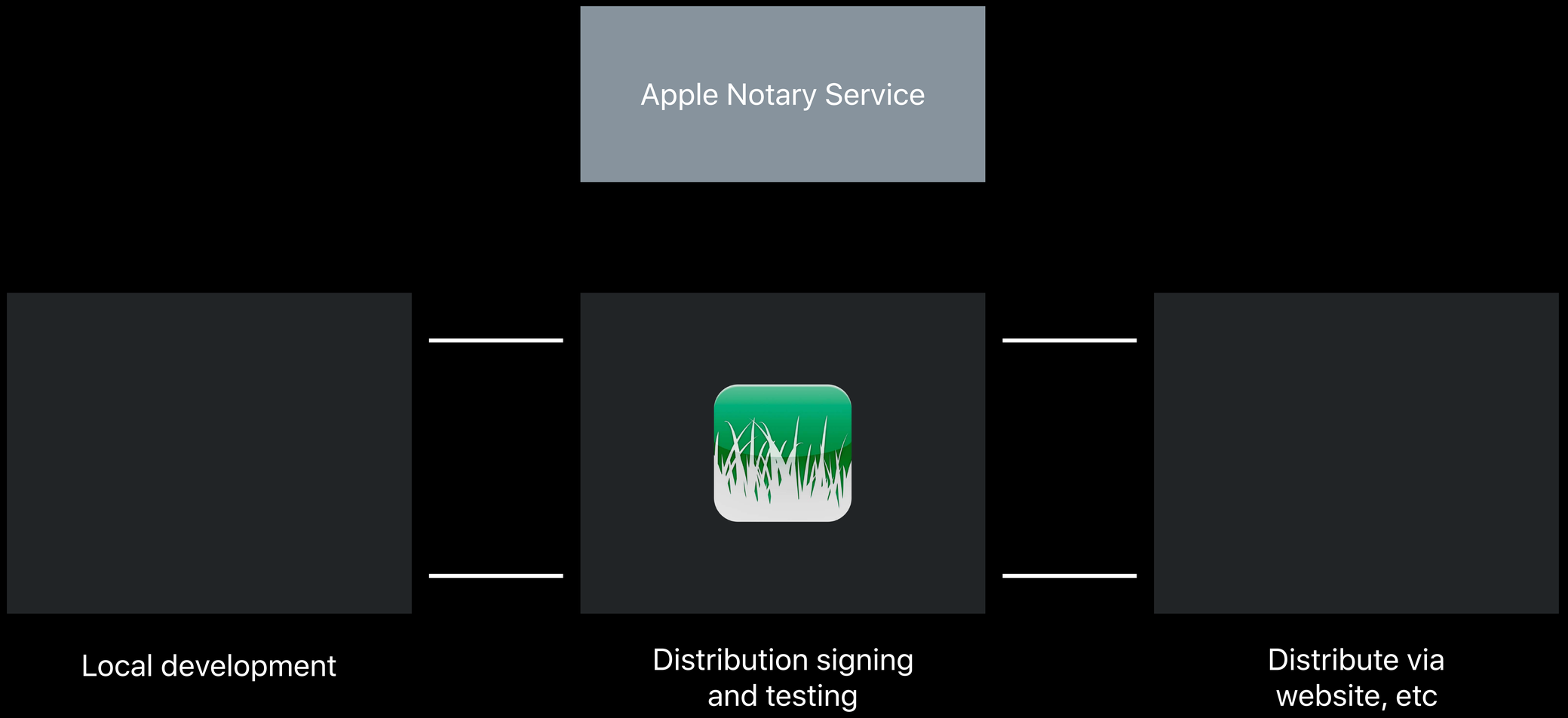
Notarization Process



Notarization Process

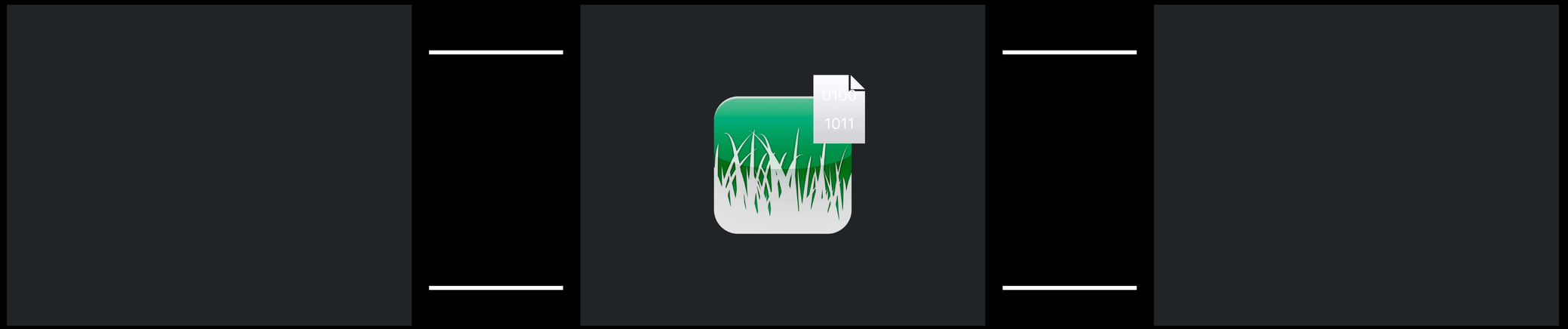


Notarization Process



Notarization Process

Apple Notary Service

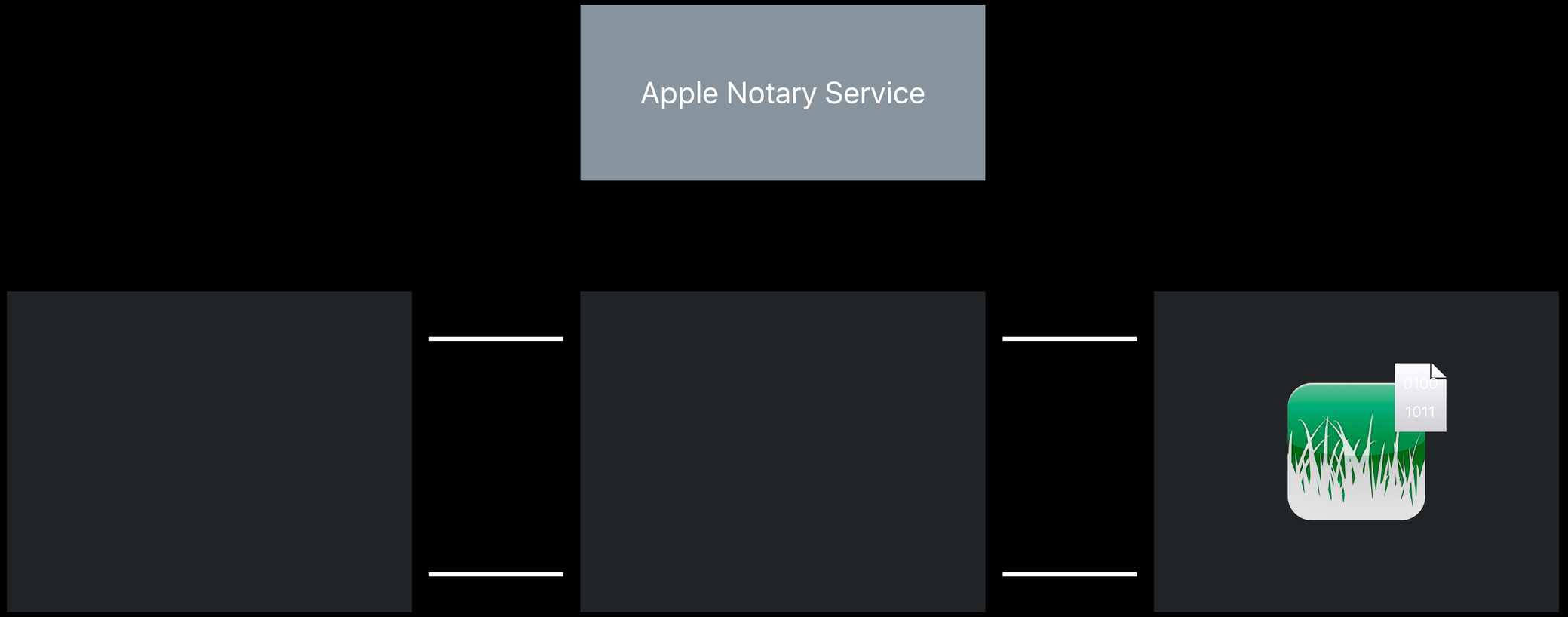


Local development

Distribution signing and testing

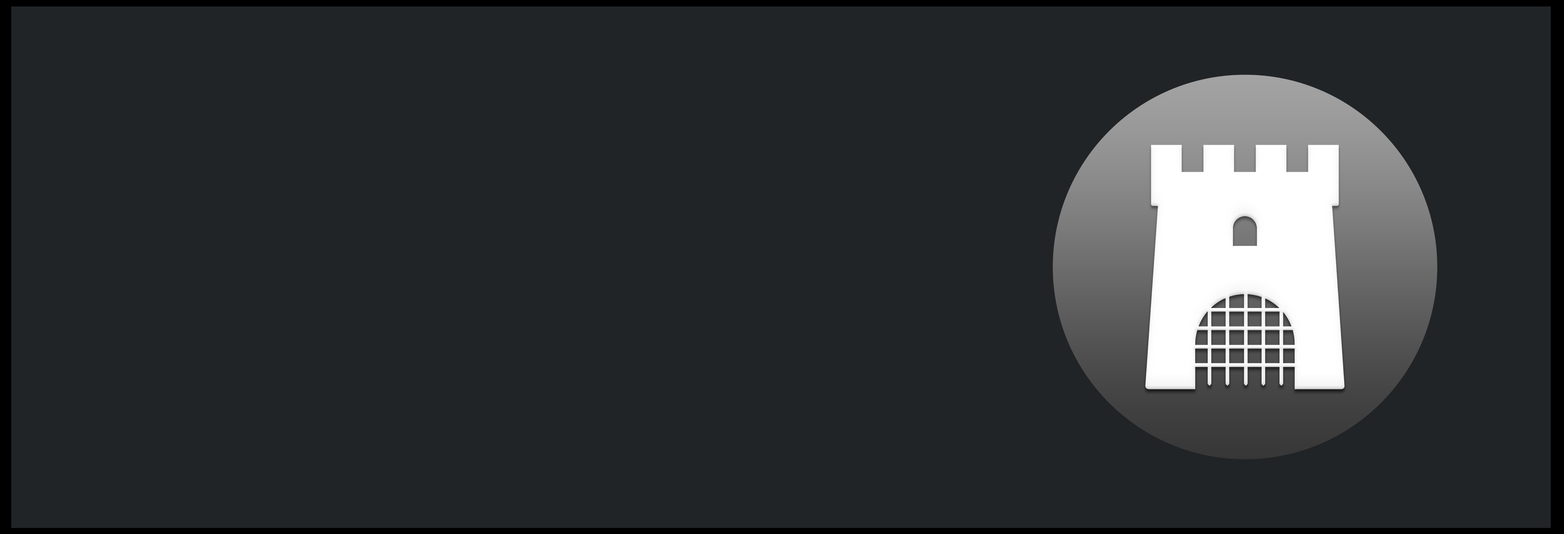
Distribute via website, etc

Notarization Process



Checking Notarization

Apple Notary Service

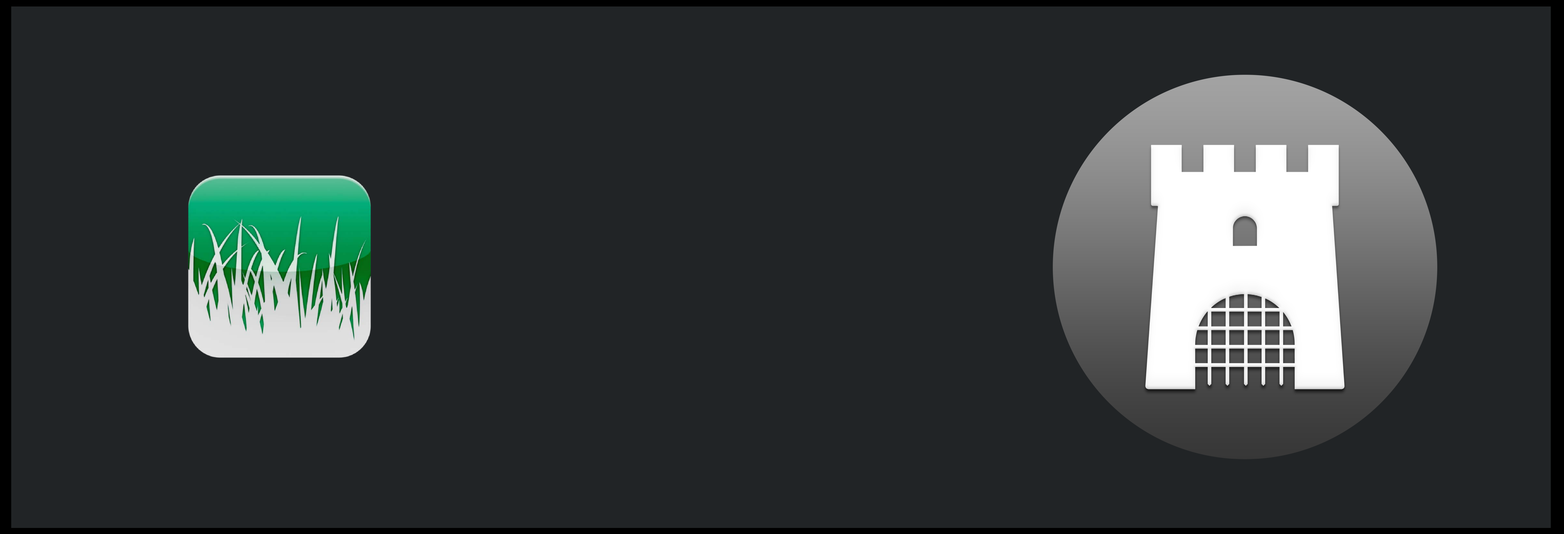


Checking Notarization

Apple Notary Service

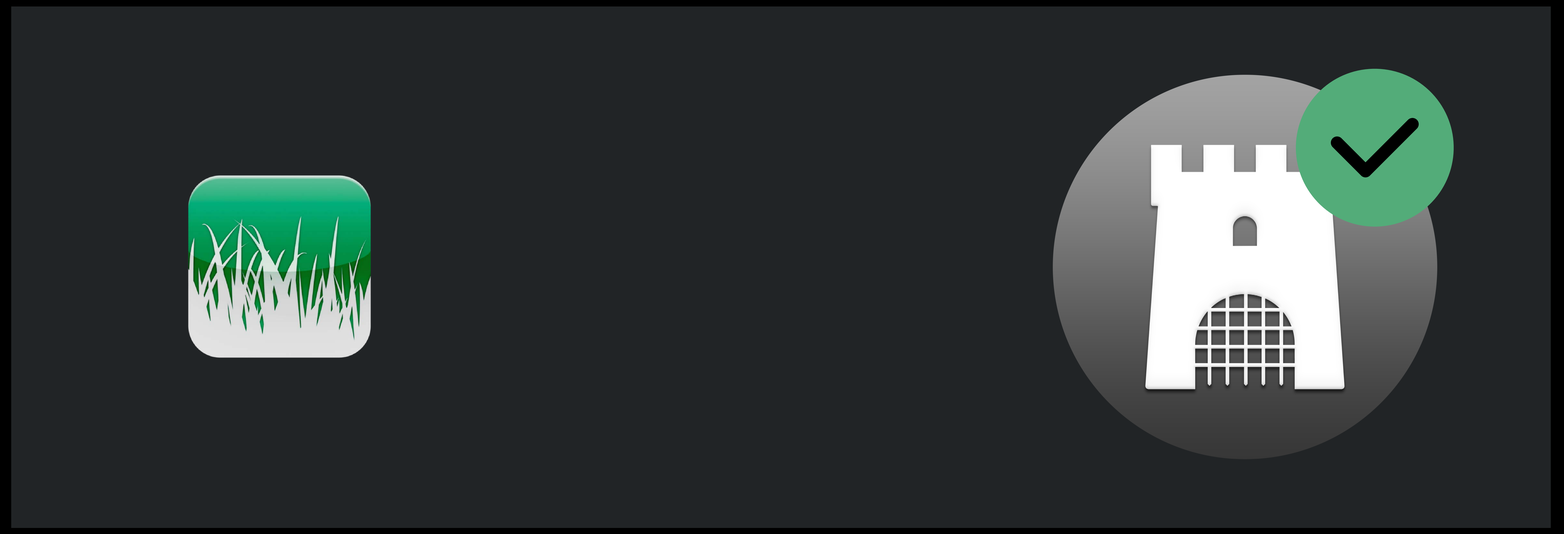
Checking Notarization

Apple Notary Service



Checking Notarization

Apple Notary Service



Not App Review

Notarization Benefits

Notarization Benefits

Help prevent you from inadvertently shipping a malicious dependency

Notarization Benefits

Help prevent you from inadvertently shipping a malicious dependency

Apps with the hardened runtime are more secure by default

Notarization Benefits

Help prevent you from inadvertently shipping a malicious dependency

Apps with the hardened runtime are more secure by default

Users are more likely to download and try new software

Notarization Benefits

Help prevent you from inadvertently shipping a malicious dependency

Apps with the hardened runtime are more secure by default

Users are more likely to download and try new software

Audit trail of software notarized by your Developer ID account

What is notarization?

Application requirements

Workflows

Application Requirements for Notarization

NEW

Previously distributed software can be submitted for notarization as-is

To protect your users, new software must adopt

- Complete and correct signing
- The Hardened Runtime

New software - signed on or after June 1, 2019

Complete and correct signing

Hardened Runtime

- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection
- Protected resource access

Complete and Correct Signing

Sign everything

Bundles

Mach-Os

Installer packages (.pkg)

Regardless of where they live in your product

Complete and Correct Signing

Signing configuration

Complete and Correct Signing

Signing configuration

Bundles, Mach-Os and “Code” files must

- be signed with your Developer ID Application Certificate
- include a secure timestamp

Complete and Correct Signing

Signing configuration

Bundles, Mach-Os and “Code” files must

- be signed with your Developer ID Application Certificate
- include a secure timestamp

Executables must opt into the Hardened Runtime

Complete and Correct Signing

Signing configuration

Bundles, Mach-Os and “Code” files must

- be signed with your Developer ID Application Certificate
- include a secure timestamp

Executables must opt into the Hardened Runtime

Installer packages (.pkg) must be signed with your Developer ID Installer Certificate

Complete and Correct Signing

Signing configuration

Bundles, Mach-Os and “Code” files must

- be signed with your Developer ID Application Certificate
- include a secure timestamp

Executables must opt into the Hardened Runtime

Installer packages (.pkg) must be signed with your Developer ID Installer Certificate

If you sign your disk images (.dmg), they must be signed with your Developer ID Application Certificate and include a secure timestamp

Complete and Correct Signing

Xcode does it for you

Use Xcode to manage the packaging and organization of your code

Turn on Automatic Code Signing

Be careful with

- Script build phases
- Copy build phases

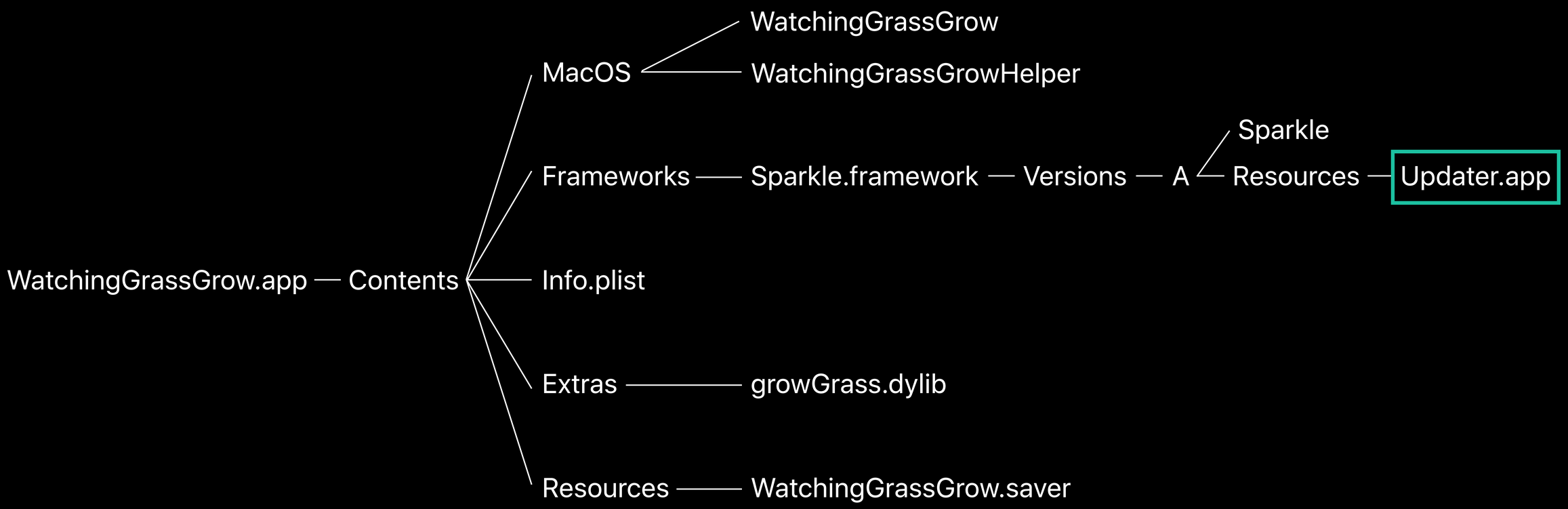
Complete and Correct Signing

Code places

Location	Description
Contents	Top content directory of the bundle
Contents/MacOS	Helper apps and tools
Contents/Frameworks	Frameworks, dylibs
Contents/PlugIns	Plug-ins, both loadable and Extensions
Contents/XPCServices	XPC services
Contents/Helpers	Helper apps and tools
Contents/Library/Automator	Automator actions
Contents/Library/Spotlight	Spotlight importers
Contents/Library/LoginItems	Installable login items

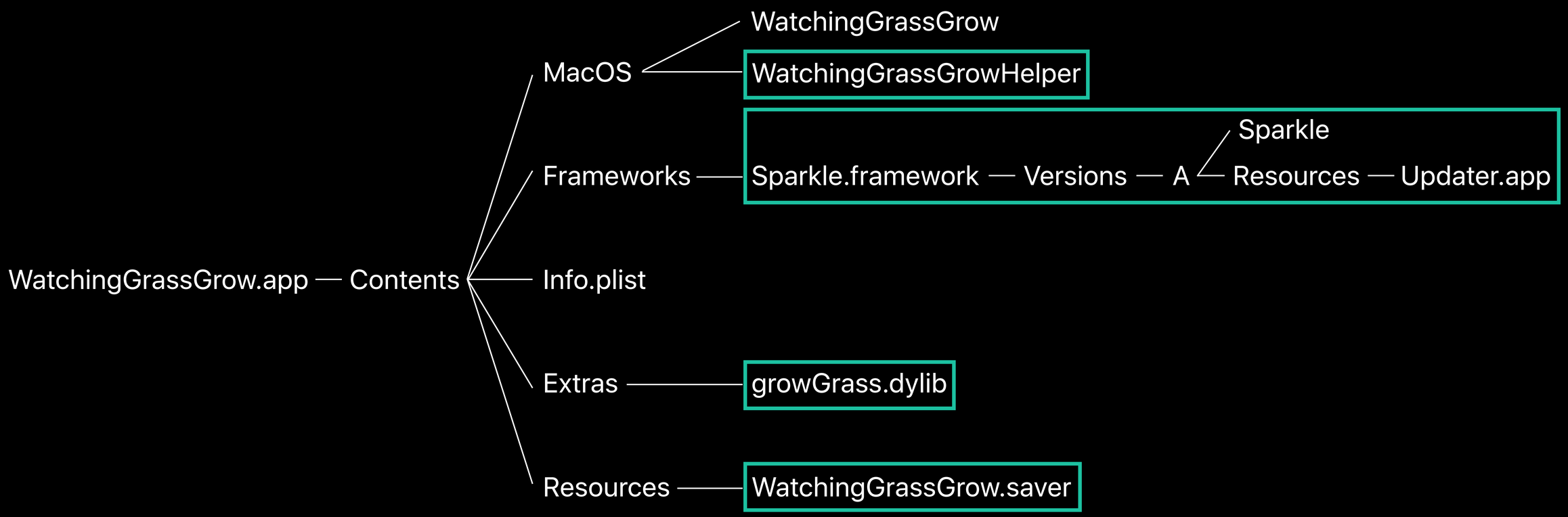
Complete and Correct Signing

Inside-out signing



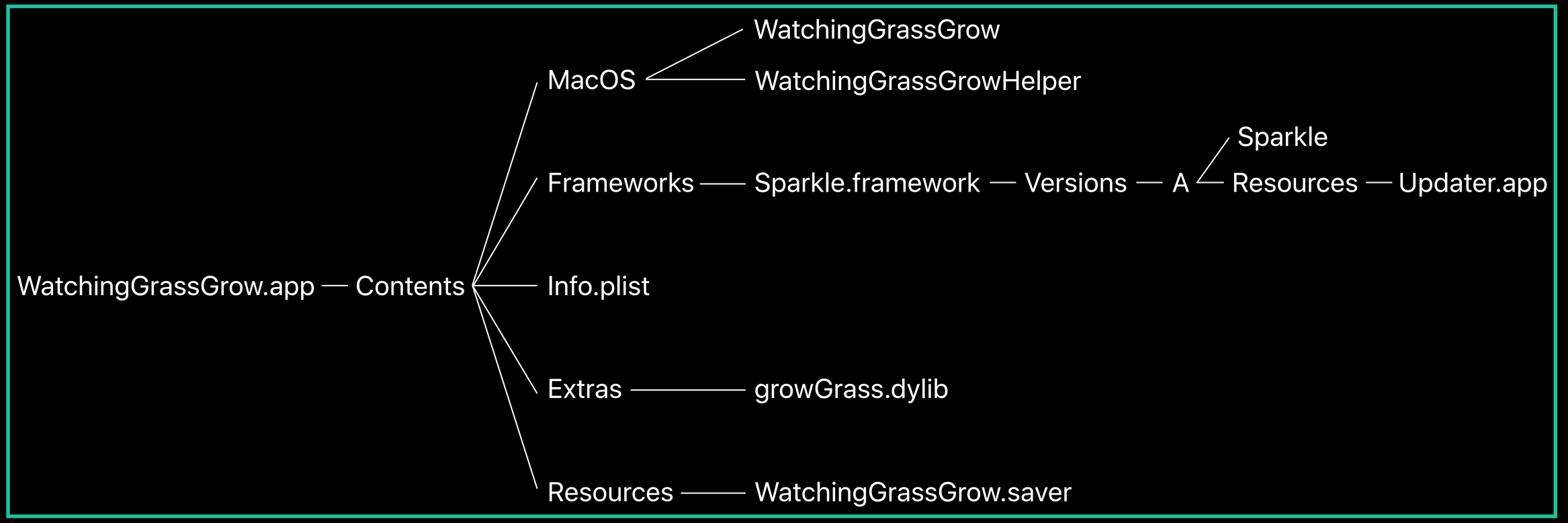
Complete and Correct Signing

Inside-out signing



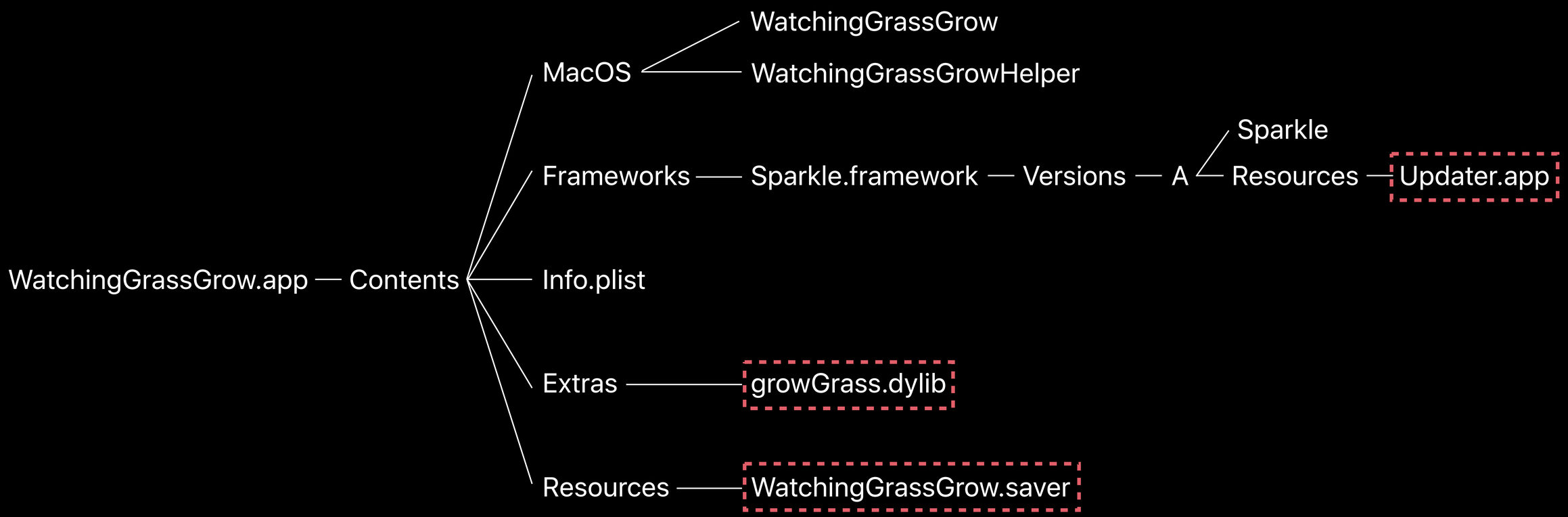
Complete and Correct Signing

Inside-out signing



Complete and Correct Signing

“-- deep” is not enough



See TN2206: Code Signing in depth for more Information

Complete and Correct Signing

Do not invalidate your signature

Complete and Correct Signing

Do not invalidate your signature

Never change files in your signed bundles except during installation or update

Complete and Correct Signing

Do not invalidate your signature

Never change files in your signed bundles except during installation or update

After an update, make sure your product still has valid signatures and is notarized

Complete and correct signing

Hardened Runtime

- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection
- Protected resource access

Adopting the Hardened Runtime

Why?

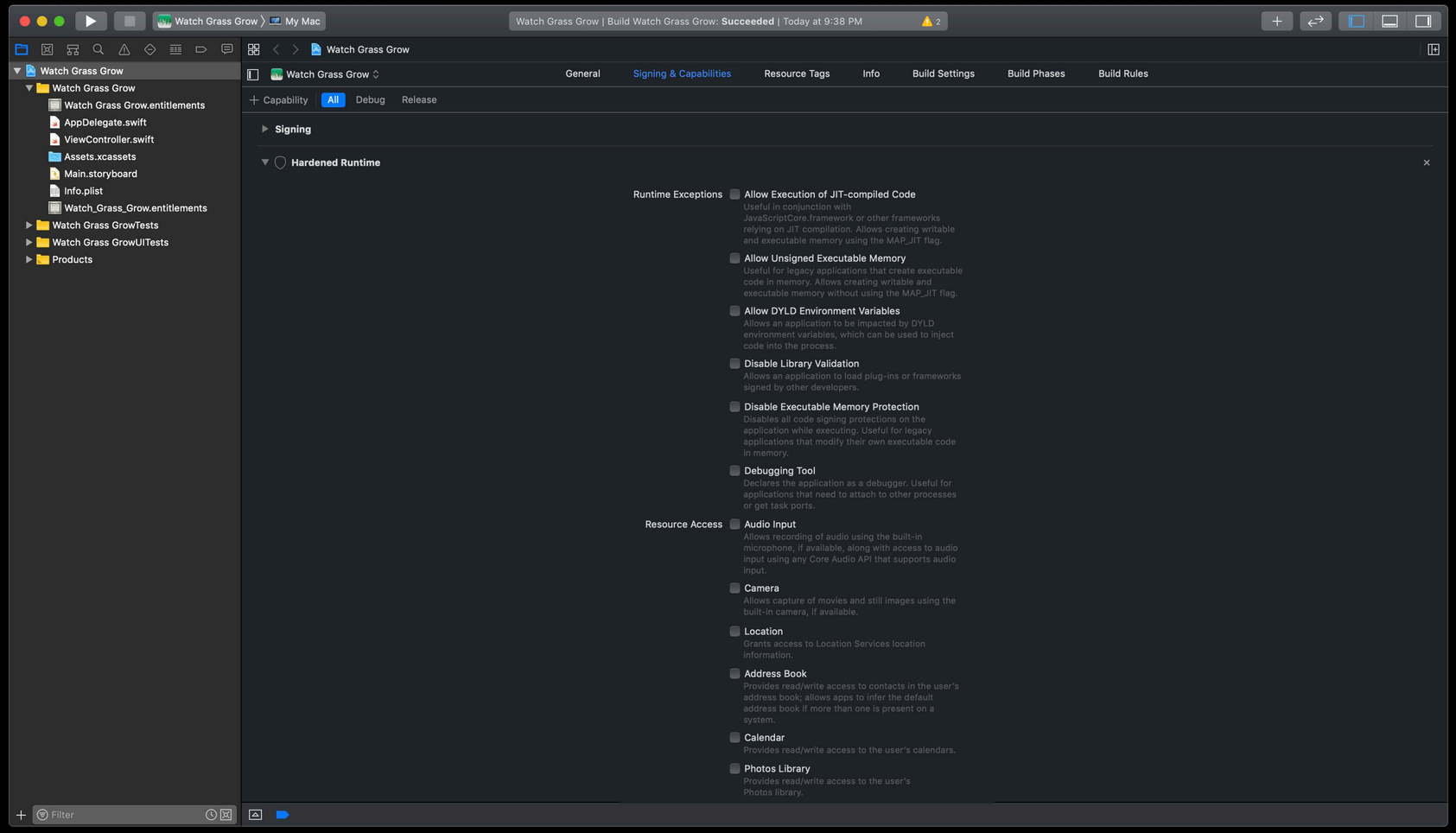
Extends many of macOS's System Integrity Protection features to your apps

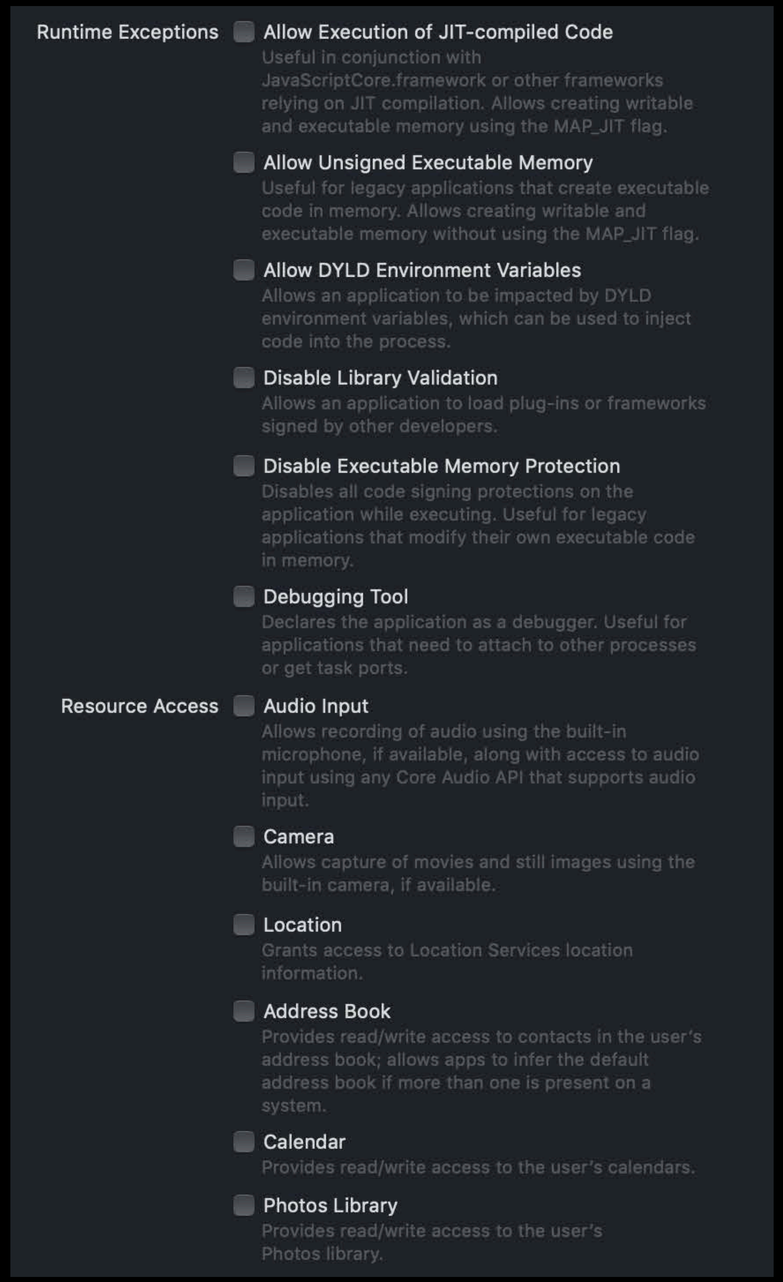
- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection

Configurable via entitlements that are available to all developers

Adopting the Hardened Runtime

Configuration in Xcode




Runtime Exceptions
Allow Execution of JIT-compiled Code
Useful in conjunction with JavaScriptCore.framework or other frameworks relying on JIT compilation. Allows creating writable and executable memory using the MAP_JIT flag.
Allow Unsigned Executable Memory
Useful for legacy applications that create executable code in memory. Allows creating writable and executable memory without using the MAP_JIT flag.
Allow DYLD Environment Variables
Allows an application to be impacted by DYLD environment variables, which can be used to inject code into the process.
Disable Library Validation
Allows an application to load plug-ins or frameworks signed by other developers.
Disable Executable Memory Protection
Disables all code signing protections on the application while executing. Useful for legacy applications that modify their own executable code in memory.
Debugging Tool
Declares the application as a debugger. Useful for applications that need to attach to other processes or get task ports.
Resource Access
Audio Input
Allows recording of audio using the built-in microphone, if available, along with access to audio input using any Core Audio API that supports audio input.
Camera
Allows capture of movies and still images using the built-in camera, if available.
Location
Grants access to Location Services location information.
Address Book
Provides read/write access to contacts in the user's address book; allows apps to infer the default address book if more than one is present on a system.
Calendar
Provides read/write access to the user's calendars.
Photos Library
Provides read/write access to the user's Photos library.

Adopting the Hardened Runtime

Using codesign

```
// Developer Workflow – Terminal

# Signature
$> codesign --sign “Developer ID” --timestamp --options runtime WatchGrassGrow.app
WatchGrassGrow.app: signed app bundle with Mach-O thin (x86_64) [com.acme.WatchGrassGrow]

# Verification
$> codesign --display --verbose=2 WatchGrassGrow.app
Executable=WatchGrassGrow.app/Contents/MacOS/WatchGrassGrow
Identifier=com.acme.WatchGrassGrow
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=566 flags=0x10000(runtime) hashes=11+3 location=embedded
Signature size=4605
Info.plist entries=22
TeamIdentifier=XXXXXXXXXX
Runtime Version=10.14.0
```

Adopting the Hardened Runtime

Using codesign

```
// Developer Workflow – Terminal

# Signature
$> codesign --sign “Developer ID” --timestamp --options runtime WatchGrassGrow.app
WatchGrassGrow.app: signed app bundle with Mach-O thin (x86_64) [com.acme.WatchGrassGrow]

# Verification
$> codesign --display --verbose=2 WatchGrassGrow.app
Executable=WatchGrassGrow.app/Contents/MacOS/WatchGrassGrow
Identifier=com.acme.WatchGrassGrow
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=566 flags=0x10000(runtime) hashes=11+3 location=embedded
Signature size=4605
Info.plist entries=22
TeamIdentifier=XXXXXXXXXX
Runtime Version=10.14.0
```

Adopting the Hardened Runtime

Using codesign

```
// Developer Workflow – Terminal

# Signature
$> codesign --sign “Developer ID” --timestamp --options runtime WatchGrassGrow.app
WatchGrassGrow.app: signed app bundle with Mach-O thin (x86_64) [com.acme.WatchGrassGrow]

# Verification
$> codesign --display --verbose=2 WatchGrassGrow.app
Executable=WatchGrassGrow.app/Contents/MacOS/WatchGrassGrow
Identifier=com.acme.WatchGrassGrow
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=566 flags=0x10000(runtime) hashes=11+3 location=embedded
Signature size=4605
Info.plist entries=22
TeamIdentifier=XXXXXXXXXX
Runtime Version=10.14.0
```

Adopting the Hardened Runtime

Using codesign

```
// Developer Workflow – Terminal

# Signature
$> codesign --sign “Developer ID” --timestamp --options runtime WatchGrassGrow.app
WatchGrassGrow.app: signed app bundle with Mach-O thin (x86_64) [com.acme.WatchGrassGrow]

# Verification
$> codesign --display --verbose=2 WatchGrassGrow.app
Executable=WatchGrassGrow.app/Contents/MacOS/WatchGrassGrow
Identifier=com.acme.WatchGrassGrow
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=566 flags=0x10000(runtime) hashes=11+3 location=embedded
Signature size=4605
Info.plist entries=22
TeamIdentifier=XXXXXXXXXX
Runtime Version=10.14.0
```

Adopting the Hardened Runtime

Using codesign

```
// Developer Workflow – Terminal

# Signature
$> codesign --sign “Developer ID” --timestamp --options runtime WatchGrassGrow.app
WatchGrassGrow.app: signed app bundle with Mach-O thin (x86_64) [com.acme.WatchGrassGrow]

# Verification
$> codesign --display --verbose=2 WatchGrassGrow.app
Executable=WatchGrassGrow.app/Contents/MacOS/WatchGrassGrow
Identifier=com.acme.WatchGrassGrow
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20500 size=566 flags=0x10000(runtime) hashes=11+3 location=embedded
Signature size=4605
Info.plist entries=22
TeamIdentifier=XXXXXXXXXX
Runtime Version=10.14.0
```

Complete and correct signing

Hardened Runtime

- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection
- Protected resource access

Adopting the Hardened Runtime

Runtime code signing enforcement

Adopting the Hardened Runtime

Runtime code signing enforcement

Prevents creation of executable memory without an associated code signature

Adopting the Hardened Runtime

Runtime code signing enforcement

Prevents creation of executable memory without an associated code signature

Ensures that all bytes mapped into your process match their associated code signature when read from disk

- Including non-executable mappings

Adopting the Hardened Runtime

Runtime code signing enforcement

Prevents creation of executable memory without an associated code signature

Ensures that all bytes mapped into your process match their associated code signature when read from disk

- Including non-executable mappings

Prevents execution from modified memory that doesn’t match its signature

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app runs non-native code, and I want that code to run blazing fast with JIT, but my app crashes when I enable Hardened Runtime

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app runs non-native code, and I want that code to run blazing fast with JIT, but my app crashes when I enable Hardened Runtime

Recommended Solution

- Adopt the “com.apple.security.cs.allow-jit” entitlement
- Use mmap and the MAP_JIT flag to allocate anonymous Read/Write/Execute memory

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app runs non-native code, and I want that code to run blazing fast with JIT, but my app crashes when I enable Hardened Runtime

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app runs non-native code, and I want that code to run blazing fast with JIT, but my app crashes when I enable Hardened Runtime

Fallback Solution

- Disable Runtime Code Signing Enforcement with the “com.apple.security.cs.allow-unsigned-executable-memory” entitlement
- Bytes mapped from disk will still be checked against any associated code signature

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app patches system frameworks it loads into memory to accomplish “...” but now my app crashes when I enable Hardened Runtime

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app patches system frameworks it loads into memory to accomplish "..." but now my app crashes when I enable Hardened Runtime

Recommended Solution

- Don't do this
- (Library Validation may meet your use case)

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app patches system frameworks it loads into memory to accomplish “...” but now my app crashes when I enable Hardened Runtime

Recommended Solution

- Don’t do this
- (Library Validation may meet your use case)

Fallback Solution

- Disable Runtime Code Signing Enforcement with the “com.apple.security.cs.allow-unsigned-executable-memory” entitlement

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app crashes when I adopt the Hardened Runtime and then run my auto update mechanism

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app crashes when I adopt the Hardened Runtime and then run my auto update mechanism

Explanation: Code signatures are latched to files on first use. Modifying files in place causes a signature mismatch

Adopting the Hardened Runtime

Developing with runtime code signing enforcement

Issue: My app crashes when I adopt the Hardened Runtime and then run my auto update mechanism

Explanation: Code signatures are latched to files on first use. Modifying files in place causes a signature mismatch

Recommended Solution

- Whenever you update a signed file, create a new file

- Complete and correct signing
- Hardened Runtime
  - Runtime code signing enforcement
  - Library validation
  - DYLD environment variable protection
  - Debugging protection
  - Protected resource access

Adopting the Hardened Runtime

Library validation

Adopting the Hardened Runtime

Library validation

Protects your app from code injection and dylibs hijacking

Adopting the Hardened Runtime

Library validation

Protects your app from code injection and dylibs hijacking

Allows your app to only load code signed by

- Your team
- Apple

Adopting the Hardened Runtime

Library validation

Protects your app from code injection and dylibs hijacking

Allows your app to only load code signed by

- Your team
- Apple

Prevents loading unsigned and adhoc signed code

Adopting the Hardened Runtime

Library validation

Protects your app from code injection and dylibs hijacking

Allows your app to only load code signed by

- Your team
- Apple

Prevents loading unsigned and adhoc signed code

Note: Make sure you use Apple Development Certificates for building and testing locally

Adopting the Hardened Runtime

Developing with library validation

Issue: My app loads plugins from other developers in-process, but plug-in loading fails when I adopt the Hardened Runtime

Adopting the Hardened Runtime

Developing with library validation

Issue: My app loads plugins from other developers in-process, but plug-in loading fails when I adopt the Hardened Runtime

Recommended Solution

- Consider moving to an out of process plugin model

Adopting the Hardened Runtime

Developing with library validation

Issue: My app loads plugins from other developers in-process, but plug-in loading fails when I adopt the Hardened Runtime

Recommended Solution

- Consider moving to an out of process plugin model

Fallback Solution

- Use the “com.apple.security.cs.disable-library-validation” entitlement
- Allows loading unsigned and adhoc signed plug-ins

Complete and correct signing

Hardened Runtime

- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection
- Protected resource access

Adopting the Hardened Runtime

DYLD environment variable protections

Adopting the Hardened Runtime

DYLD environment variable protections

DYLD environment variables can inject libraries and modify your framework and library search paths, examples

- DYLD_LIBRARY_PATH
- DYLD_INSERT_LIBRARIES
- DYLD_FRAMEWORK_PATH

Note: see “`man 1 dyld`” for the complete list

Adopting the Hardened Runtime

DYLD environment variable protections

DYLD environment variables can inject libraries and modify your framework and library search paths, examples

- DYLD_LIBRARY_PATH
- DYLD_INSERT_LIBRARIES
- DYLD_FRAMEWORK_PATH

Hardened Runtime blocks these variables by default

Note: see “`man 1 dyld`” for the complete list

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: I need to use DYLD environment variables while building and debugging my app, but they are being ignored when I enable Hardened Runtime

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: I need to use DYLD environment variables while building and debugging my app, but they are being ignored when I enable Hardened Runtime

Recommended Solution

- Use the “com.apple.security.get-task-allow” entitlement on your debug build

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: I need to use DYLD environment variables while building and debugging my app, but they are being ignored when I enable Hardened Runtime

Recommended Solution

- Use the “com.apple.security.get-task-allow” entitlement on your debug build

Note: The notary service generally refuses files signed with “com.apple.security.get-task-allow”

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: My app uses DYLD environment variables when it ships to my customers and now it doesn’t work with Hardened Runtime

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: My app uses DYLD environment variables when it ships to my customers and now it doesn’t work with Hardened Runtime

Recommend Solution

- Don’t do this

Adopting the Hardened Runtime

Developing with DYLD environment variable protections

Issue: My app uses DYLD environment variables when it ships to my customers and now it doesn’t work with Hardened Runtime

Recommend Solution

- Don’t do this

Fallback Solution

- Use the “com.apple.security.cs.allow-dyld-environment-variables” entitlement

- Complete and correct signing
- Hardened Runtime
  - Runtime code signing enforcement
  - Library validation
  - DYLD environment variable protection
  - Debugging protection
  - Protected resource access

Adopting the Hardened Runtime

Debugging protection

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory
- Modify process memory

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory
- Modify process memory

Debuggers allow attackers to

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory
- Modify process memory

Debuggers allow attackers to

- Steal sensitive user data

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory
- Modify process memory

Debuggers allow attackers to

- Steal sensitive user data
- Inject malicious code

Adopting the Hardened Runtime

Debugging protection

Debuggers allow developers to

- Inspect the state of registers and memory
- Modify process memory

Debuggers allow attackers to

- Steal sensitive user data
- Inject malicious code

Hardened Runtime prevents debugging of hardened processes by default

Adopting Hardened Runtime

Developing with debugging protections

Issue: How can I build and test with the Hardened Runtime if I cannot attach a debugger?

Adopting Hardened Runtime

Developing with debugging protections

Issue: How can I build and test with the Hardened Runtime if I cannot attach a debugger?

Solution

- Use the “com.apple.security.get-task-allow” entitlement on your debug build
- Xcode does this for you

Adopting Hardened Runtime

Developing with debugging protections

Issue: How can I build and test with the Hardened Runtime if I cannot attach a debugger?

Solution

- Use the “com.apple.security.get-task-allow” entitlement on your debug build
- Xcode does this for you

Note: Running an app under a debugger will mask Hardened Runtime related issues

- Be sure to test a release build
- If you need a debug build without “com.apple.security.get-task-allow” set CODE_SIGN_INJECT_BASE_ENTITLEMENTS=NO

Adopting Hardened Runtime

Developing with debugging protections

Issue: My app supports an in-process plug-in ecosystem. How can my plug-in developers debug their plug-ins?

Adopting Hardened Runtime

Developing with debugging protections

Issue: My app supports an in-process plug-in ecosystem. How can my plug-in developers debug their plug-ins?

Recommended Solution

- Move to an out of process plug-in model

Adopting Hardened Runtime

Developing with debugging protections

Issue: My app supports an in-process plug-in ecosystem. How can my plug-in developers debug their plug-ins?

Recommended Solution

- Move to an out of process plug-in model

Alternative Solution

- Ship a debug version to registered plug-in developers

Adopting Hardened Runtime

Developing with debugging protections

Issue: My app supports an in-process plug-in ecosystem. How can my plug-in developers debug their plug-ins?

Recommended Solution

- Move to an out of process plug-in model

Alternative Solution

- Ship a debug version to registered plug-in developers

Fallback Solution

- Combine “com.apple.security.get-task-allow” with “com.apple.security.cs.disable-library-validation”

Complete and correct signing

Hardened Runtime

- Runtime code signing enforcement
- Library validation
- DYLD environment variable protection
- Debugging protection
- Protected resource access

Adopting the Hardened Runtime

Resource access protections

Your customers use their Macs to store tons of information about their lives

Your app needs to declare its intent to access protected resources

Resource Access Requirements

Description	Entitlement	Usage string Info.plist key
Audio input and microphone	`com.apple.security.device.audio-input`	`NSMicrophoneUsageDescription`
Any camera exposed via AVFoundation	`com.apple.security.device.camera`	`NSCameraUsageDescription`
Location	`com.apple.security.personal-information.location`	`NSPhotoLibraryUsageDescription`
Contacts	`com.apple.security.personal-information.addressbook`	`NSPhotoLibraryUsageDescription`
Calendars and Reminders	`com.apple.security.personal-information.calendars`	`NSCalendarUsageDescription`
Apple Photos library	`com.apple.security.personal-information.photos-library`	`NSPhotoLibraryUsageDescription`
Sending Apple Events to other apps	`com.apple.security.automation.apple-events`	`NSAppleEventsUsageDescription`

Adopting Hardened Runtime

Recommendations

Adopting Hardened Runtime

Recommendations

Take only the entitlements you need

Adopting Hardened Runtime

Recommendations

Take only the entitlements you need

Apply entitlements only to the processes in your app that need them

Adopting Hardened Runtime

Recommendations

Take only the entitlements you need

Apply entitlements only to the processes in your app that need them

When declaring resource access, only set the entitlements
and usage strings on your main bundle

What is notarization?

Application requirements

Workflows

Notarization Workflow

Notarization Workflow

Submit your software

Notarization Workflow

Submit your software

Check processing status

Notarization Workflow

Submit your software

Check processing status

Staple ticket(s)

Notarization Workflow

Submit your software

Check processing status

Staple ticket(s)

Verify notarization

Submitting Your Software

Submitting Your Software

Submit all software you distribute

Submitting Your Software

Submit all software you distribute

OK to upload more regularly

- Not worth uploading every CI build

Submitting Your Software

Submit all software you distribute

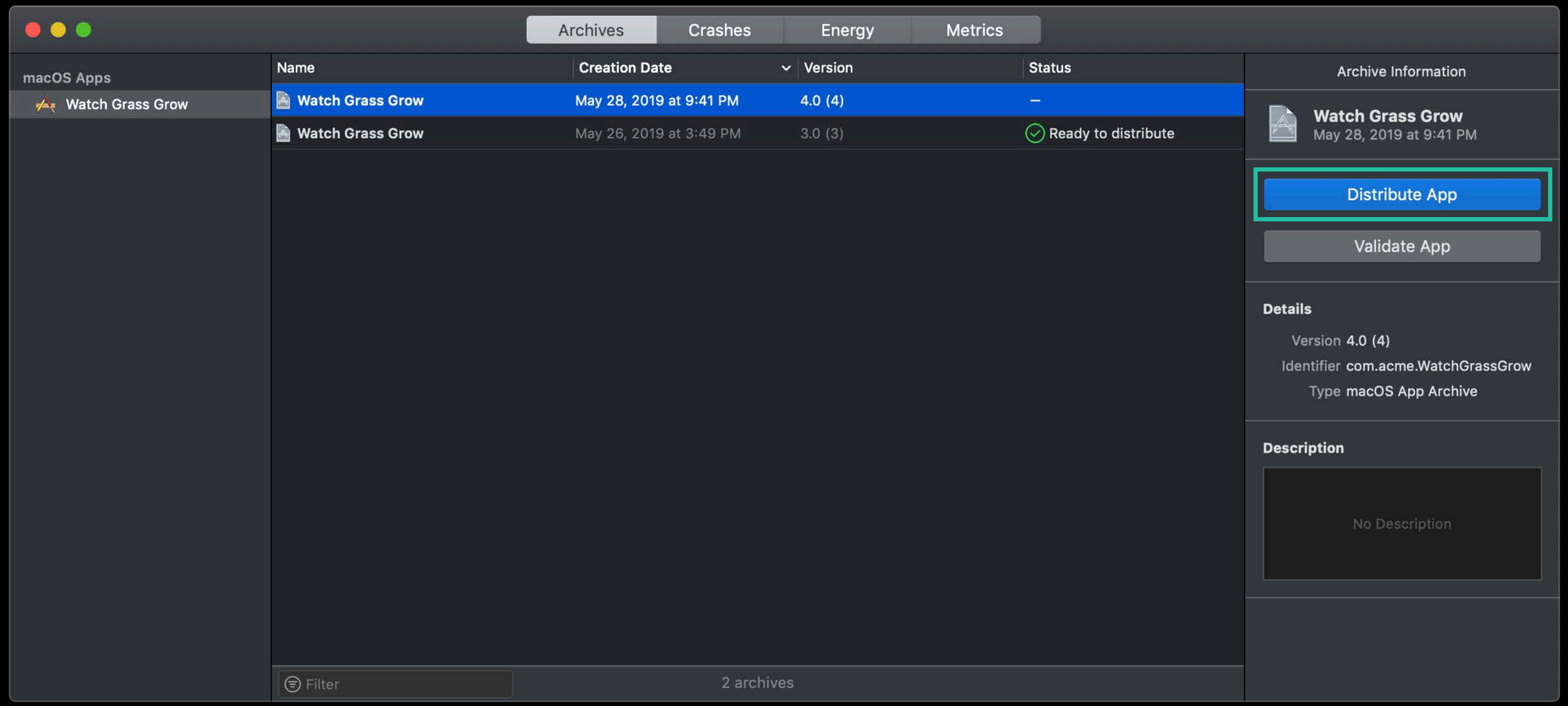
OK to upload more regularly

- Not worth uploading every CI build

Anyone on the team can submit software

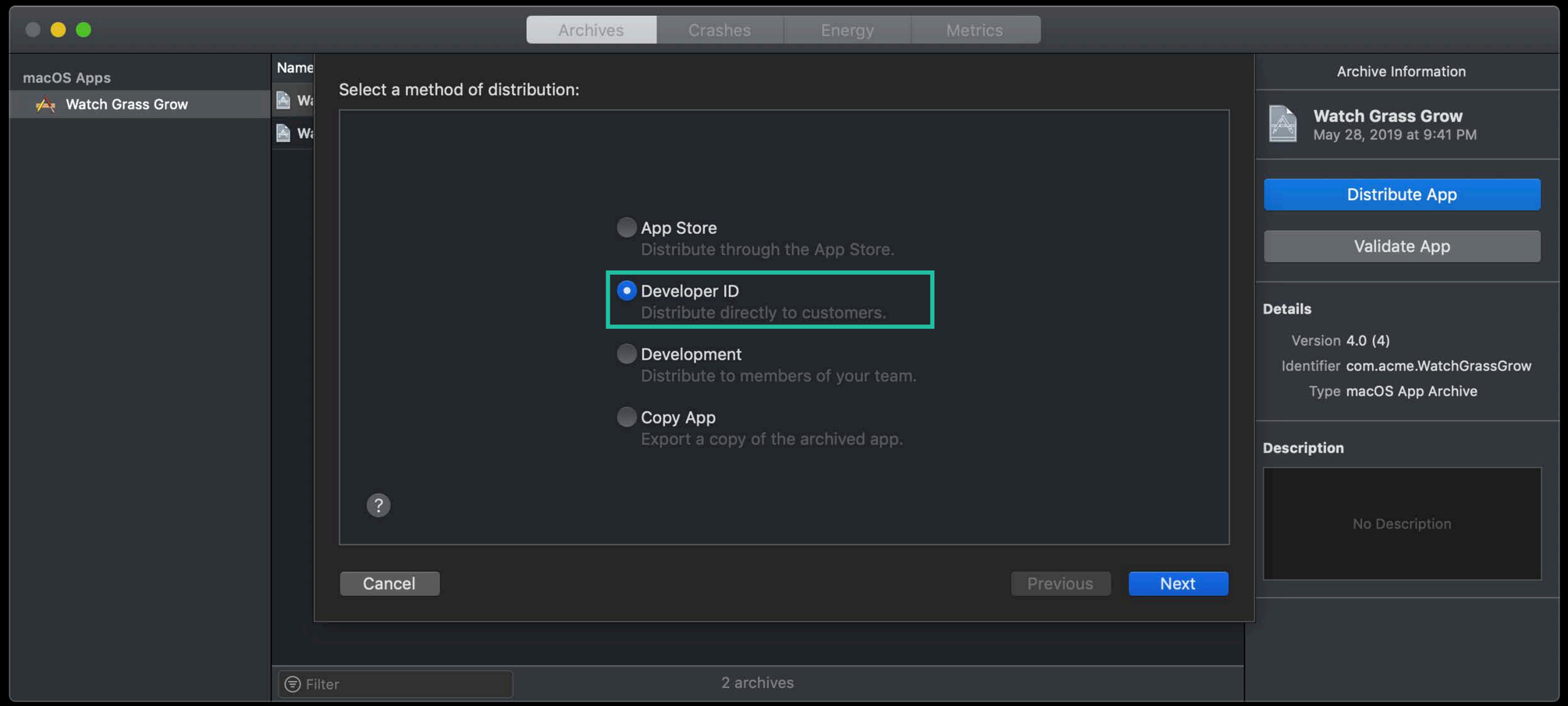
Xcode

Submit your app



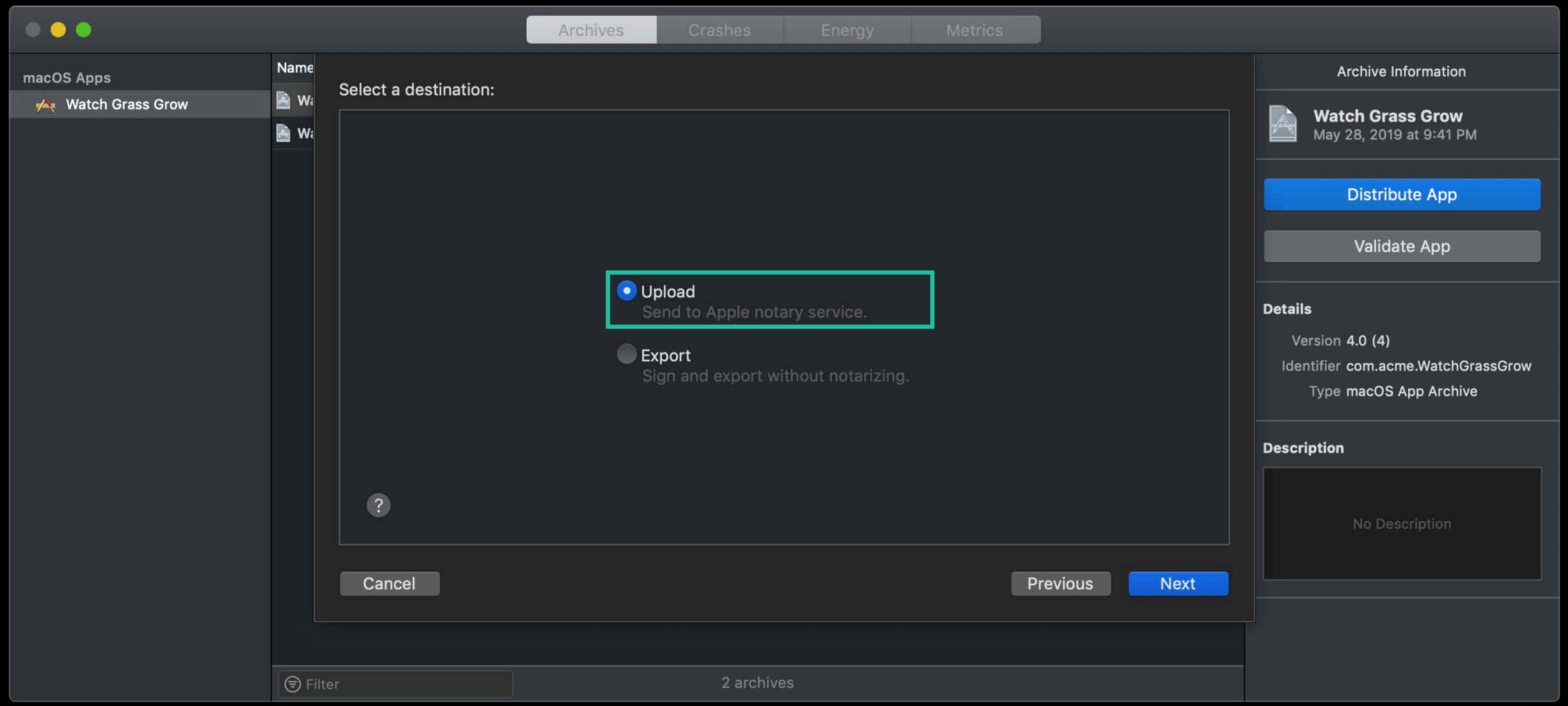
Xcode

Submit your app



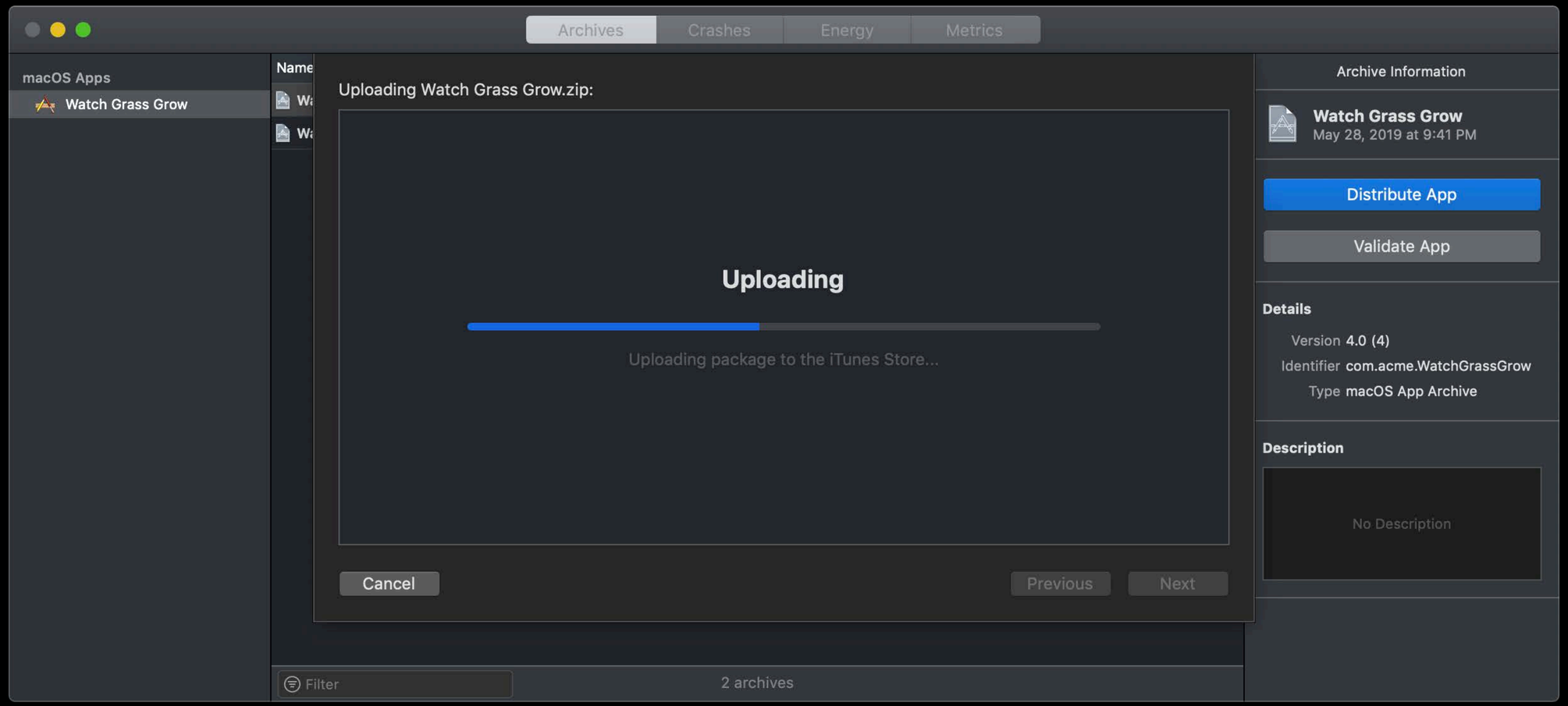
Xcode

Submit your app



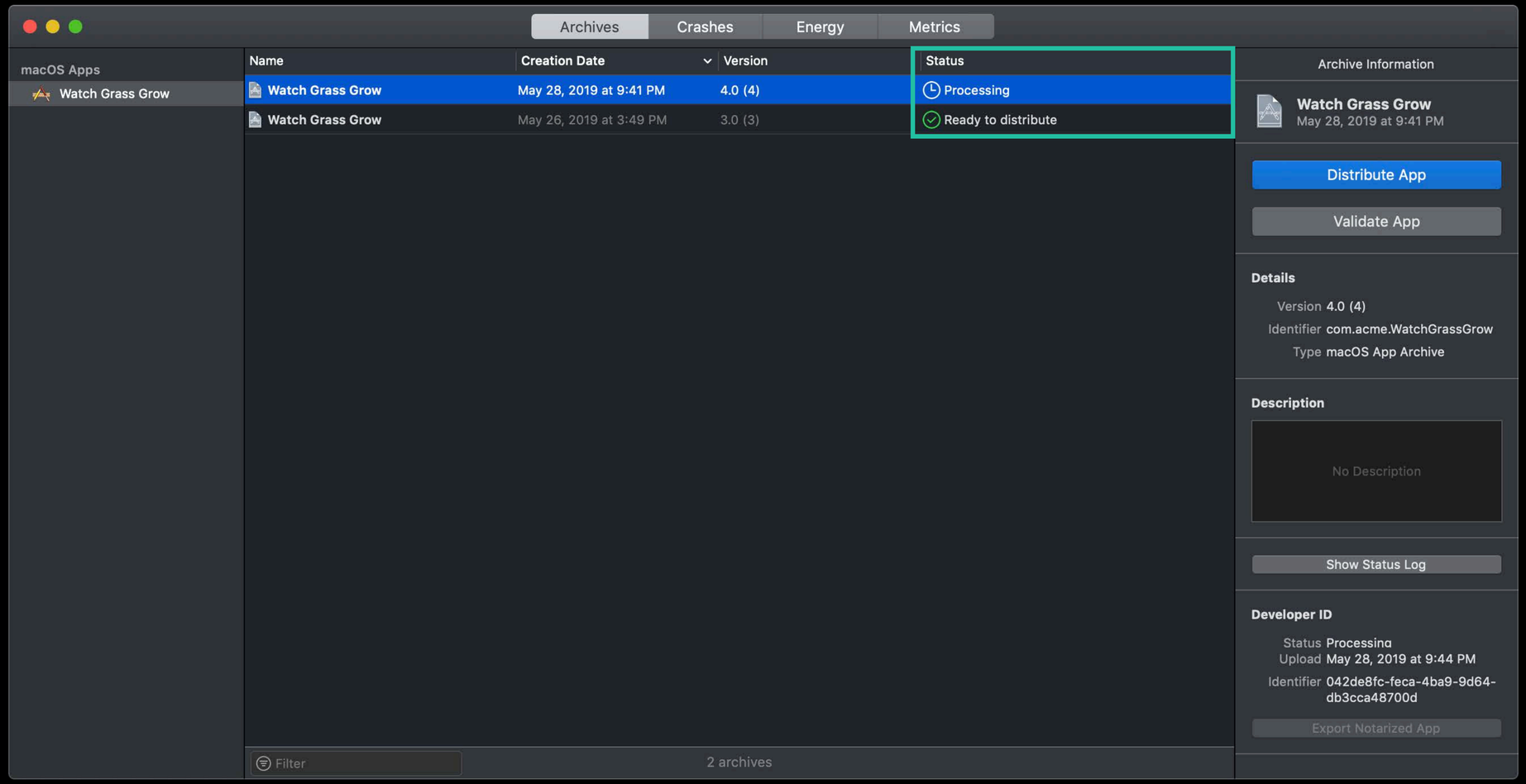
Xcode

Submit your app



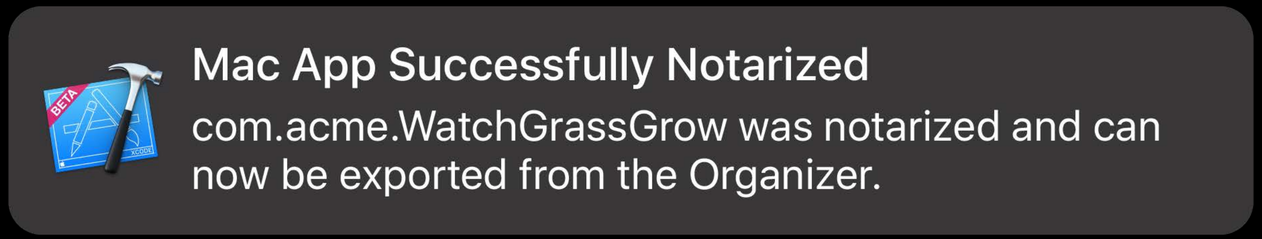
Xcode

Check status



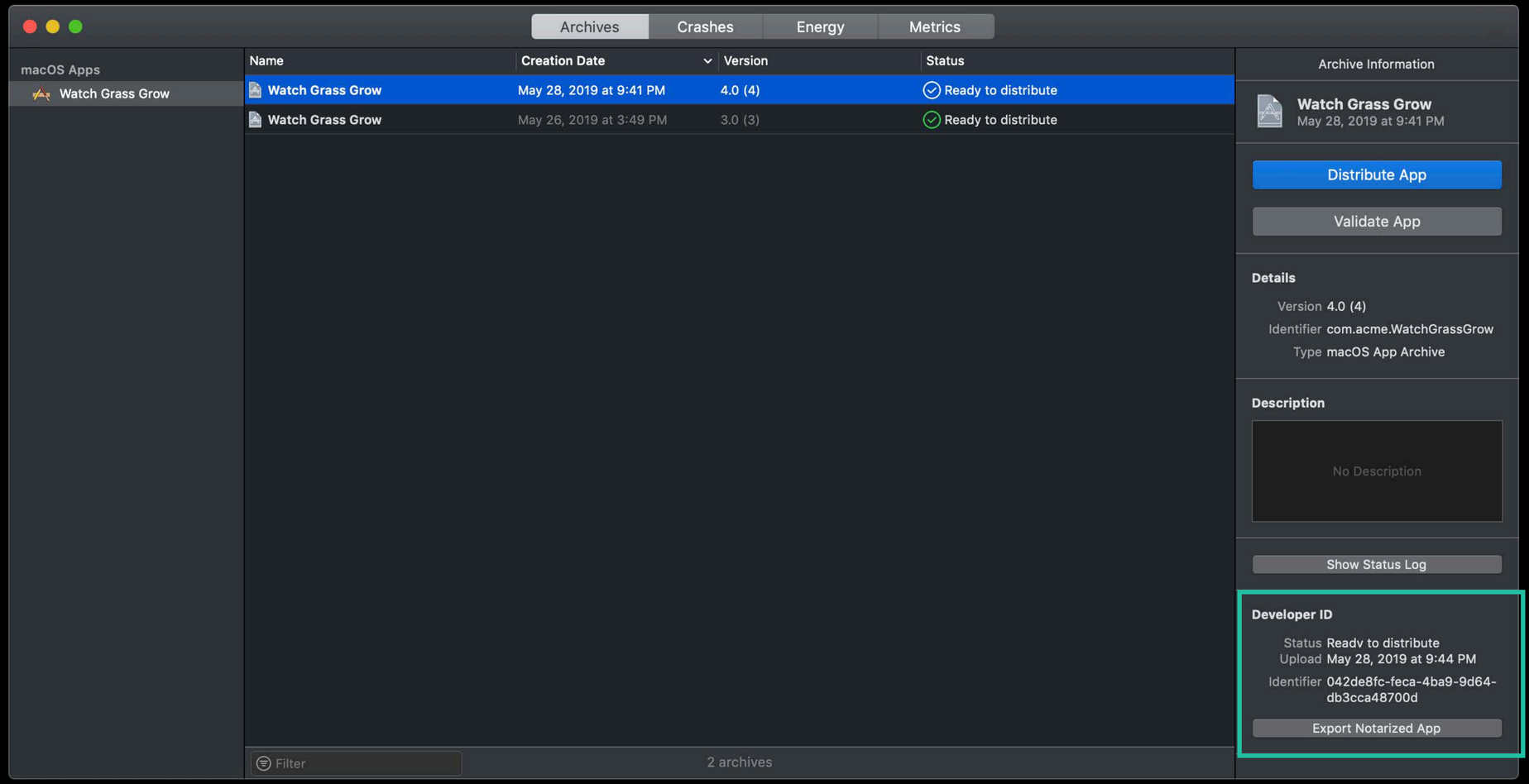
Xcode

Check status



Xcode

Staple your app



Custom Workflow

Submit your software

Supported formats

- Disk images (.dmg files)
- Installer packages (.pkg files)
- Zip archives (.zip files)

Custom Workflow

Submit your software

Supported formats

- Disk images (.dmg files)
- Installer packages (.pkg files)
- Zip archives (.zip files)

When creating an archive, ensure you preserve macOS-specific metadata

- Support in ditto and Archive Utility

Custom Workflow

Submit your software

You may need two-step notarization if your custom installer

- Pulls down additional content from the web
- Uses a custom packing format

Custom Workflow

Submit your software

You may need two-step notarization if your custom installer

- Pulls down additional content from the web
- Uses a custom packing format

Two step notarization process

- Submit the content as it will appear on disk
- Submit your custom installer

Custom Workflow

Submit your software

```
$ sudo xcode-select -s /Applications/Xcode.app
$ xcrun altool --notarize-app
               --primary-bundle-id "com.acme.WatchGrassGrow" --file "WatchGrassGrow.zip"
               --username "USERNAME" --password "@keychain:ITEM_NAME"
Result:
altool[16765:378423] No errors uploading 'WatchGrassGrow.zip'.
RequestUUID = 2EFE2717-52EF-43A5-96DC-0797E4CA1041
```

Custom Workflow

Submit your software

```
$ sudo xcode-select -s /Applications/Xcode.app
$ xcrun altool --notarize-app
               --primary-bundle-id "com.acme.WatchGrassGrow" --file "WatchGrassGrow.zip"
               --username "USERNAME" --password "@keychain:ITEM_NAME"
Result:
altool[16765:378423] No errors uploading 'WatchGrassGrow.zip'.
RequestUUID = 2EFE2717-52EF-43A5-96DC-0797E4CA1041
```

Custom Workflow

Submit your software

```
$ sudo xcode-select -s /Applications/Xcode.app
$ xcrun altool --notarize-app
               --primary-bundle-id "com.acme.WatchGrassGrow" --file "WatchGrassGrow.zip"
               --username "USERNAME" --password "@keychain:ITEM_NAME"
Result:
altool[16765:378423] No errors uploading 'WatchGrassGrow.zip'.
RequestUUID = 2EFE2717-52EF-43A5-96DC-0797E4CA1041
```

Custom Workflow

Check status

```
$ xcrun altool --notarization-info 2EFE2717-52EF-43A5-96DC-0797E4CA1041
               --username "USERNAME" –password "@keychain:ITEM_NAME"
   RequestUUID: 64e86b52-4911-4b63-9cca-590d7fb8d4fe
          Date: 2019-05-26 22:51:35 +0000
        Status: success
    LogFileURL: https://osxapps-ssl.itunes.apple.com/…
  Status Code: 0
Status Message: Package Approved
```

Custom Workflow

Check status

```
$ xcrun altool --notarization-info 2EFE2717-52EF-43A5-96DC-0797E4CA1041
               --username "USERNAME" –password "@keychain:ITEM_NAME"
   RequestUUID: 64e86b52-4911-4b63-9cca-590d7fb8d4fe
          Date: 2019-05-26 22:51:35 +0000
        Status: success
    LogFileURL: https://osxapps-ssl.itunes.apple.com/…
  Status Code: 0
Status Message: Package Approved
```

Custom Workflow

Check status

```
$ xcrun altool --notarization-info 2EFE2717-52EF-43A5-96DC-0797E4CA1041
               --username "USERNAME" –password "@keychain:ITEM_NAME"
   RequestUUID: 64e86b52-4911-4b63-9cca-590d7fb8d4fe
          Date: 2019-05-26 22:51:35 +0000
        Status: success
    LogFileURL: https://osxapps-ssl.itunes.apple.com/…
  Status Code: 0
Status Message: Package Approved
```

Custom Workflow

Check status

```
$ xcrun altool --notarization-info 2EFE2717-52EF-43A5-96DC-0797E4CA1041
               --username "USERNAME" –password "@keychain:ITEM_NAME"
   RequestUUID: 64e86b52-4911-4b63-9cca-590d7fb8d4fe
          Date: 2019-05-26 22:51:35 +0000
        Status: success
    LogFileURL: https://osxapps-ssl.itunes.apple.com/…
  Status Code: 0
Status Message: Package Approved
```

Custom Workflow

Check status


```
{
    "archiveFilename": "Watch_Grass_Grow.zip",
    "issues": [ … ],
    "jobId": "64e86b52-4911-4b63-9cca-590d7fb8d4fe",
    "logFormatVersion": 1,
    "sha256": "d18f8c91cdf08af3aae8e6983dc8f7535264f8cfe89791a03b54b22f6b6c5fbd",
    "status": "Accepted",
    "statusCode": 0,
    "statusSummary": "Ready for distribution",
    "ticketContents": [ … ],
    “uploadDate”:  “2019-05-26T22:51:35Z"
}
```

Custom Workflow

Check status


```
{
    "archiveFilename": "Watch_Grass_Grow.zip",
    "issues": [ … ],
    "jobId": "64e86b52-4911-4b63-9cca-590d7fb8d4fe",
    "logFormatVersion": 1,
    "sha256": "d18f8c91cdf08af3aae8e6983dc8f7535264f8cfe89791a03b54b22f6b6c5fbd",
    "status": "Accepted",
    "statusCode": 0,
    "statusSummary": "Ready for distribution",
    "ticketContents": [ … ],
    “uploadDate”:  “2019-05-26T22:51:35Z"
}
```

Custom Workflow

Check status


```
{
    "archiveFilename": "Watch_Grass_Grow.zip",
    "issues": [ … ],
    "jobId": "64e86b52-4911-4b63-9cca-590d7fb8d4fe",
    "logFormatVersion": 1,
    "sha256": "d18f8c91cdf08af3aae8e6983dc8f7535264f8cfe89791a03b54b22f6b6c5fbd",
    "status": "Accepted",
    "statusCode": 0,
    "statusSummary": "Ready for distribution",
    "ticketContents": [ … ],
    “uploadDate”:  “2019-05-26T22:51:35Z"
}
```

Custom Workflow

Check status


```
{
    "archiveFilename": "Watch_Grass_Grow.zip",
    "issues": [ … ],
    "jobId": "64e86b52-4911-4b63-9cca-590d7fb8d4fe",
    "logFormatVersion": 1,
    "sha256": "d18f8c91cdf08af3aae8e6983dc8f7535264f8cfe89791a03b54b22f6b6c5fbd",
    "status": "Accepted",
    "statusCode": 0,
    "statusSummary": "Ready for distribution",
    "ticketContents": [ … ],
    “uploadDate”:  “2019-05-26T22:51:35Z"
}
```

Notarization Success

Developer

Dear Garrett,

Your Mac software has been notarized. You can now export this software and distribute it directly to users.

Bundle Identifier: com.acme.WatchGrassGrow
Request Identifier: 042de8fc-feca-4ba9-9d64-db3cca48700d

For details on exporting a notarized app, visit Xcode Help or the notarization guide.

Best Regards,
Apple Developer Relations

Custom Workflow

Staple

Custom Workflow

Staple

If you submitted a pkg or dmg you can staple it directly

```
$ xcrun stapler staple <path-to-item>
```

Custom Workflow

Staple

If you submitted a pkg or dmg you can staple it directly

```
$ xcrun stapler staple <path-to-item>
```

If you submitted a zip file, it cannot be stapled directly, instead

- Unpack the zip file
- Invoke the stapler tool on each bundle individually
- Re-zip everything for distribution

Custom Workflow

Staple

If you submitted a pkg or dmg you can staple it directly

```
$ xcrun stapler staple <path-to-item>
```

If you submitted a zip file, it cannot be stapled directly, instead

- Unpack the zip file
- Invoke the stapler tool on each bundle individually
- Re-zip everything for distribution

Stapling of command line tools/standalone dylibs is not currently supported

- Standalone binaries should be notarized

Verifying Notarization

Is an application, disk image, or package stapled?

```
$ xcrun stapler validate <path-to-item>
Processing: <path>
The validate action worked!
```

Verifying Notarization

Is this application I downloaded notarized?

```
$ spctl --assess --verbose <path-to-app>
<path-to-app>: accepted
source=Notarized Developer ID
```

Verifying Notarization

Is this package I downloaded notarized?

```
$ spctl --assess --verbose --type install <path-to-pkg>
<path-to-pkg>: accepted
source=Notarized Developer ID
```

Verifying Notarization

Is this signed disk image I downloaded notarized?

```
$ spctl --assess --verbose --type open --context "context:primary-signature" <path-to-dmg>
<path-to-dmg>: accepted
source=Notarized Developer ID
```

Verifying Notarization

Is this other binary I downloaded notarized?

```
$ codesign —-verify --verbose --test-requirement="notarized" <path-to-binary>
<path-to-binary>: valid on disk
<path-to-binary>: satisfies its Designated Requirement
<path-to-binary>: explicit requirement satisfied
```

Notary Service History

Notary Service History

```
$ xcrun altool --notarization-history -u "USERNAME" -p "@keychain:ITEM_NAME"

Notarization History - page 0

Date                      RequestUUID                            Status  Status Code Status Message
------------------------- ------------------------------------ ------- ----------- ----------------
2019-05-31 05:16:55 +0000 e5652174-52b8-4a8c-9fe5-3409f78c9147 success 0           Package Approved
2019-05-26 22:51:35 +0000 64e86b52-4911-4b63-9cca-590d7fb8d4fe success 0           Package Approved
Next page value: 1558911095000
```

Notary Service History

```
$ xcrun altool --notarization-history -u "USERNAME" -p "@keychain:ITEM_NAME"

Notarization History - page 0

Date                      RequestUUID                            Status  Status Code Status Message
------------------------- ------------------------------------ ------- ----------- ----------------
2019-05-31 05:16:55 +0000 e5652174-52b8-4a8c-9fe5-3409f78c9147 success 0           Package Approved
2019-05-26 22:51:35 +0000 64e86b52-4911-4b63-9cca-590d7fb8d4fe success 0           Package Approved
Next page value: 1558911095000
```

Summary

Summary

Sign all your software properly

Summary

Sign all your software properly

Do not take Hardened Runtime entitlements you do not need

Summary

Sign all your software properly

Do not take Hardened Runtime entitlements you do not need

Notarize and staple everything you distribute

More Information

developer.apple.com/wwdc19/703

Mac App Notarization Lab	Tuesday, 4:00	
Signing and Distributing Lab	Thursday, 9:00	Friday, 9:00
Security Lab	Thursday, 2:00	


WWDC19