# Part II Tools

In the 1st part, we’ve introduced the basic concepts of iOS reverse engineering. In this part, we will introduce the toolkit of iOS reverse engineering.

Compared with App development, the main feature of iOS reverse engineering is it’s more “mixed”. When you are writing Apps, most work can be done within Xcode, since it is the descendant of Apple, it’s convenient to download, install and use. As for some other tools and plugins, they are just some kind of icing on the cake, thus useful but unnecessary.

But, in iOS reverse engineering, we have to face so many complicated tools. Let me take an example, there are two dinner tables in front of you, on the first table there’re only a bowl of noodles and a pair of chopsticks, it’s named Xcode; the other one is full of crabs, steaks and tools to eat them, such as knives and forks, in which some of the big shots are Theos, Reveal, IDA and etc etc…

Unlike Xcode, there is no tight connection among those reverse engineering tools; they are separated from each other, so we need to integrate them manually. We cannot cover all reverse engineering tools in this part, but I think you will have the ability to find and use proper tools according to the situation you face when you finish reading this book. You can also share your findings with us on http://bbs.iosre.com.

Because the tools to be introduced are quite disordered, we split this part to two chapters, one is for OSX tools, the other is for iOS. The device used in this part is iPhone 5 with iOS 8.1.

# Chapter 3 OSX toolkit

Tools used for iOS reverse engineering have different functions, and they play different roles. These tools mainly help us develop and debug on OSX. Because of the screen size of iOS devices, they are not suitable for development or debug.

In this chapter, 4 major tools will be introduced, they’re class-dump, Theos, Reveal and IDA. Other tools are assistants for them.

## 3.1 class-dump

class-dump, as the name indicates, is a tool used for dumping the class information of the specified object. It makes use of the runtime mechanism of Objective-C language to extract the headers information stored in Mach-O files, and then generates .h files.

class-dump is simple to use, firstly, you need to download the latest version from <http://stevenygard.com/projects/class-dump>, as figure 3-1 shows:

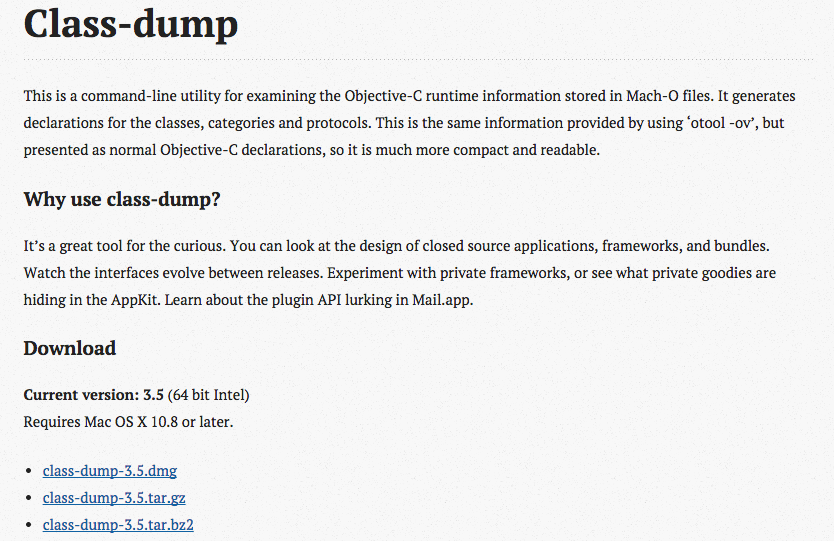


Figure 3-1 Homepage of class-dump

After downloading and decompressing class-dump-3.5.dmg, copy the class-dump executable to “/usr/bin”, and run “sudo chmod 777 /usr/bin/class-dump” in Terminal to grant it execute permission. Run class-dump, you will see its usage:

snakeninnysiMac:~ snakeninny$ class-dump

class-dump 3.5 (64 bit)

Usage: class-dump [options] <mach-o-file>

where options are:

-a show instance variable offsets

-A show implementation addresses

--arch <arch> choose a specific architecture from a universal binary (ppc, ppc64, i386, x86\_64, armv6, armv7, armv7s, arm64)

-C <regex> only display classes matching regular expression

-f <str> find string in method name

-H generate header files in current directory, or directory specified with -o

-I sort classes, categories, and protocols by inheritance (overrides -s)

-o <dir> output directory used for -H

-r recursively expand frameworks and fixed VM shared libraries

-s sort classes and categories by name

-S sort methods by name

-t suppress header in output, for testing

--list-arches list the arches in the file, then exit

--sdk-ios specify iOS SDK version (will look in /Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS<version>.sdk

--sdk-mac specify Mac OS X version (will look in /Developer/SDKs/MacOSX<version>.sdk

--sdk-root specify the full SDK root path (or use --sdk-ios/--sdk-mac for a shortcut)

The targets of class-dump are Mach-O binaries, such as library files of frameworks and executables of Apps. Now, I will show you an example of how to use class-dump.

#### Locate the executable of an App

First, copy the target App to OSX, as I placed it under “/Users/snakeninny”. Then go to App’s directory in Terminal, and use plutil, the Xcode built-in tool to inspect the “CFBundleExecutable” field in Info.plist:

snakeninnysiMac:~ snakeninny$ cd /Users/snakeninny/SMSNinja.app/

snakeninnysiMac:SMSNinja.app snakeninny$

snakeninnysiMac:SMSNinja.app snakeninny$ plutil -p Info.plist | grep CFBundleExecutable

"CFBundleExecutable" => "SMSNinja"

“SMSNinja” in the current directory is the executable of the target App.

#### class-dump the executable

class-dump SMSNinja headers to the directory of “/path/to/headers/SMSNinja/”, and sort them by name as follows:

snakeninnysiMac:SMSNinja.app snakeninny$ class-dump -S -s -H SMSNinja -o /path/to/headers/SMSNinja/

Repeat this on your own App, and compare the original headers with class-dump headers, aren’t they very similar? You will see all the methods are nearly the same except that some arguments’ types has been changed to id and their names are missing. With “-S” and “-s” options, the headers are even more readable.

class-dumping our own Apps doesn’t make much sense; since class-dump works on closed-source Apps of our own, it can also be used to analyze others’ Apps.

From the dumped headers, we can take a peek at the architecture of an App; information under the skin is the cornerstone of iOS reverse engineering. Now that App sizes have become bigger and bigger, more and more third-party libraries are integrated into our own projects, class-dump often produces hundreds and thousands of headers. It’d be great practices analyzing them one by one manually, but that’s overwhelming workload. In the following chapters, we will show you several ways to unload our workload and focus on the core problems.

It’s worth mentioning that, Apps downloaded from AppStore are encrypted by Apple, executables are “shelled” like walnuts, protecting class-dump from working, class-dump will fail in this situation. To enable it again, we need other tools to crack the shell at first, and I’ll leave this to the next chapter. To learn more about class-dump, please refer to <http://bbs.iosre.com>.

## 3.2 Theos

### 3.2.1 Introduction to Theos

Theos is a jailbreak development tool written and shared on GitHub by a friend, Dustin Howett (@DHowett). Compared with other jailbreak development tools, Theos’ greatest feature is simplicity: It’s simple to download, install, compile and publish; the bulit-in Logos syntax is simple to understand. It greatly reduces our work besides coding.

Additionally, iOSOpenDev, which runs as a plugin of Xcode is another frequently used tool in jailbreak development, developers who are familiar with Xcode may feel more interested in this tool, which is more integrated than Theos. But, reverse engineering deals with low-level knowledge a lot, most of the work can’t be done automatically by tools, it’d be better for you to get used to a less integrated environment. Therefore I strongly recommend Theos, when you use it to finish one practice after another, you will definitely gain a deeper understanding of iOS reverse engineering.

### 3.2.2 Install and configure Theos

#### Install Xcode and Command Line Tools

Most iOS developers have already installed Xcode, which contains Command Line Tools. For those who don’t have Xcode yet, please download it from Mac AppStore for free. If two or more Xcodes have been installed already, one Xcode should be specified as “active” by “xcode-select”, Theos will use this Xcode by default. For example, if 3 Xcodes have been installed on your Mac, namely Xcode1.app, Xcode2.app and Xcode3.app, and you want to specify Xcode3 as active, please use the following command:

snakeninnys-MacBook:~ snakeninny$ sudo xcode-select -s /Applications/Xcode3.app/Contents/Developer

#### Download Theos

Download Theos from GitHub using the following commands:

snakeninnysiMac:~ snakeninny$ export THEOS=/opt/theos

snakeninnysiMac:~ snakeninny$ sudo git clone git://github.com/DHowett/theos.git $THEOS

Password:

Cloning into '/opt/theos'...

remote: Counting objects: 4116, done.

remote: Total 4116 (delta 0), reused 0 (delta 0)

Receiving objects: 100% (4116/4116), 913.55 KiB | 15.00 KiB/s, done.

Resolving deltas: 100% (2063/2063), done.

Checking connectivity... done

#### Configure ldid

ldid is a tool to sign iOS executables; it replaces codesign from Xcode in jailbreak development. Download it from <http://joedj.net/ldid> to “/opt/theos/bin/”, then grant it execute permission using the following command:

snakeninnysiMac:~ snakeninny$ sudo chmod 777 /opt/theos/bin/ldid

#### Configure CydiaSubstrate

First run the auto-config script in Theos:

snakeninnysiMac:~ snakeninny$ sudo /opt/theos/bin/bootstrap.sh substrate

Password:

Bootstrapping CydiaSubstrate...

Compiling iPhoneOS CydiaSubstrate stub... default target?

failed, what?

Compiling native CydiaSubstrate stub...

Generating substrate.h header...

Here we’ll meet a bug that Theos cannot generate a working libsubstrate.dylib, which requires our manual fixes. Piece of cake: first search and install CydiaSubstrate in Cydia, as shown in figure 3-2.

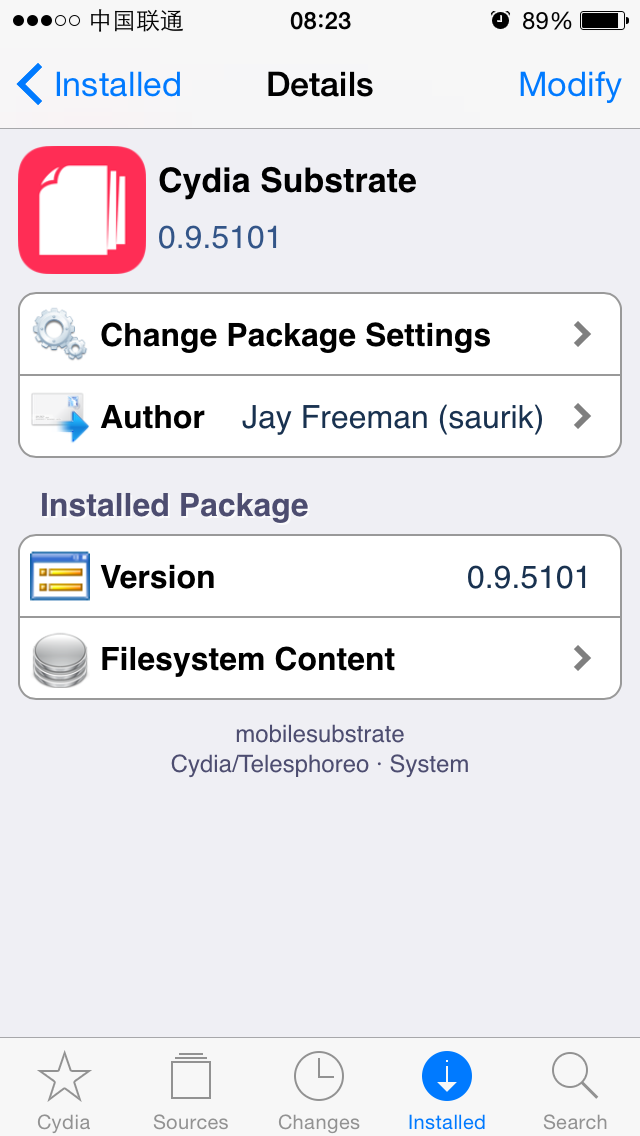


Figure 3- 2 CydiaSubstrate

Then copy “/Library/Frameworks/CydiaSubstrate.framework/CydiaSubstrate” on iOS to OSX using iFunBox or scp. Rename it libsubstrate.dylib and copy it to “/opt/theos/lib/libsubstrate.dylib” to replace the invalid file.

#### Configure dpkg-deb

The standard installation package format in jailbreak development is deb, which can be manipulated by dpkg-deb. Theos uses dpkg-deb to pack projects to debs.

Download dm.pl from [https://raw.githubusercontent.com/DHowett/dm.pl/master/dm.pl](https://raw.githubusercontent.com/dhowett/dm.pl/master/dm.pl), rename it dpkg-deb and move it to “/opt/theos/bin/”, then grant it execute permission using the following command:

snakeninnysiMac:~ snakeninny$ sudo chmod 777 /opt/theos/bin/dpkg-deb

#### Configure Theos NIC templates

It is convenient for us to create various Theos projects because Theos NIC templates have 5 different Theos project templates. You can also get 5 extra templates from [https://github.com/DHowett/theos-nic-templates/archive/master.zip](https://github.com/dhowett/theos-nic-templates/archive/master.zip) and put the 5 extracted .tar files under “/opt/theos/templates/iphone/”.

### 3.2.3 Use Theos

#### Create Theos project

1. Change Theos’ working directory to whatever you want (like mine is “/User/snakeninny/Code”), and then enter “/opt/theos/bin/nic.pl” to start NIC (New Instance Creator), as follows:

snakeninnysiMac:Code snakeninny$ /opt/theos/bin/nic.pl

NIC 2.0 - New Instance Creator

------------------------------

[1.] iphone/application

[2.] iphone/cydget

[3.] iphone/framework

[4.] iphone/library

[5.] iphone/notification\_center\_widget

[6.] iphone/preference\_bundle

[7.] iphone/sbsettingstoggle

[8.] iphone/tool

[9.] iphone/tweak

[10.] iphone/xpc\_service

There are 10 templates available, among which 1, 4, 6, 8, 9 are Theos embedded, and 2, 3, 5, 7, 10 are downloaded in the last section. At the beginning stage of iOS reverse engineering, we’ll be writing tweaks most of the time, usage of other templates can be discussed on <http://bbs.iosre.com>.

1. Chose "9" to create a tweak project:

Choose a Template (required): 9

1. Enter the name of the tweak project:

Project Name (required): iOSREProject

1. Enter a bundle identifier as the name of the deb package:

Package Name [com.yourcompany.iosreproject]: com.iosre.iosreproject

1. Enter the name of the tweak author:

Author/Maintainer Name [snakeninny]: snakeninny

1. Enter “MobileSubstrate Bundle filter”, i.e. bundle identifier of the tweak target:

[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]: com.apple.springboard

1. Enter the name of the process to be killed after tweak installation:

[iphone/tweak] List of applications to terminate upon installation (space-separated, '-' for none) [SpringBoard]: SpringBoard

Instantiating iphone/tweak in iosreproject/...

Done.

After these 7 simple steps, a folder named iosreproject is created in the current directory, which contains the tweak project we just created.

#### 2. Customize project files

It's convenient to create a tweak project with Theos, but the project is so rough that it needs further polish, more information is required. Anyway, let’s take a look at our project folder:

snakeninnysiMac:iosreproject snakeninny$ ls -l

total 40

-rw-r--r-- 1 snakeninny staff 184 Dec 3 09:05 Makefile

-rw-r--r-- 1 snakeninny staff 1045 Dec 3 09:05 Tweak.xm

-rw-r--r-- 1 snakeninny staff 223 Dec 3 09:05 control

-rw-r--r-- 1 snakeninny staff 57 Dec 3 09:05 iOSREProject.plist

lrwxr-xr-x 1 snakeninny staff 11 Dec 3 09:05 theos -> /opt/theos

There are only 4 files except one symbolic link pointing to Theos. To be honest, when I first created a tweak project with Theos as a newbie, the simplicity of this project actually attracted me instead of freaking me out, which surprised me. Less is more, Theos does an amazing job in good user experience.

4 files are enough to build a roughcast house, yet more decoration is needed to make it flawless. We’re going to extend the 4 files for now.

1. Makefile

The project files, frameworks and libraries are all specified in Makefile, making the whole compile process automatic. The Makefile of iOSREProject is shown as follows:

include theos/makefiles/common.mk

TWEAK\_NAME = iOSREProject

iOSREProject\_FILES = Tweak.xm

include $(THEOS\_MAKE\_PATH)/tweak.mk

after-install::

install.exec "killall -9 SpringBoard"

Let’s do a brief introduction line by line.

include theos/makefiles/common.mk

This is a fixed writing pattern, don’t make changes.

TWEAK\_NAME = iOSREProject

The tweak name, i.e. the “Project name” in NIC when we create a Theos project. It corresponds to the "Name" field of the control file, please don’t change it.

iOSREProject\_FILES = Tweak.xm

Source files of the tweak project, excluding headers; multiple files should be separated by spaces, like:

iOSREProject\_FILES = Tweak.xm Hook.xm New.x ObjC.m ObjC++.mm

It can be changed on demand.

include $(THEOS\_MAKE\_PATH)/tweak.mk

According to different types of Theos projects, different .mk files will be included. In the beginning stage of iOS reverse engineering, 3 types of projects are commonly created, they are Application, Tweak and Tool, whose corresponding files are application.mk, tweak.mk and tool.mk respectively. It can be changed on demand.

after-install::

install.exec "killall -9 SpringBoard"

I guess you know what’s the purpose of these two lines of code from the literal meaning, which is to kill SpringBoard after the tweak is installed, and to let CydiaSubstrate load the proper dylibs into SpringBoard when it relaunches.

The content of Makefile seems easy, right? But it's too easy to be enough for a real tweak project. How do we specify the SDK version? How do we import frameworks? How do we link libs? These questions remain to be answered. Don't worry, the bread will have of, the milk will also have of.

* Specify CPU architectures

ARCHS = armv7 arm64

Different CPU architectures should be separated by spaces in the above configuration. Note, Apps with arm64 instructions are not compatible with armv7/armv7s dylibs, they have to link dylibs of arm64. In the vast majority of cases, just leave it as “armv7 arm64”.

* Specify the SDK version

TARGET = iphone:Base SDK:Deployment Target

For example:

TARGET = iphone:8.1:8.0

It specifies the base SDK version of this project to 8.1, as well deployment target to iOS 8.0. We can also specify "Base SDK" to "latest" to indicate that the project will be compiled with the latest SDK of Xcode, like:

TARGET = iphone:latest:8.0

* Import frameworks

iOSREProject\_FRAMEWORKS = framework name

For example:

iOSREProject\_FRAMEWORKS = UIKit CoreTelephony CoreAudio

There is nothing to explain. However, as tweak developers, how to import private frameworks attracts us more for sure. It’s no much difference to importing documented frameworks:

iOSREProject\_PRIVATE\_FRAMEWORKS = private framework name

For example:

iOSREProject\_PRIVATE\_FRAMEWORKS = AppSupport ChatKit IMCore

Although it seems to be only one inserted word " PRIVATE ", there’s more to notice. Importing private frameworks is not allowed in AppStore development, most of us are not familiar with them. Private frameworks change a lot in each iOS version, so before importing them, please make sure of the existence of the imported frameworks. For example, if you want your tweak to be compatible with both iOS 7 and iOS 8, then Makefile could be written as follows:

ARCHS = armv7 arm64

TARGET = iphone:latest:7.0

include theos/makefiles/common.mk

TWEAK\_NAME = iOSREProject

iOSREProject\_FILES = Tweak.xm

iOSREProject\_PRIVATE\_FRAMEWORK = BaseBoard

include $(THEOS\_MAKE\_PATH)/tweak.mk

after-install::

install.exec "killall -9 SpringBoard"

This tweak can be compiled and linked successfully without any error. However, BaseBoard.framework only exists in SDK of iOS 8 and above, so this tweak would fail to work on iOS 7 because of the lack of specified frameworks. In this case, "weak linking" or dyld series functions like dlopen(), dlsym() and dlclose() can solve this problem.

* Link Mach-O Objects

iOSREProject\_LDFLAGS = -lx

Theos use GNU Linker to link Mach-O objects, including .dylib, .a and .o files. Input "man ld" in Terminal and locate to "-lx", it is described as follows:

*“-lx This option tells the linker to search for libx.dylib or libx.a in the library search path. If string x is of the form y.o, then that file is searched for in the same places, but without prepending `lib' or appending `.a' or `.dylib' to the filename.”*

As shown in figure 3-3, all Mach-O objects are named in the formats of "libx.dylib" and "y.o", who’re fully compatible with GNU Linker.

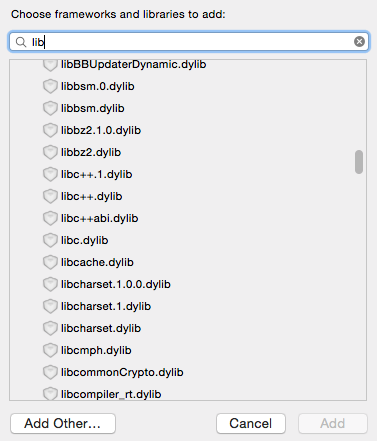


Figure 3- 3 Link Mach-O Objects

So, linking Mach-O objects becomes convenient. For example, if you want to link libsqlite3.0.dylib, libz.dylib and dylib1.o, you can do it like this:

iOSREProject\_LDFLAGS = -lz –lsqlite3.0 –dylib1.o

There is still one more field to introduce later, but without it Makefile is good to work for now. For more Makefile introductions, you can refer to [http://www.gnu.org/software/make/manual/html\_node/Makefiles.html](http://www.gnu.org/software/make/manual/html_node/makefiles.html).

1. Tweak.xm

The default source file of a tweak project created by Theos is Tweak.xm. "x" in "xm" indicates that this file supports Logos syntax; if this file is suffixed with an only "x", it means Tweak.x supports both Logos and C; if the suffix is "xm", Tweak.xm supports Logos, C and C++, just like the differences between "m" and "mm" files. The content of Tweak.xm is as follows:

/\* How to Hook with Logos

Hooks are written with syntax similar to that of an Objective-C @implementation.

You don't need to #include <substrate.h>, it will be done automatically, as will

the generation of a class list and an automatic constructor.

%hook ClassName

// Hooking a class method

+ (id)sharedInstance {

return %orig;

}

// Hooking an instance method with an argument.

- (void)messageName:(int)argument {

%log; // Write a message about this call, including its class, name and arguments, to the system log.

%orig; // Call through to the original function with its original arguments.

%orig(nil); // Call through to the original function with a custom argument.

// If you use %orig(), you MUST supply all arguments (except for self and \_cmd, the automatically generated ones.)

}

// Hooking an instance method with no arguments.

- (id)noArguments {

%log;

id awesome = %orig;

[awesome doSomethingElse];

return awesome;

}

// Always make sure you clean up after yourself; Not doing so could have grave consequences!

%end

\*/

These are the basic Logos syntax, including 3 preprocessor directives: %hook, %log and %orig. The next 3 examples show how to use them.

* %hook

%hook specifies the class to be hooked, must end with %end, for example:

%hook SpringBoard

- (void)\_menuButtonDown:(id)down

{

NSLog(@"You’ve pressed home button.");

%orig; // call the original \_menuButtonDown:

}

%end

This snippet is to hook [SpringBoard \_menuButtonDown:], write something to syslog before executing the original method.

* %log

This directive is used inside %hook to write the method arguments to syslog. We can also append anything else with the format of %log([(<type>)<expr>, …]), for example:

%hook SpringBoard

- (void)\_menuButtonDown:(id)down

{

%log((NSString \*)@"iOSRE", (NSString \*)@"Debug");

%orig; // call the original \_menuButtonDown:

}

%end

The output is as follows:

Dec 3 10:57:44 FunMaker-5 SpringBoard[786]: -[<SpringBoard: 0x150eb800> \_menuButtonDown:+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Timestamp: 75607608282

Total Latency: 20266 us

SenderID: 0x0000000100000190

BuiltIn: 1

AttributeDataLength: 16

AttributeData: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ValueType: Absolute

EventType: Keyboard

UsagePage: 12

Usage: 64

Down: 1

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

]: iOSRE, Debug

* %orig

%orig is also used inside %hook; it executes the original hooked method, for example:

%hook SpringBoard

- (void)\_menuButtonDown:(id)down

{

NSLog(@"You’ve pressed home button.");

%orig; // call the original \_menuButtonDown:

}

%end

If %orig is removed, the original method will not be executed, for example:

%hook SpringBoard

- (void)\_menuButtonDown:(id)down

{

NSLog(@"You’ve pressed home button but it’s not functioning.");

}

%end

It can also be used to replace arguments of the original method, for example:

%hook SBLockScreenDateViewController

- (void)setCustomSubtitleText:(id)arg1 withColor:(id)arg2

{

%orig(@"iOS 8 App Reverse Engineering", arg2);

}

%end

The lock screen looks like figure 3-4 with the new argument:



Figure 3- 4 Hack the lock screen

Besides %hook, %log and %orig, there are other common preprocessor directives such as %group, %init, %ctor, %new and %c.

* %group

This directive is used to group %hook directives for better code management and conditional initialization (We’ll talk about this soon). %group must end with %end, one %group can contain multiple %hooks, all %hooks that do not belong to user-specific groups will be grouped into %group \_ungrouped. For example:

%group iOS7Hook

%hook iOS7Class

- (id)iOS7Method

{

id result = %orig;

NSLog(@"This class & method only exist in iOS 7.");

return result;

}

%end

%end // iOS7Hook

%group iOS8Hook

%hook iOS8Class

- (id)iOS8Method

{

id result = %orig;

NSLog(@"This class & method only exist in iOS 8.");

return result;

}

%end

%end // iOS8Hook

%hook SpringBoard

-(void)powerDown

{

%orig;

}

%end

Inside %group iOS7Hook, it hooks [iOS7Class iOS7Method]; inside %group iOS8Hook, it hooks [iOS8Class iOS8Method]; and inside % group \_ungrouped, it hooks [SpringBoard powerDown]. Can you get it?

Notice, %group will only work with %init.

* %init

This directive is used for %group initialization; it must be called inside %hook or %ctor. If a group name is specified, it will initialize %group SpecifiedGroupName, or it will initialize %group \_ungrouped, for example:

#ifndef kCFCoreFoundationVersionNumber\_iOS\_8\_0

#define kCFCoreFoundationVersionNumber\_iOS\_8\_0 1140.10

#endif

%hook SpringBoard

- (void)applicationDidFinishLaunching:(id)application

{

%orig;

%init; // Equals to %init(\_ungrouped)

if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber\_iOS\_7\_0 && kCFCoreFoundationVersionNumber < kCFCoreFoundationVersionNumber\_iOS\_8\_0) %init(iOS7Hook);

if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber\_iOS\_8\_0) init(iOS8Hook);

}

%end

Please remember, a %group will only take effect with a corresponding %init.

* %ctor

The constructor of a tweak, it is the first function to be called in the tweak. If we don’t define a constructor explicitly, Theos will create one for us automatically, and call %init(\_ungrouped) inside it.

%hook SpringBoard

- (void)reboot

{

NSLog(@"If rebooting doesn’t work then I’m screwed.");

%orig;

}

%end

The above code works fine, because Theos has called %init implicitly like this:

%ctor

{

%init(\_ungrouped);

}

However,

%hook SpringBoard

- (void)reboot

{

NSLog(@"If rebooting doesn’t work then I’m screwed.");

%orig;

}

%end

%ctor

{

// Need to call %init explicitly!

}

This %hook never works, because we’ve defined %ctor explicitly without calling %init explicitly, there lacks a %group(\_ungrouped). Generally, %ctor is used to call %init and MSHookFunction, for example:

#ifndef kCFCoreFoundationVersionNumber\_iOS\_8\_0

#define kCFCoreFoundationVersionNumber\_iOS\_8\_0 1140.10

#endif

%ctor

{

%init;

if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber\_iOS\_7\_0 && kCFCoreFoundationVersionNumber < kCFCoreFoundationVersionNumber\_iOS\_8\_0) %init(iOS7Hook);

if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber\_iOS\_8\_0) %init(iOS8Hook);

MSHookFunction((void \*)&AudioServicesPlaySystemSound,

(void \*)&replaced\_AudioServicesPlaySystemSound,

(void \*\*)&original\_AudioServicesPlaySystemSound);

}

Attention, %ctor doesn’t end with %end.

* %new

%new is used inside %hook to add a new method to an existing class; it's the same as class\_addMethod, for example:

%hook SpringBoard

%new

- (void)namespaceNewMethod

{

NSLog(@"We’ve added a new method to SpringBoard.");

}

%end

Some of you may wonder, category in Objective-C can already adds new methods to classes, why do we still need %new? The difference between category and %new is that the former is static while the latter is dynamic. Well, does static adding or dynamic adding matter? Yes, especially when the class to be added is from a certain executable, it matters. For example, the above code adds a new method to SpringBoard. If we use category, the code should look like this:

@interface SpringBoard (iOSRE)

- (void)namespaceNewMethod;

@end

@implementation SpringBoard (iOSRE)

- (void)namespaceNewMethod

{

NSLog(@"We’ve added a new method to SpringBoard.");

}

@end

We will get “error: cannot find interface declaration for ‘SpringBoard’” when trying to compile the above code, which indicates that the compiler cannot find the definition of SpringBoard. We can compose a SpringBoard class to cheat the compiler:

@interface SpringBoard : NSObject

@end

@interface SpringBoard (iOSRE)

- (void)namespaceNewMethod;

@end

@implementation SpringBoard (iOSRE)

- (void)namespaceNewMethod

{

NSLog(@"We’ve added a new method to SpringBoard.");

}

@end

Recompile it, we’ll still get the following error:

Undefined symbols for architecture armv7:

"\_OBJC\_CLASS\_$\_SpringBoard", referenced from:

l\_OBJC\_$\_CATEGORY\_SpringBoard\_$\_iOSRE in Tweak.xm.b1748661.o

ld: symbol(s) not found for architecture armv7

clang: error: linker command failed with exit code 1 (use -v to see invocation)

ld cannot find the definition of SpringBoard. Normally, when there’s “symbol(s) not found”, most of you may think, if this is because I forget to import any framework? But, SpringBoard is a class of SpringBoard.app rather than a framework, how do we import an executable? I bet you know %new’s usage right now.

* %c

This directive is equal to objc\_getClass or NSClassFromString, it is used in %hook or %ctor to dynamically get a class by name.

Other Logos preprocessor directives including %subclass and %config are seldom used, at least I myself have never used them before. Nonetheless, if you’re interested in them, you can refer to http://iphonedevwiki.net/index.php/Logos, or go to <http://bbs.iosre.com> for discussion.

1. control

The contents of control file are basic information of the current deb package; they will be packed into the deb package. The contents of iOSREProject's control file are shown as follows:

Package: com.iosre.iosreproject

Name: iOSREProject

Depends: mobilesubstrate

Version: 0.0.1

Architecture: iphoneos-arm

Description: An awesome MobileSubstrate tweak!

Maintainer: snakeninny

Author: snakeninny

Section: Tweaks

Let me give a brief introduction of this file.

* Package field is the name of the deb package, it has the similar naming convention to bundle identifier, i.e. reverse DNS format. It can be changed on demand.
* Name field is used to describe the name of the project; it also can be changed.
* Depends field is used to describe the dependency of this deb package. Dependency means the basic condition to run this tweak, if the current environment does not meet the condition described in depends field, this tweak cannot run properly. For example, the following code means the tweak must run on iOS 8.0 or later with CydiaSubstrate installed.

Depends: mobilesubstrate, firmware (>= 8.0)

* Version field is used to describe the version of the deb package; it can be changed on demand.
* Architecture field is used to describe the target device architecture, do not change it.
* Description field is used to give a brief introduction of the deb package; it can be changed on demand.
* Maintainer field is used to describe the maintainer of the deb package, say, all deb packages on TheBigBoss are maintained by BigBoss instead of the author. This field can be changed on demand.
* Author field is used to describe the author of the tweak, which is different from the maintainer. It can be changed on demand.
* Section field is used to describe the program type of the deb package, don't change it.

There are still some other fields in control file, but the above fields are enough for Theos projects. For more information, please refer to the official site of debian, <http://www.debian.org/doc/debian-policy/ch-controlfields.html>, or control files in other deb packages. It’s worth mentioning that Theos will further edit control file when packaging:

Package: com.iosre.iosreproject

Name: iOSREProject

Depends: mobilesubstrate

Architecture: iphoneos-arm

Description: An awesome MobileSubstrate tweak!

Maintainer: snakeninny

Author: snakeninny

Section: Tweaks

Version: 0.0.1-1

Installed-Size: 104

During editing, Theos changes the Version field to indicate packaging times; adds an Installed-Size field to indicate the size of the package. This size may not be exactly the same to the actual size, but don't change it.

The information of control file will show in Cydia directly, as shown in figure 3-5:

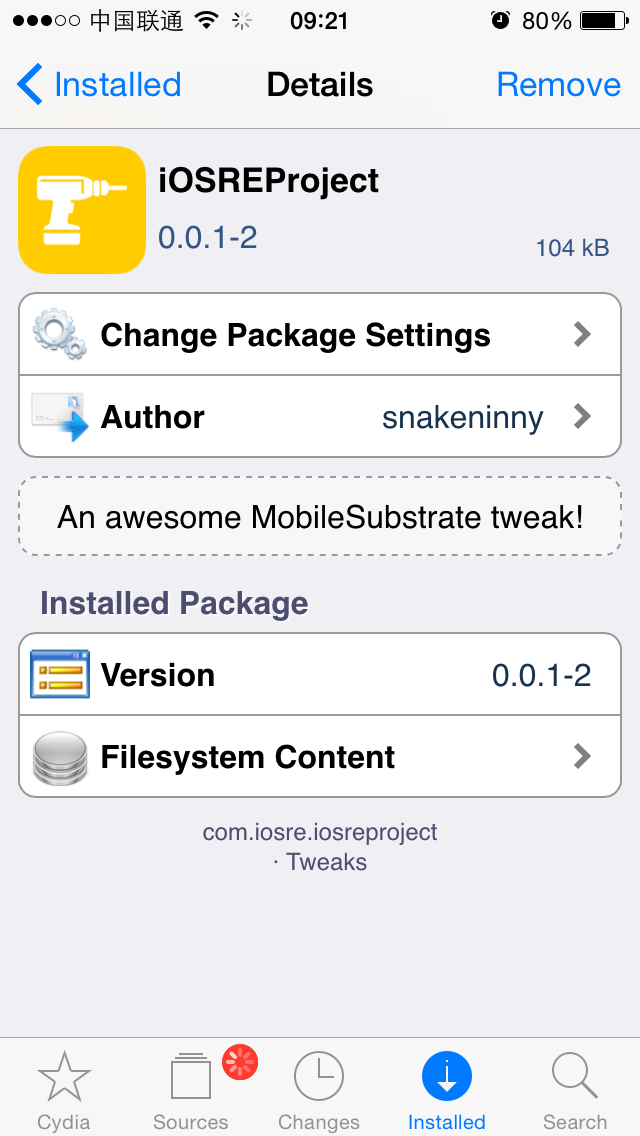


Figure 3- 5 Control informaton in Cydia

1. iOSREProject.plist

This plist file has the similar function to Info.plist of an App, which records some configuration information. Specifically in a tweak, it describes the functioning scope of the tweak. It can be edited with plutil or Xcode.

iOSREProject.plist consists of a “Root” dictionary, which has a key named "Filter", as shown in figure 3-6:

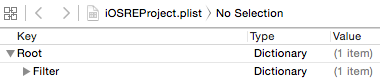


Figure 3- 6 iOSREProject.plist

There’s a series of arrays under “Filter”, which can be categorized into 3 types.

* “Bundles” specifies several bundles as the tweak's targets, as shown in figure 3-7.



Figure 3- 7 Bundles

According to figure 3-7, this tweak targets 3 bundles, i.e. SMSNinja, AddressBook.framework and SpringBoard.

* “Classes” specifies several classes as the tweak's targets, as shown in figure 3-8.

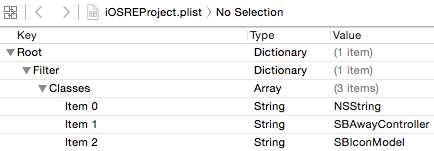


Figure 3- 8 Classes

According to figure 3-8, this tweak targets 3 classes, i.e. NSString, SBAwayController and SBIconModel.

* “Executables” specifies several executables as the tweak's targets, as shown in figure 3-9.

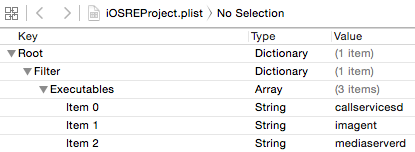


Figure 3- 9 Executables

According to figure 3-9, this tweak targets 3 executables, i.e. callservicesd, imagent and mediaserverd.

These 3 types can be used together, as shown in figure 3-10.

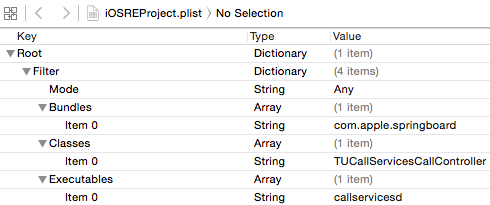


Figure 3- 10 A Mix-targeted tweak

Attention, when there’re different kinds of arrays in “Filter”, we have to add an extra “Mode : Any” key-value pair.

#### 3. Compile + Package + Install

We’ve installed Theos, created our first tweak project via NIC, and gone over all project files. In the end, we must compile the tweak and install it on iOS to start experiencing “safe mode” again and again. Are you excited?

1. Compile

“make” command is used to compile Theos project. Just run “make” under our Theos project directory:

snakeninnysiMac:iosreproject snakeninny$ make

Making all for tweak iOSREProject...

Preprocessing Tweak.xm...

Compiling Tweak.xm...

Linking tweak iOSREProject...

Stripping iOSREProject...

Signing iOSREProject...

From the output, we know Theos has finished preprocessing, compiling, linking, stripping and signing. After that, an “obj” folder appears in the current folder.

snakeninnysiMac:iosreproject snakeninny$ ls -l

total 32

-rw-r--r-- 1 snakeninny staff 262 Dec 3 09:20 Makefile

-rw-r--r-- 1 snakeninny staff 0 Dec 3 11:28 Tweak.xm

-rw-r--r-- 1 snakeninny staff 223 Dec 3 09:05 control

-rw-r--r--@ 1 snakeninny staff 175 Dec 3 09:48 iOSREProject.plist

drwxr-xr-x 5 snakeninny staff 170 Dec 3 11:28 obj

lrwxr-xr-x 1 snakeninny staff 11 Dec 3 09:05 theos -> /opt/theos

There is a .dylib file in it:

snakeninnysiMac:iosreproject snakeninny$ ls -l ./obj

total 272

-rw-r--r-- 1 snakeninny staff 33192 Dec 3 11:28 Tweak.xm.b1748661.o

-rwxr-xr-x 1 snakeninny staff 98784 Dec 3 11:28 iOSREProject.dylib

It's the core of our tweak.

1. Package

Theos uses "make package" command to pack Theos projects. In fact, “make package” executes “make” and “dpkb-deb” in sequence to finish its job.

snakeninnysiMac:iosreproject snakeninny$ make package

Making all for tweak iOSREProject...

Preprocessing Tweak.xm...

Compiling Tweak.xm...

Linking tweak iOSREProject...

Stripping iOSREProject...

Signing iOSREProject...

Making stage for tweak iOSREProject...

dm.pl: building package `com.iosre.iosreproject' in `./com.iosre.iosreproject\_0.0.1-7\_iphoneos-arm.deb'.

“make package” has created a “com.iosre.iosreproject\_0.0.1-7\_iphoneos-arm.deb” file, which is ready to be published.

There is another important function of “make package” command. After executing this command, besides “obj” folder, another “\_” folder is also created as shown below.

snakeninnysiMac:iosreproject snakeninny$ ls -l

total 40

-rw-r--r-- 1 snakeninny staff 262 Dec 3 09:20 Makefile

-rw-r--r-- 1 snakeninny staff 0 Dec 3 11:28 Tweak.xm

drwxr-xr-x 4 snakeninny staff 136 Dec 3 11:35 \_

-rw-r--r-- 1 snakeninny staff 2396 Dec 3 11:35 com.iosre.iosreproject\_0.0.1-7\_iphoneos-arm.deb

-rw-r--r-- 1 snakeninny staff 223 Dec 3 09:05 control

-rw-r--r--@ 1 snakeninny staff 175 Dec 3 09:48 iOSREProject.plist

drwxr-xr-x 5 snakeninny staff 170 Dec 3 11:35 obj

lrwxr-xr-x 1 snakeninny staff 11 Dec 3 09:05 theos -> /opt/theos

What’s this folder for? Open it, we can see 2 subfolders in it, namely “DEBIAN” and “Library”:

snakeninnysiMac:iosreproject snakeninny$ ls -l \_

total 0

drwxr-xr-x 3 snakeninny staff 102 Dec 3 11:35 DEBIAN

drwxr-xr-x 3 snakeninny staff 102 Dec 3 11:35 Library

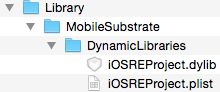
There is only an edited control file in “DEBIAN”.

snakeninnysiMac:iosreproject snakeninny$ ls -l \_/DEBIAN

total 8

-rw-r--r-- 1 snakeninny staff 245 Dec 3 11:35 control

The structure of “Library” directory is shown in figure 3-11:



Fire 3- 11 Library directory structure

If compared with the contents of deb package:

snakeninnysiMac:iosreproject snakeninny$ dpkg -c com.iosre.iosreproject\_0.0.1-7\_iphoneos-arm.deb

drwxr-xr-x snakeninny/staff 0 2014-12-03 11:35 ./

drwxr-xr-x snakeninny/staff 0 2014-12-03 11:35 ./Library/

drwxr-xr-x snakeninny/staff 0 2014-12-03 11:35 ./Library/MobileSubstrate/

drwxr-xr-x snakeninny/staff 0 2014-12-03 11:35 ./Library/MobileSubstrate/DynamicLibraries/

-rwxr-xr-x snakeninny/staff 98784 2014-12-03 11:35 ./Library/MobileSubstrate/DynamicLibraries/iOSREProject.dylib

-rw-r--r-- snakeninny/staff 175 2014-12-03 11:35 ./Library/MobileSubstrate/DynamicLibraries/iOSREProject.plist

And the files of iOSREProject seen in Cydia, as shown in figure 3-12.

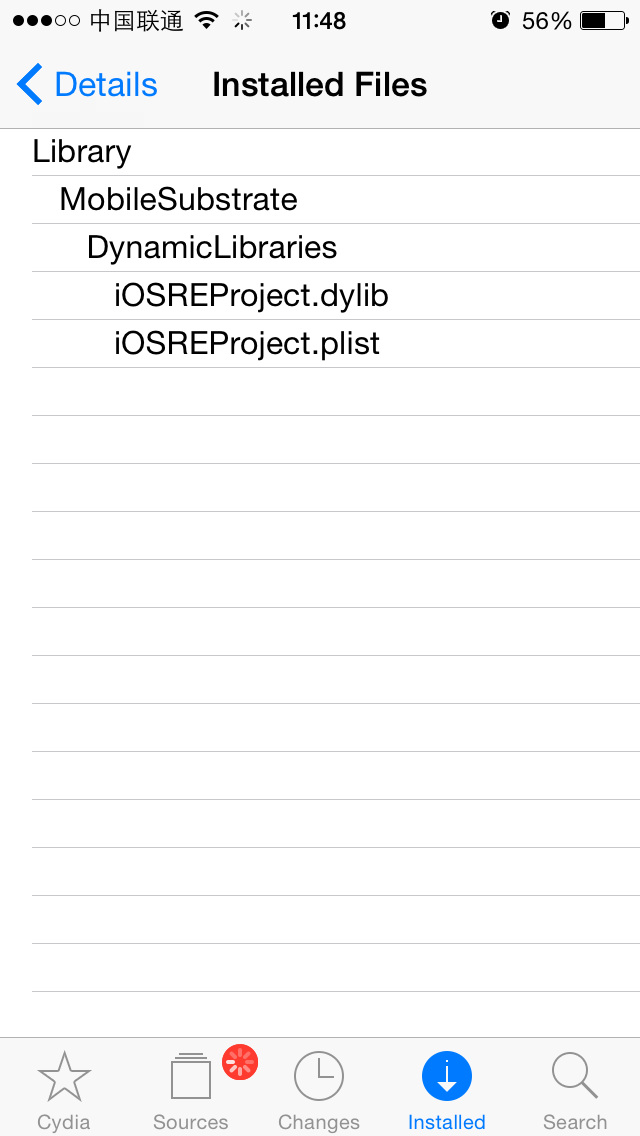


Figure 3-13 iOSREProject files

We can see that they have the same directory structures, and you may have already guessed that this deb package is simply a combination of “DEBIAN” which contains debian information, and “Library” which contains the actual files. In fact, we can also create a “layout” folder under the current project directory before packaging and installing the project on iOS. In this way, all files in “layout” will be extracted to the same positions of iOS filesystem (“layout” mentioned here acts as root directory, i.e. “/” on iOS), enhancing the functionality of deb packages lot. Let’s take an example to see the magic of “layout”.

Go back to iOSREProject, input “make clean” and “rm \*.deb” in Terminal to restore the project to the original state:

snakeninnysiMac:iosreproject snakeninny$ make clean

rm -rf ./obj

rm -rf "/Users/snakeninny/Code/iosreproject/\_"

snakeninnysiMac:iosreproject snakeninny$ rm \*.deb

snakeninnysiMac:iosreproject snakeninny$ ls -l

total 32

-rw-r--r-- 1 snakeninny staff 262 Dec 3 09:20 Makefile

-rw-r--r-- 1 snakeninny staff 0 Dec 3 11:28 Tweak.xm

-rw-r--r-- 1 snakeninny staff 223 Dec 3 09:05 control

-rw-r--r--@ 1 snakeninny staff 175 Dec 3 09:48 iOSREProject.plist

lrwxr-xr-x 1 snakeninny staff 11 Dec 3 09:05 theos -> /opt/theos

Then create a new “layout” folder:

snakeninnysiMac:iosreproject snakeninny$ mkdir layout

And put some random empty files under “layout”:

snakeninnysiMac:iosreproject snakeninny$ touch ./layout/1.test

snakeninnysiMac:iosreproject snakeninny$ mkdir ./layout/Developer

snakeninnysiMac:iosreproject snakeninny$ touch ./layout/Developer/2.test

snakeninnysiMac:iosreproject snakeninny$ mkdir -p ./layout/var/mobile/Library/Preferences

snakeninnysiMac:iosreproject snakeninny$ touch ./layout/var/mobile/Library/Preferences/3.test

At last, run “make package” to pack, then copy the deb package to iOS, and install it via iFile. Now you can inspect files of iOSREProject in Cydia, as shown in figure 3-13.

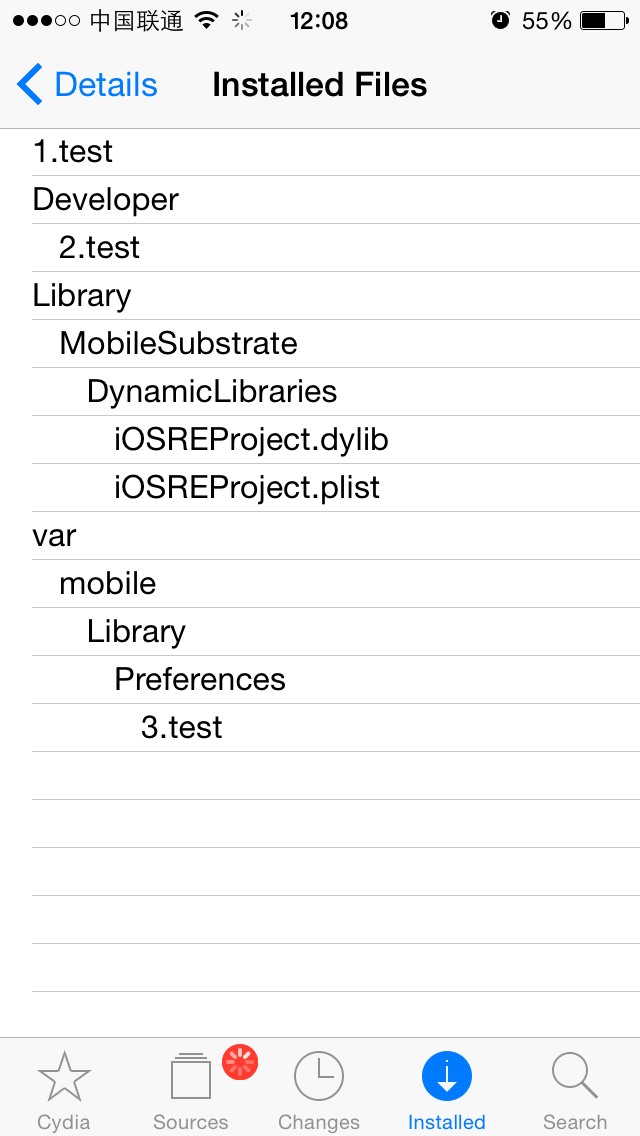


Figure 3-13 Installed files of iOSREProject

As we can see, all the files except “DEBIAN” are extracted to the same positions of iOS filesystem, all necessary subfolders are also created automatically. There are still many things about deb package we didn’t mention, please refer to <http://www.debian.org/doc/debian-policy> for more information.

1. Installation

Last but not least, we need to install this deb package on iOS. There are severals ways to install, but installation through GUI and installation through command line are two of the most typical installation methods. Most of you may think the GUI way is easier, well, let’s take a look at it first.

* Installation through GUI

This method is quite easy: First copy the deb package to iOS via iFunBox, then install it via iFile, and reboot iOS. All steps are operated on GUI, but there are too many interactions between human and device, we have to switch between PC and iPhone, which leads to inconvenience, hence is not suitable for tweak development.

* Installation through command line.

This method makes use of very simple ssh commands, which requires OpenSSH to be installed on jailbroken iOS. If you have no idea about what we were talking, go through the “OpenSSH” section in chapter 4 quickly to get some help. Let’s see how to install through command line now.

First, add your iOS IP to the first line of Makefile:

THEOS\_DEVICE\_IP = iOSIP

ARCHS = armv7 arm64

TARGET = iphone:latest:8.0

Then enter “make package install” to compile, package and install in one click:

snakeninnysiMac:iosreproject snakeninny$ make package install

Making all for tweak iOSREProject...

Preprocessing Tweak.xm...

Compiling Tweak.xm...

Linking tweak iOSREProject...

Stripping iOSREProject...

Signing iOSREProject...

Making stage for tweak iOSREProject...

dm.pl: building package `com.iosre.iosreproject:iphoneos-arm' in `./com.iosre.iosreproject\_0.0.1-15\_iphoneos-arm.deb'

install.exec "cat > /tmp/\_theos\_install.deb; dpkg -i /tmp/\_theos\_install.deb && rm /tmp/\_theos\_install.deb" < "./com.iosre.iosreproject\_0.0.1-15\_iphoneos-arm.deb"

root@iOSIP's password:

Selecting previously deselected package com.iosre.iosreproject.

(Reading database ... 2864 files and directories currently installed.)

Unpacking com.iosre.iosreproject (from /tmp/\_theos\_install.deb) ...

Setting up com.iosre.iosreproject (0.0.1-15) ...

install.exec "killall -9 SpringBoard"

root@iOSIP's password:

Among the above information, Theos has asked for the root password twice. Although it seems safe, it’s inconvenient. Fortunately, we can skip the input of password over and over by configuring the authorized\_keys on iOS, as follows:

1. Remove the entry of iOSIP in “/Users/snakeninny/.ssh/known\_hosts”.

Assume that your iOS IP address is iOSIP. Edit “/Users/snakeninny/.ssh/known\_hosts”, and locate the entry of iOSIP:

iOSIP ssh-rsa hXFscxBCVXgqXhwm4PUoUVBFWRrNeG6gVI3Ewm4dqwusoRcyCxZtm5bRiv4bXfkPjsRkWVVfrW3uT52Hhx4RqIuCOxtWE7tZqc1vVap4HIzUu3mwBuxog7WiFbsbbaJY4AagNZmX83Wmvf8li5aYMsuKeNagdJHzJNtjM3vtuskK4jKzBkNuj0M89TrV4iEmKtI4VEoEmHMYzWwMzExXbyX5NyEg5CRFmA46XeYCbcaY0L90GExXsWMMLA27tA1Vt1ndHrKNxZttgAw31J90UDnOGlMbWW4M7FEqRWQsWXxfGPk0W7AlA54vaDXllI5CD5nLAu4VkRjPIUBrdH5O1fqQ3qGkPayhsym3g0VZeYgU4JAMeFc3

Delete this entry.

1. Generate authorized\_keys.

Execute the following commands in Terminal:

snakeninnysiMac:~ snakeninny$ ssh-keygen -t rsa

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/snakeninny/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /Users/snakeninny/.ssh/id\_rsa.

Your public key has been saved in /Users/snakeninny/.ssh/id\_rsa.pub.

……

snakeninnysiMac:~ snakeninny$ cp /Users/snakeninny/.ssh/id\_rsa.pub ~/authorized\_keys

authorized\_keys will be generated under users home directory.

1. Configure iOS

Execute the following commands in Terminal:

FunMaker-5:~ root# ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/var/root/.ssh/id\_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /var/root/.ssh/id\_rsa.

Your public key has been saved in /var/root/.ssh/id\_rsa.pub.

……

FunMaker-5:~ root# logout

Connection to iOSIP closed.

snakeninnysiMac:iosreproject snakeninny$ scp ~/authorized\_keys root@iOSIP:/var/root/.ssh

The authenticity of host 'iOSIP (iOSIP)' can't be established.

RSA key fingerprint is 75:98:9a:05:a3:27:2d:23:08:d3:ee:f4:d1:28:ba:1a.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'iOSIP' (RSA) to the list of known hosts.

root@iOSIP's password:

authorized\_keys 100% 408 0.4KB/s 00:00

ssh into iOS again to see if any passwords are required. Now, “make package install” becomes “one time configuration, one click installation”, yay!

1. Clean

Theos provides a convenient command “make clean” to clean our project. It indeed excutes “rm -rf ./obj” and “rm -rf "/Users/snakeninny/Code/iosre/\_"” in turn, thereby removes folders generated by “make” and “make package”. Of course, you can further use “rm \*.deb” to remove all deb packages generated by “make package”.

### 3.2.4 An example tweak

The previous sections have introduced Theos almost thoroughly, although not all contents are covered, it is way enough for beginners. I have already talked so much about Theos without writing a single line of code, but we’re not done yet.

Now, I will take a simplest tweak to explain everything we’ve introduced. After installing this tweak, a UIAlertView will popup after each respring.

#### Create tweak project “iOSREGreetings” using Theos

Use the following commands to create iOSREGreetings project:

snakeninnysiMac:Code snakeninny$ /opt/theos/bin/nic.pl

NIC 2.0 - New Instance Creator

------------------------------

[1.] iphone/application

[2.] iphone/cydget

[3.] iphone/framework

[4.] iphone/library

[5.] iphone/notification\_center\_widget

[6.] iphone/preference\_bundle

[7.] iphone/sbsettingstoggle

[8.] iphone/tool

[9.] iphone/tweak

[10.] iphone/xpc\_service

Choose a Template (required): 9

Project Name (required): iOSREGreetings

Package Name [com.yourcompany.iosregreetings]: com.iosre.iosregreetings

Author/Maintainer Name [snakeninny]: snakeninny

[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]: com.apple.springboard

[iphone/tweak] List of applications to terminate upon installation (space-separated, '-' for none) [SpringBoard]:

Instantiating iphone/tweak in iosregreetings/...

Done.

#### Edit Tweak.xm

The edited Tweak.xm looks like this:

%hook SpringBoard

- (void)applicationDidFinishLaunching:(id)application

{

%orig;

UIAlertView \*alert = [[UIAlertView alloc] initWithTitle:@"Come to <http://bbs.iosre.com> for more fun!" message:nil delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil];

[alert show];

[alert release];

}

%end

#### Edit Makefile and control

The edited Makefile looks like this:

THEOS\_DEVICE\_IP = iOSIP

ARCHS = armv7 arm64

TARGET = iphone:latest:8.0

include theos/makefiles/common.mk

TWEAK\_NAME = iOSREGreetings

iOSREGreetings\_FILES = Tweak.xm

iOSREGreetings\_FRAMEWORKS = UIKit

include $(THEOS\_MAKE\_PATH)/tweak.mk

after-install::

install.exec "killall -9 SpringBoard"

The edited control looks like this:

Package: com.iosre.iosregreetings

Name: iOSREGreetings

Depends: mobilesubstrate, firmware (>= 8.0)

Version: 1.0

Architecture: iphoneos-arm

Description: Greetings from iOSRE!

Maintainer: snakeninny

Author: snakeninny

Section: Tweaks

Homepage: http://bbs.iosre.com

This tweak is rather simple. When [SpringBoard applicationDidFinishLaunching:] is called, SpringBoard finishes launching. We hook this method, carry out the original implementation via %orig, then display a custom UIAlertView. With this tweak, every time we relaunch SpringBoard, a UIAlertView pops up. Can you get it?

If you’re OK with this section so far, please enter “make package install” in Terminal. When the lock screen shows, you will see the magic as shown in figure 3-14:

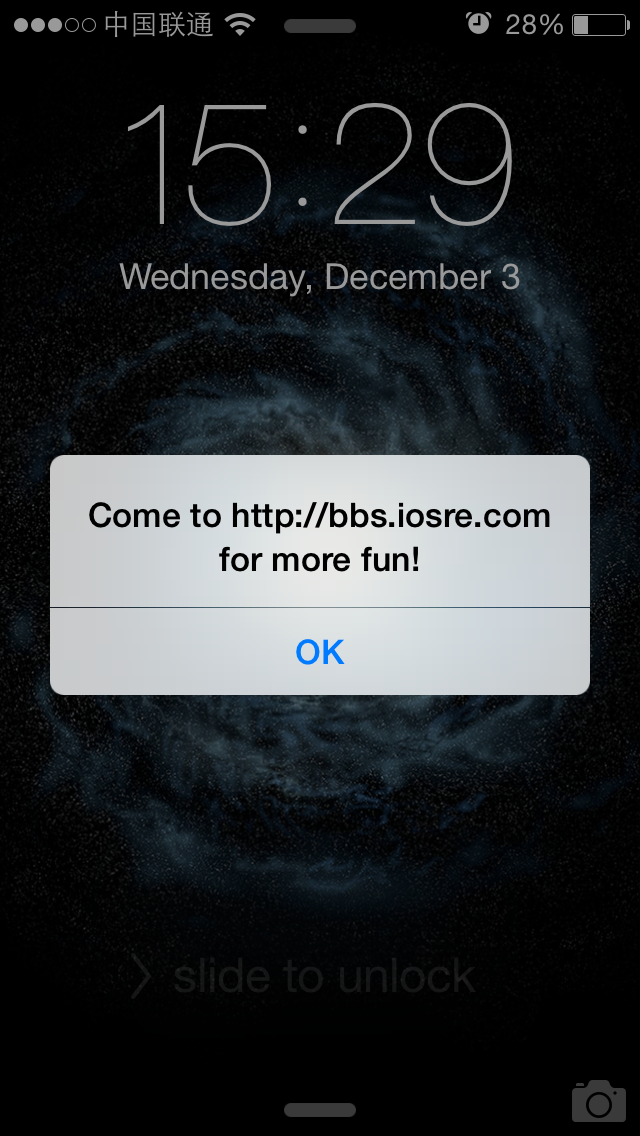


Figure 3- 14 Our first tweak

Yes, with just some tiny modifications, the behaviors of Apps are changed. Now, iOS is opening its long closed door to us…

Because of Theos, it’s never been easier to modify a closed-source App. As we have already mentioned, with the increase of App sizes, class-dump produces a greater amount of headers. It has became much more difficult to locate our targets than pure coding. Facing thousands lines of code, if there are no other tools to aid our analysis, reverse engineering would be a nightmare. Now, it’s showtime of these auxiliary analysis tools.

## 3.3 Reveal



图3- 15 Reveal

Reveal, as shown in figure 3-15, is a UI analysis tool by ITTY BITTY, enabling us to see the view hierarchy of an App intuitively. The official purpose of Reveal is to “See your application's view hierarchy at runtime with advanced 2D and 3D visualisations”, but as reverse engineers, seeing our own Apps’ view hierarchies is obviously not enough, we should be able to see other Apps' view hierarchies. Figure 3-16 shows the effect of seeing AppStore’s view hierarchy using Reveal.

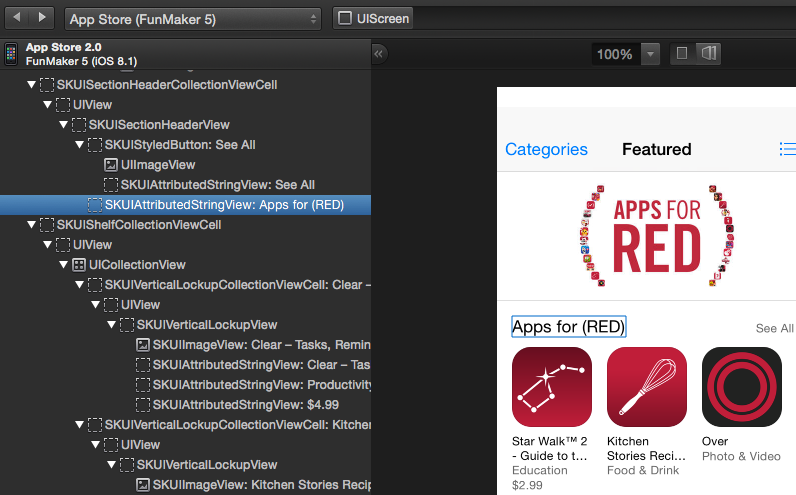


Figure 3-16 See the view hierarchy of AppStore

On the left side of Reveal, the UI layout of AppStore is presented as a tree, when choosing a control object, the corresponding UI element will be marked on the right side of Reveal in real time. At the same time, Reveal also parses the class name of this control object, as shown in figure 3-16, the class name of the selected object is SKUIAttributedStringView. To analyze the view hierarchies of other’s Apps, we need to make some configurations in Reveal.

#### Install Reveal Loader

Search and install Reveal Loader in Cydia, as shown in figure 3-17.

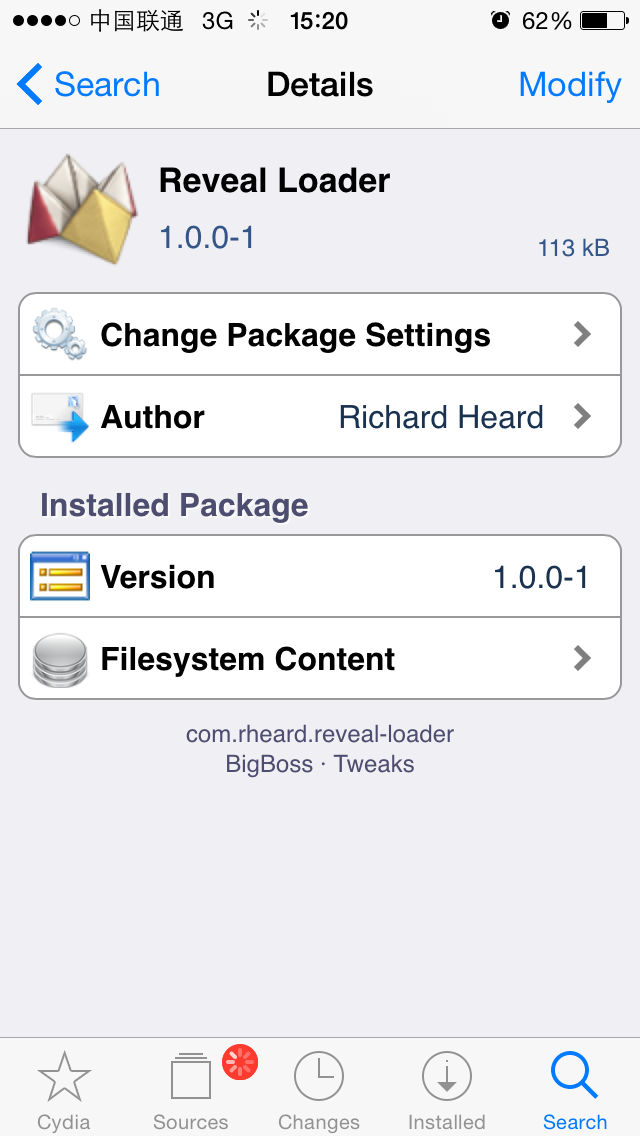


Figure 3-17 Reveal Loader

Remarkably, when installing Reveal Loader, it will download a necessary file libReveal.dylib from Reveal's official website automatically. If the network condition is not good, this file may not be downloaded successfully, and Reveal Loader is not fault tolerant to download failures. As a result, Cydia may stuck for a long time and stop responding. Therefore, after download completes, you’d better check whether there is a “RHRevealLoader” folder under the iOS directory “/Library/”.

FunMaker-5:~ root# ls -l /Library/ | grep RHRevealLoader

drwxr-xr-x 2 root admin 102 Dec 6 11:10 RHRevealLoader

If not, create one manually:

FunMaker-5:~ root# mkdir /Library/RHRevealLoader

Then open Reveal, click “Help” menu, choose “Show Reveal Library in Finder”, as shown in figure 3-18.

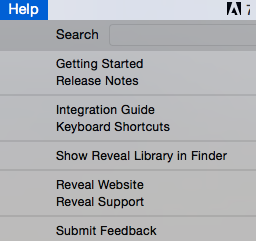


Figure 3- 18 Show Reveal Library in Finder

Then Finder will pop out just like figure 3-19.

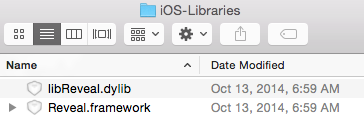


Figure 3- 19 libReveal.dylib

Copy libReveal.dylib to the RHRevealLoader folder through scp or iFunBox:

FunMaker-5:~ root# ls -l /Library/RHRevealLoader

total 3836

-rw-r--r-- 1 root admin 3927408 Dec 6 11:10 libReveal.dylib

By now, the installation of Reveal Loader completes.

#### Configure Reveal Loader

The configuration of Reveal Loader is inside the stock Settings App with the name “Reveal”, as shown in figure 3-20.

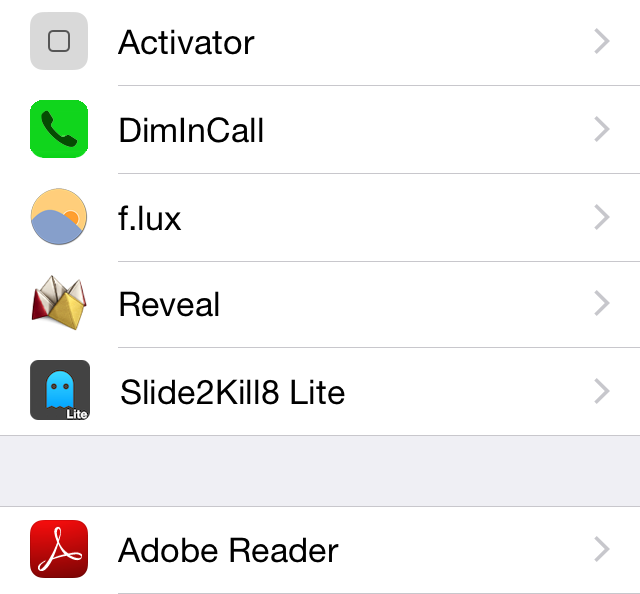


Figure 3- 20 Reveal Loader

Click “Reveal”, some declaration appears as shown in figure 3-21.

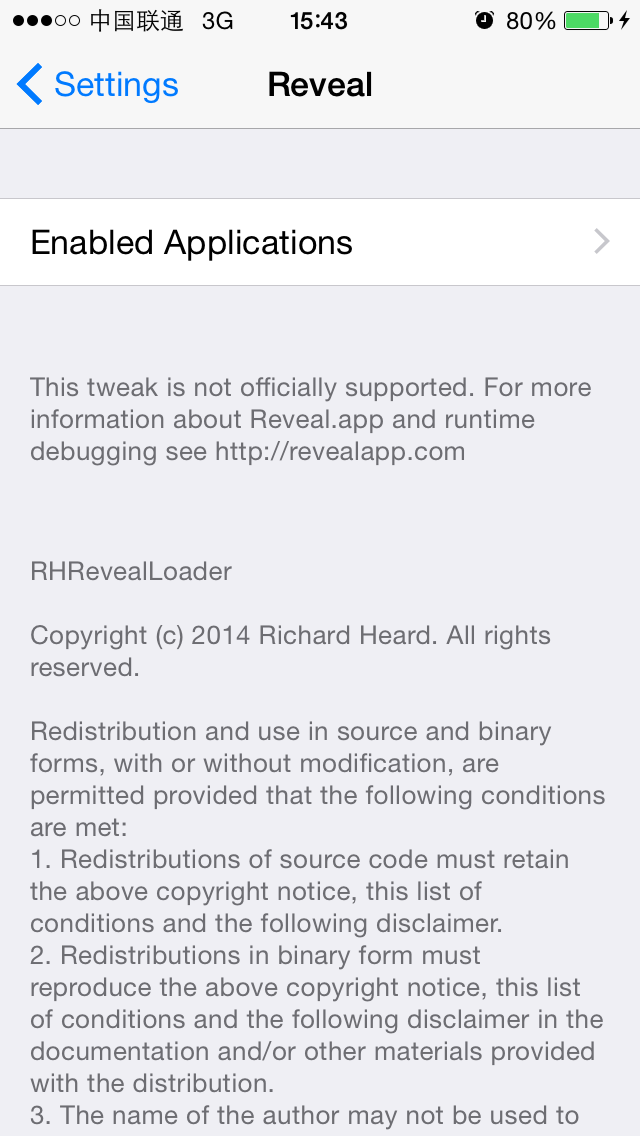


Figure 3-21 Declaration of Reveal Loader

Click “Enabled Applications” to enter the configuration view. Turn on the switch of the App you want to analyze. Here we’ve turned on AppStore and Calculator’s switches, as shown in figure 3-22.



Figure 3-22 configuration of Reveal Loader

That’s it. The configuration of Reveal Loader is simple and straightforward, isn’t it?

#### Use Reveal to see the view hierarchy of the target App

Everything is ready, now it’s the showtime of Reveal. First, one thing should be confirmed that OSX and iOS must be in the same LAN, then launch Reveal and relaunch the target App, i.e. if the target App is running, you need to terminate it first and run it again. The target App can be chosen from top left corner of Reveal. Wait a moment, Reveal will display the view hierarchy of the target App, as shown in figure 3-23.

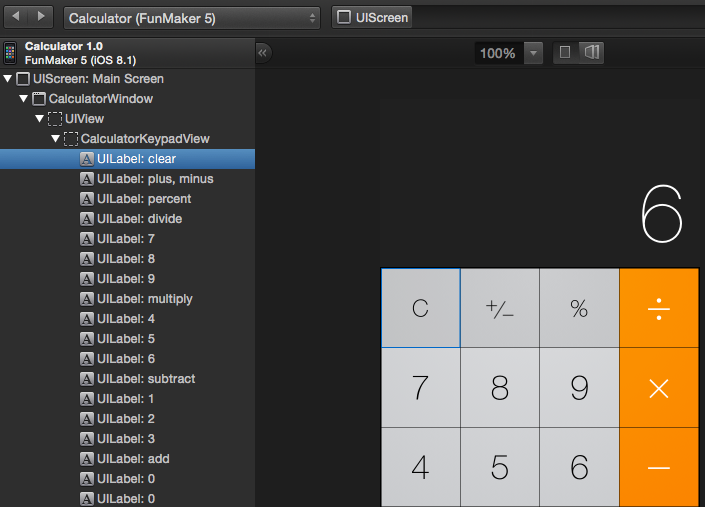


Figure 3-23 View hierarchy of Calculator

Reveal is not complicate and quite user-friendly. But in iOS reverse engineering, analysis on UI is not enough, Apps’ inner implementations under the hood are our final goals. From part 3 of this book, we will use recursiveDescription function, which is the “command line” version of Reveal, together with Cycript to find the corresponding code snippets of UI, then you will know the real power of iOS reverse engineering.

## 3.4 IDA

### Introduction to IDA

Even if you’ve never done any iOS reverse engineering before, you may have heard of IDA (The Interactive Disassembler), as shown in figure 3-24. For reverse engineers, IDA is so well-known that most of our daily work are tightly related to it. If class-dump can help us get the dots out of an App, then IDA can connect the dots to form a plane.

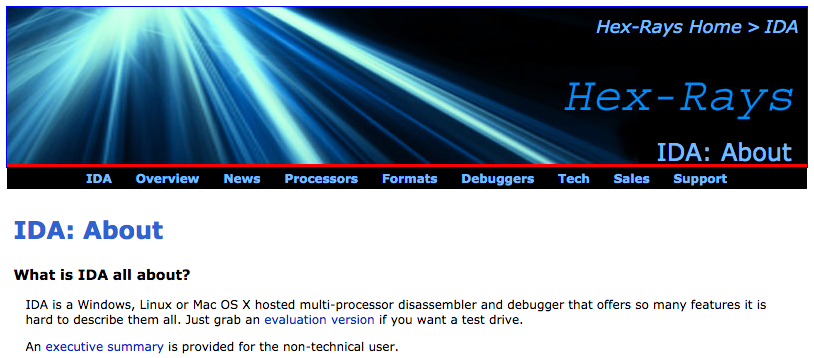


Figure 3-24 Official website of IDA

Generally speaking, IDA is a multi-processor disassembler and debugger fully supporting Windows, Linux and Mac OS X. It is so powerful that even the official site cannot give a complete function list.

To be honest, IDA is quite expensive for personal users. But the author is kind enough to offer a free evaluation version, which works well enough for beginners. It is convenient to download and install IDA, you can refer to <https://www.hex-rays.com/products/ida/index.shtml> for details.

### Use IDA

IDA will shortly display an “About” window after launch, as shown in figure 3-25.

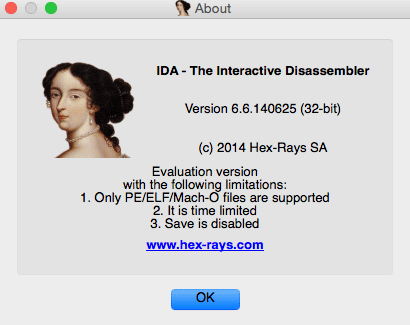


Figure 3- 25 IDA launch window

You can click “OK” or wait for a few seconds to close the window, after that you will see the main screen of IDA, as shown in figure 3-26.

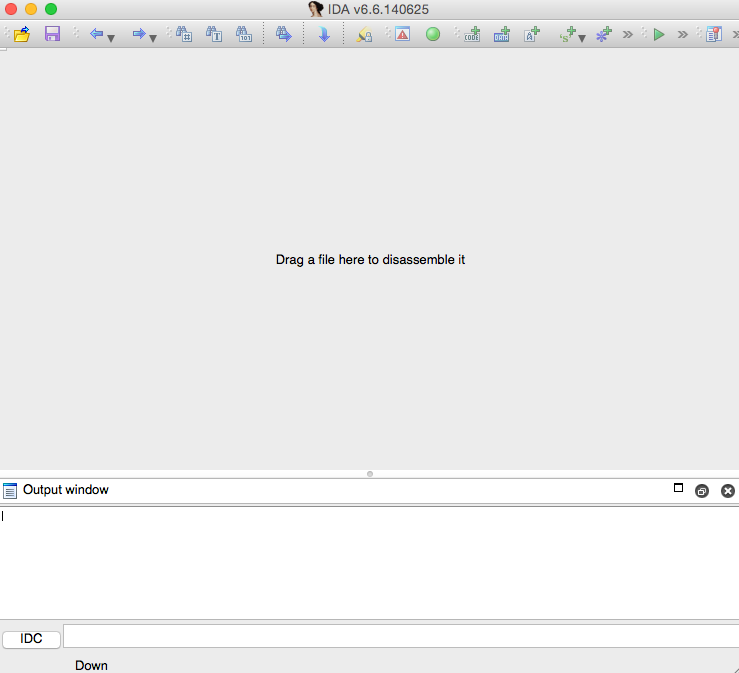


Figure 3-26 Main screen of IDA

In this screen, you don’t have to search for “Open File” in the menu and locate the file to be disassembled folder by folder, but just drag the target file to the gray zone with the placeholder “Drag a file here to disassemble it”. After opening the file, there is still something to be configured, as shown in figure 3-27.

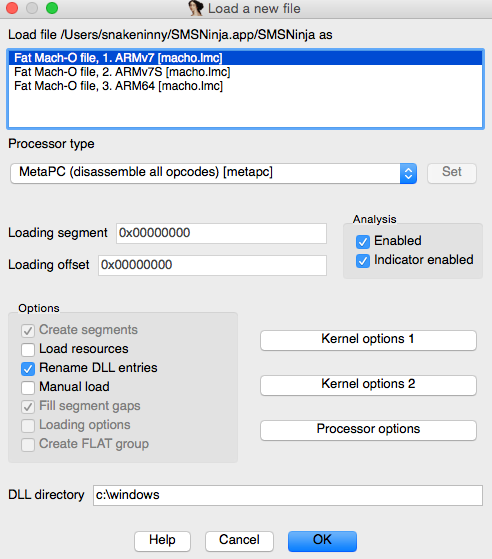


Figure 3-27 Initial configurations

There’s one thing to mention: For a fat binary (which refers to the binary that contains different instruction sets for the purpose of being compatible with different CPU architectures), the white frame in figure 3-27 will list several Mach-O files. I suggest you read table 4-1 to find the ARM type of your device. For example, my iPhone 5 corresponds to ARMv7S. If the ARM type of your device is not in the white frame, you should choose the backward compatible one, i.e. for ARMv7S devices, choose ARMv7S if there is ARMv7S in the list, otherwise choose ARMv7. This selection method handles 99% of all cases, if you happen to be the 1%, please come to <http://bbs.iosre.com>, we’ll solve the problem together.

Here, I’ve chosen ARMv7S, then click “OK”. Several windows will popup, just click “YES” or “OK” to close them, as shown in figure 3-28 and 3-29.

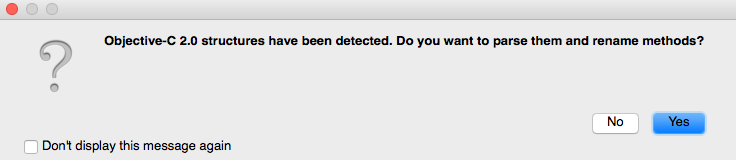


Figure 3-28 IDA launch option

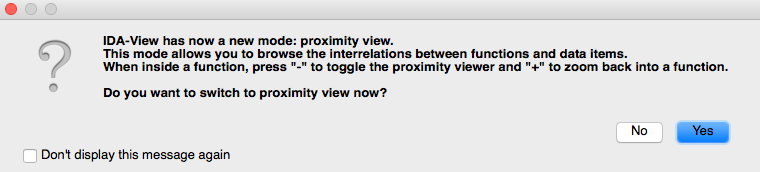


Figure 3-29 IDA launch option

Since we cannot save our configurations in the evaluation version of IDA, checking the box “Don’t display this message again” doesn’t work at all, it will still show in the next launch.

After clicking all the “OK” and “YES” buttons, the dazzling main screen shows up as in figure 3-30.

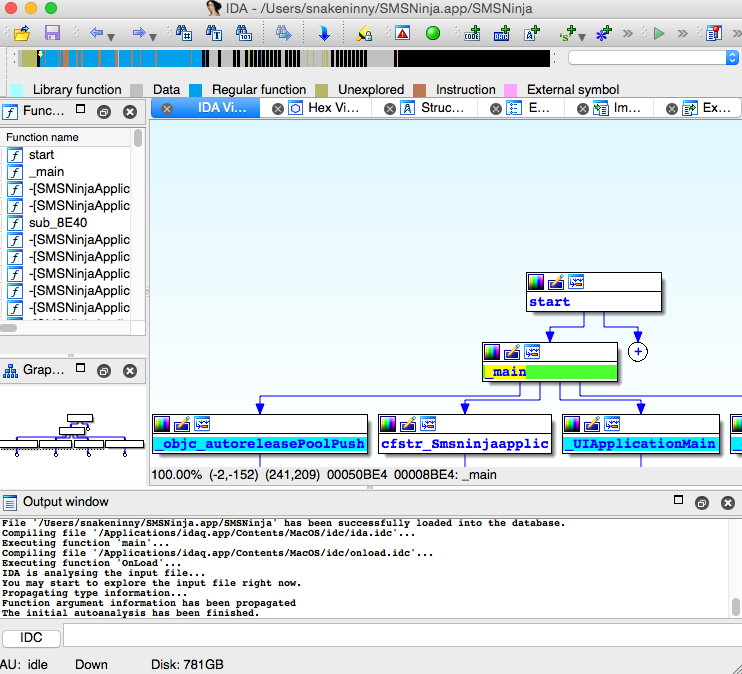


Figure 3-30 IDA main screen

When entering the screen in figure 3-30, you will see the progress bar at the top loading, the output window at the bottom printing the analysis progress. When the main color of the progress bar changes to blue, and the output window shows the message “The initial autoanalysis has been finished”, it indicates the initial analysis is completed.

At the beginning stage, IDA is mainly used for static analysis, the output window is quite useless, we can close it for now.

Now that there are two major windows, on the left is “Functions window” as shown in figure 3-31, on the right is “Main window” as shown in figure 3-32. Now, let’s take a look at them one by one.

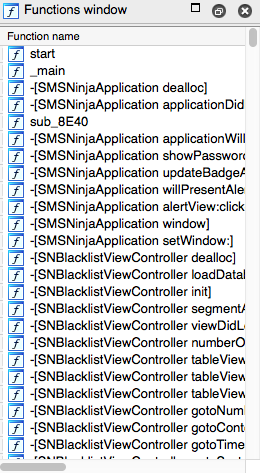


Figure 3-31 Functions window

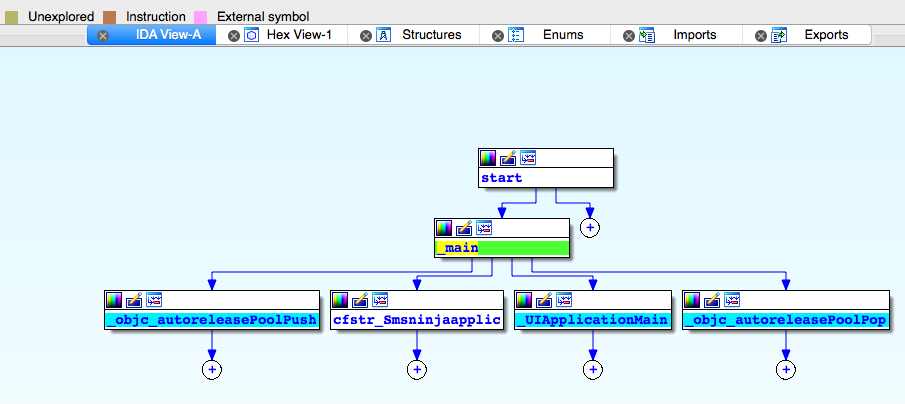


Figure 3-32 Main window

1. Functions window

As its name indicates, this window shows all functions (More accurately, Objective-C functions should be called methods, but we’re referring them to functions hereafter), double click one function name, the main window will show its implementation. When click “Search” menu of Function Window, a submenu will show up as figure 3-33.

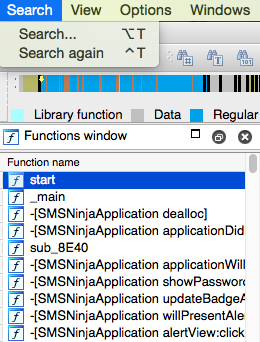


Figure 3-33 Search functions

Choose “Search…”, then type in what you want to search as shown in figure 3-34, to search for your specified string in all function names. When the string appears in several function names, you can click “Search again” to go through all of them. Of course, all above operations can be done by shortcuts.

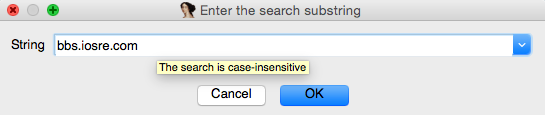


Figure 3-34 Search functions

The method names in functions window are the same as names exported by class-dump. Besides Objective-C methods, IDA lists all subroutines that we cannot get with class-dump. All class-dump contents are method names of Objective-C, it’s easy to learn and read for beginners; the names of subroutines are just combinations of “sub\_” and addresses, they don’t have any literal meaning, hence are hard to learn and read, freaking many rookies out. However, low-level iOS is implemented in C and C++, which generate subroutines rather than Objective-C methods. In this situation, class-dump is entirely defeated, our only choices are tools like IDA. If we want to go deeper into iOS, we must get familiar with IDA.

1. Main window

Most iOS developers who have never used IDA before are shocked by the “delirious” contents presented by main window. It seems a real mess for all beginners; some of them may close IDA immediately, and never open it again. This perplexed feeling is similar to the first time when you write code. In fact, it is like every project needs a main function, in iOS reverse engineering, we also need to specify the entry function that we are interested in. Double click this entry function in function window, main window will show the function body, then select main window and press space key, the main window will become much clearer and more readable as shown in figure 3-35.

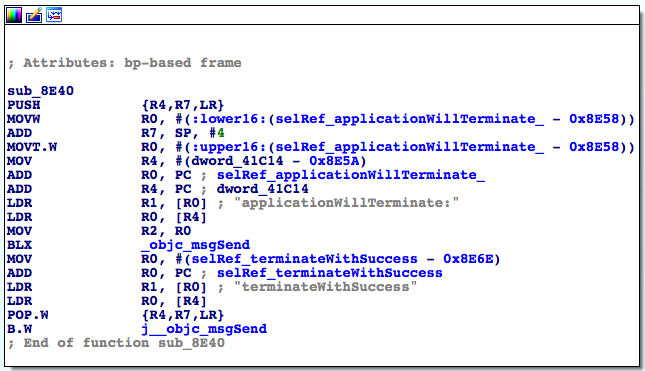


Figure 3- 35 Graph view

There are 2 display modes in main window, i.e. graph view and text view, which can be switched by space key. Graph view focuses on the logic; you can use control + mouse wheel on it to zoom in and out. Graph view provides intuitive visualization of the relationship among different subroutines. Execution flows of different subroutines are presented by lines with arrows. When there’s a conditional branch, subroutine that meets the condition will be connected with green line, otherwise with red line; for an unconditional branch, the next subroutine will be connected with blue line. For example, in figure 3-36, when the execution flow comes to the end of loc\_1C758, it judges whether R0 is equal to 0, if R0 != 0, the condition of BNE is satisfied, it will branch to the right, otherwise it will branch to the left. This is one difficult point of IDA; it will be explained again and again in the following examples.

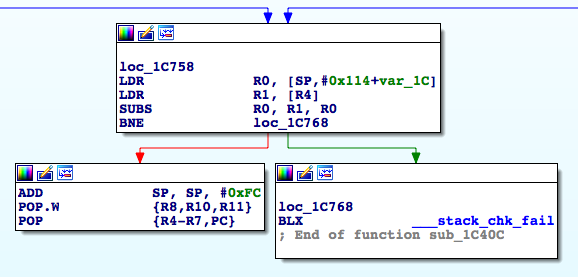


Figure 3- 36 Branches in IDA

Careful readers may have noticed that the fonts of IDA are colorful. In fact, different colors have different meanings, as shown in figure 3-37.



Figure 3-37 Color indication bar

When we choose a symbol, all the same symbols will be highlighted in yellow, making it convenient for us to track this symbol, as shown in figure 3-38.

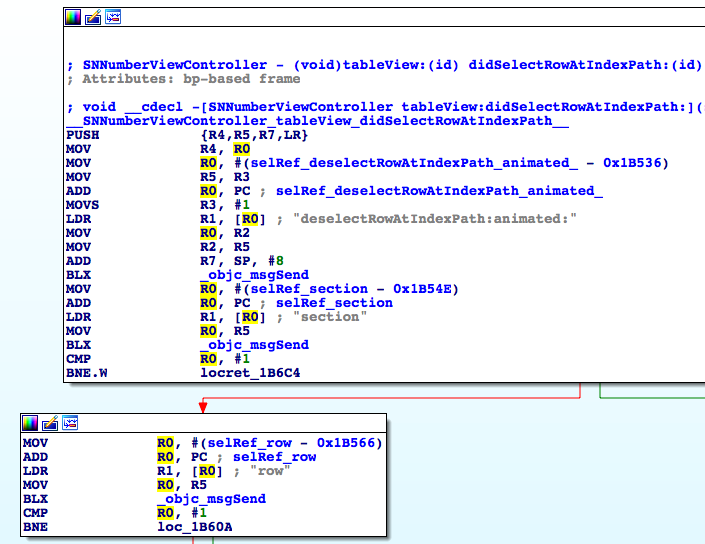


Figure 3-38 Symbol highlight

Double click a symbol to see its implementation as shown in figure 3-35. Right click a symbol to display a menu shown in figure 3-39.

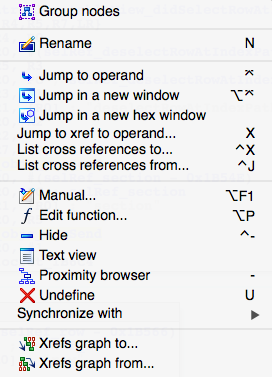


Figure 3-39 Right click on a symbol

Among the menu options, there is a very frequently used function “Jump to xref to operand…” with the shortcut X (meaning “cross”), click this option, all information explicitly cross referenced to this symbol will be displayed as shown in figure 3-40.

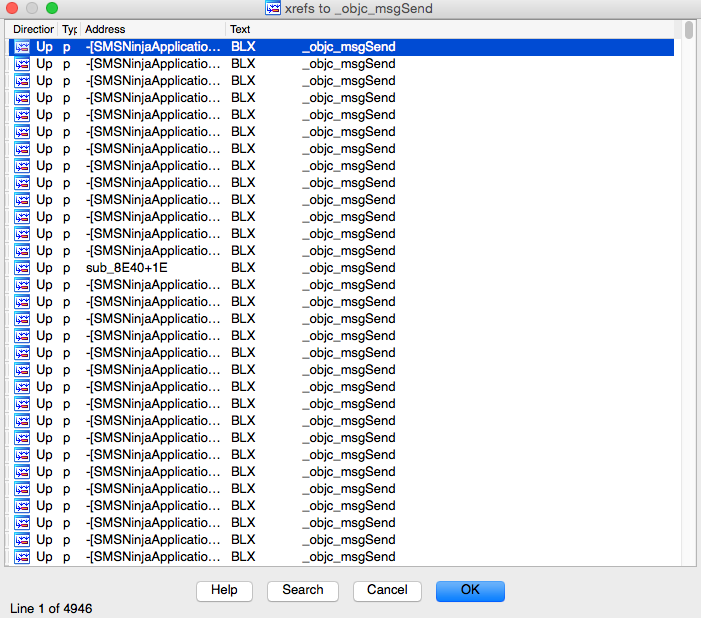


Figure 3- 40 Jump to xref to operand...

If you think this way is not straightforward and clear enough, yet prefer graph view, you can choose option “Xrefs graph to…”. However, if this symbol is cross-referenced too much, the graph view becomes a mess, just like figure 3-41 shows.

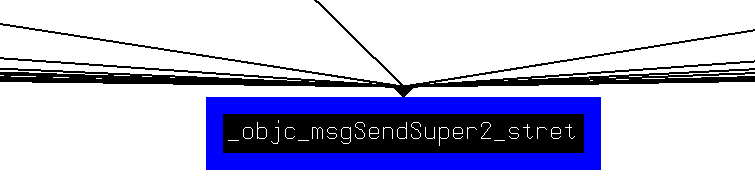


Figure 3-41 Xrefs graph to…

In figure 3-41, the irregular patterns in black are constructed by lines; lines are melting together on both sides. So we know the symbol \_objc\_msgSendSuper2\_stret is cross-referenced many times.

Relatively, if we choose “Xrefs graph from...” , it will show all symbols cross referenced by the symbol you choose, as shown in figure 3-42.

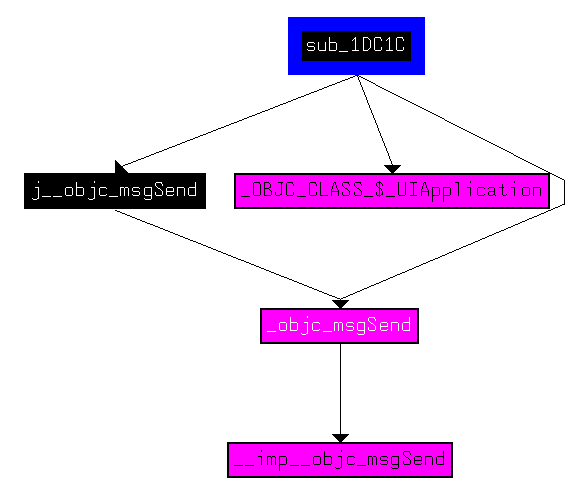


Figure 3-42 Xrefs graph from...

From figure 3-42 we know that sub\_1DC1C is a subroutine, it cross-references j\_\_objc\_msgSend, \_OBJC\_CLASS\_$\_UIApplication and \_objc\_msgSend explicitly, and \_objc\_msgSend further cross-references \_\_imp\_\_objc\_msgSend explicitly. Double click \_objc\_msgSend in main window, then double click \_\_imp\_\_objc\_msgSend, you will see it is from libobjc.A.dylib, as shown in figure 3-43.

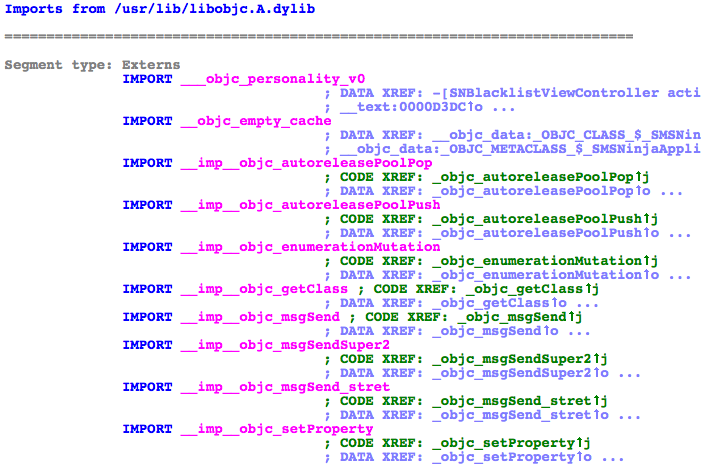


Figure 3-43 Tracking the source of external symbols

In most cases, when we discover an interesting symbol, we want to find every related clue. One clumsy but effective way is to select main window and click “Search” on the menu bar. A submenu is shown like figure 3-44.

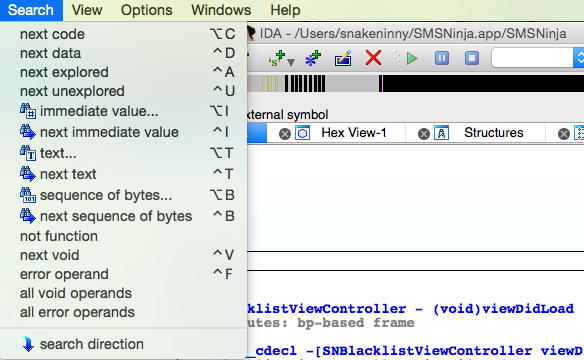


Figure 3-44 Search in Main window

Choose “text…”, a window will popup, as shown in figure 3-45.

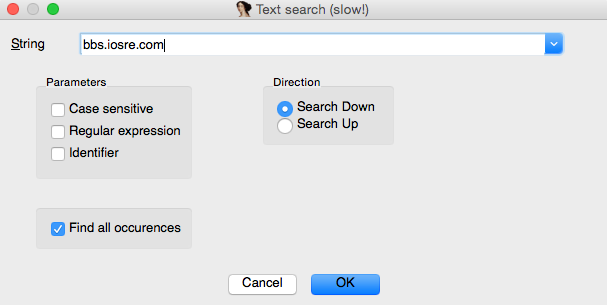


Figure 3-45 Text search

There’re other searching options available, you can check them out according to your situations. Then check “Find all occurences” and click “OK”. IDA will search the whole binary and show all the matching strings.

Graph view provides us with so many features; I’ve only introduced some common ones, proficiency in them ensures deeper research. Graph view is simple and clear, it’s easy to see the logic between different subroutines. As newbies, we mostly use graph view. When using LLDB for debugging, we’ll switch to text view to get the address of a symbol listed on the left side, as shown in figure 3-46.

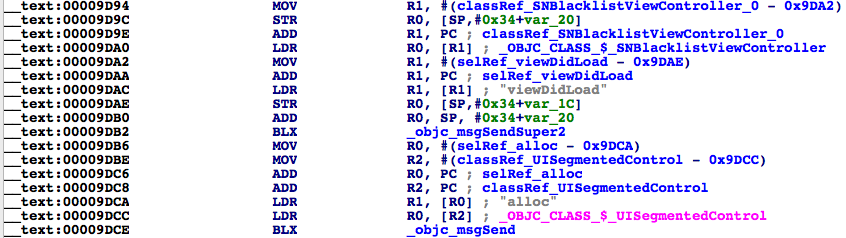


Figure 3-46 Text view

It should be noted that one bug of IDA will cause the incomplete display of a subroutine at the end of its graph view (For example, one subroutine has 100 lines of instructions but only displays 80 lines). When you are suspicious about instructions in graph view, just switch to text view to see whether some code is missing. This bug occurs by very little chance, if you happen to encounter it unfortunately, welcome to <http://bbs.iosre.com> for discussion and solution.

### An analysis example of IDA

Having introduced so many features of IDA, now I will use a simple example to show the real power of IDA. Jailbreak users know, Cydia will suggest us “Restart SpringBoard” when a tweak finishes installation. How does Cydia perform a respring? Please go through section 3.5 quickly and copy “/System/Library/CoreServices/SpringBoard.app/SpringBoard” from iOS to OSX using iFunBox, then open it with IDA. When the initial analysis is finished, search “relaunchSpringBoard” in function window, double click it to jump to its function body, as shown in figure 3-47.

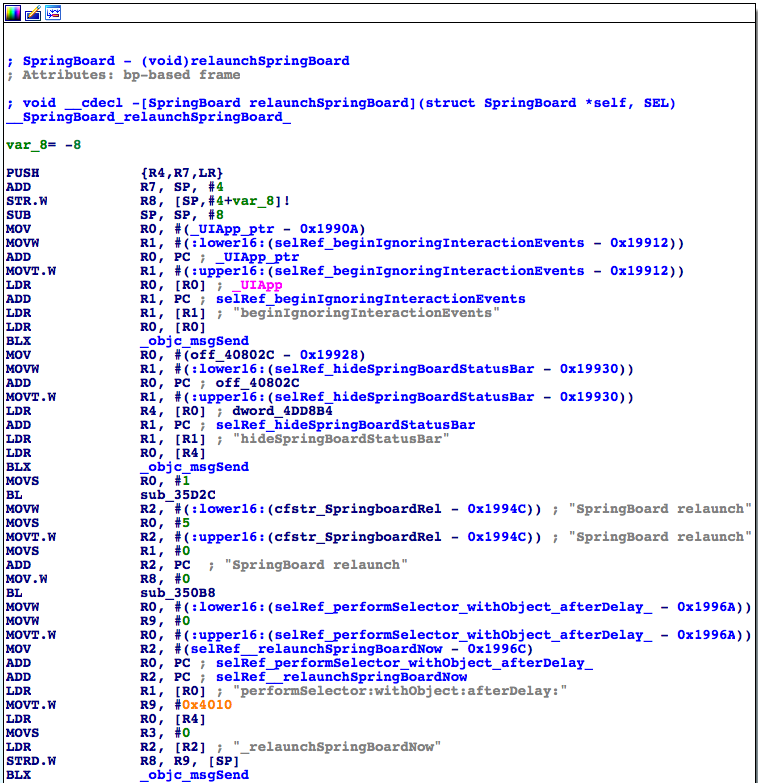


Figure 3- 47 [SpringBoard relaunchSpringBoard]

As we can see in figure 3-47, this method’s implementation is simple and clear. According the execution flow from top to bottom, firstly it calls beginIgnoringInteractionEvents to ignore all user interaction events; secondly, it calls hideSpringBoardStatusBar to hide the status bar in SpringBoard, then it executes two subroutines, they are sub\_35D2C and sub\_350B8. Now, double click sub\_35D2C to jump to its implementation, as shown in figure 3-48.

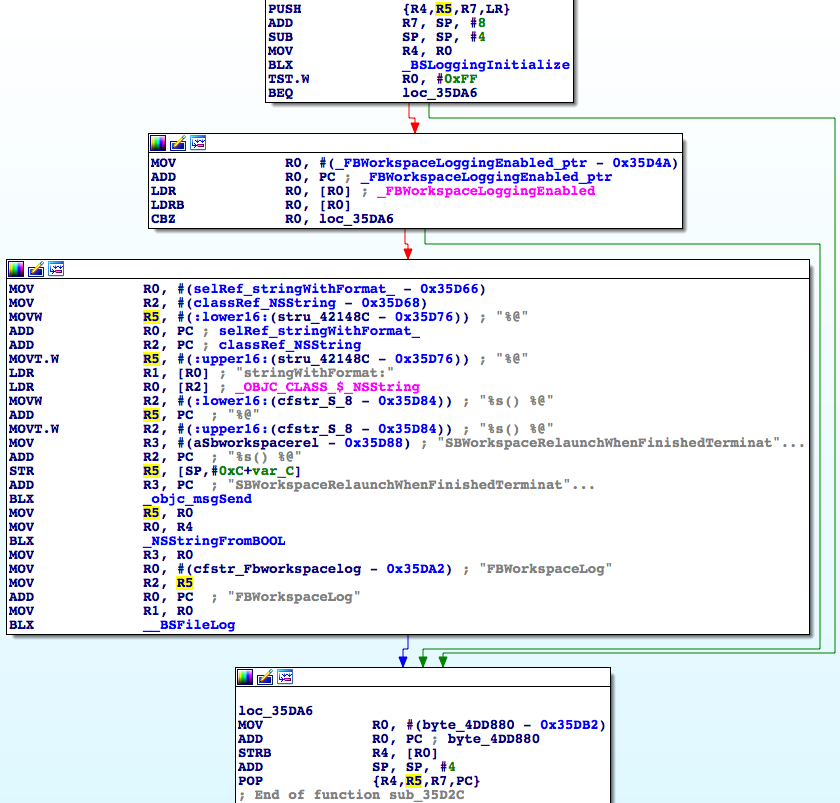


Figure 3- 48 sub\_35D2C

In figure 3-48, “log” appears a lot: First “initialize”, then check whether something is “enabled”, at last “log” something. From those keywords, we can guess that this subroutine is used for logging respring related operations, it has nothing to do with the essential function of respring. Click the blue back button of IDA menu bar (as shown in figure 3-49), or just press ESC, to go back to the implementation of “relaunchSpringBoard” and continue our analysis.



Figure 3-49 Back button

Double click sub\_350B8 to jump to figure 3-50.

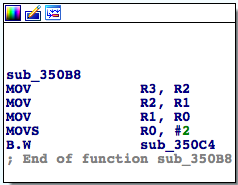


Figure 3- 50 sub\_350B8

We know from figure 3-50 that this subroutine is just preparing for calling sub\_350C4. Double click sub\_350C4 to jump to its implementation, you will find the top half of sub\_350C4 looks very similar to sub\_35D2C as shown in figure 3-48, which only does some logging job. But what’s different is that sub\_350C4 additionally does something else, as shown in figure 3-51.

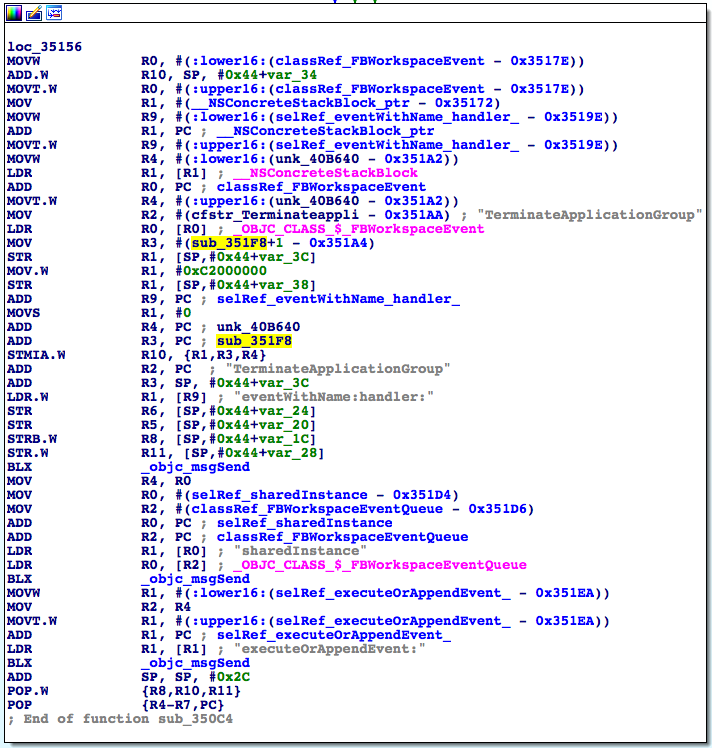


Figure 3-51 sub\_350C4

Now that we know little about assembly language, but from the literal meaning of these keywords, it can be concluded that the function of this subroutine is to generate an event named “TerminateApplicationGroup”, specify sub\_351F8 to be the handler of it, and then append this event to a queue for sequential execution, thus close all Apps by this way. This makes sense: Before a mall closes, we need to close all its shops; before respring, we need to close all Apps. Let’s go to sub\_351F8 to see its implementation, as shown in figure 3-52.

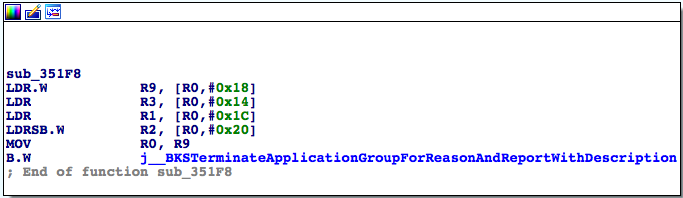


Figure 3-52 sub\_351F8

We can infer from the name of BKSTerminateApplicationGroupForReasonAndReportWithDescription that sub\_351F8 acts as a terminator, which just proves our analysis of sub\_350C4. Go back to the function body of relaunchSpringBoard, our analysis comes to the end: \_relaunchSpringBoardNow is called to finish respring.

Neither do we need to read assembly code nor be familiar with calling conventions, we’ve finished this reverse engineering task from scratch, right? However, we should not take much credit, kudos to IDA! In most cases, IDA plays the same role to the above example; you only need to be patient reading every line of code, it won’t be long before you feel the beauty of reverse engineering.

The usage of IDA is much much more complicated than I have introduced in this book, if you have any questions about it, please discuss with us on http://bbs.iosre.com, or take The IDA Pro Book as reference.

### 3.5 iFunBox

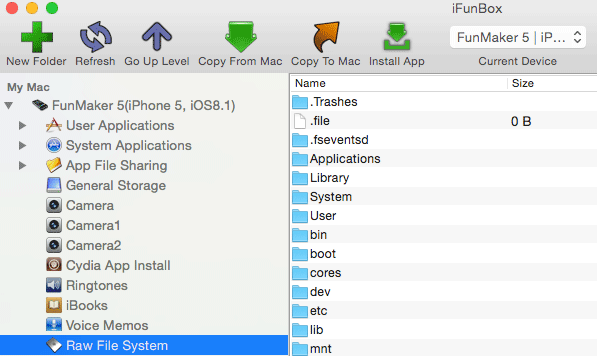


Figure 3-53 iFunBox

iFunBox (as shown in figure 3-53) is an evergreen iOS file management tool on Windows/OSX. In this book, we mainly make use of its file transfer feature. One thing to mention is that we must install “Apple File Conduit 2” (or AFC2 in short, as shown in figure 3-54) on iOS to browse the entire iOS file system, which is the prerequisite of the following operations in this book.

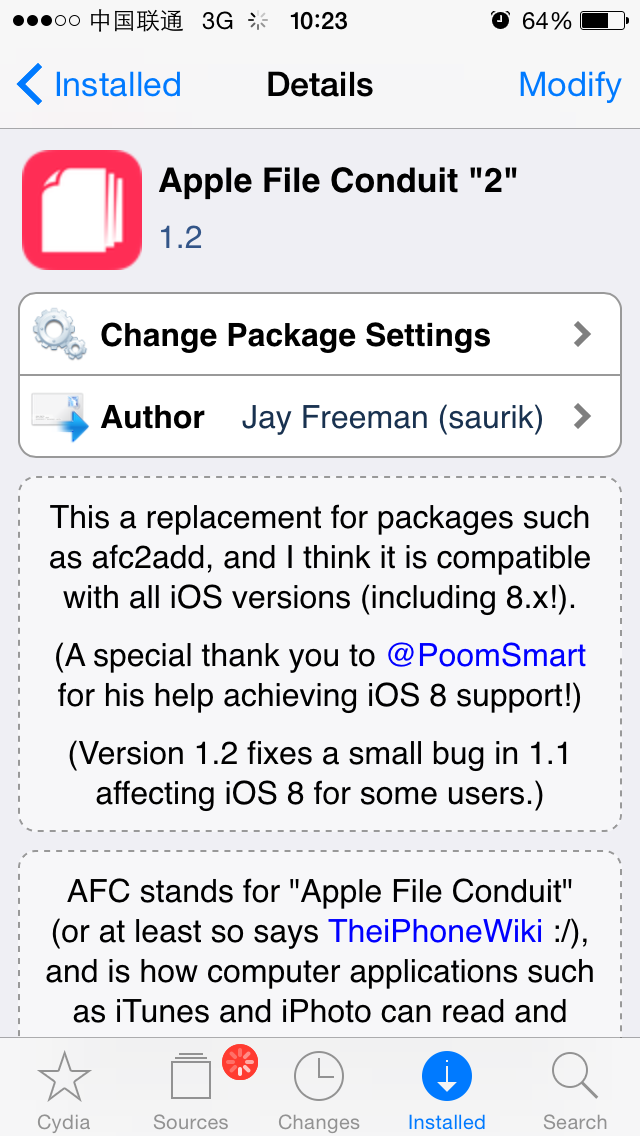


Figure 3-54 Apple File Conduit 2

### 3.6 dyld\_decache

After installing iFunBox and AFC2, most of you would be eager to start browsing the iOS filesystem to explore the secrets hidden in iOS. But soon you’ll discover that there are no library files under “/System/Library/Frameworks/” or “/System/Library/PrivateFrameworks/”. What’s going on?

From iOS 3.1, many library files including frameworks are combined into a big cache, which is located in “/System/Library/Caches/com.apple.dyld/ dyld\_shared\_cache\_armx” (i.e. dyld\_shared\_cache\_armv7, dyld\_shared\_cache\_armv7s or dyld\_shared\_cache\_arm64). We can use dyld\_decache by KennyTM to extract the separate binaries from this cache, which guarantees that the files we analyze are right from iOS, avoiding the possibility that static and dynamic analysis targets mismatch each other. More about this cache, please refer to DHowett’s blog at <http://blog.howett.net/2009/09/cache-or-check/>.

Before using dyld\_decache, please use iFunBox (not scp) to copy “/System/Library/Caches/com.apple.dyld/dyld\_shared\_cache\_armx” from iOS to OSX, then download dyld\_decache from [https://github.com/downloads/kennytm/Miscellaneous/dyld\_decache[v0.1c].bz2](https://github.com/downloads/kennytm/miscellaneous/dyld_decache%5Bv0.1c%5D.bz2) and grant execute permission to the decompressed executable:

snakeninnysiMac:~ snakeninny$ chmod +x /path/to/dyld\_decache\[v0.1c\]

Then extract binaries from the cache:

snakeninnysiMac:~ snakeninny$ /path/to/dyld\_decache\[v0.1c\] -o /where/to/store/decached/binaries/ /path/to/dyld\_shared\_cache\_armx

0/877: Dumping '/System/Library/AccessibilityBundles/AXSpeechImplementation.bundle/AXSpeechImplementation'...

1/877: Dumping '/System/Library/AccessibilityBundles/AccessibilitySettingsLoader.bundle/AccessibilitySettingsLoader'...

2/877: Dumping '/System/Library/AccessibilityBundles/AccountsUI.axbundle/AccountsUI'...

……

All the binaries are extracted into “/where/to/store/decached/binaries/”. After that, binaries to be reversed are scattered on both iOS and OSX, which leads to inconvenience. So we suggest you copy iOS filesystem to OSX with scp, a tool to be introduced in the next chapter.

## 3.7 Conclusion

This chapter focuses on 4 tools, which are class-dump, Theos, Reveal and IDA. Familiarity with them is the prerequisite of iOS reverse engineering.