Yalu چطور کار میکند؟

مقدمه

Yalu جیلبرک نیمه کاره متن باز است که برای ۸/۴/۱ عرضه شده, این فرصت خوبی است که بتوانیم از این جیلبرک برای آموختن چگونگی نوشتن یک چنین ابزاری استفاده کنیم, به همین دلیل تصمیم گرفتم که خط به خط این ابزار را برای شما توضیح دهم تا شما دوست عزیز بیشتر با این ابزار ها آشنا شوید.

در اینجا لازم میبینم که از Luca Todesco نویسنده این ابزار تشکر ویژه ای داشته باشم که به من اجازه داد تا برای اولین بار در بین سایت های داخلی و خارجی به بررسی این ابزار بپردازم.

https://github.com/kpwn/yalu

در آخر هم لازم میبینم از عمو گوگل تشکر ویژه ای داشته باشم که کمک های بسیاری در زمینه ی پیدا کردن توضیحات متعارف تر به من کردند.

مهیار رزقی

مزیت ویژه ای که این ابزار برای ما به ارمغان آورد متن باز بودن آن است که به ما اجازه میدهد مو به مو فعالیت های آن را بررسی کنیم, در گذشته وقتی میخواستیم یک چنین کاری انجام دهیم باید از روش های مهندسی معکوس استفاده میکردیم ولی حالا کارمان راحت شده.

برای درک بیشتر مطالب بهتر است که به زبان bash آشنایی داشته باشید, و همچنین نا گفته نماند که این ابزار فعلا فقط بر روی مک قابل اجراست و به همین دلیل ما هم همانجا کار را دنبال میکنیم. اگر هم مک ندارید میتوانید از بک ماشین مجازی استفاده کنید.

به علت ساختار بسیار سخت و محکم (ولی نه ضد گلوله!) iOS معمولا باید از ترکیب چند باگ استفاده کرد تا بتوان به چنین آزادی عمل و جیلبرکی دست پیدا کرد. در زیر لیست فرایند هایی که در همه ی زمان ها از گذشته تا به امروز برای انجام جیلبرک با آن ها سر و کار داریم را بیان میکنم:

- از بین بردن سندباکس: اپل به صورت دیفالت تمامی برنامه ها را محدود کرده, البته به لطف Sandbox.kext و AppleMobileFileIntegrity.kext میتوان این محدودیت را از بین برد, که در ادامه توضیح میدهم.
- به دست آوردن قابلیت اجرای کد دلخواه: چیزی شبیه مثالی که در مبحث قبل زدیم, اما با این تفاوت که این کار معمولا با استفاده از حمله کردن به یکی از فایل های امضا شده (Signed) توسط اپل و اکسپلویت (Exploit) کردن آن مانند libmis انجام میگیرد.
- گرفتن دسترسی روت (root): از هک ساده وبسایت ها تا هک کردن دستگاه های اپل همیشه هدف رسیدن به بالاترین سطح دسترسی در سیستم عامل بوده و هست, که در دستگاه های اپلی از طریق حمله کردن به یکی از دیمون هایی (daemon) که به صورت روت در حال اجرا است, بسیار محترمانه خواهش میکنیم که دیمون launchd را به صورت روت برای ما اجرا کند!
- پچ کردن کرنل (Kernel Patching): بیشتر روش های این عمل در iOS به لطف Comex ممکن شده و ما در اینجا در واقع کرنل را بعد از روی کار آمدن Userland با استفاده از دستوراتی در هنگام بوت شدن (boot-args) و نه در nvram (همانطور که گفتیم امنیت iBoot بسیار پیشرفت کرده و تمام دستورات ما را رد میکند) انجام میدهیم.

البته نکته ای باید در نظر بگیرید این است که همیشه شما نیاز به ۴ پیلود مختلف برای انجام اینکار ندارید و در بیشتر موارد میتوان با حمله به یکی از سرویس های در حال اجرا روی دستگاه به هر سه مورد اول رسید. اما پچ کردن کرنل همیشه یک پیلود جداگانه نیاز خواهد داشت, زیرا امنیت XNU در طی سال های اخیر بسیار بالاتر رفته تا جلوی جیلبرک های متعدد با یک روش یکسان گرفته شود.

خوب برای شروع کار باید فایل run.sh اجرا شود پس کارمان را از همینجا شروع میکنیم:

```
#!/bin/sh

function abort() {

exit 254;

}

./stage0.sh || abort
```

یک پروسه کاملا ساده, در ابتدا یک تابع به نام abort برای متوقف کردن عملیات ایجاد میکنیم, بعد فایل stage0.sh اجرا میشود و || بیان میکند که اگر مشکلی در اجرا پیش آمد تابع abort را فراخوانی کن. پس قدم بعدی فایل stage0.sh است.

```
SCRIPTPATH=`dirname $0`

cd $SCRIPTPATH
```

در خط اول یک متغیر به نام SCRIPTPATH ایجاد میکنیم که با استفاده از دستور dirname $0 آدرس پوشه ای که اسکریپت در آن قرار دارد را به دست می آورد و در خط دوم به این پوشه منتقل میشویم.

خوب اگر در همین ابتدای کار تابع abort اجرا شود برنامه به ما پیغام Error. Exiting... را نشان میدهد که معمولا چنین موردی پیش نمی آید.

خوب اینجا با استفاده از "DISABLE FIND MY PHONE" echo به کاربر اعلام میشود که قابلیت Find My iPhone را خاموش کند و بلافاصله فایل wait_for_device.sh برای چک کردن اتصال دستگاه اجرا میشود.

```
while ! (./bin/afcclient deviceinfo | grep FSTotalBytes >/dev/null); do sleep 5; done 2>/dev/null
```

با اجرای bin/afcclient deviceinfo یک سری اطلاعات راجع به دستگاه متصل شده مانند شکل زیر به دست می آید:

```
Mahyars-Mac:yalu-master mahyarrezghi$ ./bin/afcclient deviceinfo
AFC Device Info: - Model:iPhone7,2 FSTotalBytes:59996618752 FSFreeBytes:39223201
792 FSBlockSize:4096
```

Model: مشخص کننده نوع دستگاه است که در اینجا iPhone 7,2 یعنی آیفون ۶ است.

FSTotalBytes: نشان دهنده ی ظرفیت دستگاه است.

FSFreeBytes: نشان دهنده ی فضای خالی دستگاه است.

برای فهمیدن اینکه دستگاه متصل است یا نه یک فرایند ساده انجام میشود, با استفاده از grep | تنها مقدار FSTotalBytes فراخوانی میشود و اگر مقدار داشت معلوم میشود که دستگاه متصل شده ولی اگر null (خالی) بود معلوم میشود دستگاه متصل نیست.

```
./bin/afcclient mkdir PhotoData/KimJongCracks
./bin/afcclient mkdir PhotoData/KimJongCracks/a
./bin/afcclient mkdir PhotoData/KimJongCracks/a/a
./bin/afcclient mkdir PhotoData/KimJongCracks/Library
./bin/afcclient mkdir PhotoData/KimJongCracks/Library/PrivateFrameworks
./bin/afcclient mkdir PhotoData/KimJongCracks/Library/PrivateFrameworks/GPUToolsCore.framework
```

با استفاده از کلاینت afc در پوشه /var/mobile/Media/PhotoData/ یک پوشه موقت به نام KimJongCracks ساخته میشود, دلیل انتخاب این آدرس هم این که کلاینت afc همیشه قابلیت ایجاد تغییرات در /var/mobile/Media/ را داشته و خواهد داشت. پس از ساخت پوشه های مورد نظر فایل stage1.sh اجرا میشود. (در خط بیس و یکم فایل stage0.sh)

با اجرای دستی فایل stage1.sh با خروجی های زیر رو به رو میشویم:

```
Mahyars-Mac:yalu-master mahyarrezghi$ ./stage1.sh
Setting up environment...
Uploaded 1923 bytes to /yalu.plist

Installing app & swapping binaries...OK

Waiting..
Uploaded 962 bytes to /yalu.plist

Mahyars-Mac:yalu-master mahyarrezghi$
```

در مرحله اول فایل WWDC_Info_TOC.plist در yalu.plist ذخیره میشود که اجازه میدهد فایل ساین شده WWDC بر روی دستگاه نصب شود

```
echo "Setting up environment..."
./bin/afcclient put ./data/WWDC_Info_TOC.plist /yalu.plist | grep Uploaded || abort
```

سپس فایل WWDC-TOCTOU.ipa که همان برنامه WWDC است نصب میشود

```
echo
printf "Installing app & swapping binaries..."
./bin/mobiledevice install_app ./data/WWDC-TOCTOU.ipa || abort
```

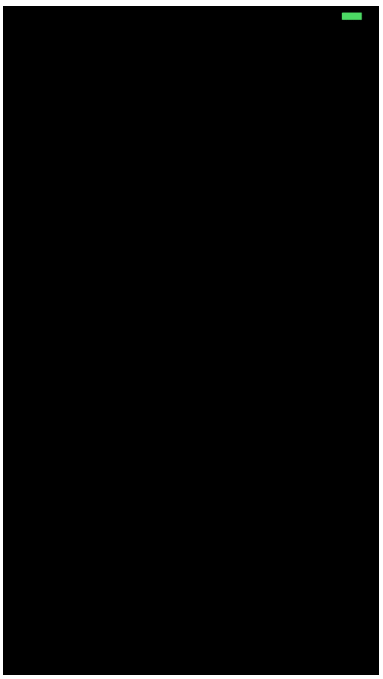
۵ ثانیه صبر میکنیم تا عملیات انجام شود

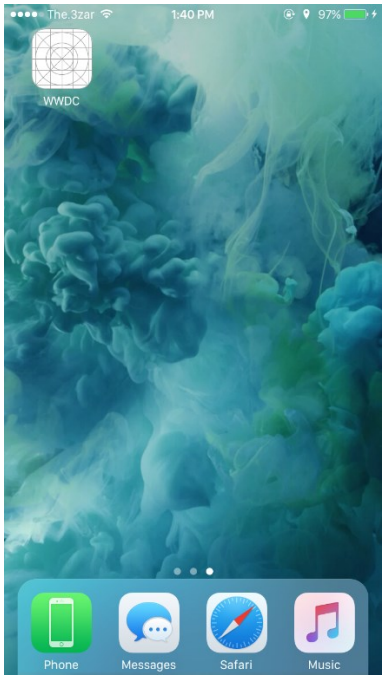
```
echo
echo "Waiting.."
sleep 5
```

حالا محتویات فایل WWDC_Info_TOU.plist در yalu.plist جایگزین میشود که به ما اجازه میدهد فایل های درون var/mobile/Media/PhotoData/KimJongCracks/a/a/MobileReplayer/ را اجرا کنیم

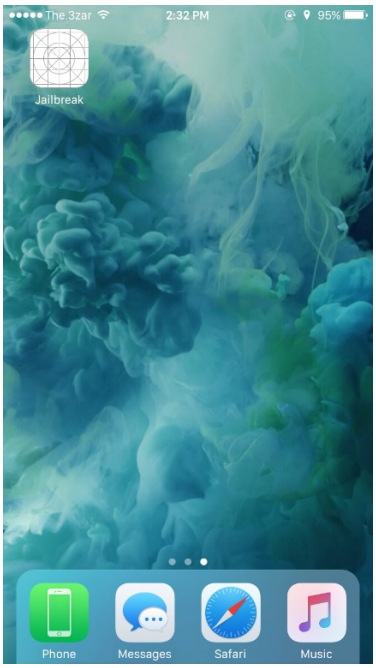
```
./bin/afcclient put ./data/WWDC_Info_TOU.plist /yalu.plist | grep Uploaded || abort
Echo
```

پس از طی کردن مرحله اول و دوم:





پس از جایگزینی yalu.plist برنامه به این شکل در می آید:



اگر نگاهی به محتویات فایل WWDC با استفاده از jtool بیاندازیم با خروجی های زیر رو به رو میشویم:

(برای بررسی باینری ۶۴ بیتی از دستور ARCH=armv8 ./jtool -l WWDC و برای بررسی باینری ۳۲ بیتی از دستور ARCH=armv7 ./jtool -l WWDC استفاده کنید)

```
Mahyars-Mac:Desktop mahyarrezghi$ ARCH=armv8 ./jtool -l WWDC
LC 00: LC_SEGMENT_64          Mem: 0x000000000-0x100000000      __PAGEZERO
LC 01: LC_SEGMENT_64          Mem: 0x100000000-0x10016c000      __TEXT
        Mem: 0x100005d40-0x100105480            __TEXT.__text   (Normal)
        Mem: 0x100105480-0x1001062c0            __TEXT.__stubs  (Symbol Stubs)
        Mem: 0x1001062c0-0x100107118            __TEXT.__stub_helper    (Normal)
        Mem: 0x100107118-0x10011f000            __TEXT.__objc_methname  (C-String Literals)
        Mem: 0x10011f000-0x100120ec0            __TEXT.__gcc_except_tab
        Mem: 0x100120ec0-0x10013cf93            __TEXT.__cstring        (C-String Literals)
        Mem: 0x10013cf93-0x10013f20d            __TEXT.__objc_classname (C-String Literals)
        Mem: 0x10013f20d-0x100143de1            __TEXT.__objc_methtype  (C-String Literals)
        Mem: 0x100143df0-0x100144e48            __TEXT.__const
        Mem: 0x100144e48-0x100168cac            __TEXT.__ustring
        Mem: 0x100168cac-0x10016bff8            __TEXT.__unwind_info
LC 02: LC_SEGMENT_64          Mem: 0x10016c000-0x1001ec000      __DATA
        Mem: 0x10016c000-0x10016c8a0            __DATA.__got    (Non-Lazy Symbol Ptrs)
        Mem: 0x10016c8a0-0x10016d220            __DATA.__la_symbol_ptr  (Lazy Symbol Ptrs)
        Mem: 0x10016d220-0x100171650            __DATA.__const
        Mem: 0x100171650-0x100187ff0            __DATA.__cfstring
        Mem: 0x100187ff0-0x1001888b8            __DATA.__objc_classlist (Normal)
        Mem: 0x1001888b8-0x100188920            __DATA.__objc_catlist   (Normal)
        Mem: 0x100188920-0x100188c10            __DATA.__objc_protolist
        Mem: 0x100188c10-0x100188c18            __DATA.__objc_imageinfo
        Mem: 0x100188c18-0x1001bfef8            __DATA.__objc_const
        Mem: 0x1001bfef8-0x1001c5f80            __DATA.__objc_selrefs   (Literal Pointers)
        Mem: 0x1001c5f80-0x1001c5ff0            __DATA.__objc_protorefs
        Mem: 0x1001c5ff0-0x1001c6b68            __DATA.__objc_classrefs (Normal)
        Mem: 0x1001c6b68-0x1001c71d0            __DATA.__objc_superrefs (Normal)
        Mem: 0x1001c71d0-0x1001c7e48            __DATA.__objc_ivar
        Mem: 0x1001c7e50-0x1001cdc80            __DATA.__objc_data
        Mem: 0x1001cdc80-0x1001e6375            __DATA.__data
        Mem: 0x1001e6378-0x1001eb8b8            __DATA.__swift1_proto
        Mem: 0x1001eb8c0-0x1001ebc18            __DATA.__bss    (Zero Fill)
        Mem: 0x1001ebc18-0x1001ebd80            __DATA.__common (Zero Fill)
LC 03: LC_SEGMENT_64          Mem: 0x1001ec000-0x100210000      __LINKEDIT
LC 04: LC_DYLD_INFO
LC 05: LC_SYMTAB
        Symbol table is at offset 0x1ff168 (2093416), 753 entries
        String table is at offset 0x202e48 (2109000), 22248 bytes
LC 06: LC_DYSYMTAB
            1 local symbols at index     0
            1 external symbols at index  1
          751 undefined symbols at index 2
           No TOC
           No modtab
          884 Indirect symbols at offset 0x202078
```

در این قسمت از خروجی هیچ چیز مشکوکی دیده نمیشود و همانند ابزار جیلبرک TaiG با یک ابزار سالم (حتی با برنامه نویسی بسیار تمیز تر نسبت به Taig) رو به رو هستیم و خبری از ویروس و بد افزار یا چیز دیگری نیست.

```
LC 07: LC_LOAD_DYLINKER         /usr/lib/dyld
LC 08: LC_UUID                  UUID: 72CF8BCC-E26F-3194-BA3C-1C1DBB9F33B1
LC 09: LC_VERSION_MIN_IPHONEOS  Minimum iOS  version:    8.3.0
LC 10: LC_SOURCE_VERSION        Source Version:          0.0.0.0.0
LC 11: LC_MAIN                  Entry Point:             0x5d40 (Mem: 100005d40)
LC 12: LC_ENCRYPTION_INFO_64    Encryption: 1 from offset 16384 spanning 1474560 bytes
LC 13: LC_LOAD_WEAK_DYLIB       /System/Library/Frameworks/LocalAuthentication.framework/LocalAuth
entication
LC 14: LC_LOAD_DYLIB            /System/Library/Frameworks/AVKit.framework/AVKit
LC 15: LC_LOAD_DYLIB            /System/Library/Frameworks/CloudKit.framework/CloudKit
LC 16: LC_LOAD_DYLIB            /System/Library/Frameworks/Accelerate.framework/Accelerate
LC 17: LC_LOAD_DYLIB            /usr/lib/libMobileGestalt.dylib
LC 18: LC_LOAD_DYLIB            /System/Library/Frameworks/Security.framework/Security
LC 19: LC_LOAD_DYLIB            /System/Library/Frameworks/MobileCoreServices.framework/MobileCore
Services
LC 20: LC_LOAD_DYLIB            /System/Library/Frameworks/AVFoundation.framework/AVFoundation
LC 21: LC_LOAD_DYLIB            /System/Library/Frameworks/PassKit.framework/PassKit
LC 22: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreTelephony.framework/CoreTelephony
LC 23: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreText.framework/CoreText
LC 24: LC_LOAD_DYLIB            /System/Library/Frameworks/QuartzCore.framework/QuartzCore
LC 25: LC_LOAD_DYLIB            /usr/lib/libz.1.dylib
LC 26: LC_LOAD_DYLIB            /System/Library/Frameworks/UIKit.framework/UIKit
LC 27: LC_LOAD_DYLIB            /System/Library/Frameworks/Foundation.framework/Foundation
LC 28: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreGraphics.framework/CoreGraphics
LC 29: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreData.framework/CoreData
LC 30: LC_LOAD_DYLIB            /System/Library/Frameworks/SystemConfiguration.framework/SystemCon
figuration
LC 31: LC_LOAD_DYLIB            @rpath/WWDCCore.framework/WWDCCore
LC 32: LC_LOAD_DYLIB            /usr/lib/libobjc.A.dylib
LC 33: LC_LOAD_DYLIB            /usr/lib/libSystem.B.dylib
LC 34: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreFoundation.framework/CoreFoundation
LC 35: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreLocation.framework/CoreLocation
LC 36: LC_LOAD_DYLIB            /System/Library/Frameworks/CoreMedia.framework/CoreMedia
LC 37: LC_LOAD_DYLIB            /System/Library/Frameworks/EventKit.framework/EventKit
LC 38: LC_LOAD_DYLIB            /System/Library/Frameworks/MediaPlayer.framework/MediaPlayer
LC 39: LC_LOAD_DYLIB            @rpath/libswiftCore.dylib
LC 40: LC_LOAD_DYLIB            @rpath/libswiftCoreGraphics.dylib
LC 41: LC_LOAD_DYLIB            @rpath/libswiftCoreImage.dylib
LC 42: LC_LOAD_DYLIB            @rpath/libswiftDarwin.dylib
LC 43: LC_LOAD_DYLIB            @rpath/libswiftDispatch.dylib
LC 44: LC_LOAD_DYLIB            @rpath/libswiftFoundation.dylib
LC 45: LC_LOAD_DYLIB            @rpath/libswiftObjectiveC.dylib
LC 46: LC_LOAD_DYLIB            @rpath/libswiftSecurity.dylib
LC 47: LC_LOAD_DYLIB            @rpath/libswiftUIKit.dylib
LC 48: LC_RPATH                 @executable_path/Frameworks
LC 49: LC_FUNCTION_STARTS       Offset: 2084240, Size: 8120 (0x1fcd90-0x1fed48) with 6281 function
s
LC 50: LC_DATA_IN_CODE          Offset: 2092360, Size: 0 (0x1fed48-0x1fed48)
LC 51: LC_DYLIB_CODE_SIGN_DRS   Offset: 2092360, Size: 1056 (0x1fed48-0x1ff168)
LC 52: LC_CODE_SIGNATURE        Offset: 2131248, Size: 21888 (0x208530-0x20dab0)
Segmentation fault: 11
Mahyars-Mac:Desktop mahyarrezghi$
```

libsystem.B: بدون این فایل هیچ برنامه یا فایلی نمیتواند پیش نیاز های خود را فراخوانی کند, میتوان کار این فایل را مانند Kernel32+NtDll+MSVCRT در ویندوز و همچنین libc.so در لینوکس در نظر گرفت.

توضیح کار تک تک DYLIB هایی که در اینجا مشاهده میکنید در سایت Theiphonewiki.com موجود است پس از توضیح مجدد کار تک تک آن ها خود داری میکنم.

یکی از مسائلی که مشاهده میکنیم این است که این جیلبرک بر روی ورژن های قبلی یعنی از ۸٫۳ به بعد قایل اجراست.

Minimum iOS version: 8.3.0

نکته ی دیگر آنکه بر خلاف TaiG این ابزار اینکریپت (کد گذاری) شده است.

Encryption: 1

خوب قبل از ادامه کار برنامه یک بکاپ از پوشه tmp/ گرفته میشود زیرا قرار است تغییراتی در این پوشه انجام شود. (خط ۲۳ فایل stage0.sh)

عملیات بکاپ گیری توسط سرویس com.apple.mobilebackup2 و از طریق پورت ۵۰۴۷۸ انجام میشود.

پوشه tmp/ هم از پوشه هایی است که سرویس afc بدون نیاز به جیلبرک میتواند در آن تغییرات انجام دهد و در واقع این پوشه سطح دسترسی ۷۷۷ دارد.

```
echo "Backing up, could take several minutes..." >&2
./bin/idevicebackup2 backup tmp || abort
udid="$(ls tmp | head -1)"
```

پس از اینکه عملیات بکاپ گیری تمام شد باید DDI را ماونت (Mount) کنیم.

خوب AFCD راه ورود را برای ما از میکند ولی فقط یکی دو قدم میتوانیم برداریم! بقیه دستورات باید توسط DeveloperDiskImage پشتیبانی بشوند. DeveloperDiskImage (DDI) زمانی روی کار می آید که شما دستگاه خود را به Xcode متصل میکنید (DDI بخشی از SDK مورد استفاده در Xcode است) که تعدادی ابزار و فریم ورک (Framework) را در اختیار ما میگذارد.

وقتی که دستگاه را به Xcode متصل میکنیم این ابزارها و فریم ورک ها در Developer/ مانت (Mount) میشود, و شما میتوانید در Developer/Library/Frameworks/ و System/Library/Frameworks/ و جند فولدر دیگر جست و جو کنید! البته فقط این فولدر نیست اپل این کار را برای فولدر های دیگر مانند AppleInternal/ نیز تکرار کرده. خوب پس اگر کسی بتواند یک DDI را شبیه سازی کند میتواند به فال های فولدر Developer/ دسترسی داشته باشد و آن ها را تغییر دهد.

یک پوشه به نام tmp_ddi ایجاد میکنیم تا DDI در آن ذخیره و ماونت شود:

```
mkdir tmp_ddi
```

این ابزار فایل DDI مورد نظر خود را مستقیما از Xcode به دست می آورد پس نیازمند این است که حتما Xcode را داشته باشید.

```
ddi="$(find /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/|grep 8.4|grep
.dmg'$'|head -1)"
```

با استفاده از ابزار hdiutil فایل DDI که به فرمت dmg از Xcode استخراج کردیم را ماونت میکنیم

```
hdiutil attach -nobrowse -mountpoint tmp_ddi "$ddi"
```

فایل MobileReplayer را از tmp_ddi به tmp انتقال میدهیم. (البته کپی میکنیم چون دستور کات کردن mv است)

```
cp tmp_ddi/Applications/MobileReplayer.app/MobileReplayer tmp/MobileReplayer
```

فایل info.plist مربوط به برنامه MobileReplayer را به tmp انتقال میدهیم و نام آن را به MobileReplayerInfo.plist تغییر میدهیم

```
cp tmp_ddi/Applications/MobileReplayer.app/Info.plist tmp/MobileReplayerInfo.plist
```

با استفاده از ابزار hdiutil فایل DDI را که ماونت کردیم را Unmount میکنیم

```
hdiutil detach tmp_ddi
```

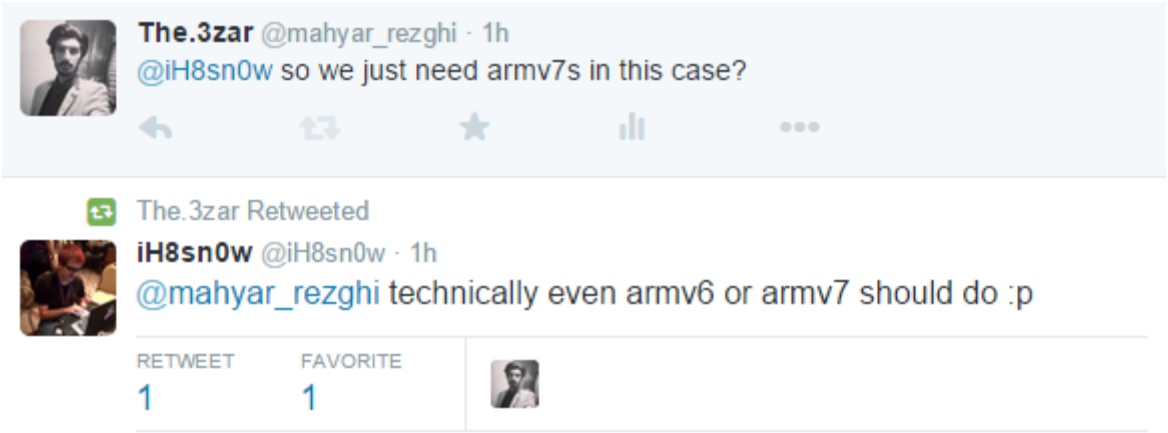
پوشه tmp_ddi را پاک میکنیم

```
rm -rf tmp_ddi
```

در خط ۳۵ به دستوری برخوردم که کمی برایم عجیب بود!

```
lipo tmp/MobileReplayer -thin armv7s -output ./tmp/MobileReplayer
```

از دستور lipo در واقع برای جدا کردن بخش ۳۲ بیتی یا ۶۴ بیتی یک باینری استفاده میشود, به طور مثال شما از سیستم عامل ۳۲ بیتی استفاده میکنید و نیازی به بخش ۶۴ بیتی نرم افزار ندارید با استفاده از این دستور فقط بخش ۳۲ بیتی را نگه میدارید و به این شکل حجم نرم افزار را کم میکنید. اما استفاده از این دستور در جایی که این ابزار قرار است هم دستگاه های ۶۴ بیتی و هم ۳۲ بیتی را جیلبرک کند, کمی عجیب است زیرا جایی هم بررسی نمیشود که دستگاهی که متصل شده ۳۲ بیتی است یا ۶۴ بیتی!



تصمیم گرفتم این موضوع را بررسی کنم و به همین دلیل از iH8sn0w در این مورد سوال کردم, او گفت که هدف استفاده از قدیمی ترین معماری پشتیبانی شده توسط xcode (که در حال حاضر armv7 است) هست. ولی در اینجا هیچ تفاوتی ندارد که از کدام معماری استفاده کنیم.

مرحله بعد انتقال فایل MobileReplayer به پوشه Media/PhotoData/KimJongCracks/a/a/ است که بتوانیم آنرا توسط برنامه ای در ابتدای مطلب جایگزین کردیم اجرا کنیم.

```
./bin/mbdbtool tmp $udid CameraRollDomain rm Media/PhotoData/KimJongCracks/a/a/MobileReplayer
```

```
./bin/mbdbtool tmp $udid CameraRollDomain put ./tmp/MobileReplayer Media/PhotoData/KimJongCracks/a/a/MobileReplayer || abort
```

در اینجا خط دوم همان کار خط اول را هم میکرد, زیرا وقتی فایلی را کپی کنید به صورت خودکار جایگزین میشود, پس احتمالا برنامه نویس اعتقاد داشته کار از محکم کاری عیب نمیکند.

بکاپی که از پوشه tmp گرفته بودیم را ریستور میکنیم و دستور ریبوت شدن دستگاه را میدهیم:

```
./bin/idevicebackup2 restore tmp --system --reboot || abort
```

پس از ۲۰ ثانیه دوباره فایل wait_for_device.sh فراخوانی میشود تا متصل بودن یا نبودن دستگاه به کامپیوتر را تعیین کند.

از خط ۴۸ به بعد عملیات کرنل پچینگ انجام میشود که در مقاله ای مجزا به بررسی آن میپردازیم زیرا بحثی کاملا تخصصی و زمان بر است, فعلا ما تنها به بررسی پیلود بسنده میکنیم زیرا بخش کرنل این ابزار هنوز کامل نشده است.

```
echo "Tap on the jailreak icon to crash the kernel (or dump it if you're in luck!)"
```

این خط پایانی این ابزار است و به ما اعلام میکند که روی آیکون Jailbreak تپ کنیم تا کرنل پچ شود.

خروجی هایی که هنگام کار ابزار دیده میشود چیزی شبیه به http://pastebin.com/vDDeSAyJ هستند, متاسفانه من نمیتوانستم ابزار را به طور کامل اجرا کنم زیرا از ورژن ۸٫۴ که با ابزار TaiG2 جیلبرک شده بود استفاده میکردم.

سخن پایانی

از همه ی شما خوانندگان عزیز ممنونم که تا اینجای مقاله را دنبال کردید, سعی میکنم در آینده نیز بتوانم با مطالب بهتر و کاملتر در خدمت شما باشم.

نیاز میبینم که نام افرادی که هم بنده در این مقاله از صحبت ها و مقالاتشان استفاده کردم و هم کمک شایانی به جامعه جیلبرک کردند را ذکر کنم:

@comex

@Technologeeks

@iH8sn0w

@qwertyoruiop

اگر خواستید بیشتر در جریان کار های آینده من قرار بگیرید میتوانید از اکانت توییتر من دیدن فرمایید:

@mahyar_rezghi

همچنین میتوانید از طریق ایمیل mahyar_rezghi@me.com با من در ازتباط باشید.

مهیار رزقی