How to Inject Code into Mach-O Apps. Part III.

jon-gabilondo-angulo-7635.medium.com/how-to-inject-code-into-mach-o-apps-part-iii-6d9eebef52e5

Jon Gabilondo July 29, 2022

Dynamic Code Injection Techniques. Summary.

Jon Gabilondo

This is the last part of the dynamic injection series aimed to learn how to inject a dynamic library into an already compiled Mac OS X App.

The main motivation for this task is to be able to inject the Organismo iOS/Mac instrumentation library. Organismo is a great tool for programming students and UI enthousiasts to learn the inners of Applications. On the way, we can learn some of Apple's great security mechanisms which make our computers safer.

The test carried out for this article have been executed in Monterey 12.5

GitHub - JonGabilondoAngulo/Organismo-Lib: Library to inspect iOS and Mac Apps. iOS driver for…

Organismo framework to be injected into Mac Apps to explore them at runtime…

github.com

Dynamic Code Injection. The Dynamic Loader — dyld.

In these series we concentrate only in the 'Dynamic Code Injection' technique, i.e. the App is already built and compiled, we want to load a dylib or framework into the App at runtime, to perform some instrumentation on it.

Loading a library into an App at launch time was very simple until OS X *Mountain Lion* (10.8. July 25, 2012). The dynamic loader was and still it is programmed to read the environment DYLD_INSERT_LIBRARIES and load the libraries defined in the paths assigned to it. Since *Mountain Lion* , the 'Hardened Applications' together with the *System Integrity Protection* (SIP) security measures, make the *dyld (dynamic likner or loader)* ignore DYLD_INSERT_LIBRARIES, more specifically all DYLD_ prefixed env variables. This is extensively covered in PartI & PartII. For programmers, this is Apple's documentation: https://developer.apple.com/documentation/security/hardened_runtime

We saw in PartI & II that some Apps, like Calculator.app, are still not hardened and the DYLD_INSERT_LIBRARIES still works. For those Apps that were hardened we removed the hardening by re-signing the Apps with *ad-hoc* or a proprietary legal *developer certificate*, but this was not a fit-all solution. Apple has a powerful security based the notion of *entitlements* which define what an App is allowed to do, and those entitlements are granted to developers via P*rovision Profiles* that follow strict rules to obtain them via the Developers Portal. On top of this, Apple has hundreds of *private entitlements* that only Apple can set their code and no other but code signed with Apple's Authority Signature would run.

All of this would make that re-signing Apps with your Dev or Distribution certificates would end up either breaking the signature and being banned by *amfid* or in a binary that will crash soon after start.

Part II would end with a suggestion to be more successful which was to disable SIP and AMFI. It indeed would increase the chances but it is not a recommended approach, it turns your computer into a decade old and highly vulnerable Mac.

Injecting Code with SIP and AMFI Enabled

In this last part of the series in dynamic code injection we are going to explore the limits of code injection, having the Mac OS X at its default-maximum security level. In the previous part we suggested to disable SIP and AMFI but this time we want to leave them on and have an opportunity to learn more about the security layer of OS X.

AMFI is the Apple Mobile File Integrity. As its name suggests it was originated in iOS and it is in charge or preventing to run code that is not porperly signed or entitled. It consists of two components:

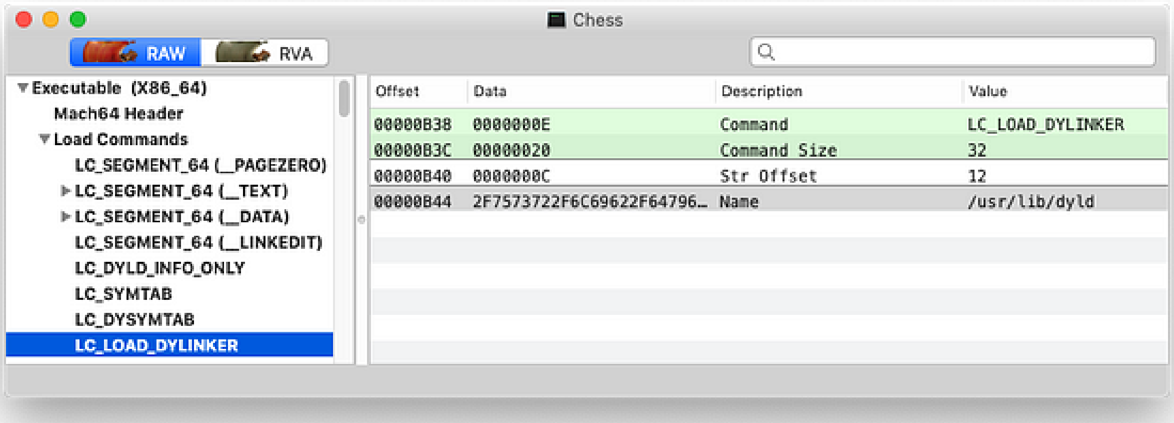
1. The is a daemon of the Apple Mobile File Integrity found in which launches .
2. Its client is the extension in /System/Library/Extension. The AMFI extension loads trusted keys from NVRAM and is also involved with rootless and DataVault controllers. amfid checks the integrity of files running on the system by calling trustd to do its checking. trustd is a Launch Agent daemon which provides services for evaluating trust in certificates for all processes, including with Static Code Checks.

For more information read the always great eclecticlight.co.

Finding Hardened Apps

Whether the *dyld* would inject libraries defined in DYLD_INSERT_LIBRARIES is not a system-wide configuration decision, it is a per App decision. The *dyld* is a library that is linked to the App and loaded at runtime. The *dyld* would react to the settings of the App which is in charge to load.

The LC_LOAD_DYLINKER load command contains the path to the *dyld* to use: */usr/lib/dyld*.



Offset	Data	Description	Value
00000B38	0000000E	Command	LC_LOAD_DYLINKER
00000B3C	00000020	Command Size	32
00000B40	0000000C	Str Offset	12
00000B44	2F7573722F6C69622F64796…	Name	/usr/lib/dyld

The command that specifies the dynamic linker to use. (Chess App)

Looking for information on this subject it is often found that the usual way to detect if an App is hardened is by using the 'codesign' command and checking the *flags* value.

The Calculator.app has flags=(none) which tells it is not hardened.

```
$ codesign -dvvv
/System/Applications/Calculator.appExecutable=/System/Applications/Calculator.app/Cont
 bundle with Mach-O universal (x86_64 arm64e)CodeDirectory v=20400 size=2125
hashes=56+7
```

And we could go on this way and see that all Apple Apps seems not to have signing flags .. *Automator, Calendar, TextEdit, FaceTime, Mail, Console, Digital Color Meter* ..

We expect the flag 'runtime' according to the manual of 'codesign':

```
OPTION FLAGS  On macOS versions >= 10.14.0, opts signed processes into a hardened
runtime environment which includes runtime code signing enforcement, library
validation, hard, kill, and debugging restrictions.  These restrictions can be
selectively relaxed via entitlements. Note: macOS  versions older than 10.14.0 ignore
the presence of this flag in the code signature.
```

Searching for the flag in non Apple Apps we do find the 'runtime' flag:

```
$ codesign -dvvv
Executable=/Applications/WhatsApp.app/Contents/MacOS/WhatsAppIdentifier=desktop.WhatsA
 bundle with Mach-O thin (x86_64)CodeDirectory v=20500 size=1820 flags=0x10000()
hashes=46+7 location=embedded.....$ codesign -dvvv
Executable=/Applications/SourceTree.app/Contents/MacOS/SourcetreeIdentifier=com.torusk
 bundle with Mach-O thin (x86_64)CodeDirectory v=20500 size=32138 flags=0x10000()
hashes=995+5 location=embedded
```

Codesign Flags do not Always Tell if App is Hardened

From these tests I would say that Apple Apps do not have the 'runtime' flag, while third party (developers) Apps do.

This would complicate to know a priori if an App is hardened and if we could inject Organismo.

From my experiements *Calculator* is not hardened but *Digital Color Meter* is. So the question is what really tells the *dyld* to read DYLD_INSERT_LIBRARIES or not ?

The answer was in the article in Part II, entitlements can harden an Application.

We can see in the source code of *dyld* how is parsing and deleting the variables that match DYLD_* and LD_LIBRARY_PATH, and what is more interesting we can see the three reasons for the restiction:

```
//// For security, setuid programs ignore DYLD_* environment variables.//
Additionally, the DYLD_* enviroment variables are removed// from the environment, so
that any child processes don't see them.//  pruneEnvironmentVariables( * envp[],  ***
applep){// delete all DYLD_* and LD_LIBRARY_PATH environment variables removedCount =
0; ** d = envp;( ** s = envp; *s != ; s++) {  ( (strncmp(*s, "DYLD_", 5) != 0) &&
(strncmp(*s, "LD_LIBRARY_PATH=", 16) != 0) ) {*d++ = *s;} {++removedCount;}}*d++ =
;// <rdar://11894054> Disable warnings about DYLD_ env vars being ignored.  The
warnings are causing too much confusion.#if 0 ( removedCount != 0 ) {dyld::log("dyld:
DYLD_ environment variables being ignored because "); (sRestrictedReason) {
restrictedNot:; restrictedBySetGUid:dyld::log("main executable (%s) is setuid or
setgid\n", sExecPath);; restrictedBySegment:dyld::log("main executable (%s) has
__RESTRICT/__restrict section\n", sExecPath);;
restrictedByEntitlements:dyld::log("main executable (%s) is code signed with
entitlements\n", sExecPath);;}}#endif...
```

These are the three reasons for the restriction:

1. . These are flags can be assigned to an App to run with the privileges of the owning user or group, i.e. not in the current user's context. Instead of creating an entry in the sudoers file, which must be done by root, any user can specify the setuid or setgid flag to be set for their own applications. These bits are indicated with an "s" instead of an "x" when viewing a file's attributes via `ls -l`. The `chmod` program can set these bits with via bitmasking, `chmod 4777 [file]` or via shorthand naming, `chmod u+s [file]`. This is a vulnerability that can be exploited.
2. . These is a segment in the Mach-O file that can be created at link time. No specific content is needed. Acts like a flag to harden the process.
3. . In the code signing process of the App an entitlements flag can define the hardening of the App. The entitlements can be configured by Xcode enabling the runtime hardening:

Making ‘Digital Color Meter.app’ Injectable

Digital Color Meter is an interesting case for injection, while most of the small utility apps do accept DYLD_INSERT_LIBRARIES and we can inject *Organismo*, it is not the case for *Digital Color Meter.*

This is the command to inject *Organismo*:

```
$ DYLD_INSERT_LIBRARIES=/path_to/Organismo-mac.framework/Versions/A/Organismo-mac
/path_to/Digital\ Color\ Meter.app/Contents/MacOS/Digital\ Color\ Meter
```

The *Digital Color Meter* entitlements:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST
1.0//EN" ""><plist version="1.0"><dict><key></key><array><string></string></array>
</dict></plist>%
```

The **com.apple.private.tcc.allow** entitlements are available only to Apple and sets a privilege to bypass the TCC, in the case of **kTCCServiceScreenCapture** the binary will be granted to perform screen recording without user consent.

It seems that the presence of a TCC entitlement makes the App hardened/restricted.

Solution. Remove the Signature of the App

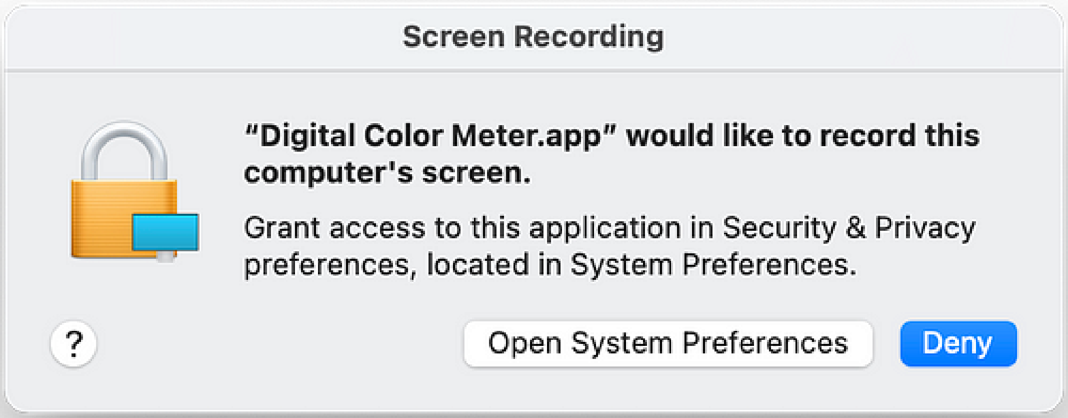
In order to make *Digital Color Meter* injectable we could resign the App with our *Dev Certificate*, but we could not assign the **com.apple.private.tcc.allow** entitlement, therefore we could just as well remove the signature and the entitlements and turn it into an unsigned App, i.e. non restricted and non hardened.

This is the command to remove the signature from the App:

```
$ codesign --remove-signature /aWorkingFolder/Digital\ Color\ Meter.appor using $
macho_inject --remove-signature -o ~/ModifiedApps /Applications/Digital\ Color\
Meter.app
```

The consequence of removing the signature is that the TCC entitlement to allow screen capture will be lost (**com.apple.private.tcc.allow — kTCCServiceScreenCapture)**.

When our modified D*igital Color Meter* starts and attempts a screen capture, OS X blocks the operation because the binary has no entitlement for it, therefore it asks the user if it allows the App to record the computer’s screen.



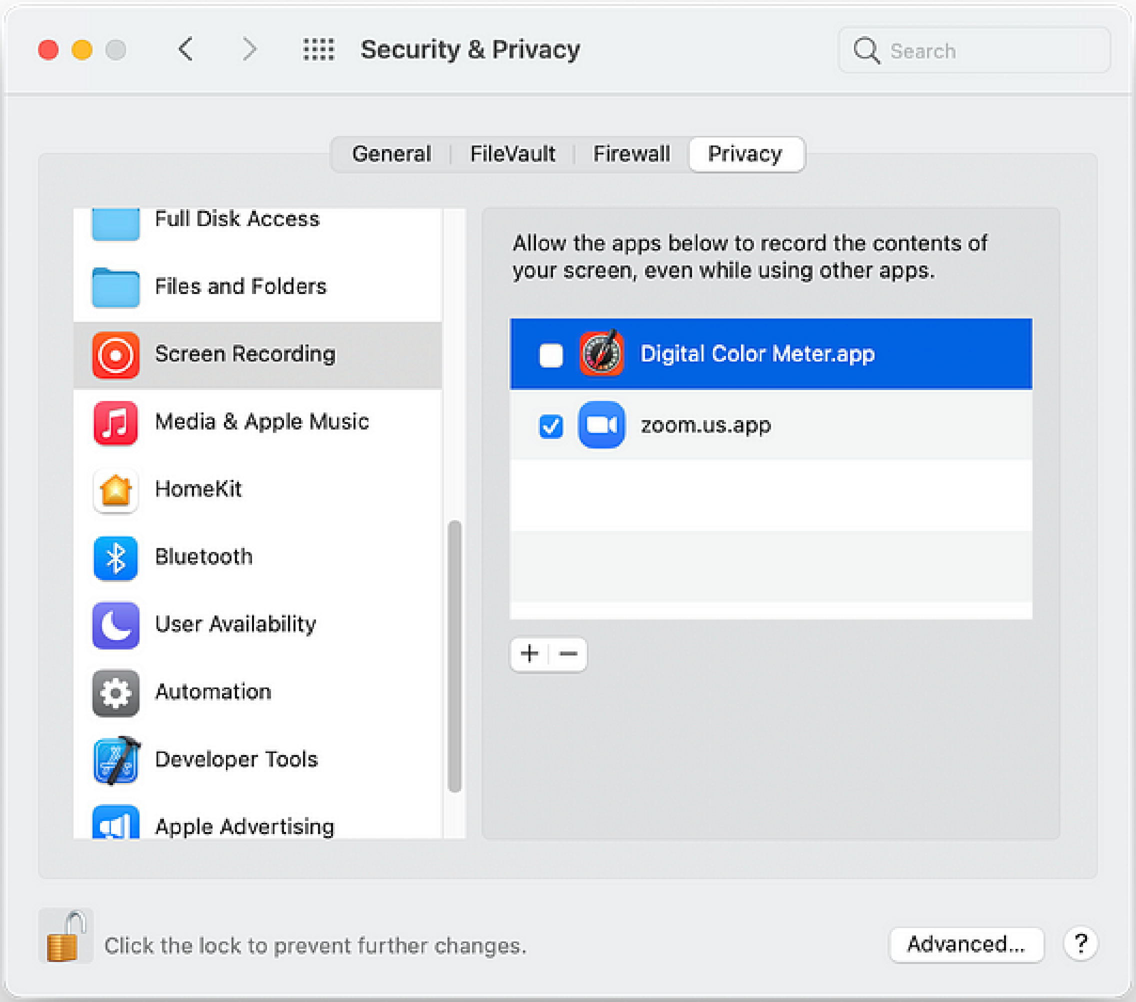
Screen Recording

"Digital Color Meter.app" would like to record this computer's screen.

Grant access to this application in Security & Privacy preferences, located in System Preferences.

Dialog prompted by OS X to ask user if accepts Digital Color Meter to get screenshots.

We now can allow our modified *Digital Color Meter* to record the screen. See the *Security & Privacy* Settings bellow.



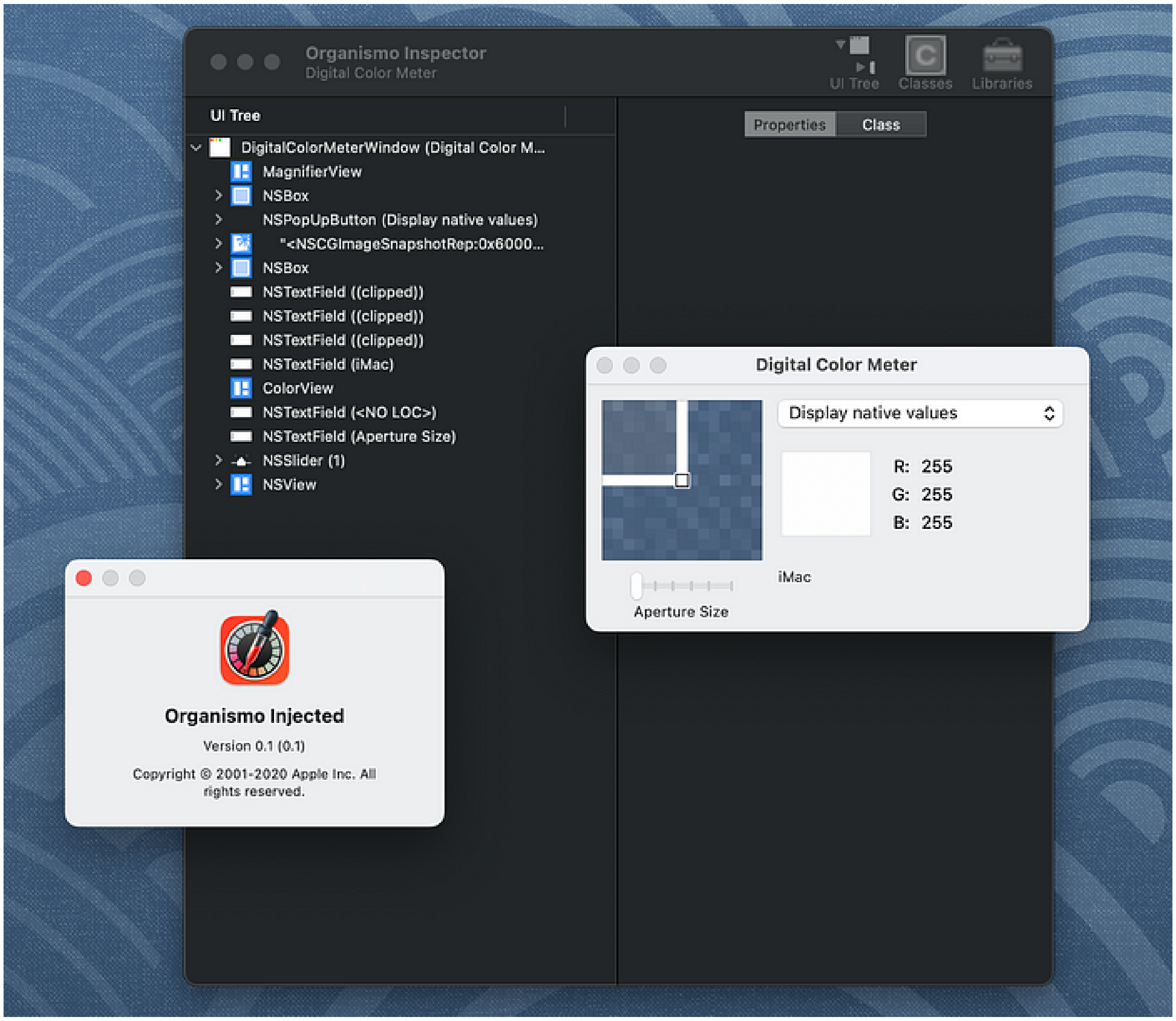
Allow the apps below to record the contents of your screen, even while using other apps.

- [ ] Digital Color Meter.app
- [x] zoom.us.app

Click the lock to prevent further changes.

Security & Privacy settings for Screen Recording.

Now *Digital Color Meter* is injectable. See bellow *Organismo Inpector* within the *Digital Color Meter* App:



```
UI Tree
DigitalColorMeterWindow (Digital Color M...
  MagnifierView
  NSBox
  NSPopUpButton (Display native values)
  "<NSCGImageSnapshotRep:0x6000...
  NSBox
  NSTextField ((clipped))
  NSTextField ((clipped))
  NSTextField ((clipped))
  NSTextField (iMac)
  ColorView
  NSTextField (<NO LOC>)
  NSTextField (Aperture Size)
  NSSlider (1)
  NSView
```

R: 255
G: 255
B: 255

Organismo Injected
Version 0.1 (0.1)
Copyright © 2001-2020 Apple Inc. All rights reserved.

Organismo instrumentation showing inners of Digital Color Meter.

Advanced Apple Apps Can’t be Injected

Most of the advanced Apps like *Stocks* can’t be injected (with SIP AMFI enabled). The amount of private entitlements it has will make it impossible. Either removing the signature or resigning with our *Dev Certificate* we will never be able to enable the features it requires.

Here an example of the entitlements of *Stocks.app,* which most of them will only be accepted in Apple signed binaries:

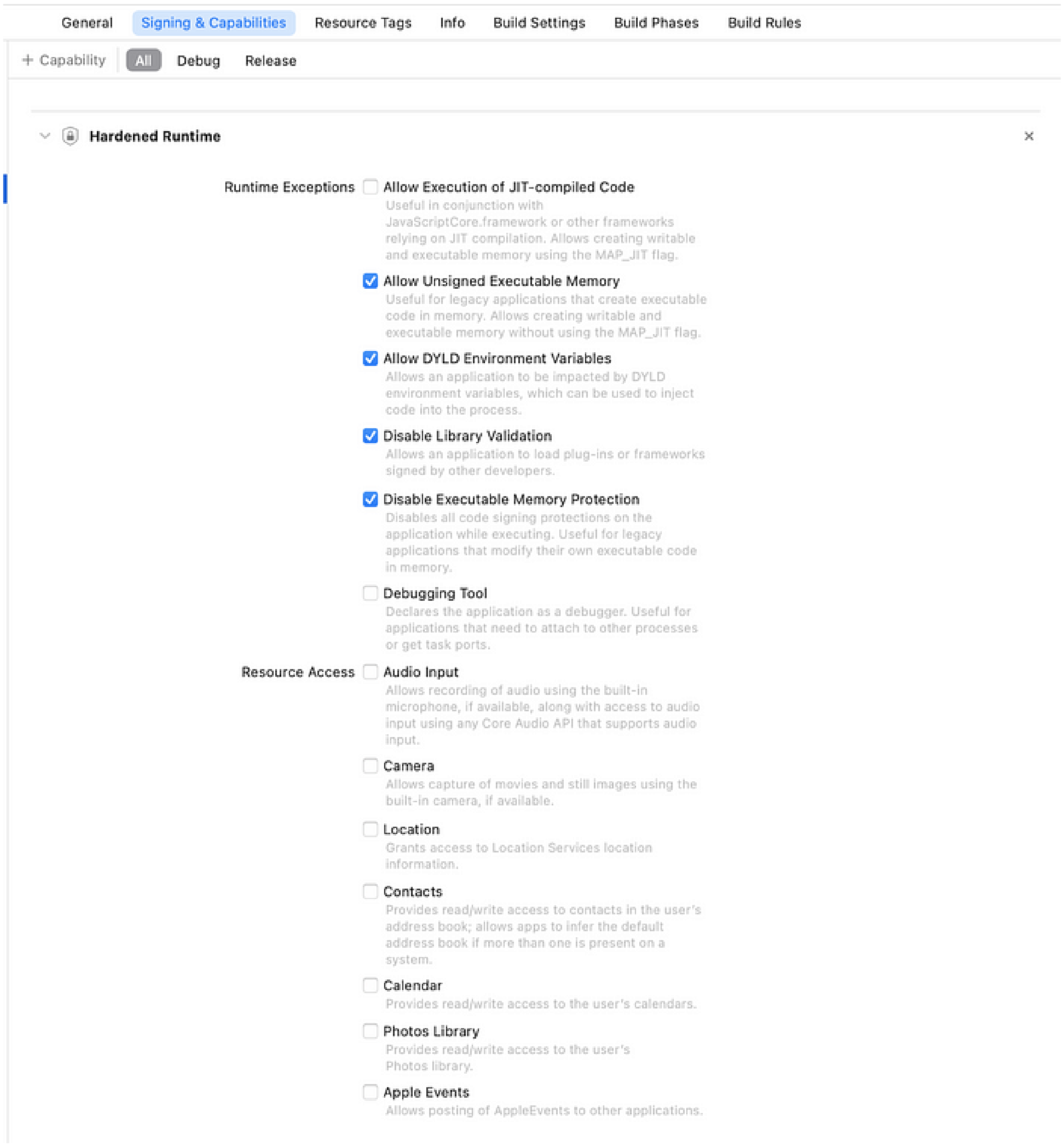
```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST
1.0//EN" ""><plist version="1.0"><dict><key>aps-environment</key>
<string>production</string><key>fairplay-client</key><string>1417937365</string>
<key>com.apple.itunesstored.private</key><true/><key>com.apple.security.app-
sandbox</key><true/><key>com.apple.authkit.client.private</key><true/>
<key>com.apple.private.swc.system-app</key><true/>
<key>com.apple.proactive.eventtracker</key><true/><key>com.apple.private.iad.news-
client</key><true/><key>com.apple.private.iosmac.unscaled</key><true/>
<key>com.apple.security.network.client</key><true/>
<key>com.apple.private.cloudkit.spi</key><array><string>true</string></array>
<key>com.apple.private.canGetAppLinkInfo</key><true/>
<key>com.apple.coretelephony.Identity.get</key><true/>
<key>com.apple.locationd.effective_bundle</key><true/>
<key>com.apple.private.applemediaservices</key><true/>
<key>com.apple.private.appstorecomponents</key><true/>
<key>com.apple.private.associated-domains</key><true/><key>com.apple.private.iad.opt-
in-control</key><true/><key>com.apple.CommCenter.fine-grained</key><array>
<string>spi</string></array><key>com.apple.companionappd.connect.allow</key><true/>
<key>com.apple.developer.associated-domains</key><array></array>
<key>com.apple.private.appstored</key><array><string>IAPHistory</string></array>
<key>com.apple.private.accounts.allaccounts</key><true/>
<key>com.apple.private.xpc.domain-extension</key><true/>
<key>com.apple.springboard.opensensitiveurl</key><true/>
<key>com.apple.frontboard.launchapplications</key><true/>
<key>com.apple.mediaremote.set-playback-state</key><true/>
<key>com.apple.private.cloudkit.systemService</key><true/>
<key>com.apple.private.cloudkit.setEnvironment</key><true/>
<key>com.apple.private.network.socket-delegate</key><true/>
<key>com.apple.private.security.storage.Stocks</key><true/>
<key>com.apple.accounts.appleaccount.fullaccess</key><true/>
<key>com.apple.private.cfnetwork.har-capture-amp</key><true/>
<key>com.apple.developer.icloud-services</key><array><string>CloudKit</string>
</array><key>com.apple.private.canModifyAppLinkPermissions</key><true/>
<key>com.apple.private.security.container-required</key><true/>
<key>com.apple.private.security.system-application</key><true/>
<key>com.apple.private.coreservices.canopenactivity</key><true/>
<key>com.apple.coretelephony.CTCarrierSettings.allow</key><true/>
<key>com.apple.private.coreservices.canmaplsdatabase</key><true/><key>application-
identifier</key><string>ZL6BUSYGB3.com.apple.stocks</string>
<key>com.apple.private.networkextension.configuration</key><true/>
<key>com.apple.springboard.allowIconVisibilityChanges</key><true/>
<key>com.apple.security.personal-information.calendars</key><true/>
<key>com.apple.private.tcc.allow</key><array><string>kTCCServiceAddressBook</string>
</array><key>com.apple.security.personal-information.addressbook</key><true/>
<key>com.apple.private.hid.client.event-dispatch.internal</key><true/>
<key>com.apple.private.mobileinstall.xpc-services-enabled</key><true/>
<key>com.apple.notificationcenter.widgetcontrollerhascontent</key><true/>
<key>com.apple.developer.icloud-container-environment</key>
<string>Production</string><key>com.apple.application-identifier</key>
<string>ZL6BUSYGB3.com.apple.stocks</string><key>com.apple.developer.ubiquity-
kvstore-identifier</key><string>com.apple.stocks</string>
<key>com.apple.private.appleaccount.app-hidden-from-icloud-settings</key><true/>
```

```
<key>com.apple.springboard.hardware-button-service.event-consumption</key><true/>
<key>com.apple.telephonyutilities.callservicesd</key><array><string>access-call-
providers</string></array>
<key>com.apple.private.MobileGestalt.AllowedProtectedKeys</key><array>
<string>UniqueDeviceID</string></array>
<key>com.apple.private.DistributedEvaluation.RecordAccess-com.apple.stocks.des</key>
<true/><key>com.apple.security.exception.shared-preference.read-only</key><array>
<string>com.apple.AdPlatforms</string></array><key>com.apple.security.application-
groups</key><array><string>group.com.apple.stocks</string>
<string>group.com.apple.stocks-news</string></array>
<key>com.apple.security.exception.files.home-relative-path.read-write</key><array>
<string>/Library/Caches/com.apple.AppleMediaServices/</string>
<string>/Library/News/</string></array><key>com.apple.security.temporary-
exception.files.home-relative-path.read-write</key><array>
<string>/Library/Caches/com.apple.AppleMediaServices/</string>
<string>/Library/News/</string></array><key>com.apple.security.exception.shared-
preference.read-write</key><array><string>com.apple.AdPlatforms</string>
<string>com.apple.newscore</string><string>com.apple.newscore2</string>
<string>com.apple.stocks.account</string><string>com.apple.stocks.iAd</string>
</array><key>com.apple.security.temporary-exception.shared-preference.read-
write</key><array><string>com.apple.newscore</string>
<string>com.apple.newscore2</string><string>com.apple.stocks.account</string>
<string>com.apple.stocks.iAd</string><string>com.apple.AppStoreComponents</string>
<string>com.apple.AdPlatforms</string></array>
<key>com.apple.private.cloudkit.serviceNameForContainerMap</key><dict>
<key>com.apple.stocks.private</key><string>com.apple.stocks.private</string>
<key>com.apple.news.private.secure</key>
<string>com.apple.news.private.secure2</string>
<key>com.apple.news.private.secure.qa</key>
<string>com.apple.news.private.secure2</string>
<key>com.apple.news.private.secure.staging</key>
<string>com.apple.news.private.secure2</string></dict>
<key>com.apple.security.temporary-exception.mach-lookup.global-name</key><array>
<string>com.apple.commerce</string><string>com.apple.siri-distributed-
evaluation</string><string>com.apple.appstored.xpc</string>
<string>com.apple.appstored.xpc.request</string><string>com.apple.ap.adprivacyd.opt-
out</string><string>com.apple.appstoreagent.xpc</string>
<string>com.apple.companionappd</string>
<string>com.apple.ap.adservicesd.server</string>
<string>com.apple.ap.adprivacyd.attribution</string>
<string>com.apple.ap.promotedcontent.pcinterface</string>
<string>com.apple.ap.promotedcontent.mescalinterface</string>
<string>com.apple.ap.promotedcontent.metrics</string>
<string>com.apple.ak.anisette.xpc</string><string>com.apple.newsd.analytics</string>
</array><key>com.apple.security.exception.mach-lookup.global-name</key><array>
<string>com.apple.commerce</string><string>com.apple.siri-distributed-
evaluation</string><string>com.apple.appstored.xpc</string>
<string>com.apple.appstored.xpc.request</string>
<string>com.apple.appstoreagent.xpc</string><string>com.apple.companionappd</string>
<string>com.apple.ap.adservicesd.server</string>
<string>com.apple.ap.adprivacyd.attribution</string>
<string>com.apple.ap.adprivacyd.opt-out</string>
```

```
<string>com.apple.ak.anisette.xpc</string><string>com.apple.newsd.analytics</string>
<string>com.apple.ap.promotedcontent.pcinterface</string>
<string>com.apple.ap.promotedcontent.metrics</string>
<string>com.apple.ap.promotedcontent.mescalinterface</string>
<string>com.apple.ap.promotedcontent.supportinterface</string>
<string>com.apple.appstorecomponentsd.xpc</string></array>
<key>com.apple.developer.icloud-container-identifiers</key><array>
<string>com.apple.news.public</string><string>com.apple.news.private</string>
<string>com.apple.news.private.secure</string>
<string>com.apple.news.public.staging</string>
<string>com.apple.news.private.staging</string>
<string>com.apple.news.private.secure.staging</string>
<string>com.apple.news.public.qa</string><string>com.apple.news.private.qa</string>
<string>com.apple.news.private.secure.qa</string>
<string>com.apple.news.public.sandbox</string>
<string>com.apple.news.private.sandbox</string>
<string>com.apple.news.private.secure.sandbox</string>
<string>com.apple.news.public.demo1</string>
<string>com.apple.news.private.demo1</string>
<string>com.apple.news.private.secure.demo1</string>
<string>com.apple.news.public.demo2</string>
<string>com.apple.news.private.demo2</string>
<string>com.apple.news.private.secure.demo2</string>
<string>com.apple.stocks.private</string></array></dict></plist
```

> When a developer wants to distribute an App it should mark it as hardened, it would make the App much less vulnerable. This is done by adding the “Hardened Runtime” capability in Xcode (Signing & Capabilities section).

The *Runtime Exceptions*, would relax some of the security features imposed by the hardening. As we saw earlier, the Apps signed with the *Hardened Runtime* would have the signature flag ‘runtime’.


General | Signing & Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

+ Capability | All | Debug | Release

Hardened Runtime

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.
- [ ] Contacts
  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.
- [ ] Apple Events
  Allows posting of AppleEvents to other applications.

Runtime Exceptions in Xcode.

These are the entitlements for the *Runtime Exceptions*. They would be inserted into the signature of our binary:

com.apple.security.cs.allow-dyld-environment-variablescom.apple.security.cs.disable-library-validationcom.apple.security.cs.disable-executable-page-protectioncom.apple.security.cs.disable-executable-page-protection

This is a must read Apple documentation in Hardened Runtime. Manage security protections and resource access for your macOS apps.

1. Allow DYLD Environment Variables Entitlement

A Boolean value that indicates whether the app may be affected by dynamic linker environment variables, which you can use to inject code into your app's process.

Key: **com.apple.security.cs.allow-dyld-environment-variables**

2. Disable Library Validation Entitlement

A Boolean value that indicates whether the app loads arbitrary plug-ins or frameworks, without requiring code signing.

Key: **com.apple.security.cs.disable-library-validation**

3. Disable Executable Memory Protection Entitlement

A Boolean value that indicates whether to disable all code signing protections while launching an app, and during its execution.

Key: **com.apple.security.cs.disable-executable-page-protection**

4. Allow Unsigned Executable Memory Entitlement

A Boolean value that indicates whether the app may create writable and executable memory without the restrictions imposed by using the `MAP_JIT`flag.

Key: **com.apple.security.cs.allow-unsigned-executable-memory**

Injecting into Third Party Apps

Let's pick an non-Apple App to experiment with injection: *WhatsApp.* For a start is correctly signed as hardened (*runtime* flag):

```
$ codesign -dvvv
/Applications/WhatsApp.appExecutable=/Applications/WhatsApp.app/Contents/MacOS/WhatsAp
 bundle with Mach-O thin (x86_64)CodeDirectory v=20500 size=1820 flags=0x10000()
hashes=46+7 location=embedded
```

WhatsApp entitlements:

```
$ codesign -d — entitlements :-
/Applications/WhatsApp.appExecutable=/Applications/WhatsApp.app/Contents/MacOS/WhatsAp
xml version=”1.0" encoding=”UTF-8"?><!DOCTYPE plist PUBLIC “-//Apple//DTD PLIST
1.0//EN” “"><plist version=”1.0"><dict><key>com.apple.security.device.usb</key>
<true/><key>com.apple.security.app-sandbox</key><true/>
<key>com.apple.security.device.camera</key><true/>
<key>com.apple.security.network.client</key><true/>
<key>com.apple.security.network.server</key><true/>
<key>com.apple.security.device.bluetooth</key><true/>
<key>com.apple.security.device.microphone</key><true/>
<key>com.apple.security.device.audio-input</key><true/><key>com.apple.developer.team-
identifier</key><string>XXXXXXXXX</string>
<key>com.apple.security.files.downloads.read-write</key><true/>
<key>com.apple.security.files.user-selected.read-write</key><true/>
<key>com.apple.security.cs.allow-unsigned-executable-memory</key><true/>
<key>com.apple.application-identifier</key><string>XXXXXXXX.desktop.WhatsApp</string>
<key>com.apple.security.application-groups</key>
<string>XXXXXXXX.desktop.WhatsApp</string></dict></plist>
```

Those entitlements are familiar to any developer, they can be set in Xcode. We can see that it is sandboxed with some permission to reach the camera, being a network server & client etc.

It does have as well a hardened relaxation to allow unsigned code execution: **com.apple.security.cs.allow-unsigned-executable-memory**.

Review of Our Usual Options

These are our usual options :

1. Remove all signatures.
2. Resign without the hardening. Resign with our Dev certificate, without the ‘runtime’ option.
3. Resign with Hardening and add relaxing entitlements. Resign with our Dev certificate, with the ‘runtime’ option and add extra entitlements to allow injection. i.e to add the entitlement .

By experience I know that resigning with a *Dev Certificate* makes the *amfi* validate the code against a provision profile that validates security entitlements. This is the message we get after resigning the App:

```
Failure validating against provisioning profiles: No eligible provisioning profiles
found during verification
```

Actually the simplest approach has given the best result.

Solution. Removing the Signature.

In this particular case of *WhatsApp,* the simplest approach, to remove the signature, yields a modified *WhatsApp* with no restrictions that allows dynamic code injection.

Removing a signature is done buy using the command:

```
$ codesign --remove-signature pathtoApp
```

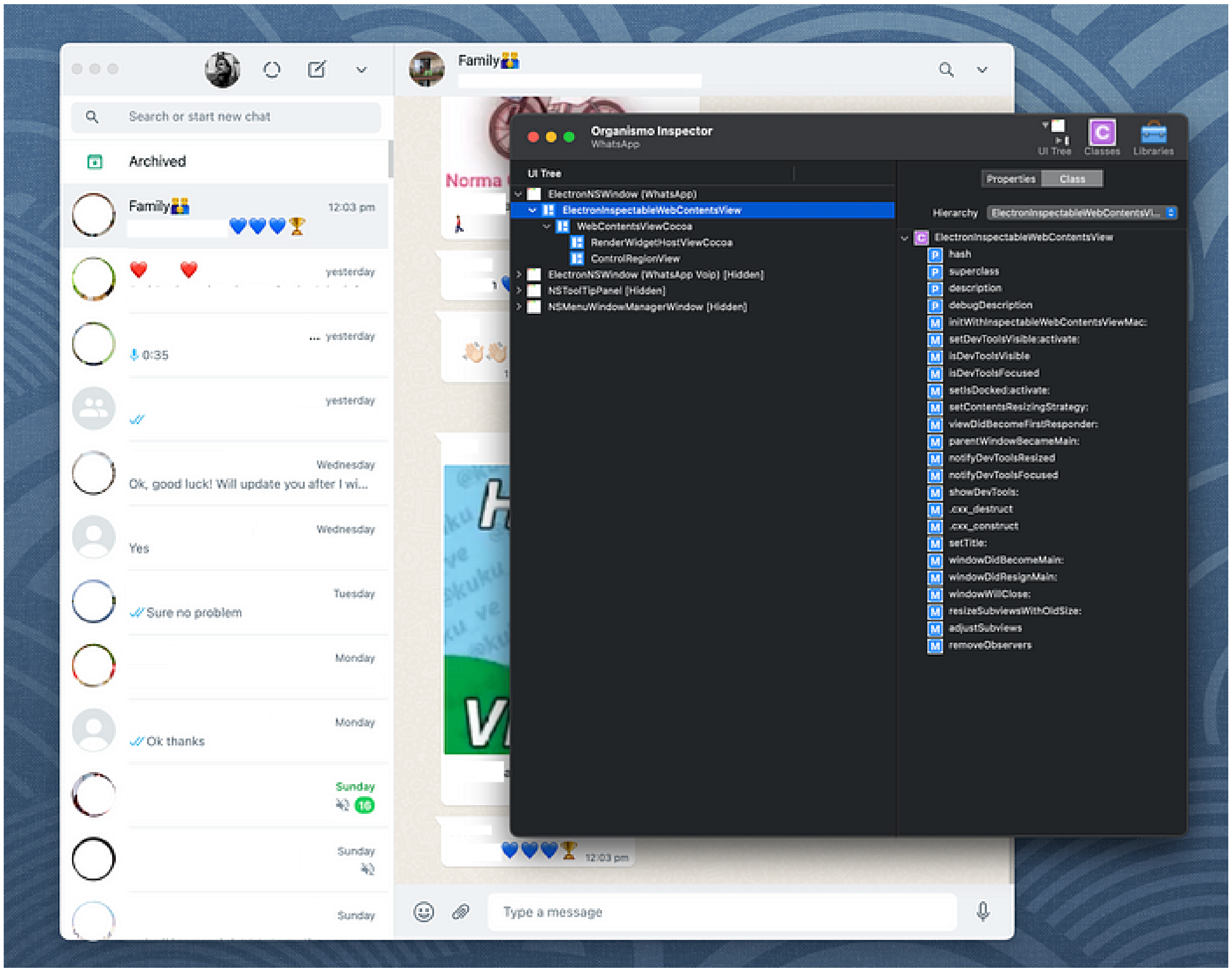
However tha command will not reach to all binaries. The complex folder structure and distribution of binaries would make the removal of all signatures in all binaries within *WhatsApp* quite a lengthy task. Using a tool like simplifies complex codesigning to a single command:

```
$ macho_inject --remove-signature  -o /YourWorkingFolder/Out -i
/pathToACopyOf/WhatsApp.app
```

The resulting *WhatsApp* has no signing, no entitlements, no hardening. The following command will inject *Organismo*:

```
$ DYLD_INSERT_LIBRARIES=/pathTo/Organismo-mac.framework/Versions/A/Organismo-mac
/pathToModified/WhatsApp.app/Contents/MacOS/WhatsApp
```

Here we can see the modified *WhatsApp* with *Organismo* inspector:



```
UI Tree
ElectronNSWindow (WhatsApp)
  ElectronInspectableWebContentsView
    WebContentsViewCocoa
      RenderWidgetHostViewCocoa
      ControlRegionView
ElectronNSWindow (WhatsApp Voip) [Hidden]
NSToolTipPanel [Hidden]
NSMenuWindowManagerWindow [Hidden]
```

```
ElectronInspectableWebContentsView
  P hash
  P superclass
  P description
  P debugDescription
  M initWithInspectableWebContentsViewMac:
  M setDevToolsVisible:activate:
  M isDevToolsVisible
  M isDevToolsFocused
  M setIsDocked:activate:
  M setContentsResizingStrategy:
  M viewDidBecomeFirstResponder:
  M parentWindowBecameMain:
  M notifyDevToolsResized
  M notifyDevToolsFocused
  M showDevTools:
  M .cxx_destruct
  M .cxx_construct
  M setTitle:
  M windowDidBecomeMain:
  M windowDidResignMain:
  M windowWillClose:
  M resizeSubviewsWithOldSize:
  M adjustSubviews
  M removeObservers
```

WhatsApp with Organismo Inspector injected.

Conclusion

Having SIP and AMFI enabled makes the dynamic injection of code in signed-hardened Apps impossible, and very difficult even trying modifications of the App with code-signing techniques. Although some simple Apple Apps are still not protected, most of the important Apple Apps are. In addition, the special entitlements that only Apple can use make the tampering of Apple Apps near impossible.

In the case of Developers Apps, the situation is different due to two main reasons:

1. The entitlements are available to all developers and through codesiging and creation it could be possible to resign the App with new entitlements.
2. Sandboxing and hardening are easy to remove. If the App is not programmed to be aware when signing, sandboxing, hardening are missing, the modified App could operate without any constraints as we saw in the case of

Developers should always harden their Apps and consider *Runtime Integrity Checks* to add protection against tampering by App impersonation (modified bundleid, provision profile, entitlements..) or signature removal.

Update ! All Apps Must be Signed on Apple Silicon

The simplicity to tamper *WhatsApp* was worrisome. The test for this article were made in Monterey 12.5 Intel. Apple is aware and moving forward to make our computers safer and as a result the *WhatsApp* tampering as described above will not be possible on Apple silicon Macs, all executables will have to be signed :

(https://developer.apple.com/documentation/macos-release-notes/macos-big-sur-11_0_1-universal-apps-release-notes)

“New in macOS 11 on Apple silicon Mac computers, and starting in the next macOS Big Sur 11 beta, the operating system will enforce that any executable must be signed with a valid signature before it’s allowed to run. There isn’t a specific identity requirement for this signature: a simple ad-hoc signature issued locally is sufficient, which includes signatures which are now generated automatically by the linker. This new behavior doesn’t change the long-established policy that our users and developers can run arbitrary code on their Macs, and is designed to simplify the execution policies on Apple silicon Mac computers and enable the system to better detect code modifications.
This new policy doesn’t apply to translated x86 binaries running under Rosetta, nor does it apply to macOS 11 running on Intel platforms.”

Thanks !

How to Inject Code into Mach-O Apps. Part I.

In this story we are going to research how to add your own code to already compiled Apps, i.e. any App you may have on…

jon-gabilondo-angulo-7635.medium.com

How to Inject Code into Mach-O Apps. Part II.

In Part I we saw how easy it is to inject code into Mac Apps, from Calculator to Mail, even more surprisingly, into…

jon-gabilondo-angulo-7635.medium.com

GitHub - JonGabilondoAngulo/macho-inject: OS X command line tool to inject Frameworks and dylibs on…

OS X command line tool to inject Frameworks and dylibs on mach-o binaries. It does the injection of the framework and…

---

github.com