

Модифицируем поведение оригинальных приложений из AppStore

**Кто мы такие и чем
занимаемся?**

Задачи нашей КОМАНДЫ

Работаем над тем, что индустрия
называет UEM

Помогаем удалённо настраивать
работу устройств в компании

Работаем с бизнесом (B2B)

Проводим исследования

Используем необычный для iOS-
разработчиков стек

Помогаем заказчикам с
необычными хотелками

Какие хотелки бывают

Мы видели в документации вот такую возможность, реализуйте!

Мы видели у других ребят вот такую функциональность, дайте нам тоже!

Мы видели, что согласно документации так нельзя, а нам хотелось бы. Придумайте как!

Мы видели вот такое приложение, нам надо его, но по-другому!

То же самое, но с изменениями

Disclaimer

Исключительно в образовательных целях

Материалы доклада предназначены для обучения, анализа механизмов защиты и повышения квалификации специалистов по информационной безопасности.

Соблюдение законодательства

Автор не призывает к совершению противоправных действий. Все операции реверс-инжиниринга выполнены в рамках ст. 1280 ГК РФ (исследование функционирования программы) либо на специально созданных тестовых стендах.

Отсутствие вредоносного умысла

Целью подготовки материала не был обход средств защиты для нанесения ущерба правообладателям. Никакое коммерческое ПО не модифицировалось для незаконного использования.

Что может быть нужно?

Убрать лишнюю функциональность

Добавить новую функциональность

Примеры

Убрать лишнюю авторизацию

Убрать целую фичу из приложения

Ограничить возможность делиться экраном и делать скриншоты

Добавить новый тип авторизации

Добавить логирование

Добавить взаимодействие с новым бэкендом

Что делаем мы

Проводим анализ
осуществимости

Реализуем

План на сегодня

01

Получаем расшифрованный IPA
из App Store

02

Анализируем приложение.

03

Вносим изменения.

04

Переподписываем.

05

Запускаем и тестируем.

1

Получаем расшифрованный іра

Откуда взять іра?

Конечно же, из App Store

Звучит логично, но все IPA в App Store защищены DRM.

Все они зашифрованы и получить ключ с устройства невозможно

До тех пор, пока у вас нет возможности выполнять код из-под рута.

Немного про DRM

Digital Rights Management

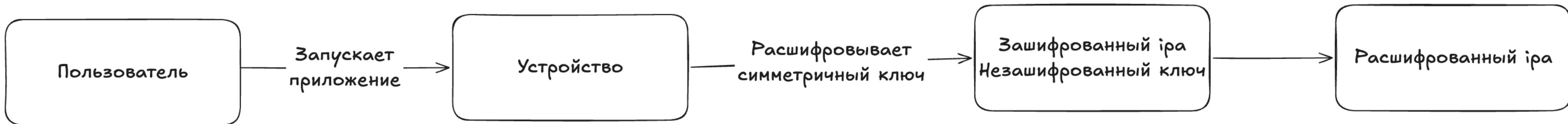
Немного про DRM



Немного про DRM



Немного про DRM



Немного про DRM

Все эти этапы ОС выполняет при каждом запуске приложения, чтобы на диске всегда лежала его зашифрованная версия.

Мы не замечаем дешифрования бинарников, так как:

Это происходит на специальном чипе, задача которого — выполнение криптографических операций.

Бинарники дешифруются не целиком, а частично, по мере необходимости.

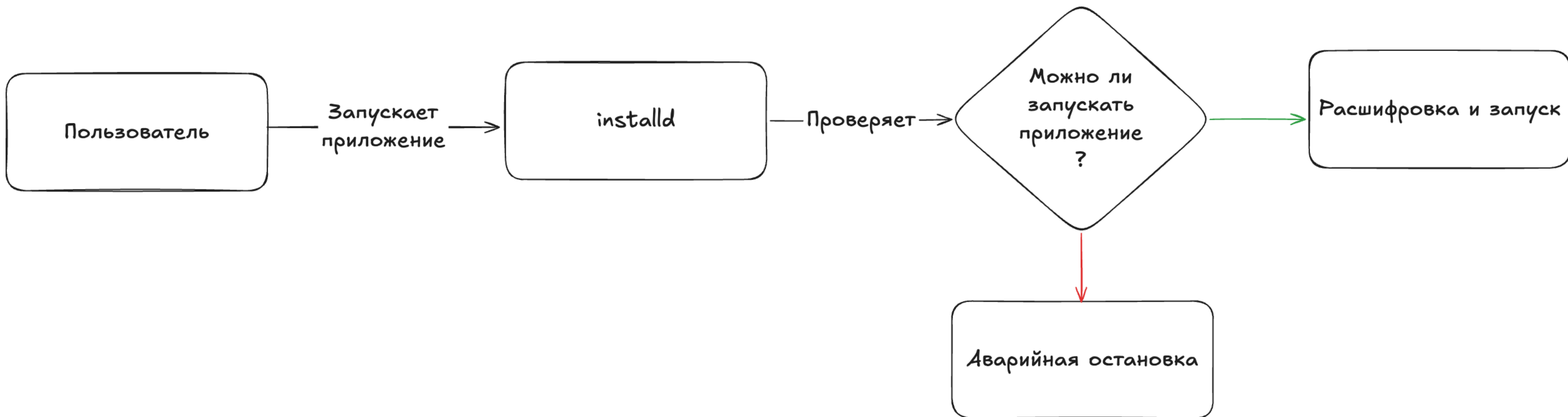
Для расшифровки іра
необходимо инициировать
запуск приложения.



Подготовка к запуску

Как запускается .app

Подготовка к запуску



Подготовка к запуску

Какие проверки осуществляет installd?

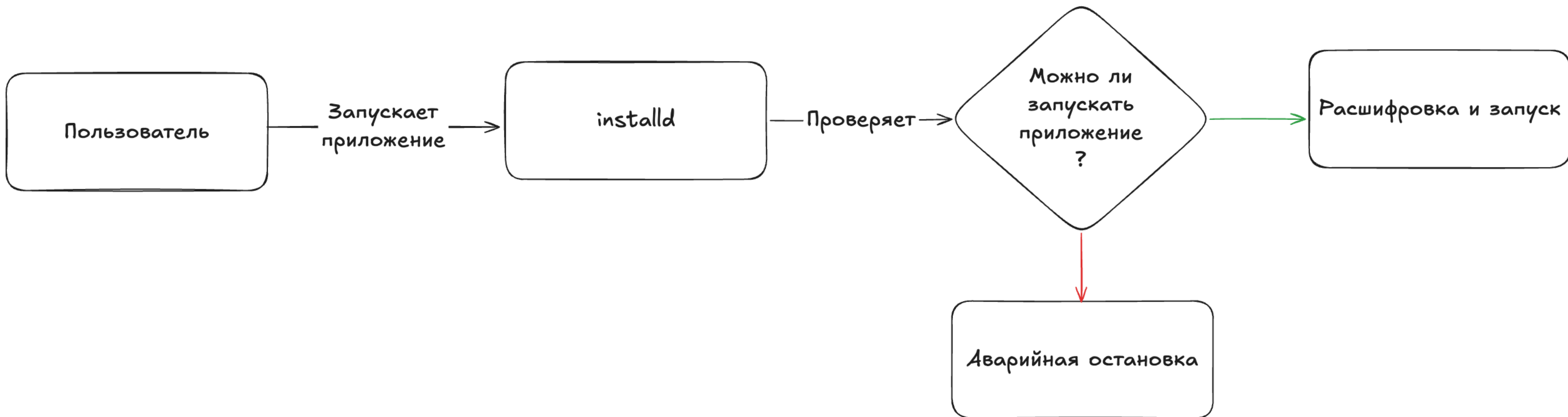
Указано ли в приложении, что оно предназначено для текущей ОС?

Поддерживается ли минимальный deployment target?

Все ли компоненты приложения теоретически могут запускаться на этом устройстве?

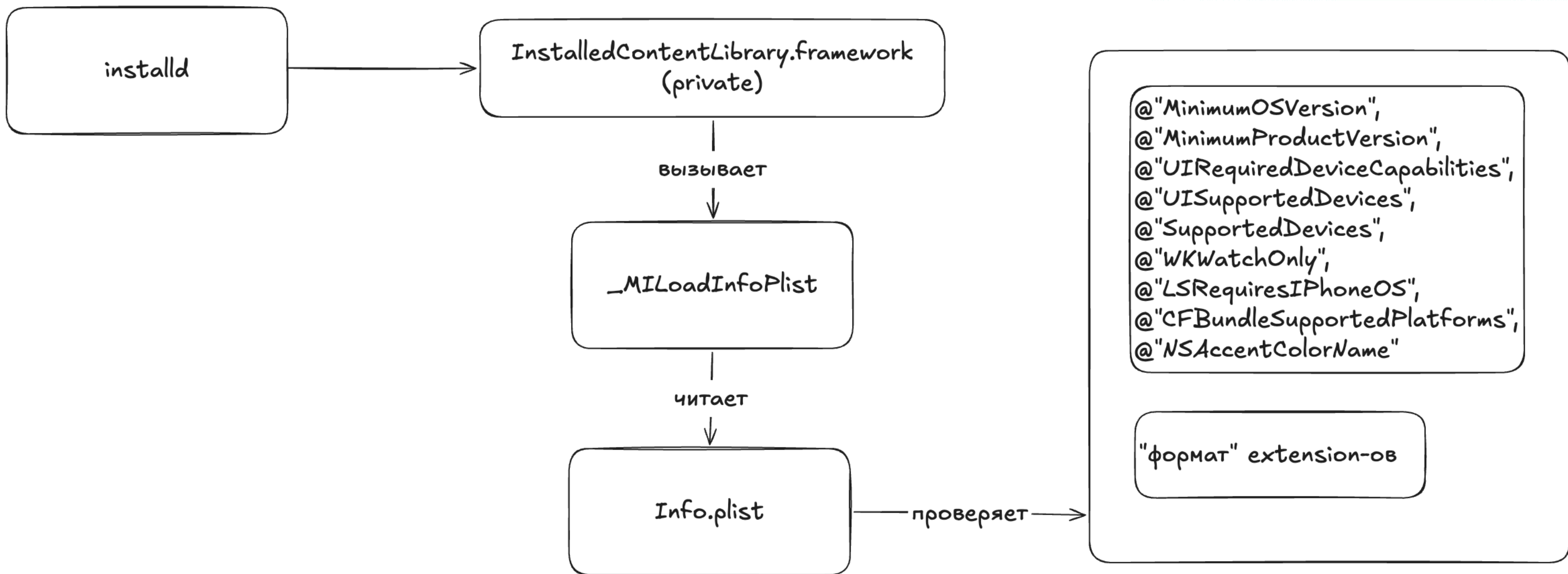
Подготовка к запуску

Работа installd



Подготовка к запуску

Работа installd



Как доставали іра раньше?

01

Берём устройство.

02

Накатываем на него джейлбрейк.

03

С помощью популярных утилит дешифруем приложение.

04

Загружаем его на мак.

Как сейчас?

01

Для современных ОС и устройств джейлбрейков нет.

Последнее, что есть, — iPhone X с iOS 18.

02

А что, если приложение написано для iOS 26, а в перспективе — для iOS 27?

Оно может содержать новые расширения, которые устройство не поддерживает. Как с ними поступить?

03

installd всё это отсечёт и запускать не даст.

Конечно, выход есть.

“

**installD — всего лишь
небольшой сервис в
огромном джейлбрейкнутом
устройстве.**

Патчим installd

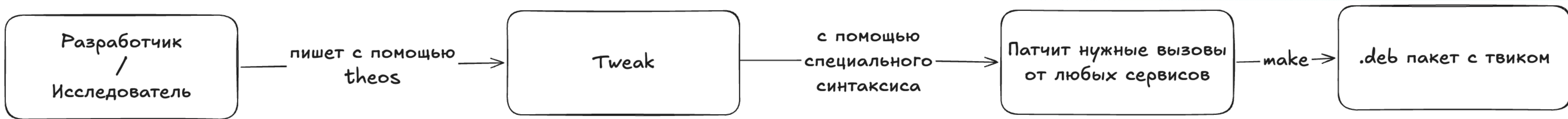
Теоретически на jб-устройстве можно сделать всё что угодно — вопрос лишь во времени и силах.

Удобные способы патчинга, в том числе и приватных бинарников, существуют давно.

Один из них — Theos + ElleKit.

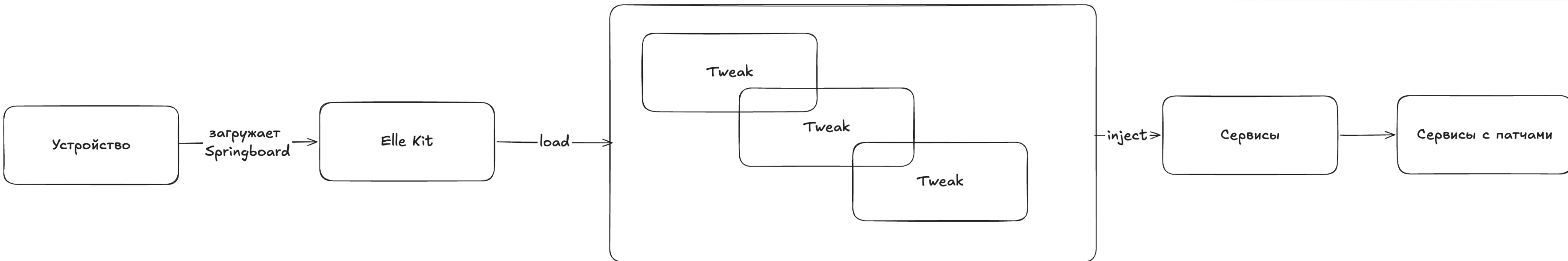
Патчим installD

Принцип работы theos



Патчим installD

Принцип работы Elle Kit / Substrate / ...



Патчим installd

Ищем нужный символ

```
%ctor {  
    ...  
    MSImageRef image = MSGetImageByName("/.../InstalledContentLibrary");  
    ...  
    void *sym1 = MSFindSymbol(image, "_MILoadInfoPlist");  
    ...  
    %init(_MILoadInfoPlist = sym1);  
}
```

Патчим installd

Патчим info.plist

```
void addPrincipleClassForExtension(NSMutableDictionary *plist) {  
    ...  
  
    NSString *bundle_id = [plist objectForKey:@"CFBundleIdentifier"];  
    ...  
  
    NSSet<NSString *> *extensionsRequiredAddingPrincipalClass = [[NSSet<NSString *> alloc] initWithObjects:  
                                                                    @"com.BUNDLE_ID.CalendarWidgetExt",  
                                                                    @"com.BUNDLE_ID",  
                                                                    nil];  
  
    ...  
    extensionDict[@"NSExtensionPrincipalClass"] = @"SOMECLASS";  
    plist[@"NSExtension"] = extensionDict;  
}
```


После патча `installD`

`installD` будет думать, что все extension-ы одного формата

Все бинарники будут якобы для одного deployment target

Успешно установит приложение

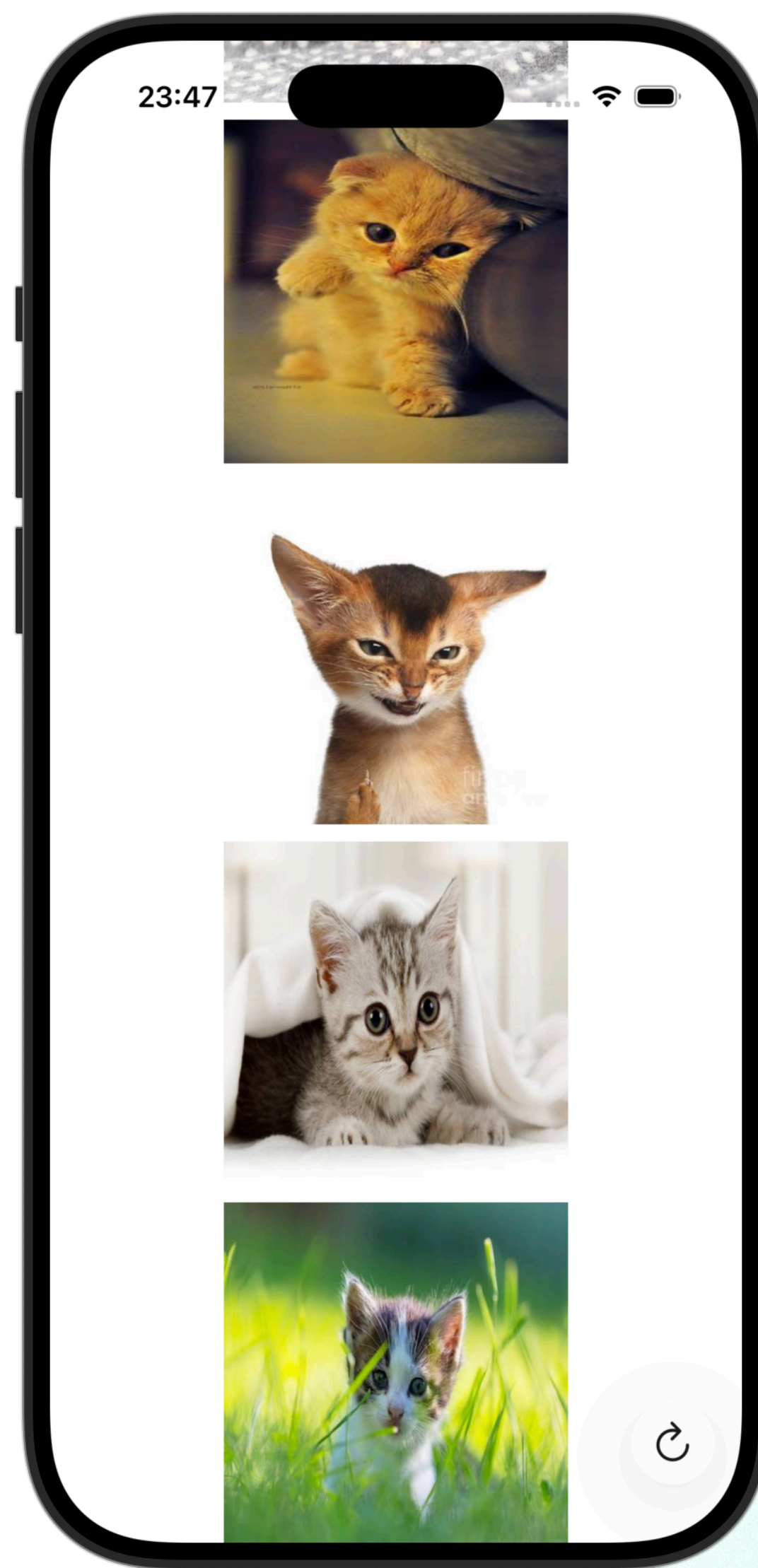
2

Постановка задачи

Постановка задачи

Реальное приложение из AppStore нам брать нельзя, поэтому сами дадим себе приложение для экспериментов

Что может быть лучше котиков



Постановка задачи

Попробуем дооснастить эту компанию котят своим шпионом



Постановка задачи

Изменим поведение кнопки, при нажатии на которую в данные для таблицы добавится наш item.

При перезагрузке страницы item должен отобразиться.

3

Анализируем ПО

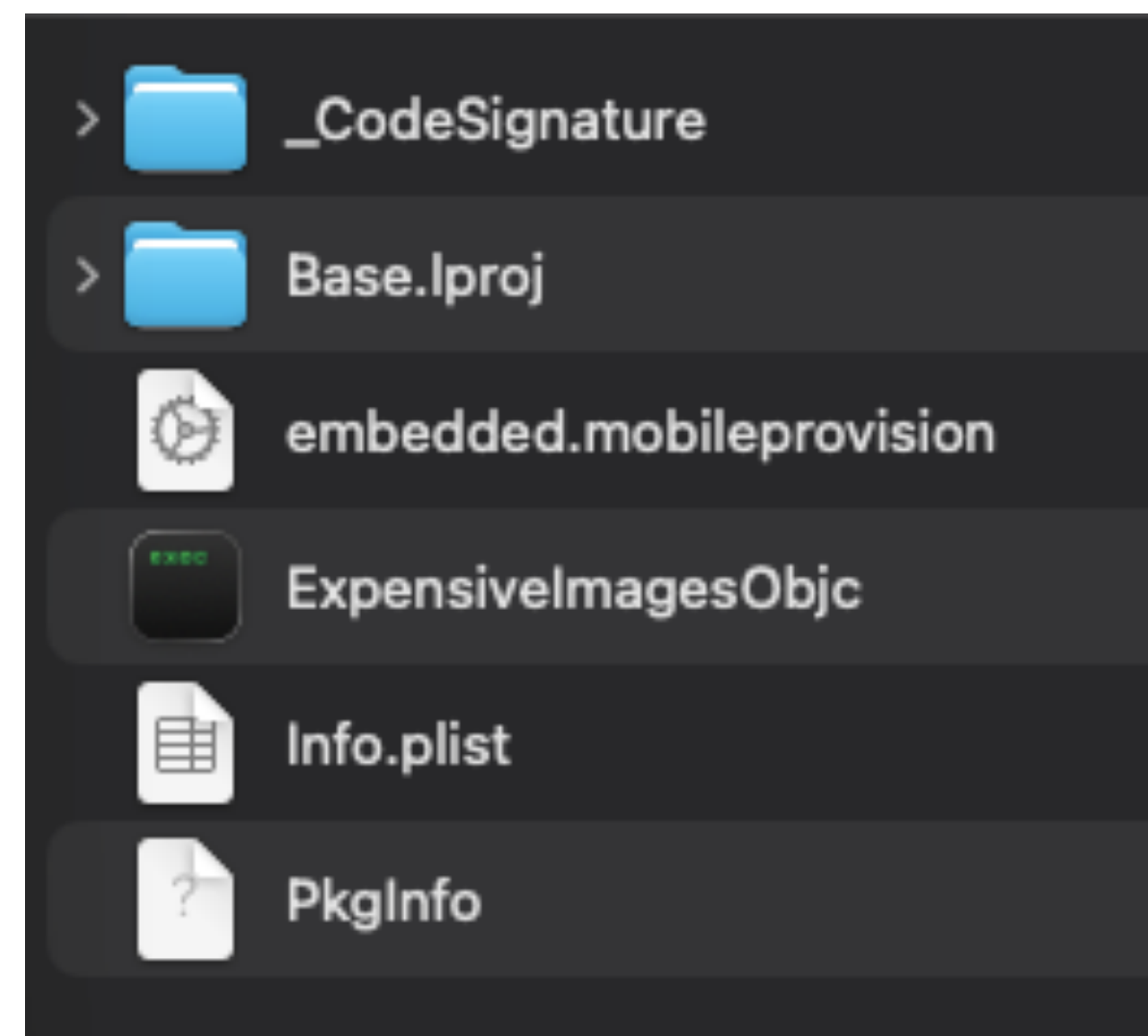
Внутри іра



Внутри іра

Внутри іра лежит как минимум 1 бинарник Mach-O.

Внутри іра



Внутри іра

С полным разбором строения бинарника можно ознакомиться в моём прошлом докладе на Mobius:



Инструменты для анализа



Ghidra

Мощный инструмент от NSA

Open-source

Кросс-платформенный (java)

Работает с практически любыми target-архитектурами и платформами

Для Objc существует отдельный плагин

Качество анализа именно для Objc / Swift ниже, чем у Hopper

Hopper

Closed-source

Платный

Ориентирован только на macOS /
iOS RE

Более богатый анализ, чем у Ghidra
(даже с плагином)

Возможность отладки

Более user-friendly для новичка

Пути анализа

Строки

Имена символов
(актуально для ObjC)

Обращения к системным
фреймворкам

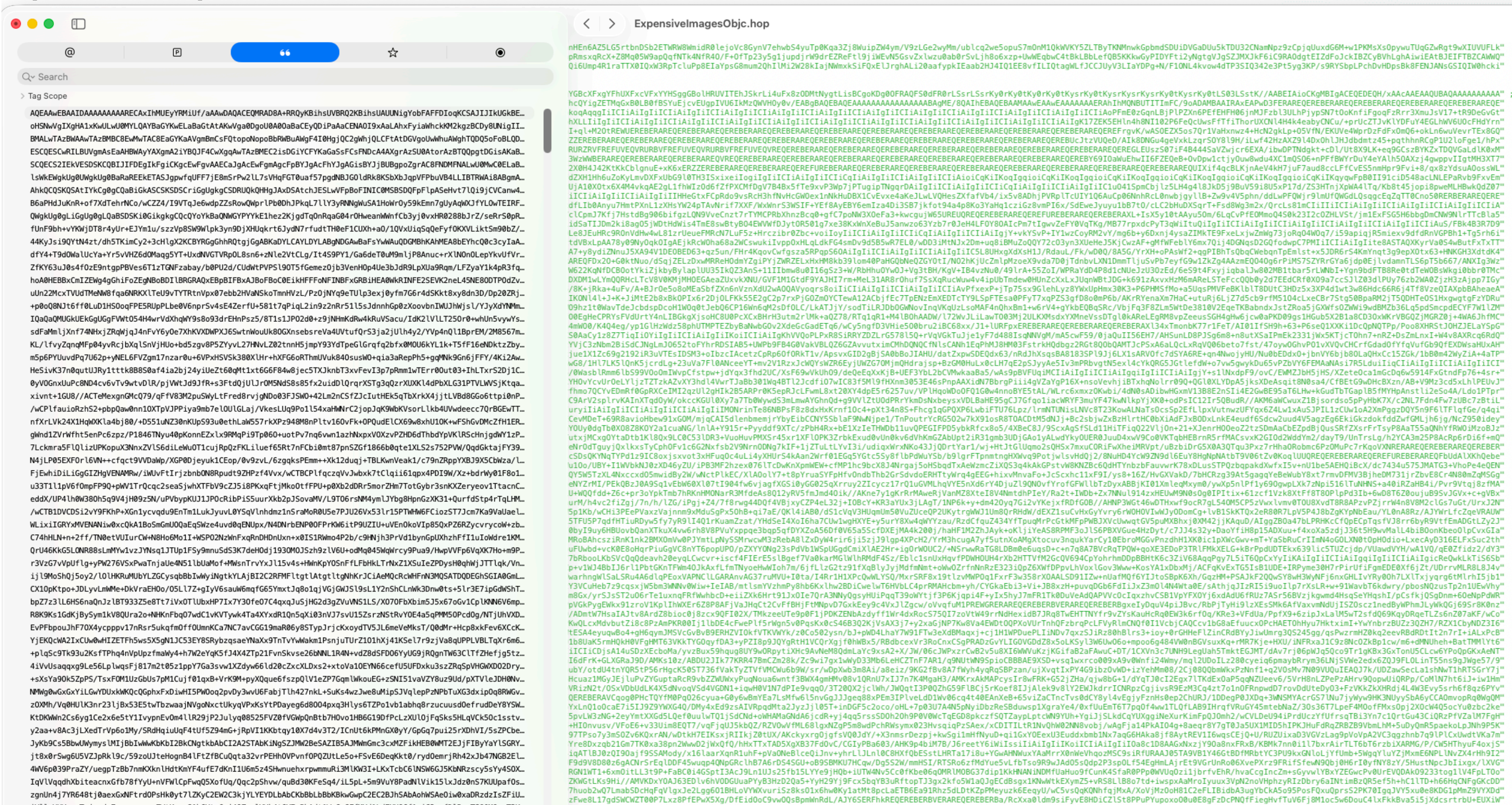
UI-трейсинг

Анализ

Смотрим строки

Анализ

Смотрим строки



Анализ

Смотрим строки

/pk1Eaab2HJ41Q1E8v1L1IQttagWLTJCCJUyV3L1aYDPg+N/F10NL4kvow4dIP3S1Q342e3Pt5yg3KP/s9RY5bpLPchDvHDpsBk8FENJANS6S1Q1W0hck1"

Rename "a9j4aaqskzjrgab" (at 0x10000a385)

Rename "cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBxMTEhUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBolHRU" (at 0x10004c178)

References to "a9j4aaqskzjrgab" (at 0x10000a385)

References from "a9j4aaqskzjrgab" (at 0x10000a385)

References to "cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBxMTEhUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBolHRU" (at 0x10004c178)

References from "cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBxMTEhUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBolHRU" (at 0x10004c178)

Default

Hexadecimal

Decimal

Float

Octal

Characters

Stack Variable

Offset

Address

Address Difference

Binary

Structure

Enumeration

Toggle Signed

Toggle Leading Zeroes

Toggle Negate

⬆ @

⬆ H

⬆ Q

⬆ F

⬆ O

⬆ R

⬆ V

⬆ Z

⬆ Z

⬆ Z

⬆ B

>

>

-

0

⬆ N

LNP6k+y+dm420yq7G12vYKexfRDfG0B//AHNP3WGt46wDTHxwf9ocR7gLS40M5CP5zVwxlvmv0T0U8XvdT8R8APzvPZjrrW4n8V8M2clGs7uGt/UrxJ2N"

JKytrgWwJ1Um8QrRHdW/dEXZ1suCvHxGyYvry6rW0H0V1wWJy0DomCg+lvB1SkKTQx2eR80R7LpV5P4J8bZgKYpNbEau/YL0nA8Rz/AJYWrLfcZqeVRAUW"

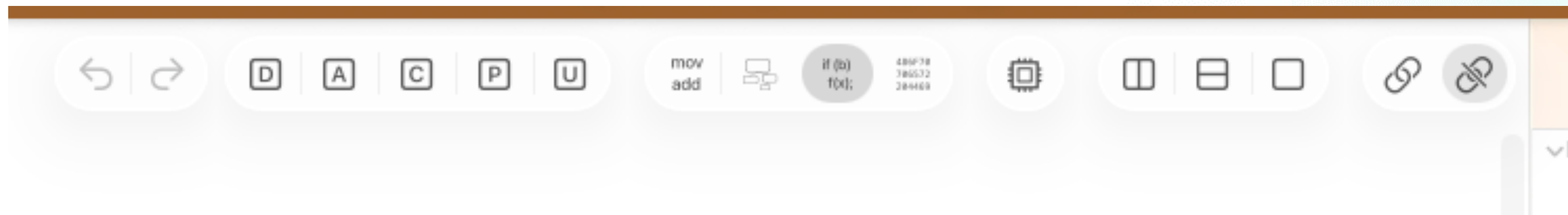
Анализ

Смотрим строки

```
-[ImageStorage loadImage]:
0000000100004000      sub     sp, sp, #0xd0                ; Objective C Implementation defined at 0x100005708 (instance method), DATA XREF=0x100005708
0000000100004004      stp     x28, x27, [sp, #0x70]
0000000100004008      stp     x26, x25, [sp, #0x80]
000000010000400c      stp     x24, x23, [sp, #0x90]
0000000100004010      stp     x22, x21, [sp, #0xa0]
0000000100004014      stp     x20, x19, [sp, #0xb0]
0000000100004018      stp     fp, lr, [sp, #0xc0]
000000010000401c      add     fp, sp, #0xc0
0000000100004020      mov     x25, x0
0000000100004024      adrp    x8, #0x10004c000                ; __stack_chk_guard_10004c0f0@PAGE
0000000100004028      ldr     x8, [x8, #0xf0]                ; __stack_chk_guard_10004c0f0@PAGEOFF, __stack_chk_guard_10004c0f0, __stack_chk_guard
000000010000402c      ldr     x8, [x8]
0000000100004030      stur    x8, [fp, var_58]
0000000100004034      adrp    x2, #0x10004c000                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8@PAGE
0000000100004038      add     x2, x2, #0x158                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8@PAGEOFF, @"/9j/4AA
000000010000403c      bl      _objc_msgSend$parseB64ToImage: ; _objc_msgSend$parseB64ToImage:
0000000100004040      mov     fp, fp
0000000100004044      bl      imp__stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
0000000100004048      mov     x20, x0
000000010000404c      str     x0, [sp, #0xc0 + var_B0]
0000000100004050      adrp    x2, #0x10004c000                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwMTEhUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBoLHRU@PAGE
0000000100004054      add     x2, x2, #0x178                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwMTEhUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBoLHRU@PAGEOFF, @"/9j/4AA
0000000100004058      mov     x0, x25                ; argument #1 for method _objc_msgSend$parseB64ToImage:
000000010000405c      bl      _objc_msgSend$parseB64ToImage: ; _objc_msgSend$parseB64ToImage:
0000000100004060      mov     fp, fp
0000000100004064      bl      imp__stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
0000000100004068      mov     x21, x0
000000010000406c      adrp    x2, #0x10004c000                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8_10004c198@PAGE
0000000100004070      add     x2, x2, #0x198                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8_10004c198@PAGEOFF,
0000000100004074      mov     x0, x25                ; argument #1 for method _objc_msgSend$parseB64ToImage:
0000000100004078      bl      _objc_msgSend$parseB64ToImage: ; _objc_msgSend$parseB64ToImage:
000000010000407c      mov     fp, fp
0000000100004080      bl      imp__stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
0000000100004084      mov     x22, x0
0000000100004088      adrp    x2, #0x10004c000                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8_10004c1b8@PAGE
000000010000408c      add     x2, x2, #0x1b8                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8_10004c1b8@PAGEOFF,
0000000100004090      mov     x0, x25                ; argument #1 for method _objc_msgSend$parseB64ToImage:
0000000100004094      bl      _objc_msgSend$parseB64ToImage: ; _objc_msgSend$parseB64ToImage:
0000000100004098      mov     fp, fp
000000010000409c      bl      imp__stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
00000001000040a0      str     x0, [sp, #0xc0 + var_B8]
00000001000040a4      adrp    x2, #0x10004c000                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8_10004c1b8@PAGE
```


Анализ

Смотрим строки



Анализ

Смотрим строки

```
/* @class ImageStorage */
-(void *)loadImages {
    r31 = r31 - 0xd0;
    var_50 = r28;
    decomp_var_48 = r27;
    var_40 = r26;
    decomp_var_38 = r25;
    var_30 = r24;
    decomp_var_28 = r23;
    var_20 = r22;
    decomp_var_18 = r21;
    var_10 = r20;
    decomp_var_8 = r19;
    saved_fp = r29;
    decomp_var_m8 = r30;
    var_58 = **__stack_chk_guard;
    r0 = [self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/RD"];
    r0 = [r0 retain];
    r20 = r0;
    var_B0 = r0;
    r21 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwMTExUTEhMWFhUVGBYXGBgYGBcXFxgYFhUXFxcVFxYYHSggGBoLHRUVITEhJSkrLi4uFx8z"];
    r22 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    var_B8 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6I"];
    r23 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBw4NCg4IDQgICAgJBwoHBwckBw8ICQcKIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    r24 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    r26 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwoJCAgJCQkICAgHCAoJCAgICBsICQcWIBEWIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    r27 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    r28 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwgHBgkIBwgKCgkLDRYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    r19 = [[self parseB64ToImage:@" /9j /4AAQSkZJRgABAQAAQABAAD /2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU30jo6Iys/I"];
    var_A8 = r20;
    r25 = var_B8;
    r0 = *@class(NSArray_10004c0f8);
    r0 = [r0 arrayWithObjects:&var_A8 count:0xa];
    r20 = [r0 retain];
    [r19 release];
    [r28 release];
    [r27 release];
    [r26 release];
    [r24 release];
    [r23 release];
    [r25 release];
    [r22 release];
    [r21 release];
    r0 = [var_B0 release];
    if (**__stack_chk_guard != var_58) {
        __stack_chk_fail();
    }
    r0 = [r20 autorelease];
    return r0;
}
```


Анализ

Обращения к системным фреймворкам

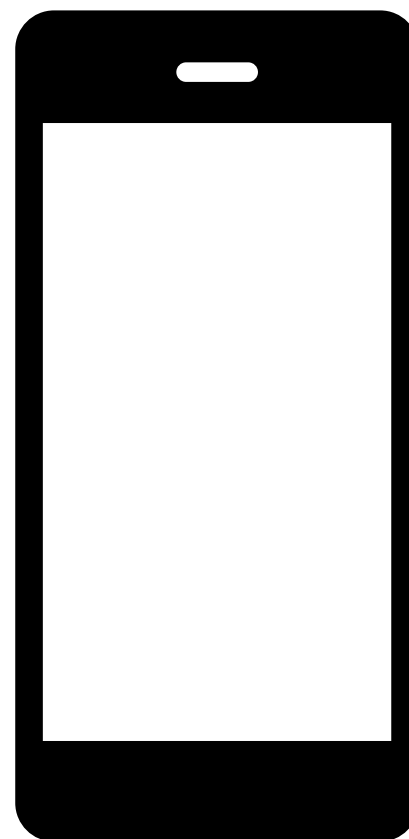
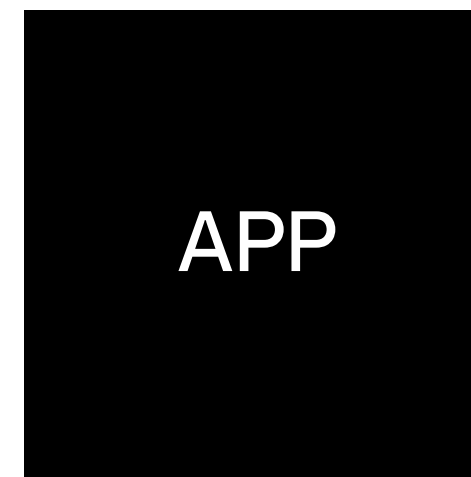
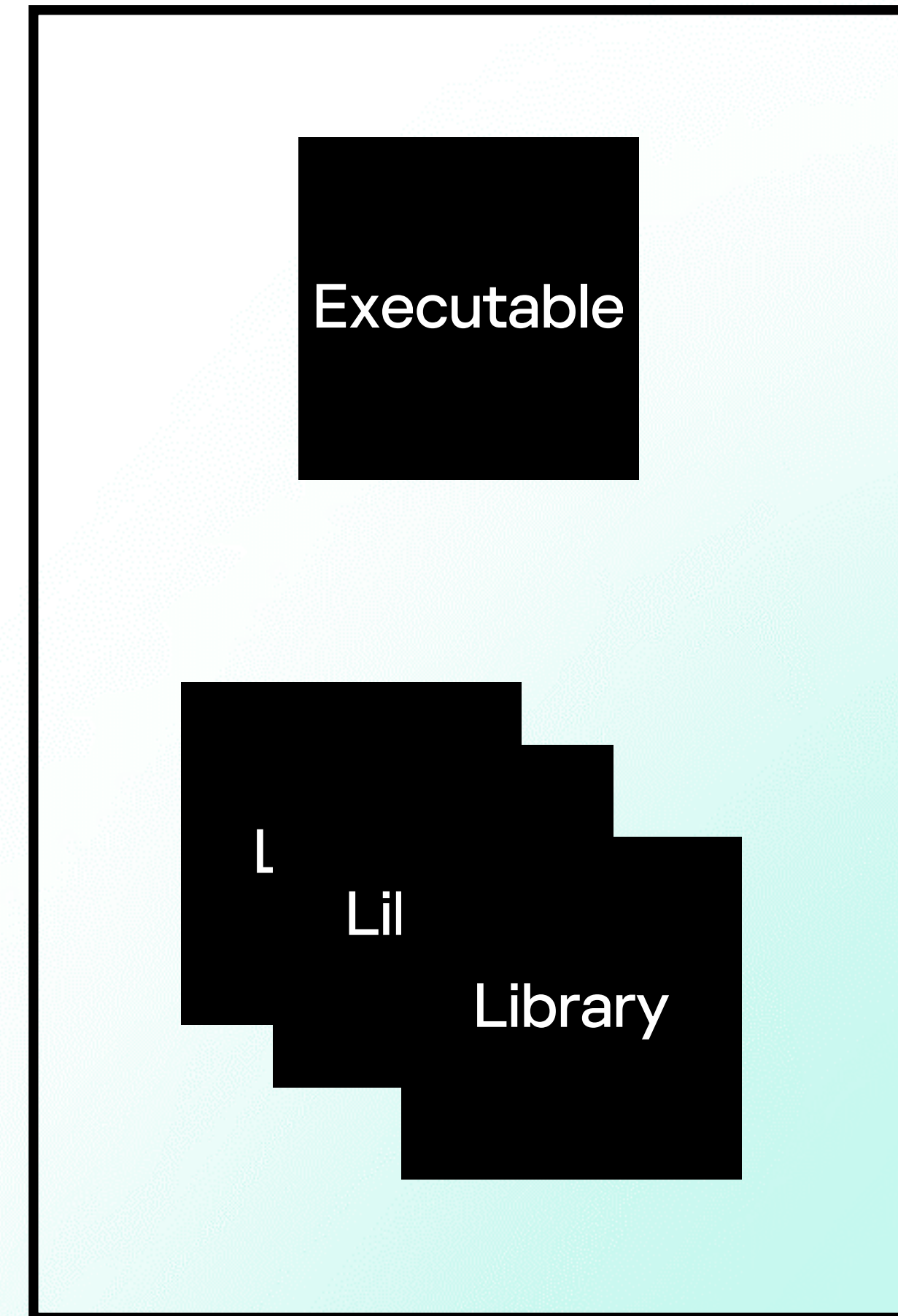
Обращения к системным фреймворкам

Если мы знаем, что приложение точно в нужном нам месте взаимодействует с системным фреймворком, можно посмотреть список импортированных символов.

Естественно, их реализацию мы не увидим: они лежат только на устройстве и могут быть в shared cache.

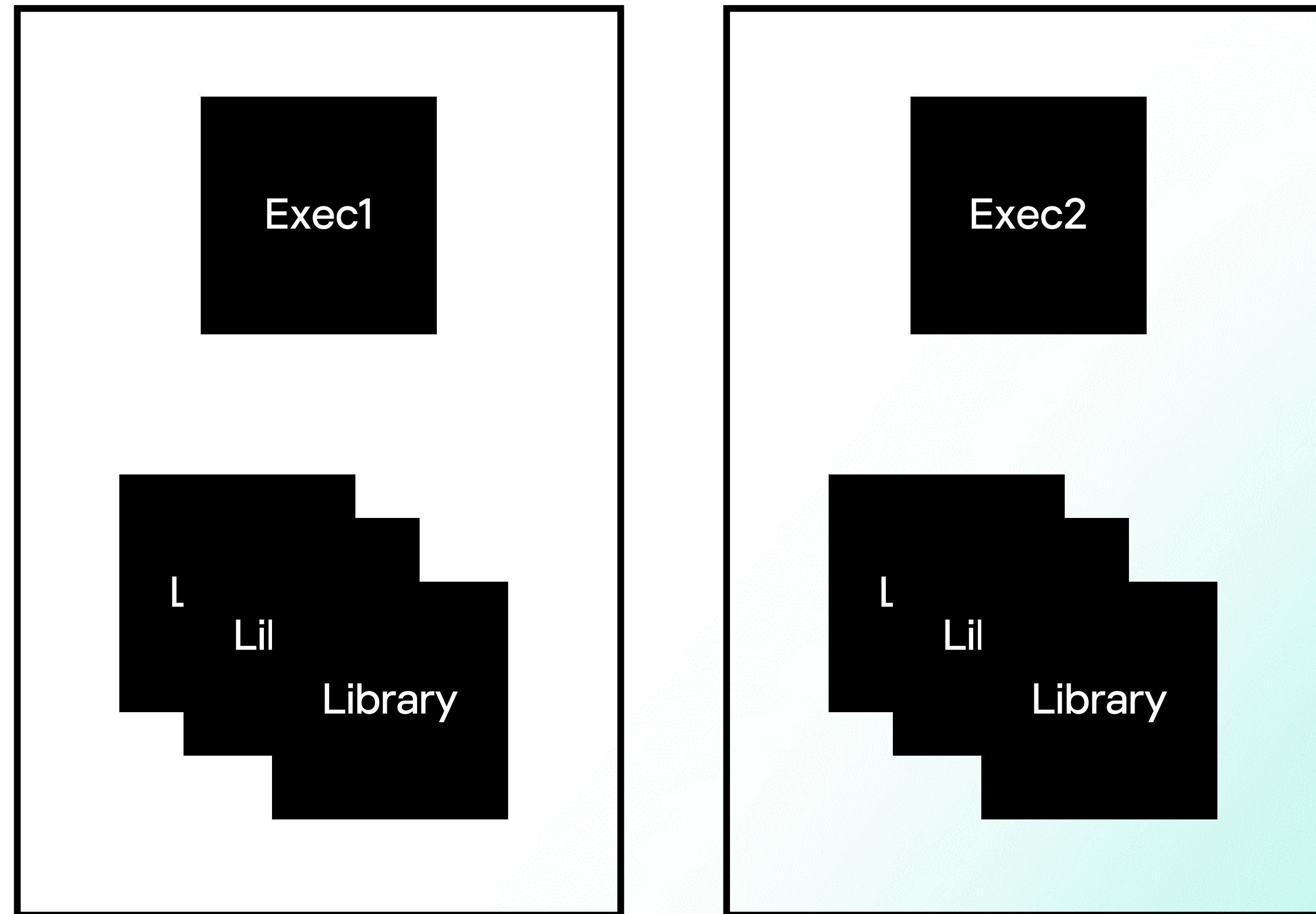
Анализ

shared cache



Анализ

shared cache



Анализ

shared cache

Exec2

Exec1

Shared Cache

L

Lil

Library

Анализ

Системные фреймворки

Imported Symbols				
Q Search				
Address	Ordinal	Symbol	From	
0x10005c000	-	_CGRectZero	/System/Library/Frameworks/CoreGraphics.framework/CoreGraphics	
0x10005c008	-	_NSStringFromClass	/System/Library/Frameworks/Foundation.framework/Foundation	
0x10005c010	-	_OBJC_CLASS_\$_NSArray	/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation	
0x10005c018	-	_OBJC_CLASS_\$_NSData	/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation	
0x10005c020	-	_OBJC_CLASS_\$_NSLayoutConstraint	/System/Library/Frameworks/Foundation.framework/Foundation	
0x10005c028	-	_OBJC_CLASS_\$_NSObject	/usr/lib/libobjc.A.dylib	
0x10005c030	-	_OBJC_CLASS_\$_UIBarButtonItem	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c038	-	_OBJC_CLASS_\$_UICollectionViewCell	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c040	-	_OBJC_CLASS_\$_UICollectionViewController	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c048	-	_OBJC_CLASS_\$_UICollectionViewFlowLayout	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c050	-	_OBJC_CLASS_\$_UIColor	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c058	-	_OBJC_CLASS_\$_UIImage	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c060	-	_OBJC_CLASS_\$_UIImageView	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c068	-	_OBJC_CLASS_\$_UIResponder	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c070	-	_OBJC_CLASS_\$_UISceneConfiguration	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c078	-	_OBJC_CLASS_\$_UIToolbar	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c080	-	_OBJC_CLASS_\$_UIWindow	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c088	-	_OBJC_METAClass_\$_NSObject	/usr/lib/libobjc.A.dylib	
0x10005c090	-	_OBJC_METAClass_\$_UICollectionViewCell	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c098	-	_OBJC_METAClass_\$_UICollectionViewController	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c0a0	-	_OBJC_METAClass_\$_UIResponder	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c0a8	-	_UIApplicationMain	/System/Library/Frameworks/UIKit.framework/UIKit	
0x10005c0b0	-	___CFCConstantStringClassReference	/System/Library/Frameworks/CoreFoundation.framework/CoreFoundation	
0x10005c0b8	-	___stack_chk_fail	/usr/lib/libSystem.B.dylib	
0x10005c0c0	-	___stack_chk_guard	/usr/lib/libSystem.B.dylib	
0x10005c0c8	-	___objc_empty_cache	/usr/lib/libobjc.A.dylib	
0x10005c0d0	-	_objc_alloc	/usr/lib/libobjc.A.dylib	
0x10005c0d8	-	_objc_alloc_init	/usr/lib/libobjc.A.dylib	

Анализ

Системные фреймворки

```
;  
_CGRectZero:  
0000000010005c000      extern function code  
_NSStringFromClass:  
0000000010005c008      extern function code  
_OBJC_CLASS_$_NSArray:  
0000000010005c010      extern function code  
_OBJC_CLASS_$_NSData:  
0000000010005c018      extern function code  
_OBJC_CLASS_$_NSLayoutConstraint:  
0000000010005c020      extern function code  
_OBJC_CLASS_$_NSObject:  
0000000010005c028      extern function code  
_OBJC_CLASS_$_UIBarButtonItem:  
0000000010005c030      extern function code  
_OBJC_CLASS_$_UICollectionViewCell:  
0000000010005c038      extern function code  
_OBJC_CLASS_$_UICollectionViewController:  
0000000010005c040      extern function code  
_OBJC_CLASS_$_UICollectionViewFlowLayout:  
0000000010005c048      extern function code  
_OBJC_CLASS_$_UICollectionView:
```


Системные фреймворки

Ищем нужный нам символ (в нашем случае — просто использование класса NSData).

Смотрим, какие есть XREF (перекрёстные ссылки) на символ.

Попадаем в код, который использует этот символ.

Анализ

UI-трейсинг



UI-трейсинг

У каждого бинарника (только для executable) есть `entrypoint` — место, с которого начинается выполнение программы.

Он прописан в структуре Mach-O как адрес первой инструкции.

Анализ

UI-трейсинг

```
int EntryPoint(int arg0, int arg1) {  
    r21 = objc_autoreleasePoolPush();  
    r22 = [NSStringFromClass(objc_opt_class(@class(AppDelegate))) retain];  
    objc_autoreleasePoolPop(r21);  
    r19 = UIApplicationMain(arg0, arg1, 0x0, r22);  
    [r22 release];  
    r0 = r19;  
    return r0;  
}
```


UI-трейсинг

Здесь видно и создание основного экземпляра `UIApplication`, и создание `AppDelegate`, который в этом `UIApplication` используется. `AppDelegate` в iOS-приложениях — это входная точка, поэтому заглянем в него.

Если поддерживаются `UIScene`, то смотрим в `Info.plist` имя класса по ключу «`UISceneDelegateClassName`».

Анализ

UI-трейсинг

```
<key>UISceneDelegateClassName</key>  
<string>SceneDelegate</string>
```


Анализ

UI-трейсинг

```
/* @class SceneDelegate */
-(void)scene:(void *)arg2 willConnectToSession:(void *)arg3 options:(void *)arg4 {
...
    r20 = [[ViewController alloc] initWithCollectionViewLayout:var_28];
...
    return;
}
```

Анализ

UI-трейсинг

```
/* @class ViewController */
-(void)viewDidLoad {
...
    [self reload];
    [r23 release];
    [r22 release];
    [r20 release];
    if (**__stack_chk_guard != var_60) {
        __stack_chk_fail();
    }
    return;
}
```


Анализ

UI-трейсинг

```
/* @class ViewController */
-(void)reload {
    r0 = [self imageStorage];
    ...
    r21 = [[r0 loadImages] retain];
    [self setImages:r21];
    ...
    r0 = [self collectionView];
    ...
    [r0 reloadData];
    ...
    return;
}
```

Анализ

UI-трейсинг

```
/* @class ImageStorage */
-(void *)loadImages {
...
    r0 = [self parseB64ToImage:@"..."];
    ...
    r26 = [[self parseB64ToImage:@"..."] retain];
    r27 = [[self parseB64ToImage:@"..."] retain];
    r28 = [[self parseB64ToImage:@"..."] retain];
    r19 = [[self parseB64ToImage:@"..."] retain];
...
    r0 = [r0 arrayWithObjects:&var_A8 count:0xa];
...
    return r0;
}
```


Анализ

UI-трейсинг

Таким образом, мы можем попасть в искомый код через UI

4

Переподпись

Зачем нужна ПОДПИСЬ

Подпись IPA (его частей) позволяет убедиться, что приложение, которое было залито в App Store, точно совпадает с тем, которое устанавливается на устройство.

Формально, если Apple подписывает приложение и оно попадает в App Store, то Apple может гарантировать, что приложение свободно от вредоносного ПО.

Благодаря подписи все стороны знают, кем именно было разработано приложение.

Установка неподписанных приложений не разрешена.

Как переподписать приложение

Фактически достаточно одной утилиты codesign

01

Получить provisioning profile для каждого бинарника.

02

Сгенерировать entitlements.

03

Пройтись по всем бинарникам: основному и рекурсивно по всем внутри директорий Plugins, Extensions, Watch, Frameworks.

04

Запустить для них codesign с указанием сертификата и entitlements.

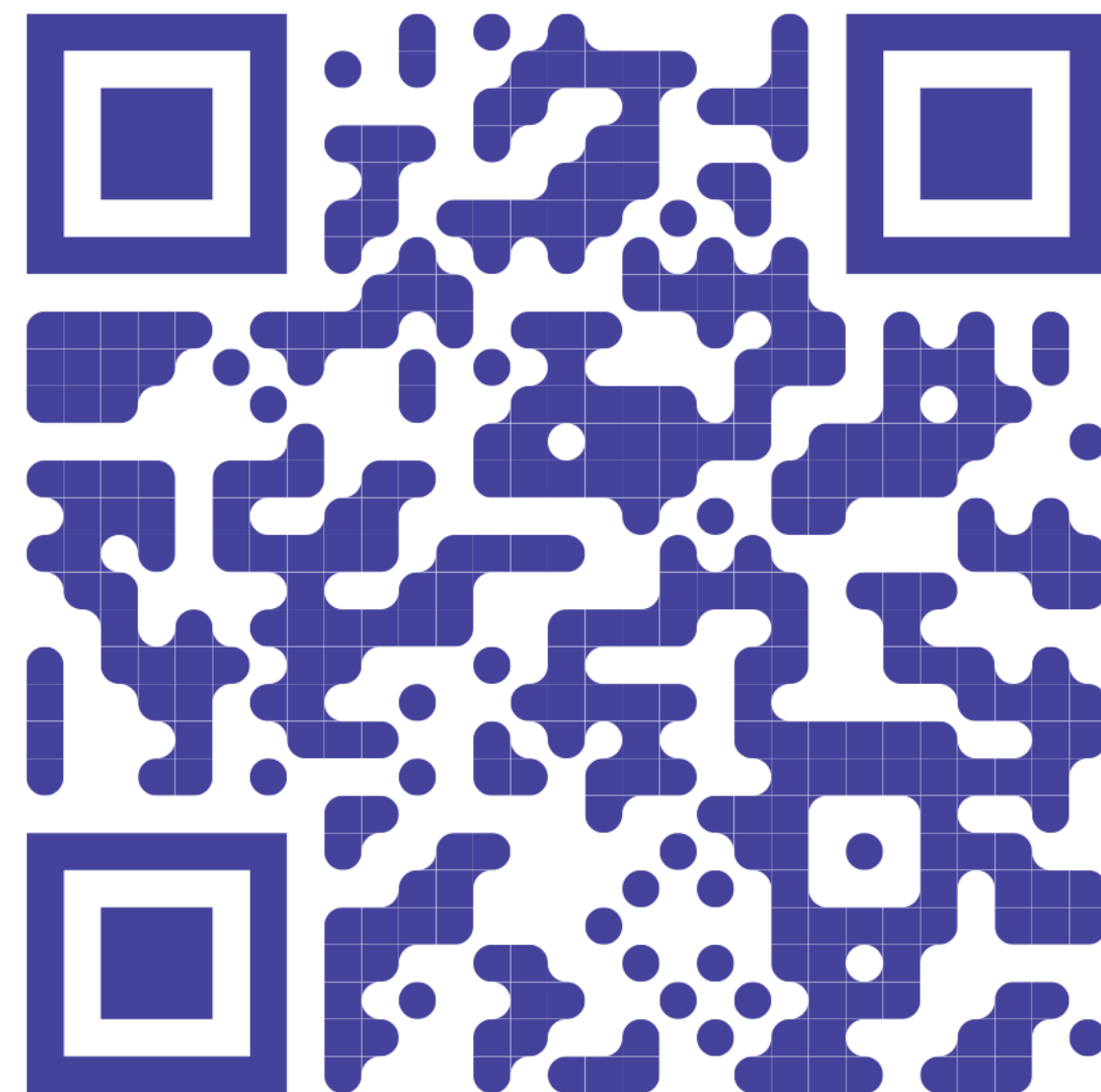
05

Также для executable-файла понадобится скопировать provisioning profile.

Переподпись

В интернете есть разные варианты подписи (например, «sign» от fastlane), но весь процесс можно реализовать и самостоятельно.

В качестве референса — мой пример на GitHub:



5

Вносим изменения в ПО

Что можем сделать?

Изменять текущее поведение

Добавлять новое поведение

Изменение в существующем коде

Попробуем изменить текущую функциональность приложения.

Для начала просто подменим одну из картинок на свою.

Изменение в существующем коде

Найдём любую строку, из которой генерируется изображение.

Изменение в существующем коде

Референс на строку

```
00000000100004124    mov     fp, fp
00000000100004128    bl      imp___stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
0000000010000412c    mov     x28, x0
00000000100004130    adrp    x2, #0x10004c000                                ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8@PAGE
00000000100004134    add     x2, x2, #0x280                                  ; cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8@PAGE0FF, @" /s
00000000100004138    mov     x0, x25                                         ; argument #1 for method _objc_msgSend$parseB64ToImage:
0000000010000413c    bl      _objc_msgSend$parseB64ToImage:                  ; _objc_msgSend$parseB64ToImage:
00000000100004140    mov     fp, fp
00000000100004144    bl      imp___stubs__objc_retainAutoreleasedReturnValue ; objc_retainAutoreleasedReturnValue
00000000100004148    mov     x19, x0
0000000010000414c    stp     x20, x21, [sp, #0x18]
```


Изменение в существующем коде

Строка в памяти

```
000000010004c260      dq      ____CFConstantStringClassReference, 0x7c8, a9j4aaqskzjrgab_10003824c, 0xa55c ; "/9j/4AAQSkZJRgABA
cfstring__9j_4AAQSkZJRgABAQAAQABAAD_2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8:
000000010004c280      dq      ____CFConstantStringClassReference, 0x7c8, a9j4aaqskzjrgab_1000427a9, 0x55b0 ; "/9j/4AAQSkZJRgABA
cfstring_Generating_image_from_string____:
000000010004c2a0      dq      ____CFConstantStringClassReference, 0x7c8, aGeneratingImag, 0x20 ; "Generating image from string:
cfstring_Default_Configuration:
```


Изменение в существующем коде

Строка в hex

42750	35	56	31	64	51	4A	50	44	63	55	78	4A	4F	6D	76	4B	5V1dQJPDcUxJ0mvK
42760	6C	58	62	39	78	48	41	6E	4D	47	4B	79	47	37	37	31	lXb9xHANMGKyG771
42770	31	64	56	4B	69	34	57	4C	58	77	57	4D	41	68	59	6E	1dVKi4WLXwWMAhYn
42780	6C	46	4A	53	2F	64	52	4C	53	69	34	78	42	33	6B	31	lFJS/dRLSi4xB3k1
42790	46	64	57	6D	56	2B	7A	69	6E	46	70	52	43	2F	53	70	FdWmV+zinFpRC/Sp
427a0	72	71	36	67	2F	39	6B	3D	00	2F	39	6A	2F	34	41	41	rq6g/9k=;/9j/4AA
427b0	51	53	6B	5A	4A	52	67	41	42	41	51	41	41	41	51	41	QSkZJRgABAQAAAQA
427c0	42	41	41	44	2F	32	77	43	45	41	41	6B	47	42	77	6B	BAAD/2wCEAAkGBwk
427d0	4A	43	41	67	4A	43	51	6B	49	43	41	67	49	43	42	59	JCAgJCQkICAgICBY
427e0	4A	43	41	67	49	43	42	73	49	46	51	6F	57	49	42	30	JCAgICBsIFQoWIB0
427f0	69	49	69	41	64	48	78	38	6B	4B	44	51	73	4A	43	59	iIiAdHx8kKDQsJCY
42800	78	4A	78	38	66	4C	54	45	74	4D	54	55	33	4F	6A	6F	xJx8fLTEtMTU30jo
42810	36	49	79	73	7A	4F	44	4D	34	4E	7A	51	35	4F	6A	63	6Iysz0DM4NzQ50jc
42820	42	43	67	6F	4B	44	67	30	4F	47	68	41	51	47	69	30	BCgoKDg00GhAQGi0
42830	64	46	78	30	72	4C	53	30	72	4C	53	30	74	4C	69	30	dFx0rLS0rLS0tLi0
42840	72	4C	53	73	74	4C	69	30	74	4C	53	30	74	4C	53	30	rLSstLi0tLS0tLS0
42850	74	4C	54	67	74	4F	43	30	34	4C	53	30	74	4C	53	30	tLTgt0C04LS0tLS0
42860	78	4C	53	30	72	4C	53	73	74	4C	53	73	74	4C	53	30	xLS0rLSstLSstLS0
42870	72	4C	53	30	74	4C	66	2F	41	41	42	45	49	41	57	67	rLS0tLf/AABEIAWg
42880	43	48	41	4D	42	49	67	41	43	45	51	45	44	45	51	48	CHAMBIgACEQEDEQH
42890	2F	78	41	41	63	41	41	45	41	41	67	4D	42	41	51	45	/xAAcAAEAAGMBAQE
428a0	41	41	41	41	41	41	41	41	41	41	41	41	41	42	41	55	AAAAAAAAAAAAABAU
428b0	43	41	77	59	48	41	51	6A	2F	78	41	41	2F	45	41	41	CAwYHAQj/xAA/EAA
428c0	43	41	51	4D	43	41	77	55	48	41	67	59	42	41	77	49	CAQMCAwUHAgyBAwI
428d0	48	41	41	41	42	41	67	41	44	42	42	45	53	49	51	55	HAAABAgADBBESIUQ
428e0	78	51	52	4D	69	55	57	46	78	42	6A	4B	42	6B	61	47	xQRMiUWFxBjKBkaG
428f0	78	38	42	54	52	49	30	4A	53	77	65	48	78	42	79	51	x8BTRI0JSweHxByQ
42900	7A	59	68	56	54	46	68	63	6C	4E	45	4E	79	77	76	2F	zYhVTFhc lNENyww
42910	45	41	42	67	42	41	51	45	42	41	51	45	41	41	41	41	EABgBAQEBAQEAAAA

Изменение в существующем коде

Изменяем строку

Теперь напишем простой скрипт, который будет патчить наш бинарник — заменять строку на новую, добавляя \0 в конце.

```
import sys

address = 0x1000427a9

new_string = "... "

string_bytes = new_string.encode('utf-8')

padded_bytes = string_bytes + b'\x00'

doc = Document.getCurrentDocument()
for i, byte in enumerate(padded_bytes):
    doc.writeByte(address + i, byte)
print(f"Successfully patched {len(padded_bytes)} bytes at address {hex(address)}!")
```


Изменение в существующем коде

Дифф от изменённой строки

```
42750 35 56 31 64 51 4A 50 44 63 55 78 4A 4F 6D 76 4B 5V1dQJPDcUxJ0mvK
42760 6C 58 62 39 78 48 41 6E 4D 47 4B 79 47 37 37 31 lXb9xHAnMGKyG771
42770 31 64 56 4B 69 34 57 4C 58 77 57 4D 41 68 59 6E 1dVKi4WLXwMAhYn
42780 6C 46 4A 53 2F 64 52 4C 53 69 34 78 42 33 6B 31 lFJS/dRLSi4xB3k1
42790 46 64 57 6D 56 2B 7A 69 6E 46 70 52 43 2F 53 70 FdWmV+zinFpRC/Sp
427a0 72 71 36 67 2F 39 6B 3D 00 69 56 42 4F 52 77 30 rq6g/9k=;1VBORw0
427b0 4B 47 67 6F 41 41 41 41 4E 53 55 68 45 55 67 41 KGgoAAAANSUhEUgA
427c0 41 41 51 41 41 41 41 45 41 43 41 49 41 41 41 44 AAQAAAAEACAIAAAD
427d0 54 45 44 38 78 41 41 41 44 4D 45 6C 45 51 56 52 TED8xAAADMElEQVR
427e0 34 6E 4F 7A 56 77 51 6E 41 49 42 51 46 51 59 58 4n0zVwQnAIBQFQYX
427f0 66 66 38 31 52 55 6B 51 43 4F 79 44 6A 31 59 4F ff81RUkQC0yDj1Y0
42800 50 6E 62 58 57 50 6D 65 54 52 65 66 2B 2F 33 4F PnbXWPmeTRef+/30
42810 2F 4F 79 42 6A 7A 68 33 43 44 39 35 42 66 71 49 /0yBjzh3CD95BfqI
42820 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42830 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42840 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42850 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42860 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42870 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42880 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42890 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428a0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428b0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428c0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428d0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428e0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
428f0 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42900 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42910 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42920 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
42930 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 43 4D 4B 30 CMK0CMK0CMK0CMK0
```


Тестируем изменения

01

Строку заменили, бинарник сохранили.

02

Переподписали, собрали новый ipa, или залили .app как есть.

03

Проверяем.

04

Видим, что приложение падает.

05

Хотелось бы увидеть почему.

Отладка

На помощь нам приходит отладчик, но так просто подключиться к любому приложению нельзя (например, к скачанному из App Store).

Этому препятствует ключ `get-task-allow`.

get-task-allow

Ключ в entitlements, который позволяет другим приложениям — в том числе дебаггеру Xcode (debugserver) — получать доступ ко всей памяти процесса.

При выполнении кем-либо функции `task_for_pid(parent, pid, *task_out)` ОС проверяет, есть ли в entitlement необходимый ключ со значением `true`.

Если есть, то в `task_out` помещается `mach_port` для управления процессом.

Если нет, то результат функции не равен нулю, а `task_out` останется пустым.

get-task-allow

```
<?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>application-identifier</key>
```

```
  <string>2Y8XE5CQ94.com.kaspersky.ios.Mobius.Expensivelmages</string>
```

```
  <key>get-task-allow</key>
```

```
  <true/>
```

```
</dict>
```

```
</plist>
```


Теперь к
процессу можно
подключаться
отладчиком для:

Просмотра консольного
вывода.

Просмотра стектрейса и
состояния регистров.

RW-доступа к памяти.

В общем, всего, что
можно делать при
обычной отладке вашего
приложения.

С ПОМОЩЬЮ ОТЛАДЧИКА ПОСМОТРИМ, ЧТО НЕ ТАК В НАШЕМ КОДЕ

Изменение в существующем коде

1. Заходим в Xcode → Debug → Attach to Process.

2. Пишем имя основного процесса.

3. Запускаем приложение на iPhone.

Аварийная остановка приложения

[illegible]

```
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '*** -[__NSPlaceholderArray initWithObjects:count:]: attempt to insert nil object from objects[9]'
```

```
(0x186e34244 0x183901224 0x186d20f5c 0x186d20c54 0x102158178 0x102158d30 0x102158cbc 0x18c97fc60 0x18c96638c 0x18c965fa4 0x18c9f4194 0x18caa690c 0x102158468
0x18c99f450 0x18caa7c68 0x18ca9d6f0 0x18caa8820 0x1a7fb0438 0x1a7fab8ac 0x1a7fb01dc 0x1a801a2c4 0x1a7faf520 0x1a7faf898 0x1a7faf694 0x1a7faf520 0x1c18771e4
0x1c1862694 0x19f307f84 0x19f307e00 0x186dbf360 0x186dbf2d4 0x186d845a8 0x186d4e1a0 0x186d4d54c 0x22cd35498 0x18ca81670 0x18c9ec158 0x10215834c 0x183959c1c)
```


Результаты отладки

Видим логи приложения (уже неплохо).

Также понятно, что мы пытаемся вставить nil в NSArray (а этого он совсем не любит).

Что-то недосмотрели — смотрим обратно в наш дизассемблированный код.

Анализируем падение

```
r21 = [[self parseB64ToImage:@"..."] retain];
r22 = [[self parseB64ToImage:@"..."] retain];
var_B8 = [[self parseB64ToImage:@"..."] retain];
r23 = [[self parseB64ToImage:@"..."] retain];
r24 = [[self parseB64ToImage:@"..."] retain];
r26 = [[self parseB64ToImage:@"..."] retain];
r27 = [[self parseB64ToImage:@"..."] retain];
r28 = [[self parseB64ToImage:@"..."] retain];
r19 = [[self parseB64ToImage:@"iVB0Rw0KGgoAAAANSUgAAQAAAEACIAAA..." ] retain];
var_A8 = r20;
r25 = var_B8;
r0 = *@class(NSArray_10004c100);
r0 = [r0 arrayWithObjects:&var_A8 count:0xa];
```

Анализируем падение

Строка выглядит так же, как и все остальные.

Почему-то из неё
получается nil вместо NSData.

Вернёмся на пару шагов назад.

Анализируем падение

```
000000010004c260      dq      ____CFConstantStringClassReference, 0x7c8, a9j4aaqskzjrgab_10003824c, 0xa55c ; "/9j/4AAQSkZJRgABAQAAQABAAAD_2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8:
000000010004c280      dq      ____CFConstantStringClassReference, 0x7c8, a9j4aaqskzjrgab_1000427a9, 0x55b0 ; "/9j/4AAQSkZJRgABAQAAQABAAQAAAD_2wCEAAkGBwkJCAgJCQkICAgICBYJCAgICBsIFQoWIB0iIiAdHx8kKDQsJCYxJx8:
000000010004c2a0      dq      ____CFConstantStringClassReference, 0x7c8, aGeneratingImag, 0x20 ; "Generating image from string:
cfstring_Default_Configuration:"
```

Подозрительное место

Обращаем внимание на байты 0x55b0.

`__CFConstantStringClassReference` — тип в Core Foundation, который представляет константную строку.

Помимо прочей информации (флагов кодировки и сырого указателя на данные), есть ещё один важный параметр — длина строки.

0x55b0 — это 21936 (именно столько символов ранее занимала строка).

А теперь СЛОЖНОСТЬ

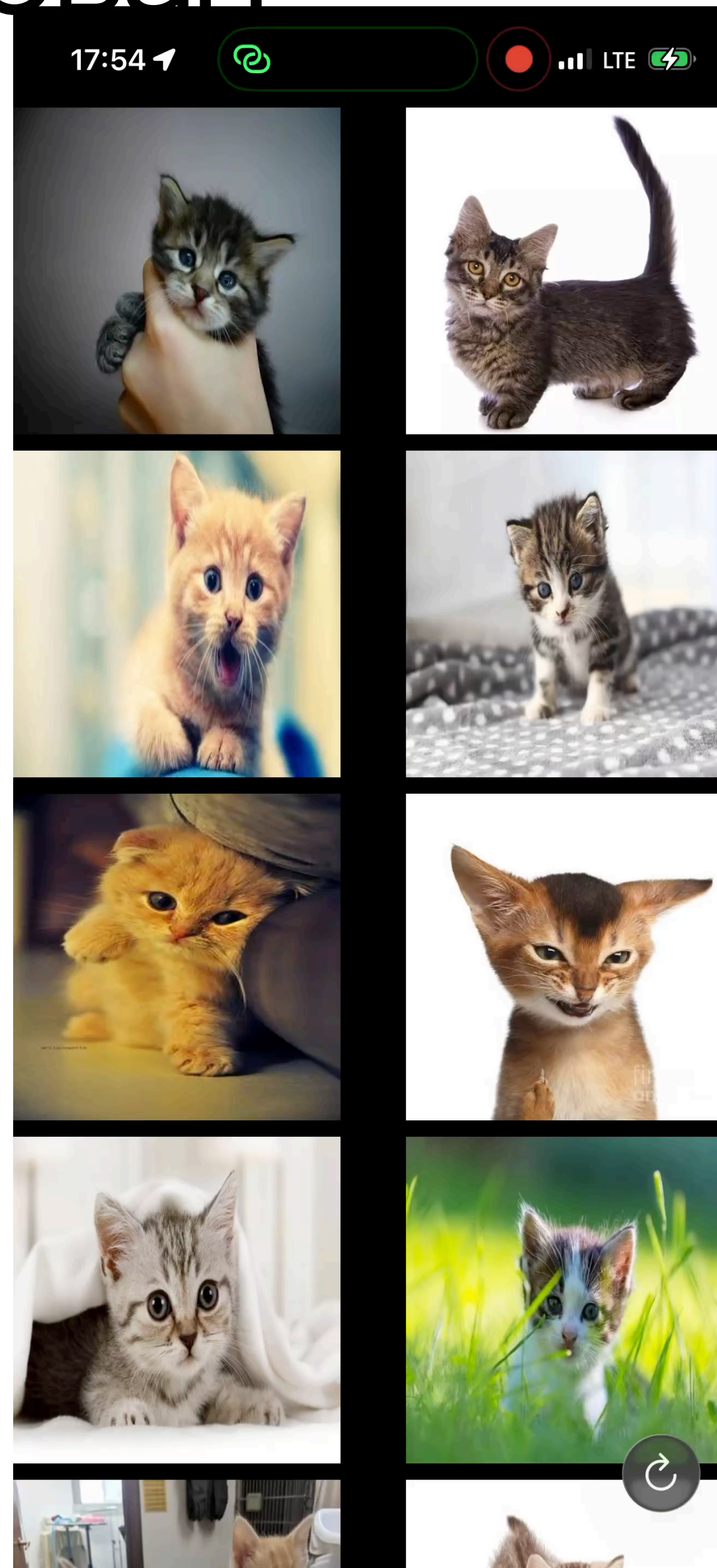
При таком подходе новая строка не может быть длиннее старой (без дополнительных трюков), так как структура памяти в бинарнике очень плотная.

Сразу за этой строкой могут быть байты другой строки, поэтому менять можно только на строку длиной не больше исходной.

Чиним падение

1. Патчим длину строки.
2. Запускаем приложение снова.

Шпион интегрирован



Изменение в существующем коде

Плюсы:

- простота;
- скорость;
- подходит для незначительных правок.

Минусы:

- имеет ряд ограничений;
- не подходит для крупных изменений.

Можно ли так изменить логику, а не константы?

Можно, но ограничения те же:

Можно писать код короче того, что есть в функции, — остальное место заполняем пор-ами.

Можно ссылаться (например, через jmp) на другие места в бинарнике.

Нельзя писать код длиннее, чем уже есть: это может сломать бинарник или привести к UB во время работы.

Мы поняли, что
изменения в
бинарнике
вносить можно,
но это не всегда
удобно.

Что бы хотелось:

Не ограничивать себя в размере кода.

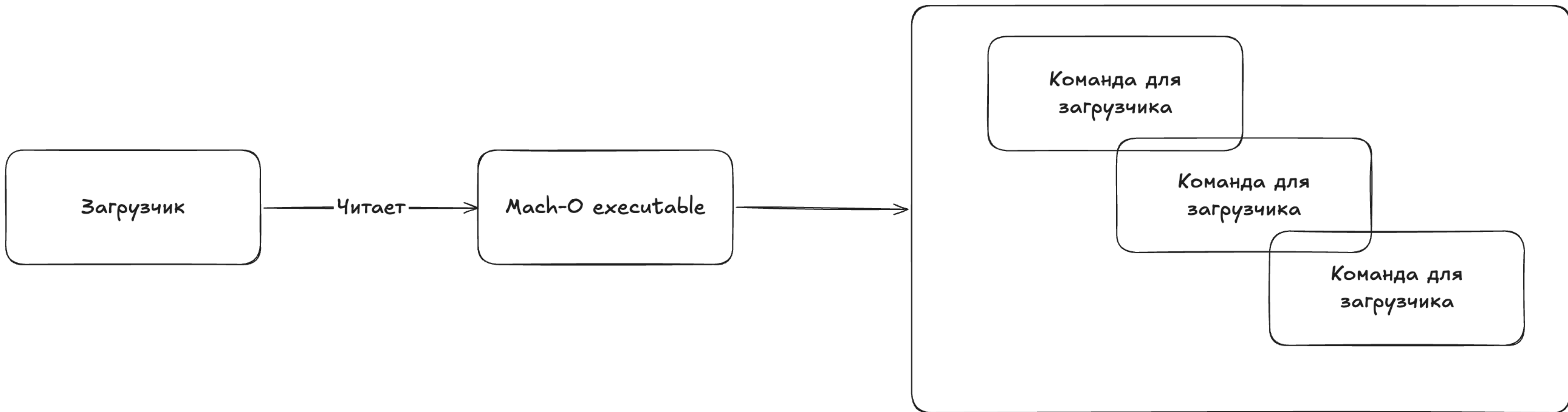
Иметь возможность влиять на рантайм.

Разрабатывать и отлаживать
привычными нам способами (через IDE).

Расширяем возможности

На помощь нам придёт динамическая линковка библиотек.

Расширяем возможности



Какие бывают команды?

LC_SEGMENT

Указывает на то, что какой-то участок бинарника нужно поместить в память, а также с какими правами и по какому относительному адресу.

LC_MAIN

Основная команда для executable-бинарников; в ней указано смещение до entrypoint приложения (мы видели его в Hopper).

LC_LOAD_DYLIB

Команда для загрузки динамической библиотеки по определённому пути.

LC_LOAD_DYLIB

Позволяет инъектировать произвольный код.

Длина этого кода не ограничена.

Доступ к памяти и функциям ОС будет точно такой же, как и у бинарника-хоста.

Позволяет использовать любые привычные runtime-трюки.

LC_LOAD_DYLIB

Инжектировать свою библиотеку в бинарник можно любым способом:

- вручную (понимая, что для этого могут потребоваться значительные правки в структуре Mach-O);
- автоматически, например, через утилиту `insert_dylib`.

Например: `insert_dylib <new_framework_binary_path> <executable_path>`.

Создаём свою dylib

Подойдёт обычная динамическая библиотека, собранная через Xcode.

Для разработки можно будет выбрать любой язык (Swift / ObjC).

Для инициализации понадобится ObjC.

Создаём свою dylib

```
+ (void)initialize {  
    NSLog(@"injected_dylib successfully injected");  
}
```

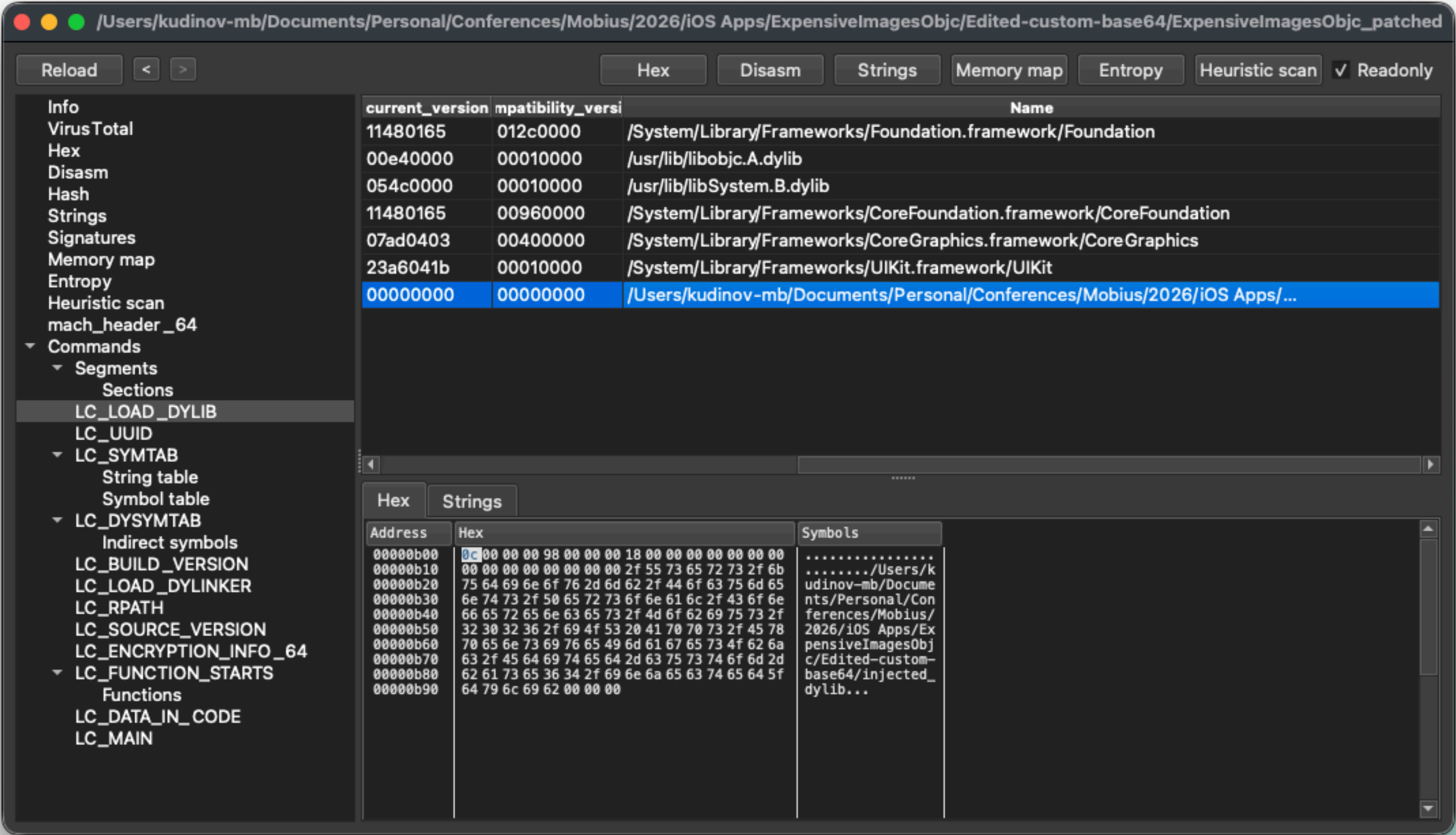

Патчим бинарник

```
insert_dylib ./injected_dylib ./ExpensiveImages0bjc
```


Патчим бинарник

```
ExpensiveImagesObjc*  
ExpensiveImagesObjc_patched*
```


Патчим бинарник



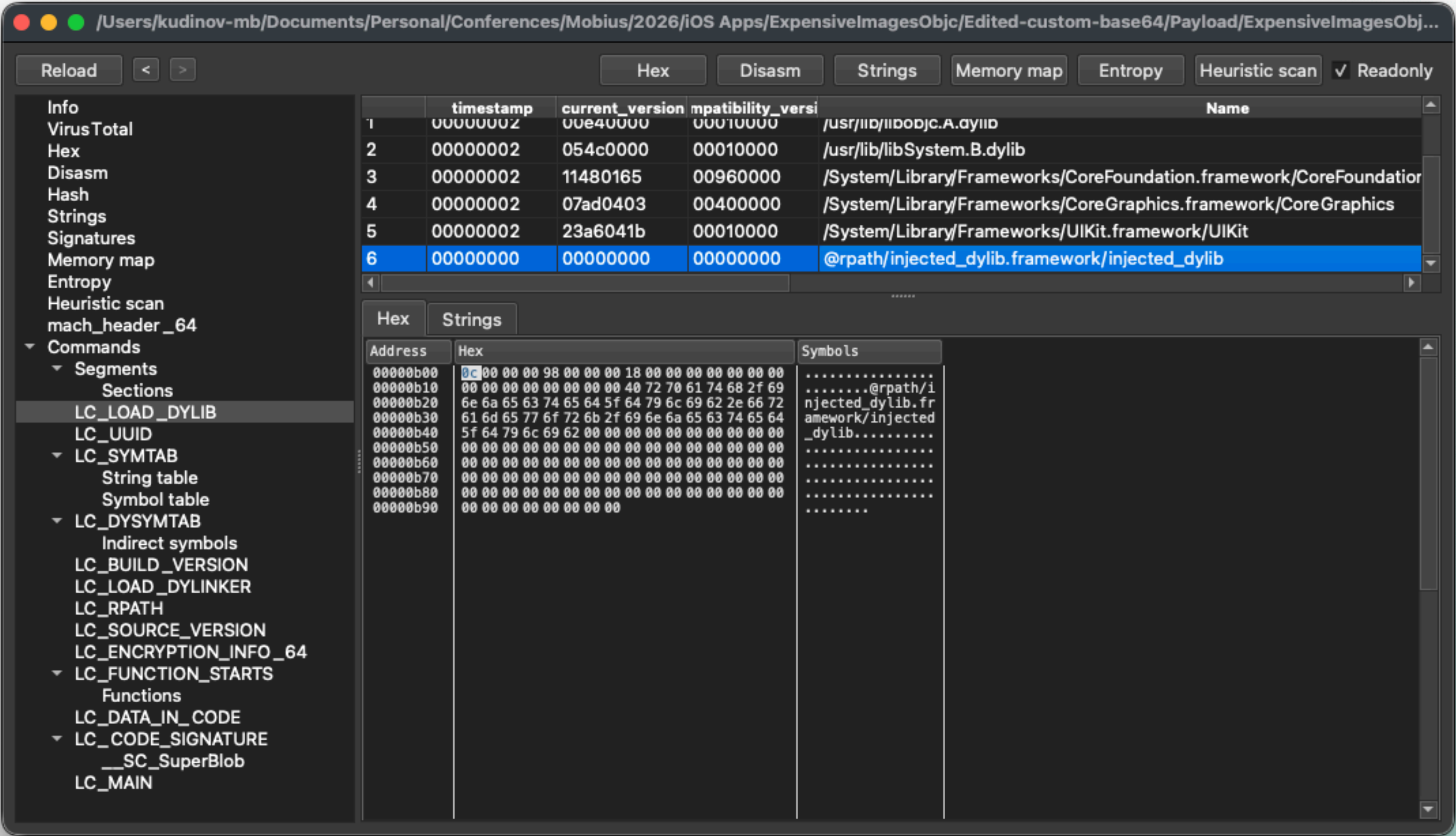
Патчим бинарник

Увидели, что наша dylib уже прописана в Mach-O, но у неё всё ещё довольно странный путь.

По такому пути загрузчик dyld не сможет найти нашу библиотеку.

Поэтому следует изменить её путь на @rpath.

Патчим бинарник



Патчим бинарник

Путь должен быть строго в формате `@rpath/framework_name.framework/binary_name`.

Также обязательно подложить любой Info.plist от подходящего фреймворка. Заполнить в нём минимальную версию ОС, Bundle ID и имя бинарника.

Патчим бинарник

Если всё сделали правильно, то увидим вывод нашего NSLog в консоли.

injected_dylib successfully injected

Generating image from string:

[illegible]

Generating image from string:

Патчим бинарник

I'm in.



Кадр из ф. «Матрица», реж. Братя Вачовски , 1999 г.

Теперь перед нами открыты все дороги — мы можем:

01

Пользоваться рантаймом
Objective-C.

02

Напрямую влиять на память.

03

Использовать любые
системные фреймворки.

04

Пользоваться теми же
разрешениями, что и основное
приложение (например, такими
чувствительными, как
микрофон, камера, геопозиция
и т. д.).

05

Отсылать любые данные,
используя URLSession.

Решаем поставленную задачу

Можно пойти несколькими путями:

- найти нужный UI-компонент (кнопку) и изменить её обработчик;
- изменить функцию перезагрузки данных так, чтобы она возвращала нужные элементы в списке.

Решаем поставленную задачу

В ходе анализа в Hopper мы уже видели, что у нас есть некий ImageStorage.

Попробуем собрать методы, которые у него есть.

Напишем нехитрый код для анализа класса.

Анализ класса

```
var numClasses: Int32 = 0
...

numClasses = objc_getClassList(nil, 0)
if numClasses > 0 {
    ...
    for i in 0..
```

Анализ класса

```
[injected_dylib] [ ObjcClassEnumerator.swift enumerateAllClasses() : 26 ] Class Name: ImageStorage
[injected_dylib] [ ObjcClassEnumerator.swift enumerateIvars(for:) : 43 ]      No ivars found
[injected_dylib] [ ObjcClassEnumerator.swift enumerateProperties(for:) : 63 ] Properties:
[injected_dylib] [ ObjcClassEnumerator.swift enumerateProperties(for:) : 81 ]      (No properties found)
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 87 ] Methods:
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 101 ]      -(@)loadImages, encoding: @16@0:8
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 108 ]      ↳ arg[0]: @
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 108 ]      ↳ arg[1]: :
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 101 ]      -(@)parseB64ToImage:, encoding: @24@0:8@16
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 108 ]      ↳ arg[0]: @
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 108 ]      ↳ arg[1]: :
[injected_dylib] [ ObjcClassEnumerator.swift enumerateMethods(for:) : 108 ]      ↳ arg[2]: @
[injected_dylib] [ ObjcClassEnumerator.swift enumerateClassMethods(for:) : 119 ] Class Methods:
[injected_dylib] [ ObjcClassEnumerator.swift enumerateClassMethods(for:) : 151 ]      (No class methods found)
```


Решаем поставленную задачу

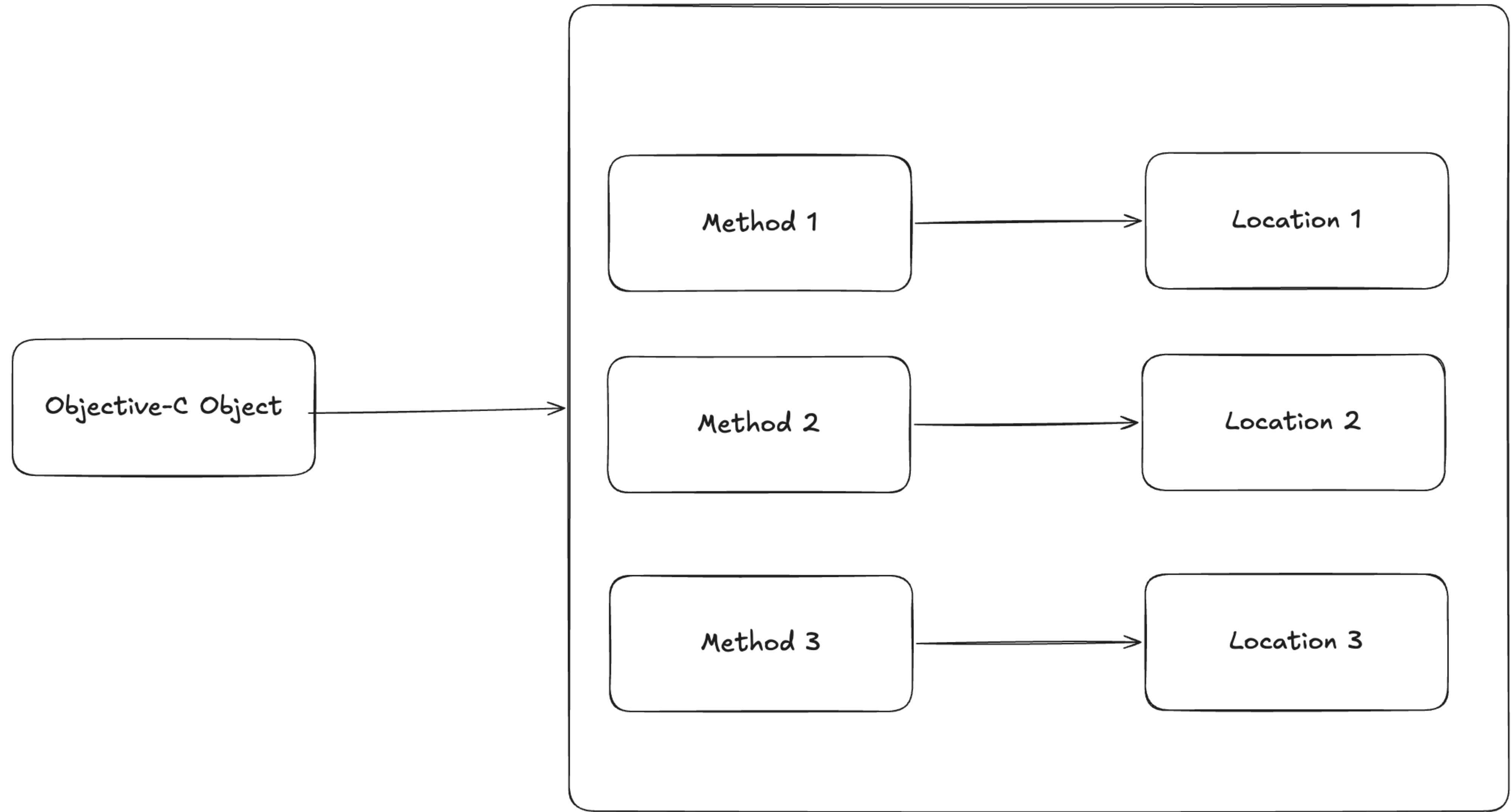
Видим, что у `ImageStorage` есть instance метод `loadImages`.

Метод не принимает аргументов и возвращает указатель (`id`).

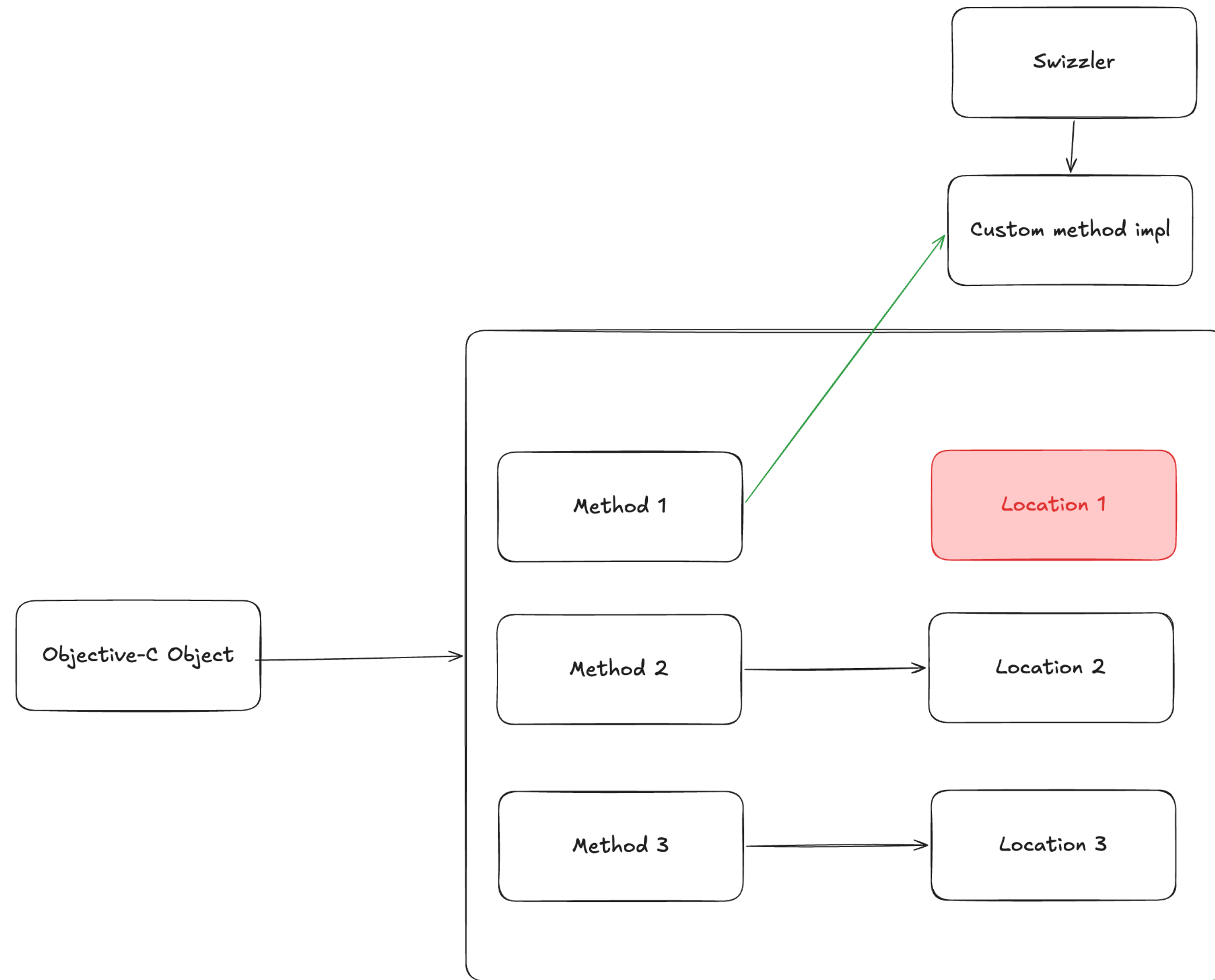
Можем применить немного рантайма ObjC (swizzling).

Спидран Swizzling-а за 20 секунд

Swizzling



Swizzling



Спидран Swizzling-a за 20 секунд

Можно ли свизлить Swift-методы?

Нет, в Swift применимы другие, более сложные механизмы.

Но из Swift свизлить Objective-C можно без проблем.

Спидран Swizzling-а за 20 секунд

```
func swizzleInstanceMethod<T>(for cls: AnyClass, selector: String,
                               newImplementationBuilder block: (Selector, IMP) -> T) {
    let originalSelector = NSSelectorFromString(selector)
    ...
    let originalIMP = method_getImplementation(originalMethod)

    let newIMP = imp_implementationWithBlock(block(originalSelector, originalIMP))
    let newSelector = NSSelectorFromString("injected_dylib_swizzled_\(selector)")

    guard class_addMethod(cls, newSelector, newIMP, method_getTypeEncoding(originalMethod)) else {
        err("")
        return
    }

    guard let replacedMethod = class_getInstanceMethod(cls, newSelector) else {
        err("")
        return
    }

    method_exchangeImplementations(originalMethod, replacedMethod)
}
```


Спидран Swizzling-а за 20 секунд

```
swizzler.swizzleInstanceMethod(for: NSClassFromString("ImageStorage")!,
                                selector: "loadImages") {
    selector, oldImpl in
        ...

        let newImplementation: Tb = { _self in
            let original = unsafeBitCast(oldImpl, to: Tc.self)
            let images = original(_self, selector)
            return images
        }

        return newImplementation
}
```

Решаем поставленную задачу

Что лежит в images?

```
✓ L images = 10 elements: __NSArrayI
  > [0] = 0x0000000010c402940: UIImage?
  > [1] = 0x0000000010c402a80: UIImage?
  > [2] = 0x0000000010c402b20: UIImage?
  > [3] = 0x0000000010c402bc0: UIImage?
  > [4] = 0x0000000010c4029e0: UIImage?
  > [5] = 0x0000000010c402da0: UIImage?
  > [6] = 0x0000000010c402c60: UIImage?
  > [7] = 0x0000000010c402d00: UIImage?
  > [8] = 0x0000000010c402e40: UIImage?
  > [9] = 0x0000000010c402ee0: UIImage?
```

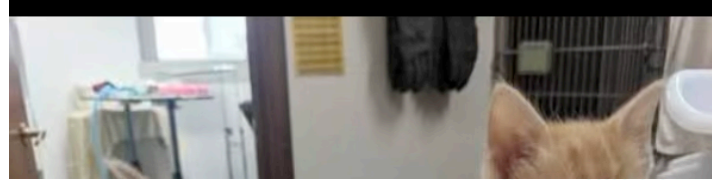

Решаем поставленную задачу

```
let newImplementation: Tb = { _self in
    ...
    let spyImageData = Data(base64Encoded: SpyB64String.b64String),
    let spyImage = UIImage(data: spyImageData) else {
        return images
    }
    newArray.append(spyImage)
    return (newArray as NSArray)
}
```


17:54



LTE



Решаем поставленную задачу

Формально задача решена:

- наш агент внедрён;
- исходная функциональность приложения сохранена и немного изменена.

Но фактически простор для экспериментов не ограничен.

6

Запуск и тестирование

Проверки со стороны ОС

Проверяется:

Без этих пунктов система
даже не установит ваш
бинарник.

Подпись всех бинарников, в том числе
наличие нужных capabilities.

Корректность минимальной версии ОС.

Платформа.

Может ли что-то сломаться в рантайме, если приложение уже запустилось?

Как и при обычной разработке, да:

- неправильная линковка;
- проблемы с capabilities;
- ошибки при свизлинге;
- ошибки при редактировании памяти.

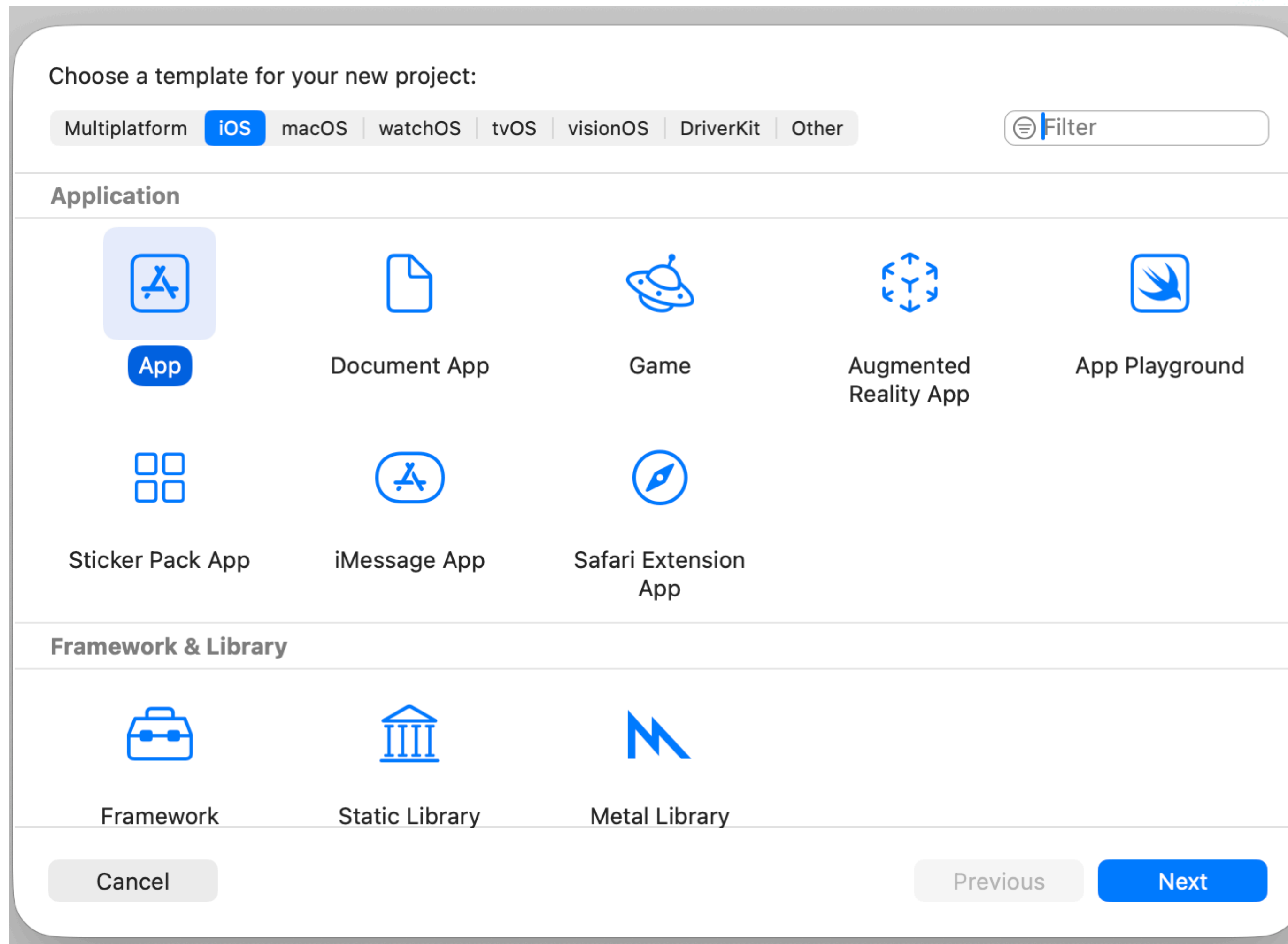
Например, при изменении указателей или данных, на которые они указывают, в Swift.

Отладка

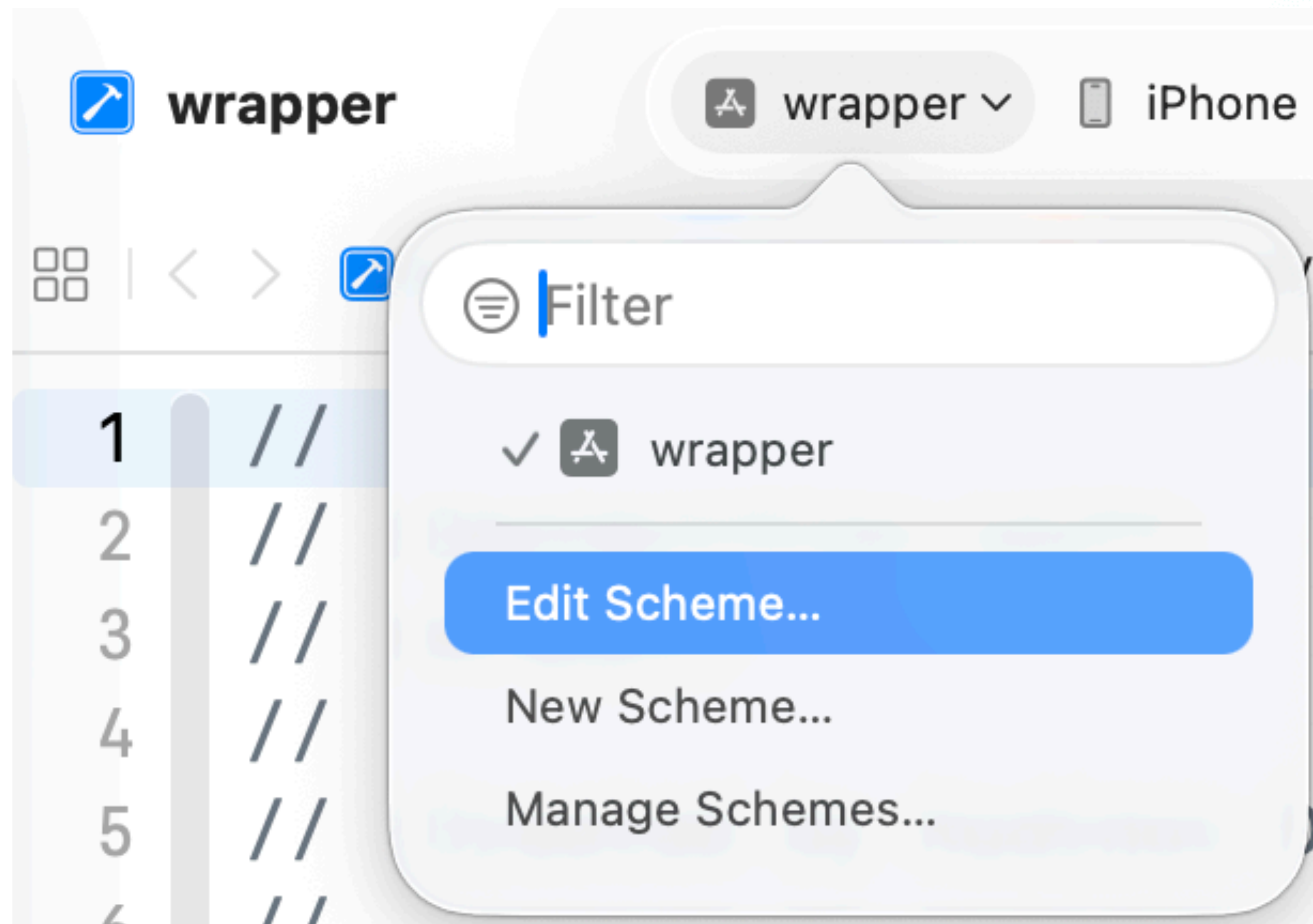
Мы уже поняли, что присоединиться отладчиком можно, но каждый раз нужно копировать бинарник библиотеки в .app / .ipa и переподписывать приложение вручную.

Можно ли это автоматизировать?

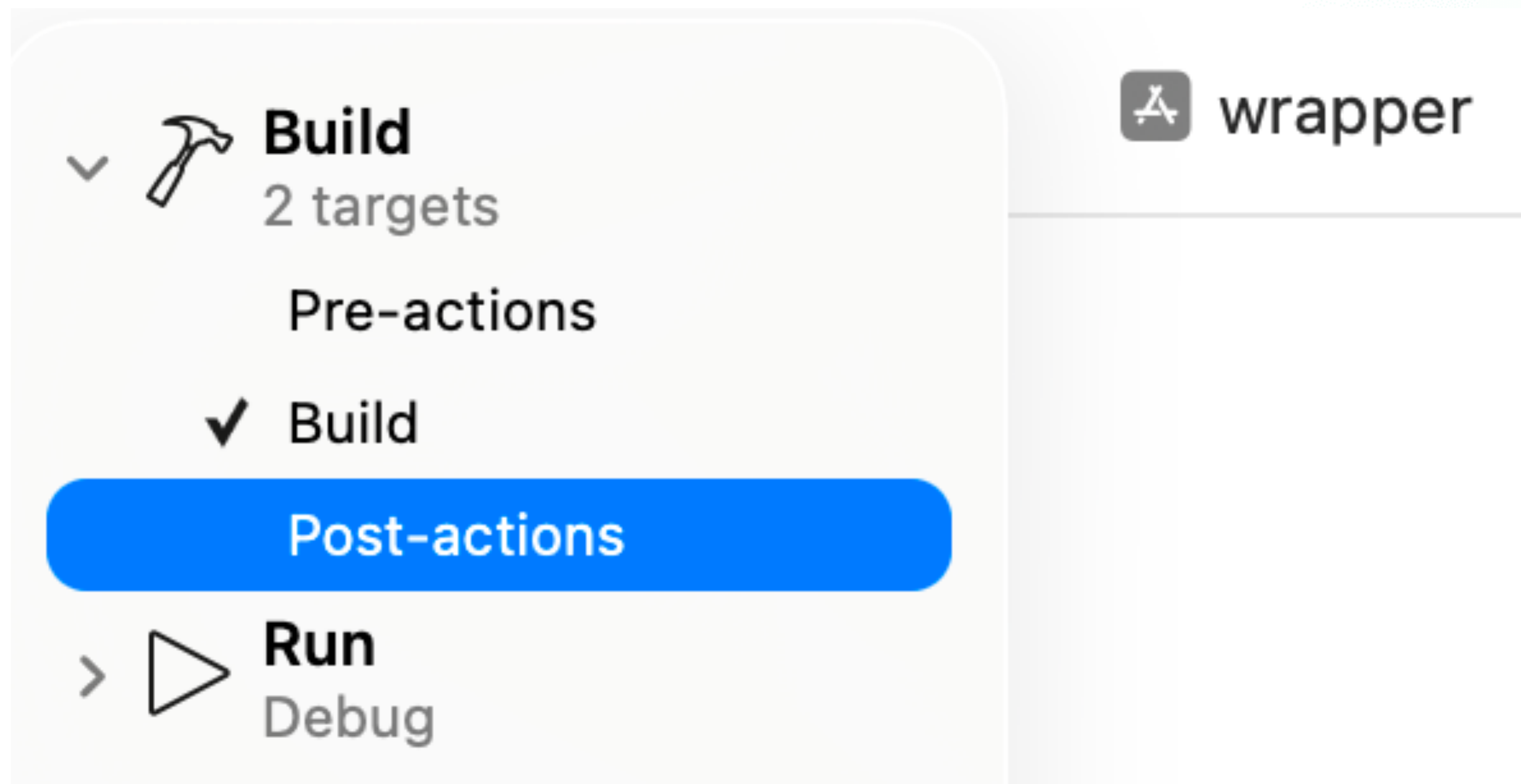
Автоматическая отладка



Автоматическая отладка



Автоматическая отладка



Автоматическая отладка

▼ Run Script

Shell

Provide build settings from

```
1 # Type a script or drag a script file from your workspace to insert its path.
```

```
2
```

Автоматическая отладка

Три простых шага

1 шаг

Удаляем старую dylib.

2 шаг

Копируем новую dylib.

3 шаг

Переподписываем.

Автоматическая отладка

```
TARGET_APP_PATH="/path/to/ExpensiveImagesObjc.app"
```

```
rm -f "$TARGET_APP_PATH/Frameworks/injected_dylib.framework/injected_dylib"
```


Автоматическая отладка

```
TARGET_APP_PATH="/path/to/ExpensiveImagesObjc.app"
```

```
rm -f "$TARGET_APP_PATH/Frameworks/injected_dylib.framework/injected_dylib"
```

```
cp "$BUILT_PRODUCTS_DIR/injected_dylib.framework/injected_dylib" \  
"$TARGET_APP_PATH/Frameworks/injected_dylib.framework/"
```


Автоматическая отладка

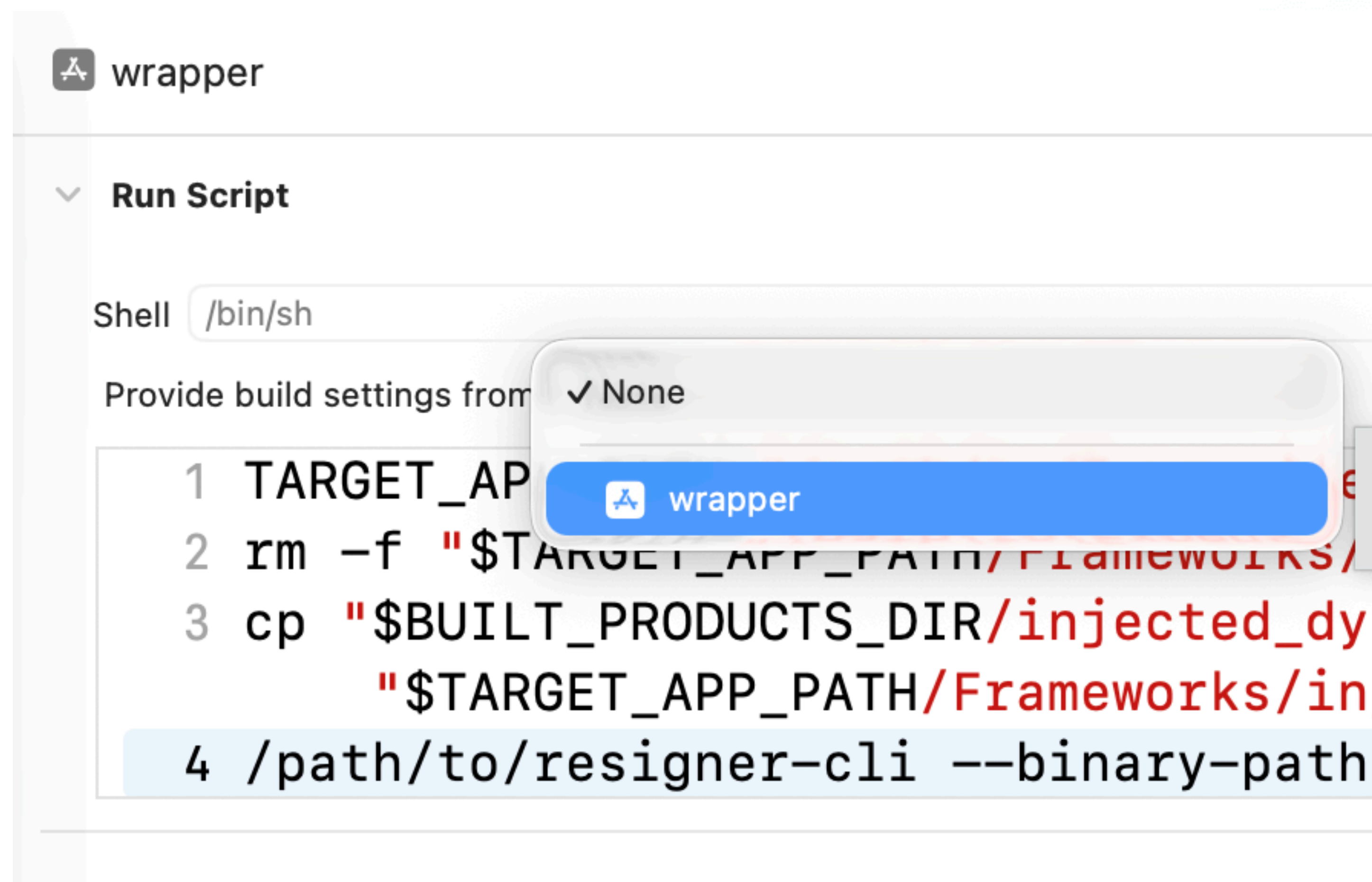
```
TARGET_APP_PATH="/path/to/ExpensiveImagesObjc.app"

rm -f "$TARGET_APP_PATH/Frameworks/injected_dylib.framework/injected_dylib"

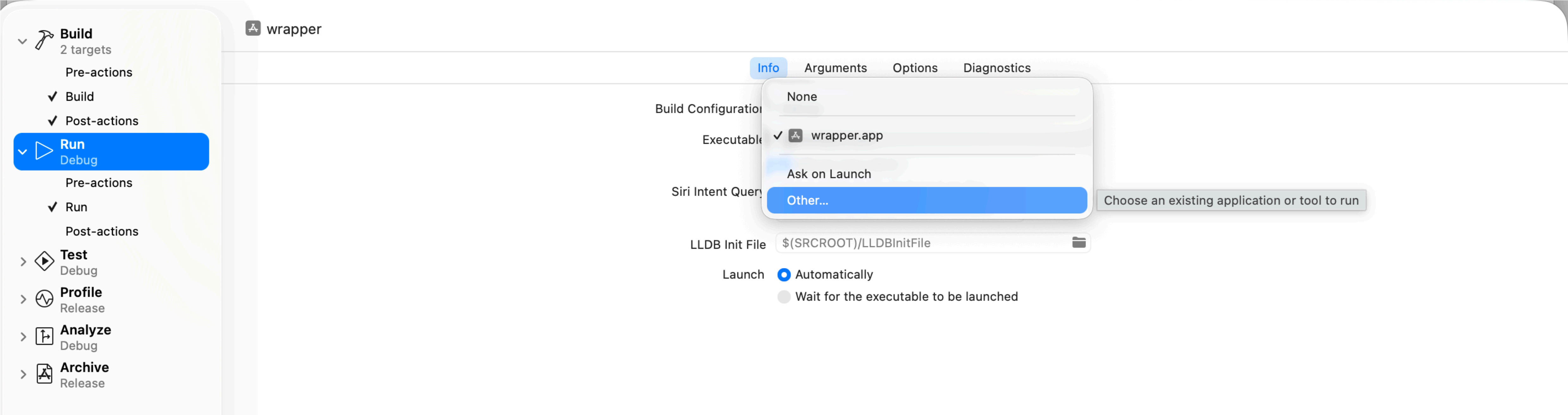
cp "$BUILT_PRODUCTS_DIR/injected_dylib.framework/injected_dylib" \
"$TARGET_APP_PATH/Frameworks/injected_dylib.framework/"

/path/to/resigner-cli --binary-path "$TARGET_APP_PATH"
```


Автоматическая отладка



Автоматическая отладка



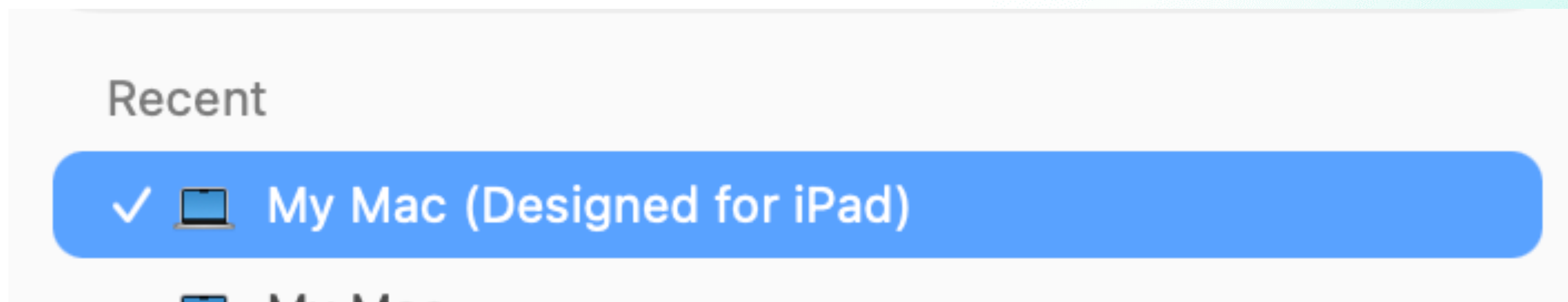
Автоматическая отладка

```
[injected_dylib] successfully injected
```


Автоматическая отладка

Бонусом при такой отладке получаем возможность запуска ipa из AppStore, собранного под iPhone на macOS.

Для этого нужно запустить в режиме My Mac (Designed for iPad).



«Подарки» от разработчиков

Что помогает при анализе

Оставленные логи:

- часто логируют работу внутренних компонентов;
- пишут, чего не хватает для нормальной работы (например, забыли про доступ к keychain или shared-группу для хранения файлов);
- логируют работу сети — не нужно перехватывать трафик для анализа;
- пишут секреты в открытом виде, так как «никто их прочитать не сможет».

Необфусцированный код.

7

Сложности и ограничения

Ограничения

Чтобы достать расшифрованное приложение, нужно устройство с джейлбрейком.

С каждой новой версией ОС увидеть джейлбрейк всё сложнее, а механизмы запуска и дешифровки IPA меняются всё чаще.

Иногда для анализа нужно глубоко понимать внутреннее устройство ОС или особенности работы встроенных фреймворков.

Ограничения

Работу усложняют (но не останавливают):

- Swift;
- оптимизация кода;
- отсутствие символов;
- обфускация кода.

8

ИТОГИ

Итоги

- Любое приложение, даже собранное под современные ОС, можно достать из App Store и расшифровать.
- Для анализа можно применять различные инструменты — как с открытым исходным кодом, так и с закрытым.
- Есть несколько возможностей для внесения изменений.
- Разрабатывать инжектируемую библиотеку можно в Xcode.
- Переподпись — довольно простой механизм, который можно написать самостоятельно.
- Отлаживать также можно быстро и удобно в Xcode.

Спасибо за внимание!

Development Team Lead

tg: @kudinovdenis

denis.kudinov@kaspersky.com

kaspersky