# 第三部分 理 论 篇

当你从第一部分了解了iOS应用逆向工程的基本概念，并跟着第二部分把玩过一些逆向工具之后，就已经具备了iOS应用逆向工程的基本知识。当你完成了书上的例子之后，接下来可能会有一种无从下手的感觉，不知道下一步该做些什么。逆向工程是一门需要动手的学问，而从哪里动手、该怎么动手，其实是有套路可循的。第5章和第6章分别尝试从Objective-C和ARM的角度出发，用iOS应用逆向工程独有的理论知识把我们介绍过的工具串联起来，总结出一套通用的逆向工程方法论。这就开始吧！

# 第5章 Objective-C相关的iOS逆向工程

Objective-C语言是一门面向对象的高级语言，想必大家都能较为熟练地掌握它的基本用法，在逆向工程的入门阶段采用Objective-C语言有助于大家更平稳地从App开发进阶到逆向工程。幸运地是，iOS采用的文件格式Mach-O中包含了足够多的原始数据，让我们能够用class-dump等工具还原出二进制文件的头文件；有了这些信息，我们就可以开始Objective-C级别的逆向工程了，而撰写 tweak 无疑是这个阶段最受欢迎的项目，下面就从它开始吧！

## 5.1 tweak在Objective-C中的工作方式

在第3章中介绍Theos时，已经介绍了tweak的概念。依据维基百科的定义，tweak 指的是对电子系统进行轻微调整来增强其功能的工具；在iOS中，tweak特指那些能够增强其他进程功能的dylib，是越狱iOS的最重要组成部分。

正是因为tweak的存在，越狱iOS用户才能依照自己的喜好打造独一无二的个性化系统，iOS开发者才有机会站在优秀软件的肩膀上为它们添砖加瓦，丰富它们的功能，而这些便利都是原版iOS和AppStore无法提供的。Cydia中最受欢迎的软件几乎全是创意各异的tweak（图5-1是Cydia中的tweak图标），如Activator、Barrel、SwipeSelection等。一般来说，一个tweak的核心是各种“hook”，而绝大部分的hook是针对Objective-C方法的，那么tweak是如何工作的呢？

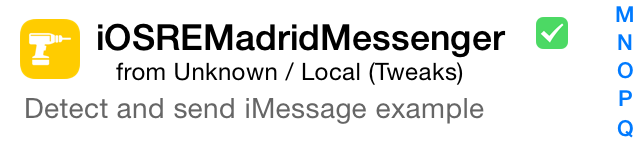


图5- 1 tweak图标

Objective-C是典型的面向对象语言。iOS是由一个个小的组件构成的，这些组件其实就是一个个对象。举个例子，iOS里的每个图标、每条信息和每张照片都是对象；除了这些用户能够看到的对象以外，还有很多对象一直在后台工作，为前台对象提供各种支持。例如有些对象负责跟苹果的服务器通信，有的对象负责读写文件。一个对象可以拥有其他对象，例如图标对象就拥有一个标签对象，用来显示这个图标代表的App名称。一般来说，每个对象都有自己存在的意义，工程师通过对不同对象的组合排序，实现不同的功能；在 Objective-C里，我们称对象的功能为“方法”，“方法”的具体行为称为“实现”。对象、方法和实现的关系，就是tweak大做文章的地方。

对象具备了某种功能，代码就可以发出指令“[object method]”，让一个对象去执行它的功能，也就是“调用对象的方法”。看到这里，可能有朋友会说，指令里的“对象”和“方法” 都是名词，而执行一个功能需要的不应该是一个动词吗?说得没错，我们还缺少一个动词，需要去“实现”这个方法。需要的动词已经出现了——“实现”指的是当某个方法得到调用时，iOS实际干了些什么，也就是执行了什么代码。在 Objective-C里，方法和实现的关系不是在编译时决定的，而是在运行时决定的。

在实际使用中，“[object method]”中的 method 不一定是一个名词，它也可能是一个动词。但仅凭简短的 [object method]，我们还是不知道要怎么实现这个方法，比如：“妈妈，接一下电话”，翻译成 Objective-C语言是“[ 妈妈 接电话 ]”，这里的对象是妈妈，方法是“接电话”，实现是“放下手里的炒菜铲子，把炉火关小一点，然后走到客厅去接电话”；“snakeninny，过来搬个东西”，翻译成Objective-C语言是“[snakeninny 搬东西]”，这里的对象是 snakeninny，方法是“搬东西”，实现是“停下手里的工作，从椅子上起来，走到老板的办公室里把一个箱子抬到楼下”。上面的两个例子如果没有“实现”的具体描述，即使调用了“方法”，“对象”也不知道具体该干嘛。“实现”是“方法”的释义，“方法”是词语，“实现”是词语的意义——这不就是词典吗？

随着时代的进步，词典的内容产生了变化，一些旧词语被赋予了新解释，“灌水”跟液体已经没有太大关系，“粉丝”也从一种食物变成了一类人。这些现象在 iOS 中也有体现，我们可以通过改变“实现”和“方法”的对应关系，赋予一个方法新的意义，从而达到更改对象功能的目的；只要别人在查询一个词语意义的时候参考了你修改过的词典，那么他的方法就有了新的实现，例如，笔者开发的LowPowerBanner（如图5-2所示）会在低电量时以横幅代替弹窗，提醒用户没电了——哈哈，那正是因为笔者更改了低电量提醒的实现，善意地欺骗系统“弹窗”的意思是“横幅”。



图5- 2 LowPowerBanner

笔者的另一个短信防火墙SMSNinja（如图 5-3 所示）能在收到垃圾短信时将其自动放进垃圾箱，这是通过更改iOS收到短信的动作实现，在原有基础上增加了检测垃圾短信的功能。这种“更改词典内容”的方式就是通过CydiaSubstrate 进行 hook 操作来实现的。CydiaSubstrate的用法已经在前两章详细介绍过了，想必大家都还记得。

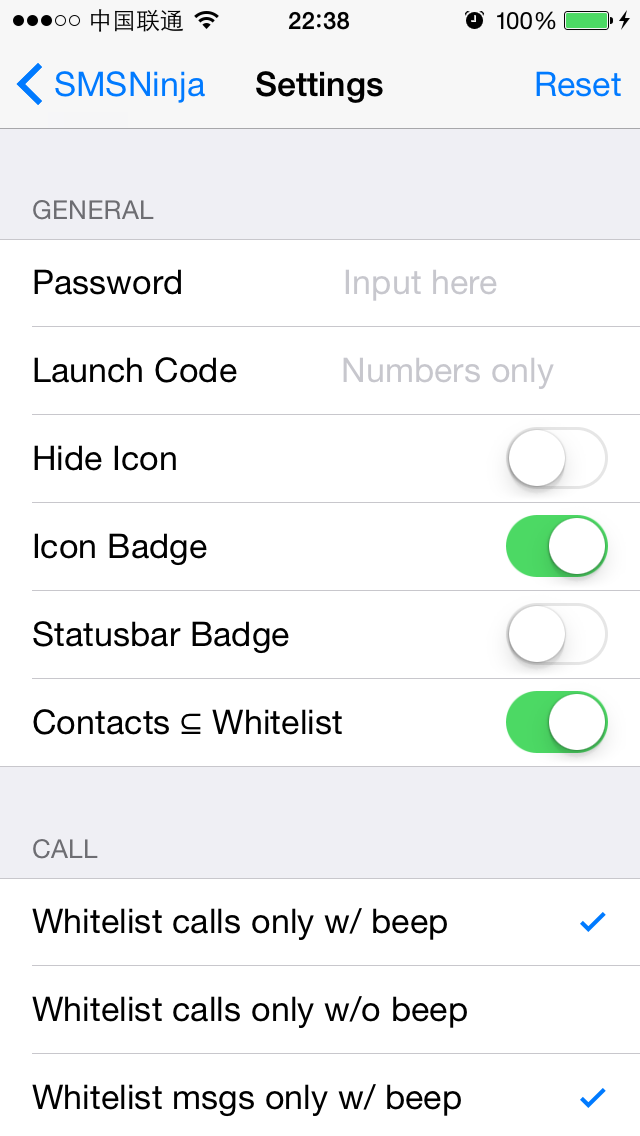


图5- 3 SMSNinja

## 5.2 tweak 的编写套路

只有理解了 tweak 的工作方式，才能在编写 tweak 时清楚地知道自己想干什么、在干什么。一般来说，编写 tweak 会用到 C、C++ 和 Objective-C 三种语言，当我们有了一个灵感时，该如何自如地运用这三种语言来把灵感变成一个好用的 tweak呢？事实上，编写 tweak 的思路是有规律可循的，而且随着你对iOS的了解愈加深入，对编程语言的掌握愈加熟练，这种规律会变得越来越明显。下面将围绕一个简单的 tweak 例子，从 iOS 工程师使用最多的 Objective-C 语言开始分析，总结归纳 Objective-C 级别的逆向工程理论。

### 5.2.1 寻找灵感

可能有部分iOS工程师读到这里时，已经能够结合前几章的知识开始开发 tweak 了，但可能也还有部分人感到无从下手，不知道该写些什么东西。这种有劲儿没处使的感觉确实挺难受，面对这种情况，该怎么办呢？一般情况下，可以从这几个方面找灵感。

#### 1. 多使用，多观察

没事就把你的手机拿出来把玩把玩，把系统的每个角落都扫一遍，别光顾着刷朋友圈。虽然iOS 的功能已足够多，但也不可能符合每个用户的要求，所以，你用得越多，你对 iOS 的了解就越多，哪些地方用着不爽的感觉就会更强烈。上网看看吧，iOS 的用户基数巨大，他们中一定有跟你想法相同的人——你碰到需要解决的实际问题了，这不就是灵感吗？笔者在iOS 6时代开发的 Characount for Notes（如图5-4所示）就是这样得来的。当时，笔者经常把微博的内容存成记事本，但微博是有140字限制的，于是就做了一个这样的 tweak，用来统计记事本每页的字数，从而控制微博的长度。曾有一位阿拉伯用户还专门给笔者发邮件说很喜欢这个插件，希望加入更多功能把记事本改造成一个 Word，但笔者对这个想法不大感兴趣，所以只好对他说声抱歉了。

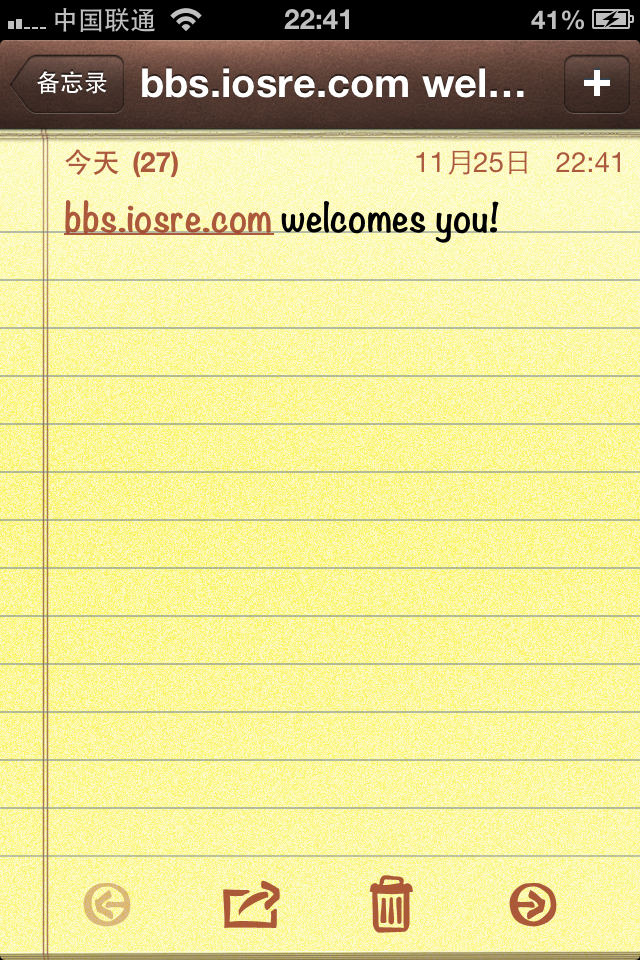


图5- 4 Characount for Notes

#### 2. 倾听用户的声音

每个人使用 iOS 的方式不同，他们的需求各异。如果你自己没有太多灵感，那就多听听果粉们的需求；只要有需求，tweak 就有用户。大的项目已经有人做了，我们就针对少数人群定制 tweak；水平不足做不了底层的复杂功能，就从高层的简单功能做起；每一版发布后，虚心听取用户的意见和建议，及时改进，快速迭代，你的付出不会没有回报。 LowPowerBanner 这个iOS 6插件就是笔者听取用户PrimeCode的建议编写的，完成第 1 版仅用了约 5 小时，写代码不到 50 行，但发布后 8 小时下载量即突破 3 万次（如图5-5所示），受欢迎程度大大超出笔者的预期。同志们，群众的眼睛是雪亮的，群众的智慧也是无穷的，如果你没有什么灵感，就走到群众中去吧！

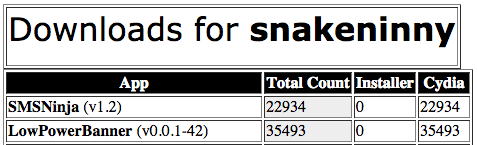


图5- 5 LowPowerBanner的第一版下载量

#### 3. 解剖 iOS

当你的能力越大时，能做的事情也就越多。千里之行始于足下，从小程序做起，经过层层磨炼，你对 iOS 的理解会不断加深；iOS 是个封闭的系统，它暴露给我们的只是冰山一角，有太多太多的功能还有待我们进一步挖掘。每次越狱发布后，都会有人把最新的头文件发布出来，Google一下“iOS private headers”即可轻松找到下载链接，省去了自己class-dump的麻烦。Objective-C语言的函数命名很规律，大多数函数都可以望名生义，如SpringBoard.h里的：

- (void)reboot;

- (void)relaunchSpringBoard;

和UIViewController.h里的：

- (void)attentionClassDumpUser:(id)arg1

yesItsUsAgain:(id)arg2

althoughSwizzlingAndOverridingPrivateMethodsIsFun:(id)arg3

itWasntMuchFunWhenYourAppStoppedWorking:(id)arg4

pleaseRefrainFromDoingSoInTheFutureOkayThanksBye:(id)arg5;

通览这些函数名，是灵感的重要来源之一，也是你了解iOS底层的便捷渠道。掌握越多的iOS实现细节，意味着你手里握有的零件就越多，因此你就能组装出与别人不同的设备。limneos 开发的“Audio Recorder”就是最好的例子，iOS早在2007年面世，但通话录音的功能直到7年后才由这位希腊开发者实现。有这个想法的人很多很多，已经动手的人也肯定不在少数，但为什么只有 limneos 成功了？因为他对 iOS 的解剖比别人更彻底！说起来很简单，做起来不简单。

### 5.2.2 定位目标文件

当我们知道自己想要实现什么功能后，就要开始寻找实现这个功能的二进制文件，用到的方法一般有以下几种。

#### 1. 固定位置

现阶段我们的逆向目标一般是 dylib、bundle 或 daemon，它们在系统中的位置几乎是固定的：

❏  基于CydiaSubstrate的dylib全部位于“/Library/MobileSubstrate/ DynamicLibraries/”下， 几乎不费吹灰之力就可以轻松定位。

❏ bundle主要分为App 和framework 两类，其中AppStore App全部位于 “/var/mobile/Containers/Bundle/Application/”下，framework 全部位于 “/System/Library/Frameworks”或“/System/Library/ PrivateFramework”下。关于其他类型App的定位，可以来http://bbs.iosre.com一起讨论。

❏  daemon的配置文件均位于“/System/Library/LaunchDaemons/”、“/Library/LaunchDaemons”或“/Library/LaunchAgents/”下，是一个plist格式的文件。其中的“ProgramArguments”字段，即是daemon可执行文件的绝对路径：

snakeninnys-MacBook:~ snakeninny$ plutil -p /Users/snakeninny/Desktop/com.apple.backboardd.plist

{

……

"ProgramArguments" => [

0 => "/usr/libexec/backboardd"

]

……

}

#### 2. Cydia定位

通过 “dpkg -i”命令安装的deb包，其内容会被 Cydia 如实记录，若想要查看，在 Cydia 的“Installed”项中选择“Expert”，如图5-6所示。

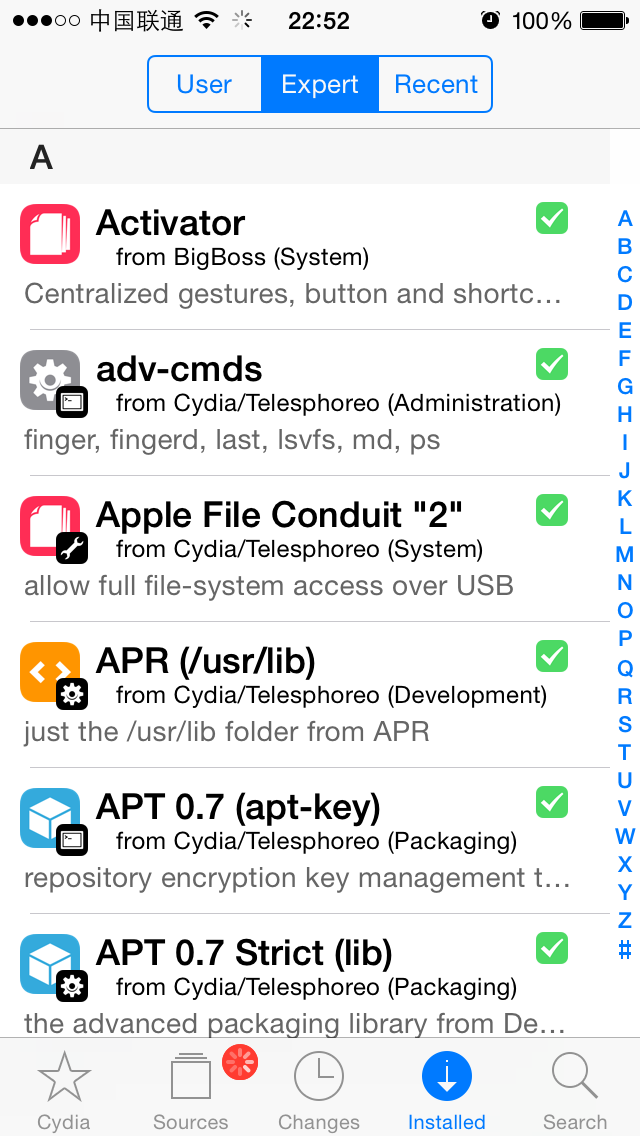


图5- 6 Cydia的Expert界面

然后选择目标软件，进入“Details”界面，如图5-7所示。

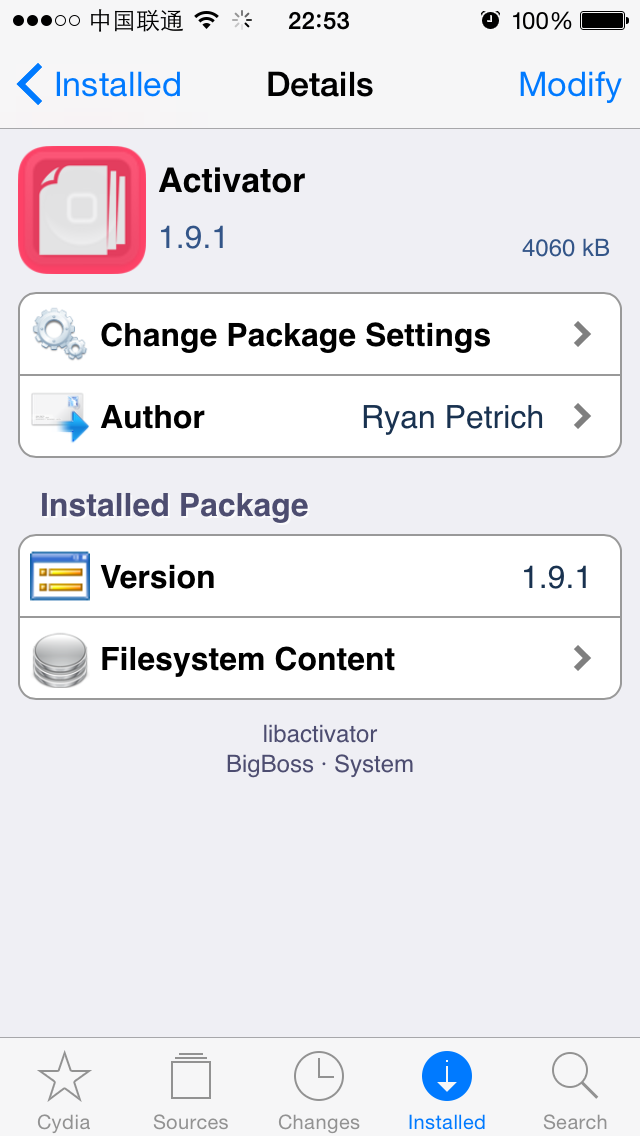


图5- 7 Details界面

之后选择“Filesystem Content”，即可浏览软件包里的所有文件，如图5-8所示。



图5- 8 Filesystem Content界面

deb包中的每个文件都被放在了iOS的哪个路径下，一目了然。

#### 3. PreferenceBundle

PreferenceBundle是寄生在Settings程序里的App，它的功能界定有些模糊，既可以作为单纯的配置文件，由别的进程读取后执行，如图5-9所示的“DimInCall”界面。

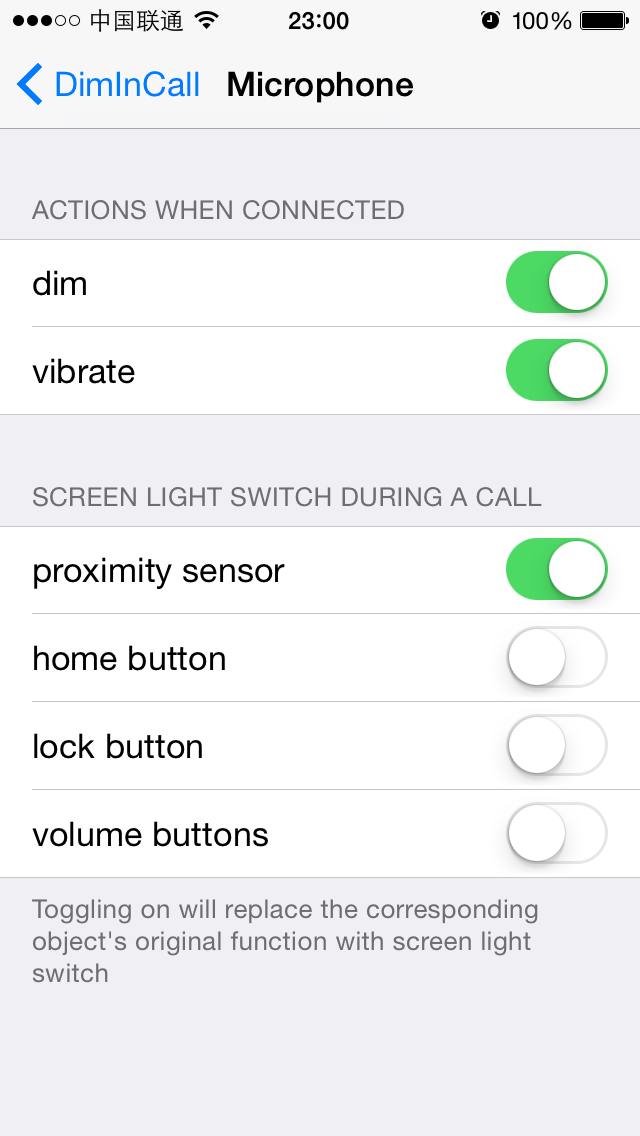


图5- 9 DimInCall界面

也可以含有实际功能，自己来执行一些操作，如图5-10所示的“WLAN”界面。

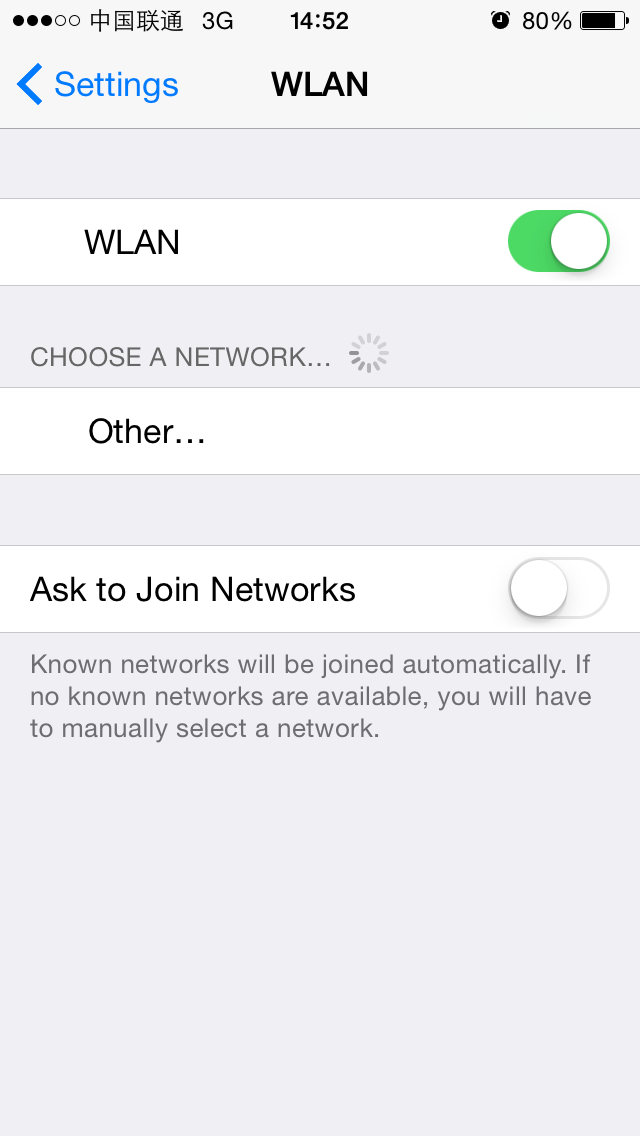


图5- 10 WLAN界面

我们关注的重点是应用的实际功能，因此如何定位PreferenceBundle执行实际功能的二进制文件，就是需要研究的课题之一。来自AppStore的第三方PreferenceBundle仅可作为配置文件存在，不会含有实际功能；来自Cydia的也不是问题，刚才介绍的Cydia定位方式已经完全够用了；但对于iOS自带的PreferenceBundle来说，定位的过程就要复杂一些。

PreferenceBundle的界面可以用代码编写，也可以用具有固定格式的plist 文件构造（格式请参考 http:// iphonedevwiki.net/index.php/Preferences\_specifier\_plist）。当我们在逆向此类程序时，如果发现界面中的控件类型全部来自 preference specifier plist罗列的标准控件类型，如“About”界面（如图5-11所示），则需注意分辨此界面是用代码编写的，还是用plist构造的。对于iOS自带的PreferenceBundle来说，如果是用代码编写的，一般情况下实际功能就已经包含在二进制文件里了，它们位于“/System/Library/PreferenceBundles/”下；如果是用plist构造的，就需要分析plist和实际功能间的关系，从中找到切入点，定位含有实际功能的二进制文件。总之，PreferenceBundle的情况相对复杂，并不适合作为新手练习。如果你对上面的内容一知半解，不要紧，稍后本章会以一个实例来提供参考。更多关于PreferenceBundle的讨论，尽在<http://bbs.iosre.com>。



图5- 11 About界面

#### 4. grep命令

grep 是一个来自 UNIX 系统的命令行工具，能够搜索文件中是否含有给定的正则表达式。OSX自带greo命令，iOS上的grep命令则是由Saurik移植过来的，随着Cydia默认安装。在寻找一个字符串的出处时，grep能够快速缩小查找范围。例如，想知道都有哪些地方调用了[IMDAccount initWithAccountID:de faults:service:]，可以ssh到iOS后使用grep命令查看一下：

FunMaker-5:~ root# grep -r initWithAccountID:defaults:service: /System/Library/

Binary file /System/Library/Caches/com.apple.dyld/dyld\_shared\_cache\_armv7s matches

grep: /System/Library/Caches/com.apple.dyld/enable-dylibs-to-override-cache: No such file or directory

grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libCGCorePDF.dylib: No such file or directory

grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libCMSBuiltin.dylib: No such file or directory

grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libCMaps.dylib: No such file or directory

grep: /System/Library/Frameworks/System.framework/System: No such file or directory

从运行结果得知，要查找的函数在 dyld\_shared\_cache\_armv7s中出现了。再次对decache 过的 dyld\_shared\_cache\_armv7s使用grep命令：

snakeninnysiMac:~ snakeninny$ grep -r initWithAccountID:defaults:service: /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5

Binary file /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/dyld\_shared\_cache\_armv7s matches

grep: /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/System/Library/Caches/com.apple.xpc/sdk.dylib: Too many levels of symbolic links

grep: /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/System/Library/Frameworks/OpenGLES.framework/libLLVMContainer.dylib: Too many levels of symbolic links

Binary file /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/System/Library/PrivateFrameworks/IMDaemonCore.framework/IMDaemonCore matches

可以看到，在“/System/Library/”目录下，[IMDAccount initWithAccountID:de faults:service:]出现在了IMDaemonCore中，可以就从它着手开始分析。

### 5.2.3 定位目标函数

在找到含有目标功能的二进制文件之后，可以通过class-dump导出头文件，在里面寻找自己感兴趣的函数。具体做法比较简单，可分为以下两种。

#### 1. OSX自带的搜索功能

不得不承认，OSX自带的搜索功能在笔者用过的操作系统中是最强大的，强大到既能搜索文件名，又能搜索文件内容，而且不论是搜目录还是搜全盘，速度都非常快。利用这一便利工具，可以在大量文件中快速定位目标文件，例如，对iPhone自带的距离感应器（Proximity Sensor）很感兴趣，想看看相关的函数可能会提供哪些功能，可以在Finder中打开你保存所有class-dump头文件的文件夹，然后在右上角的搜索栏中输入proximity（大小写不敏感），如图5-12所示。

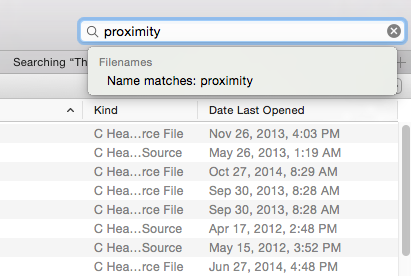


图5- 12 在搜索栏输入关键词

默认情况下Finder会把本机所有内容中含有 proximity关键词的文本文件罗列出来，如图5-13 所示。

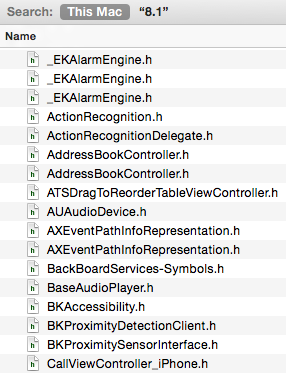


图5-13 搜索结果

也可以缩小搜索范围，选择在当前目录下递归搜索文件名。剩下的工作就是找出你感兴趣的文件，然后开始分析喽！

#### 2. grep命令

是的，你没看错，强大的grep再一次出现了。既然grep能搜索出二进制文件里的字符串，对付文本文件就更不在话下了。对于刚才的例子，使用grep来试试：

snakeninnysiMac:~ snakeninny$ grep -r -i proximity /Users/snakeninny/Code/iOSPrivateHeaders/8.1

/Users/snakeninny/Code/iOSPrivateHeaders/8.1/Frameworks/CoreLocation/CDStructures.h: char proximityUUID[512];

/Users/snakeninny/Code/iOSPrivateHeaders/8.1/Frameworks/CoreLocation/CLBeacon.h: NSUUID \*\_proximityUUID;

……

/Users/snakeninny/Code/iOSPrivateHeaders/8.1/SpringBoard/SpringBoard.h:- (\_Bool)proximityEventsEnabled;

/Users/snakeninny/Code/iOSPrivateHeaders/8.1/SpringBoard/SpringBoard.h:- (void)\_proximityChanged:(id)arg1;

虽然grep 显示出的结果大而全，但看起来有些乱。推荐使用 Finder 的搜索功能，毕竟在便捷程度相差无几的情况下，图形界面比命令行界面用起来更方便。

### 5.2.4 测试函数功能

在逆向工程中，我们感兴趣的绝大多数函数都是私有的，没有文档可供参考，如果运气足够好，谷歌可能会帮上你的忙，但也可能说明你想做的东西别人已经做过了；如果搜索不到，那么恭喜， 你可能发现了一块新大陆，但是，函数的用法和功能需要你自己测试。

Objective-C 函数的功能测试相对于 C/C++ 函数来说要简单得多，有CydiaSubstrate 和 Cycript 两种方法可供选择。

#### 1. CydiaSubstrate

在测试函数功能时，主要利用 CydiaSubstrate来钩住（hook）住一个函数，从而判断这个函数的调用时机。假设怀疑 SBScreenShotter.h 中的 saveScreenshot:在截屏时得到了调用，就可以撰写以下代码来验证：

%hook SBScreenShotter

- (void)saveScreenshot:(BOOL)screenshot

{

%orig;

NSLog(@"iOSRE: saveScreenshot: is called");

}

%end

将 filter 设置成“com.apple.springboard”，并用 Theos 制作成 deb，安装在 iOS 中，然后注销（respring）一次。如果感觉有些生疏，不要着急，这是正常的，不求快，但求稳。等锁屏界面完全出现后，同时按下 home 键和 lock 键截屏，然后ssh 到 iOS查看syslog：

FunMaker-5:~ root# grep iOSRE: /var/log/syslog

Nov 24 16:22:06 FunMaker-5 SpringBoard[2765]: iOSRE: saveScreenshot: is called

可以看到，syslog 中出现了我们的自定义信息，说明在截屏时，saveScreenshot:得到了调用。 此时，你一定会跟笔者一样好奇：这个函数名的含义太明显了，调用这个函数，是不是真就能实现截屏的功能呢？

在 iOS 的世界中，好奇不会害死猫，就怕你失去好奇心。要满足好奇心，就用Cycript！

#### 2. Cycript

在知道 Cycript 之前，笔者测试函数功能的工具是 Theos。比如，针对上面的例子，笔者会编写这样 一个 tweak。

%hook SpringBoard

- (void)\_menuButtonDown:(id)down

{

%orig;

SBScreenShotter \*shotter = [%c(SBScreenShotter) sharedInstance];

[shotter saveScreenshot:YES]; // 这里参数传YES是我猜的，等会我们试验一下传NO是什么效果

}

%end

当 tweak 生效后，按下 home 键，就会调用 saveScreenshot: 函数，然后观察屏幕是不是白光一闪，相册里是不是多了一张截屏。再进入 Cydia 把 tweak 删掉，把 home 键的单纯还给它……

其实如果没有对比，这种方法看起来还算简单，但是当笔者用 Cycript 达到了相同目的时，才后知后觉地发现，以前浪费了多少“井猜”的“绳命”！

Cycript 的用法前面已经介绍过了，因为 SBScreenShotter是 SpringBoard 里的类，所以这里将 Cycript 注入 SpringBoard 进程，然后直接调用待测试函数观察实际效果即可，整个编译过程对我们是透明的，测试后无须任何清理工作，简直会让人忍不住哼唱：“测一个简单函数，让我的心情快乐，逆向就像一条河，难免会碰到波折。”

ssh 到 iOS后输入：

FunMaker-5:~ root# cycript -p SpringBoard

cy# [[SBScreenShotter sharedInstance] saveScreenshot:YES]

你的屏幕是不是也白光一闪，“咔嚓”一声，截屏一张，与同时按下2个键截屏的效果如出一辙？好了，现在可以确认这个函数能完成截屏操作了。为了进一步满足我们的好奇心和求知欲，在Cycript 提示符下按“↑”键，重复上一次输入的命令，然后把 YES 改成 NO，看看是什么效果。下一节将会继续说明。

### 5.2.5 解析函数参数

在上面的例子中，函数的参数明确，函数名的含义明显，但我们还是拿不准在调用时到底是传 YES 还是 NO，只能靠猜。浏览通过class-dump导出的头文件时，你会发现绝大多数函数的参数类型是 id，也就是 Objective-C 里的泛型，它是在运行时动态决定的，猜都没法猜。我们从感兴趣的功能开始，一路分析到了对应的函数，只差一步就能闯过第一关了，难道要就此放弃？“不要放弃！”CydiaSubstrate 和 Theos 异口同声地说。

还记得我们是怎样判断函数调用时机的吧？既然能打印一个自定义字符串，就完全能打印出函数参数的信息——description 函数能够把对象的内容表示成一个 NSString，object\_getClassName函数能够把对象的类名表示成一个 char\*，两者可分别用 %@ 和 %s 打印出来，这就为解析参数提供了足够参考。对于刚才完成的截屏操作，saveScreenshot: 的参数是YES 还是 NO，唯一的区别好像在于屏幕是否闪现白光。依据这个线索，我们很快就能定位到可疑的 SBScreenFlash 类，其中有一个有意思的函数 flashColor:——是否闪光可以选择，难道闪光的颜色也可以改变？而且，参数类型似乎就是 UIColor 吧？编写下面的代码，来满足一下自己的好奇心。

%hook SBScreenFlash

- (void)flashColor:(id)arg1 withCompletion:(id)arg2

{

%orig;

NSLog(@"iOSRE: flashColor: %s, %@", object\_getClassName(arg1), arg1); // [arg1 description] 可以直接写成arg1

}

%end

作为练习，请读者把上面的代码变成一个可用的 tweak。

安装完成后，注销（respring ）一次，截个屏，再通过ssh命令连接到iOS上看看syslog，你所看到的内容应该如下所示：

FunMaker-5:~ root# grep iOSRE: /var/log/syslog

Nov 24 16:40:33 FunMaker-5 SpringBoard[2926]: iOSRE: flashColor: UICachedDeviceWhiteColor, UIDeviceWhiteColorSpace 1 1

可 以 看 出，color 是一个UICachedDeviceWhiteColor类型的对象，它的description是“UIDevice WhiteColorSpace 1 1”。根据命名规则，UICachedDeviceWhiteColor 是 UIKit 中的一个类，但在文档中搜索不到这个类，因此可以断定它是个私有类。在 class-dump 出的 UIKit 头文件中找到 UICachedDeviceWhiteColor.h，打开看看：

@interface UICachedDeviceWhiteColor : UIDeviceWhiteColor

{

}

- (void)\_forceDealloc;

- (void)dealloc;

- (id)copy;

- (id)copyWithZone:(struct \_NSZone \*)arg1;

- (id)autorelease;

- (BOOL)retainWeakReference;

- (BOOL)allowsWeakReference;

- (unsigned int)retainCount;

- (id)retain;

- (oneway void)release;

@end

它继承自 UIDeviceWhiteColor，于是继续找到 UIDeviceWhiteColor.h：

@interface UIDeviceWhiteColor : UIColor

{

float whiteComponent;

float alphaComponent;

struct CGColor \*cachedColor;

long cachedColorOnceToken;

}

- (BOOL)getHue:(float \*)arg1 saturation:(float \*)arg2 brightness:(float \*)arg3 alpha:(float \*)arg4;

- (BOOL)getRed:(float \*)arg1 green:(float \*)arg2 blue:(float \*)arg3 alpha:(float \*)arg4;

- (BOOL)getWhite:(float \*)arg1 alpha:(float \*)arg2;

- (float)alphaComponent;

- (struct CGColor \*)CGColor;

- (unsigned int)hash;

- (BOOL)isEqual:(id)arg1;

- (id)description;

- (id)colorSpaceName;

- (void)setStroke;

- (void)setFill;

- (void)set;

- (id)colorWithAlphaComponent:(float)arg1;

- (struct CGColor \*)\_createCGColorWithAlpha:(float)arg1;

- (id)copyWithZone:(struct \_NSZone \*)arg1;

- (void)dealloc;

- (id)initWithCGColor:(struct CGColor \*)arg1;

- (id)initWithWhite:(float)arg1 alpha:(float)arg2;

@end

UIDeviceWhiteColor 继承自 UIColor，因为 UIColor 是一个公开类，所以对参数类型的解析到这个程度就可以了。对其他 id 类型参数的解析均可重复上述思路。

知道了一个函数的调用效果，解析了它的参数，它的使用文档就可以由我们自行撰写了，建议大家对自己分析的函数作简单记录，这样在下次使用时就能迅速回想起它的用法。

接下来要用 Cycript 来测试这个函数，看看传进去一个 [UIColor magentaColor] 是什么效 果：

FunMaker-5:~ root# cycript -p SpringBoard

cy# [[SBScreenFlash mainScreenFlasher] flashColor:[UIColor magentaColor] withCompletion:nil]

一抹紫红色的光在屏幕上散开，比白色的闪光有个性多了。检查相册，并没有看到新截屏，因此自然地猜测，这个函数仅仅负责截屏时的闪光功能，而不进行实际截屏操作——一个新的 tweak 灵感就此产生：我们可以钩住（hook）这个 flashColor: 函数，把自定义的颜色作为参数传递给它，从而使截屏闪光变得丰富多彩起来。这个 tweak 作为练习，请读者独立完成。

以上的套路是笔者5年多以来的总结，因为iOS逆向工程没有任何官方资料可供参考，笔者个人经验难免有失偏颇，不可能面面俱到，所以，<http://bbs.iosre.com>的大门向任何讨论开放，欢迎提问！

### 5.2.6 class-dump的局限性

分析通过class-dump导出的头文件，我们找到了感兴趣的东西，并在 5.2.4 节的 Cycript 试 验中看到了对 SBScreenShotter 类中 saveScreenshot: 函数传 YES 和 NO 两种参数时函数的不同执行效果。

在 5.2.5 节里，解析了SBScreenFlash 类的 flashColor: 函数参数。从 flashColor:的效果来看，我们猜测它应该发生在 saveScreenshot: 的内部，而如果仅根据 class-dump 的头文件，结合 CydiaSubstrate，最多也只能判断出saveScreenshot: 和 flashColor:的先后调用顺序，至于两者的实现细节和调用关系则不得而知。

完成了一个tweak，应该小小庆祝一下。从灵感，到文件，到函数，最后到成型的tweak，所有Objective-C级别的逆向工程都遵循这个套路，只是实现细节不同而已，即使你完全不懂越狱开发，相信你也能掌握这个套路，它一点都不难。难度低，门槛就低，竞争就多，压力就大，当你掌握了Objective-C级别的逆向工程思路，想要进阶更高的级别，就会发现class-dump不够用了。

在完成一个小tweak之后，我们还应清楚地意识到，与这个 tweak 相关的很多知识点我们还没有弄清楚，而通过 class-dump 得到的信息并不足以支撑我们弄清这些未知的东西，就好像我们身处逆向工程这片茂密的原始森林中，class-dump 给我们提供了可以落脚的小屋， 但要走出这片森林，还需要一张地图和一个指南针——它们就是 IDA 和LLDB。这两款工具就像两座挡在我们面前的大山，绝大多数逆向工程初学者都没能成功翻越它们，爬到半山腰就打道回府了，而翻越大山的人们顺利跨过逆向工程的门槛，欣赏到了别样的风景。梦想还是要有的，万一实现了呢？我们鼓起勇气，试试看能不能征服它们。

## 5.3 实例演示

在翻山越岭之前，本节将针对刚才讲过的理论作一次全面的实战演练，让大家更牢固地掌握所学的知识，以便更平稳地过渡到第6章。本次实战演练的内容是一个真实的示例，它按 照 5.2 节所示的套路，完整地讲述了笔者iOS 6插件“Speaker SBSettings Toggle”（如图 5-14 所示）的开发过程。当时笔者还不会使用IDA和LLDB，所有的线索几乎都来自class-dump和误打误撞，能够比较好的代表iOS逆向工程初学者的状态。



图5- 14 Speaker SBSettings Toggle

注意：下面的具体步骤已不适用于iOS 8，请大家当做案例，了解思路，作为参考就好。

### 5.3.1 得到灵感

2012年3月底，笔者收到一个伊朗裔加拿大人Shoghian 发来的邮件，邮件中分享了一个创意：iOS 通话时用户可以从听筒切换到免提，但很少有人知道，接听来电时是可以默认打开免提的，这个功能对那些开车、做饭或工作时双手不方便接电话的人非常有用。但是这么有用的功能却被 iOS 藏在了“设置”→“通用”→“ 辅助功能”→“ 来电使用”的四级目录里（如图5-15所示），设置起来非常繁琐。SBSettings 上各种各样的开关就是为解决这类问题而存在的，因此笔者打算把这个功能做成一个 toggle，帮这家酒香不怕巷子深的饭店找一个临街的门面，把好的事物呈现在更多人的面前。



图5- 15 Incoming Calls界面

### 5.3.2 定位文件

因为这个功能位于“设置”中，所以笔者的第一反应自然是在 “/Applications/Preferences. App”和“/System/Library/PreferenceBundles/”里寻找可疑文件，大致步骤如下。

#### 1. 将 iOS 系统语言换成英文

因为iOS的文件系统是全英文的，所以在开始分析之前，笔者先把 iOS 的系统语言设置成了英文，这样在浏览文件系统时看到的关键词与UI上显示的关键词就更有可能产生对应关系。

#### 2. 发现“Accessibility”关键词

切换语言后，“设置”→“通用”→“ 辅助功能”→“ 来电使用”翻译成了“Settings” →“General” →“Accessibility” →“Incoming Calls”，其中 Accessibility 关键词引起了笔者的注意， 因为不结合语境的话，Accessibility 是不会直译成“辅助功能”的。于是笔者 ssh 到 iOS 中，以 Accessibility 为关键词进行了一次 grep 操作：

FunMaker-4s:~ root# grep -r Accessibility /

grep: /Applications/Activator.app/Default-568h@2x~iphone.png: No such file or directory

grep: /Applications/Activator.app/Default.png: No such file or directory

grep: /Applications/Activator.app/Default~iphone.png: No such file or directory

grep: /Applications/Activator.app/LaunchImage-700-568h@2x.png: No such file or directory

Binary file /Applications/Activator.app/en.lproj/Localizable.strings matches

grep: /Applications/Activator.app/iOS7-Default-Landscape@2x.png: No such file or directory

grep: /Applications/Activator.app/iOS7-Default-Portrait@2x.png: No such file or directory

Binary file /Applications/AdSheet.app/AdSheet matches

Binary file /Applications/Compass.app/Compass matches

……

得到的结果很多，但最吸引笔者的是下面这几个以 strings 为后缀的文件：

Binary file /Applications/Preferences.app/English.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/English.lproj/General~iphone.strings matches

Binary file /Applications/Preferences.app/General-Simulator.plist matches

Binary file /Applications/Preferences.app/General.plist matches

Binary file /Applications/Preferences.app/Preferences matches

Binary file /Applications/Preferences.app/en\_GB.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/en\_GB.lproj/General~iphone.strings matches

如果不出意外，它们是App字符串本地化的配置文件，里面应该含有 Accessibility 在代码中的符号名。用Xcode自带的plutil工具查看strings文件非常方便，先来看看“/Applications/Preferences.app/English.lproj/General~iphone.strings”：

snakeninnys-MacBook:~ snakeninny$ plutil -p ~/General\~iphone.strings

{

"Videos..." => "• Videos..."

"Wallpaper" => "Wallpaper"

"TV\_OUT" => "TV Out"

"SOUND\_EFFECTS" => "Sound Effects"

"d\_MINUTES" => "%@ Minutes"

……

"ACCESSIBILITY" => "Accessibility"

"Multitasking\_Gestures" => "Multitasking Gestures"

……

}

由" ACCESSIBILITY " => " Accessibility "基本可以断定，“ACCESSIBILITY”就是代码中使用的符号名。

#### 3. 发现 General.plist

有了新的线索后，以大写的ACCESSIBILITY为关键词，又grep了一遍：

FunMaker-4s:~ root# grep -r ACCESSIBILITY /

grep: /Applications/Activator.app/Default-568h@2x~iphone.png: No such file or directory

grep: /Applications/Activator.app/Default.png: No such file or directory

grep: /Applications/Activator.app/Default~iphone.png: No such file or directory

grep: /Applications/Activator.app/LaunchImage-700-568h@2x.png: No such file or directory

grep: /Applications/Activator.app/iOS7-Default-Landscape@2x.png: No such file or directory

grep: /Applications/Activator.app/iOS7-Default-Portrait@2x.png: No such file or directory

Binary file /Applications/Preferences.app/Dutch.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/Dutch.lproj/General~iphone.strings matches

Binary file /Applications/Preferences.app/English.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/English.lproj/General~iphone.strings matches

Binary file /Applications/Preferences.app/French.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/French.lproj/General~iphone.strings matches

Binary file /Applications/Preferences.app/General-Simulator.plist matches

Binary file /Applications/Preferences.app/General.plist matches

Binary file /Applications/Preferences.app/German.lproj/General-Simulator.strings matches

Binary file /Applications/Preferences.app/German.lproj/General~iphone.strings matches

……

得到的结果与刚才grep结果的重合度很高，其中，刚才没有留意的“/Applications/Preferences.app/General.plist”显得格外醒目。在 5.2.2 节中，特意提到了PreferenceBundle的概念，此处 General.plist 既是 plist 格式的文件，又包含关键词，我们看看它里面有什么：

snakeninnys-MacBook:~ snakeninny$ plutil -p ~/General.plist

{

"title" => "General"

"items" => [

0 => {

"cell" => "PSGroupCell"

}

1 => {

"detail" => "AboutController"

"cell" => "PSLinkCell"

"label" => "About"

}

2 => {

"cell" => "PSLinkCell"

"id" => "SOFTWARE\_UPDATE\_LINK"

"detail" => "SoftwareUpdatePrefController"

"label" => "SOFTWARE\_UPDATE"

"cellClass" => "PSBadgedTableCell"

}

……

24 => {

"detail" => "PSInternationalController"

"cell" => "PSLinkCell"

"label" => "INTERNATIONAL"

}

25 => {

"cell" => "PSLinkCell"

"bundle" => "AccessibilitySettings"

"label" => "ACCESSIBILITY"

"requiredCapabilities" => [

0 => "accessibility"

]

"isController" => 1

}

26 => {

"cell" => "PSGroupCell"

}

……

]

}

#### 4. 发现 AccessibilitySetting.bundle

果不其然，这个文件就是一个标准的 preference specifier plist，大写的“ACCESSIBILITY”来自25号单元。对比 preferences specifier plist 格式，将目标锁定在 AccessibilitySettings这个 bundle 中；由 AccessibilitySettings 的名字，自然地猜测这个 bundle 可能负责 Accessibility 下的所有功能。根据 5.2.2 节中的文件固定位置理论，AccessibilitySettings.bundle 一定安安静静地躺在 “/System/Library/PreferenceBundles/”中，对将要发生在自己身上的事情浑然不知。

看看“/System/Library/PreferenceBundles/AccessibilitySetting.bundle”里面有什么：

FunMaker-4s:~ root# ls -la /System/Library/PreferenceBundles/AccessibilitySettings.bundle

total 240

drwxr-xr-x 37 root wheel 2414 Mar 10 2013 .

drwxr-xr-x 40 root wheel 1360 Jan 14 2014 ..

-rw-r--r-- 1 root wheel 2146 Mar 10 2013 Accessibility.plist

-rwxr-xr-x 1 root wheel 438800 Mar 10 2013 AccessibilitySettings

-rw-r--r-- 1 root wheel 238 Dec 22 2012 BluetoothDeviceConfig.plist

-rw-r--r-- 1 root wheel 252 Mar 10 2013 BrailleStatusCellSettings.plist

-rw-r--r-- 1 root wheel 4484 Dec 22 2012 ColorWellRound@2x.png

-rw-r--r-- 1 root wheel 916 Dec 22 2012 ColorWellSquare@2x.png

drwxr-xr-x 2 root wheel 646 Feb 7 2013 Dutch.lproj

drwxr-xr-x 2 root wheel 646 Dec 22 2012 English.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 French.lproj

drwxr-xr-x 2 root wheel 646 Dec 22 2012 German.lproj

-rw-r--r-- 1 root wheel 703 Mar 10 2013 GuidedAccessSettings.plist

-rw-r--r-- 1 root wheel 807 Mar 10 2013 HandSettings.plist

-rw-r--r-- 1 root wheel 652 Mar 10 2013 HearingAidDetailSettings.plist

-rw-r--r-- 1 root wheel 507 Mar 10 2013 HearingAidSettings.plist

-rw-r--r-- 1 root wheel 383 Dec 22 2012 HomeClickSettings.plist

-rw-r--r-- 1 root wheel 447 Dec 22 2012 IconPlay@2x.png

-rw-r--r-- 1 root wheel 1113 Dec 22 2012 IconRecord@2x.png

-rw-r--r-- 1 root wheel 170 Dec 22 2012 IconStop@2x.png

-rw-r--r-- 1 root wheel 907 Mar 10 2013 Info.plist

drwxr-xr-x 2 root wheel 646 Feb 7 2013 Italian.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 Japanese.lproj

-rw-r--r-- 1 root wheel 364 Dec 22 2012 LargeFontsSettings.plist

-rw-r--r-- 1 root wheel 217 Mar 10 2013 NavigateImagesSettings.plist

-rw-r--r-- 1 root wheel 1030 Dec 22 2012 QuickSpeakSettings.plist

-rw-r--r-- 1 root wheel 346 Dec 22 2012 RegionNamesNonLocalized.strings

drwxr-xr-x 2 root wheel 646 Feb 7 2013 Spanish.lproj

-rw-r--r-- 1 root wheel 394 Dec 22 2012 SpeakerLoad1@2x.png

-rw-r--r-- 1 root wheel 622 Mar 10 2013 TripleClickSettings.plist

-rw-r--r-- 1 root wheel 467 Dec 22 2012 VoiceOverBrailleOptions.plist

-rw-r--r-- 1 root wheel 2477 Mar 10 2013 VoiceOverSettings.plist

-rw-r--r-- 1 root wheel 540 Mar 10 2013 VoiceOverTypingFeedback.plist

-rw-r--r-- 1 root wheel 480 Dec 22 2012 ZoomSettings.plist

drwxr-xr-x 2 root wheel 102 Dec 22 2012 \_CodeSignature

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ar.lproj

-rw-r--r-- 1 root wheel 8371 Dec 22 2012 bottombar@2x~iphone.png

-rw-r--r-- 1 root wheel 2701 Dec 22 2012 bottombarblue@2x~iphone.png

-rw-r--r-- 1 root wheel 2487 Dec 22 2012 bottombarblue\_pressed@2x~iphone.png

-rw-r--r-- 1 root wheel 2618 Dec 22 2012 bottombarred@2x~iphone.png

-rw-r--r-- 1 root wheel 2426 Dec 22 2012 bottombarred\_pressed@2x~iphone.png

-rw-r--r-- 1 root wheel 2191 Dec 22 2012 bottombarwhite@2x~iphone.png

-rw-r--r-- 1 root wheel 2357 Dec 22 2012 bottombarwhite\_pressed@2x~iphone.png

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ca.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 cs.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 da.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 el.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 en\_GB.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 fi.lproj

-rw-r--r-- 1 root wheel 955 Dec 22 2012 hare@2x.png

drwxr-xr-x 2 root wheel 646 Feb 7 2013 he.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 hr.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 hu.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 id.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ko.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ms.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 no.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 pl.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 pt.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 pt\_PT.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ro.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 ru.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 sk.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 sv.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 th.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 tr.lproj

-rw-r--r-- 1 root wheel 998 Dec 22 2012 turtle@2x.png

drwxr-xr-x 2 root wheel 646 Feb 7 2013 uk.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 vi.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 zh\_CN.lproj

drwxr-xr-x 2 root wheel 646 Feb 7 2013 zh\_TW.lproj

这里的 GuidedAccess、HearingAid 和 HomeClick 等字眼和我们在“Accessibility”中看到的内容吻合（如图5-16所示），它们印证了笔者的猜测。



图5- 16 关键词重合度高

#### 5. 发现“ACCESSIBILITY\_DEFAULT\_HEADSET”关键词

借助强大的 grep，以 “Incoming”为关键词搜索一下这个 bundle：

FunMaker-4s:~ root# grep -r Incoming /System/Library/PreferenceBundles/AccessibilitySettings.bundle

Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/English.lproj/Accessibility~iphone.strings matches

Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/en\_GB.lproj/Accessibility~iphone.strings matches

搜索的结果同本小节开始时的场景如出一辙。打开 “/System/Library/PreferenceBundles/ AccessibilitySettings.bundle/English.lproj/Accessibility~iphone.strings”看看：

snakeninnys-MacBook:~ snakeninny$ plutil -p ~/Accessibility\~iphone.strings

{

"HAC\_MODE\_POWER\_REDUCTION\_N90" => "Hearing Aid Mode improves performance with some hearing aids, but may reduce cellular reception."

"LEFT\_RIGHT\_BALANCE\_SPOKEN" => "Left-Right Stereo Balance"

"QUICKSPEAK\_TITLE" => "Speak Selection"

"LeftStereoBalanceIdentifier" => "L"

"ACCESSIBILITY\_DEFAULT\_HEADSET" => "Incoming Calls"

"HEADSET" => "Headset"

"CANCEL" => "Cancel"

"ON" => "On"

"CUSTOM\_VIBRATIONS" => "Custom Vibrations"

"CONFIRM\_INVERT\_COLORS\_REMOVAL" => "Are you sure you want to disable inverted colors?"

"SPEAK\_AUTOCORRECTIONS" => "Speak Auto-text"

"DEFAULT\_HEADSET\_FOOTER" => "Choose route for incoming calls."

"HEARING\_AID\_COMPLIANCE\_INSTRUCTIONS" => "Improves compatibility with hearing aids in some circumstances. May reduce 2G cellular coverage."

"DEFAULT\_HEADSET" => "Default to headset"

"ROOT\_LEVEL\_TITLE" => "Accessibility"

"HEARING\_AID\_COMPLIANCE" => "Hearing Aid Mode"

"CUSTOM\_VIBES\_INSTRUCTIONS" => "Assign unique vibration patterns to people in Contacts. Change the default pattern for everyone in Sounds settings."

"VOICEOVERTOUCH\_TEXT" => "VoiceOver is for users with

blindness or vision disabilities."

"IMPORTANT" => "Important"

"COGNITIVE\_HEADING" => "Learning"

"HAC\_MODE\_EQUALIZATION\_N94" => "Hearing Aid Mode improves audio quality with some hearing aids."

"SAVE" => "Save"

"HOME\_CLICK\_TITLE" => "Home-click Speed"

"AIR\_TOUCH\_TITLE" => "AssistiveTouch"

"CONFIRM\_ZOT\_REMOVAL" => "Are you sure you want to disable Zoom?"

"VOICEOVER\_TITLE" => "VoiceOver"

"OFF" => "Off"

"GUIDED\_ACCESS\_TITLE" => "Guided Access"

"ZOOMTOUCH\_TEXT" => "Zoom is for users with low-vision acuity."

"INVERT\_COLORS" => "Invert Colors"

"ACCESSIBILITY\_SPEAK\_AUTOCORRECTIONS" => "Speak Auto-text"

"LEFT\_RIGHT\_BALANCE\_DETAILS" => "Adjust the audio volume balance between left and right channels."

"MONO\_AUDIO" => "Mono Audio"

"CONTRAST" => "Contrast"

"ZOOM\_TITLE" => "Zoom"

"TRIPLE\_CLICK\_HEADING" => "Triple-click"

"OK" => "OK"

"SPEAKER" => "Speaker"

"AUTO\_CORRECT\_TEXT" => "Automatically speak auto-corrections

and auto-capitalizations."

"HEARING" => "Hearing"

"LARGE\_FONT" => "Large Text"

"CONFIRM\_VOT\_USAGE" => "VoiceOver"

"CONFIRM\_VOT\_REMOVAL" => "Are you sure you want to disable VoiceOver?"

"HEARING\_AID\_TITLE" => "Hearing Aids"

"FLASH\_LED" => "LED Flash for Alerts"

"VISION" => "Vision"

"CONFIRM\_ZOOM\_USAGE" => "Zoom"

"DEFAULT" => "Default"

"MOBILITY\_HEADING" => "Physical & Motor"

"TRIPLE\_CLICK\_TITLE" => "Triple-click Home"

"RightStereoBalanceIdentifier" => "R"

}

"ACCESSIBILITY\_DEFAULT\_HEADSET" => "Incoming Calls"给了我们非常明显的提示，以它为线索继续查找。

#### 6. 定位Accessibility.plist

FunMaker-4s:~ root# grep -r ACCESSIBILITY\_DEFAULT\_HEADSET /System/Library/PreferenceBundles/AccessibilitySettings.bundle

Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Accessibility.plist matches

Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Dutch.lproj/Accessibility~iphone.strings matches

……

除了一个plist文件外，其他都是strings文件，那就是它了。看看它里面有什么：

snakeninnys-MacBook:~ snakeninny$ plutil -p ~/Accessibility.plist

{

"title" => "ROOT\_LEVEL\_TITLE"

"items" => [

0 => {

"label" => "VISION"

"cell" => "PSGroupCell"

"footerText" => "AUTO\_CORRECT\_TEXT"

}

1 => {

"cell" => "PSLinkListCell"

"label" => "VOICEOVER\_TITLE"

"detail" => "VoiceOverController"

"get" => "voiceOverTouchEnabled:"

}

2 => {

"cell" => "PSLinkListCell"

"label" => "ZOOM\_TITLE"

"detail" => "ZoomController"

"get" => "zoomTouchEnabled:"

}

……

18 => {

"cell" => "PSLinkListCell"

"label" => "HOME\_CLICK\_TITLE"

"detail" => "HomeClickController"

"get" => "homeClickSpeed:"

}

19 => {

"detail" => "PSListItemsController"

"set" => "accessibilitySetPreference:specifier:"

"validValues" => [

0 => 0

1 => 1

2 => 2

]

"get" => "accessibilityPreferenceForSpecifier:"

"validTitles" => [

0 => "DEFAULT"

1 => "HEADSET"

2 => "SPEAKER"

]

"requiredCapabilities" => [

0 => "telephony"

]

"cell" => "PSLinkListCell"

"label" => "ACCESSIBILITY\_DEFAULT\_HEADSET"

"key" => "DefaultRouteForCall"

}

]

}

又是一个标准的 preference specifier plist，而且我们知道了这个配置的setter和getter分别是accessibilitySetPreference:specifier:和accessibilityPreferenceForSpecifier:，可以进入下一环节了。

### 5.3.3 定位函数

根据 preference specifier plist 标准，在选择“Incoming Calls”中的某一行时，其 setter，即 accessibilitySetPreference:specifier: 函数得到调用。但问题随之而来，这个函数存在于 AccessibilitySettings.bundle 里，笔者当时不知道怎么将这个 bundle 加载进内存，因此没法调用这个函数；也不会用 IDA 和 LLDB，在 class-dump 的函数里找了又找，仍没有发现任何线索，感觉这个问题的难度已经超出笔者的能力范围，一时解决不了，还沮丧地给 Shoghian 发了封邮件，如图 5-17 所示。



图5- 17 我和Shoghian之间的交流

这个问题卡了笔者近两个星期，期间笔者一直在想，iOS 能在这个函数里干些什么呢？因为 preferences specifier plist 中提供了 PostNotification 这一方式来通知别的进程配置文件发生了变动，而 AccessibilitySettings 的配置与电话相关，正好也是进程间通信的模式，那么，accessibilitySetPreference:specifier: 的作用会不会是改动配置文件，然后发出一个通知？于是笔者利用limneos 开发的 LibNotifyWatch，在手动改变“来电使用”配置时观察系统中是否出现了相关的通知，没想到，还真让笔者歪打正着了：

FunMaker-4s:~ root# grep LibNotifyWatch: /var/log/syslog

Nov 26 00:09:20 FunMaker-4s Preferences[6488]: LibNotifyWatch: <CFNotificationCenter 0x1e875600 [0x39b4b100]> postNotificationName:UIViewAnimationDidCommitNotification object:UIViewAnimationState userInfo:{

Nov 26 00:09:20 FunMaker-4s Preferences[6488]: LibNotifyWatch: <CFNotificationCenter 0x1e875600 [0x39b4b100]> postNotificationName:UIViewAnimationDidStopNotification object:<UIViewAnimationState: 0x1ea74f20> userInfo:{

……

Nov 26 00:09:21 FunMaker-4s Preferences[6488]: LibNotifyWatch: CFNotificationCenterPostNotification center=<CFNotificationCenter 0x1dd86bd0 [0x39b4b100]> name=com.apple.accessibility.defaultrouteforcall userInfo=(null) deliverImmediately=1

Nov 26 00:09:21 FunMaker-4s Preferences[6488]: LibNotifyWatch: notify\_post com.apple.accessibility.defaultrouteforcall

……

笔者发现了2条名为“com.apple.accessibility.defaultrouteforcall”的通知！结合前面的一系列推导，想来没有必要再多作解释了。发现了最可疑的通知后，面对的就是另一个同样重要的问题：配置文件在哪里？

第 2 章说过，“/var/mobile/” 中存放了大量用户数据。“/var/mobile/Containers/” 中全是 App相关数据，“/var/mobile/Media/”中全是媒体文件，而在“/var/mobile/ Library/”中稍加浏览就很容易发现 “/var/mobile/Library/Preferences/”目录，进而找到“com.apple. Accessibility.plist”，其内容如下：

snakeninnys-MacBook:~ snakeninny$ plutil -p ~/com.apple.Accessibility.plist

{

……

"DefaultRouteForCallPreference" => 2

"VOTQuickNavEnabled" => 1

"CurrentRotorTypeWeb" => 3

"PunctuationKey" => 2

……

"ScreenCurtain" => 0

"VoiceOverTouchEnabled" => 0

"AssistiveTouchEnabled" => 0

}

在 iOS 中改变“来电使用”的配置，观察 DefaultRouteForCallPreference 值的变化规律， 很容易得出结论：0 对应 default，1 对应 headset，2 对应 speaker，与Accessibility.plist的内容吻合。

### 5.3.4 测试函数

在经过漫长的推理之后，笔者总算得出了一个可能的解决方案，仅需极少代码即可修改配置文件，然后发出一个通知，就这么简单。这个方案可行吗？怀揣一颗惴惴不安又蠢蠢欲动的心，用激动的双手敲出了为数不多的几行代码（那时候还不会用 Cycript，所以用 tweak 测试）：

%hook SpringBoard

- (void)menuButtonDown:(id)down

{

%orig;

NSMutableDictionary \*dictionary = [NSMutableDictionary dictionaryWithContentsOfFile:@"/var/mobile/Library/Preferences/com.apple. Accessibility.plist"];

[dictionary setObject:[NSNumber numberWithInt:2] forKey:@"DefaultRouteForCallPreference"];

[dictionary writeToFile:@"/var/mobile/Library/Preferences/com.apple. Accessibility.plist" atomically:YES];

notify\_post("com.apple.accessibility.defaultrouteforcall");

}

%end

编译、运行、安装、respring，闭着眼睛按下 home 键，然后伴着极快的心跳依次打开“Settings” →“General” →“Accessibility” →“Incoming Calls”——已选项变成了“Speaker”，成功啦！

### 5.3.5 编写实例代码

程序的核心功能已经验证完毕，写代码就不用费脑子了。按照 SBSettings toggle 的编写规范完成代码（http://thebigboss.org/guides-iphone-ipod-ipad/sbsettings-toggle-spec），完整代码如下：

#import <notify.h>

#define ACCESSBILITY @"/var/mobile/Library/Preferences/com.apple.Accessibility. plist"

// Required

extern "C" BOOL isCapable() {

if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber\_iOS\_5\_0 && [[[UIDevice currentDevice] model] isEqualToString:@"iPhone"])

return YES;

return NO;

}

// Required

extern "C" BOOL isEnabled() {

NSMutableDictionary \*dictionary = [[NSMutableDictionary alloc] initWithCont entsOfFile:ACCESSBILITY];

BOOL result = [[dictionary objectForKey:@"DefaultRouteForCallPreference"] intValue] == 0 ? NO : YES;

[dictionary release];

return result;

}

// Optional

// Faster isEnabled. Remove this if it's not necessary. Keep it if isEnabled() is expensive and you can make it faster here.

extern "C" BOOL getStateFast() {

return isEnabled();

}

// Required

extern "C" void setState(BOOL enabled) {

NSMutableDictionary \*dictionary = [[NSMutableDictionary alloc] initWithCont entsOfFile:ACCESSBILITY];

[dictionary setObject:[NSNumber numberWithInt:(enabled ? 2 : 0)] forKey:@"D efaultRouteForCallPreference"];

[dictionary writeToFile:ACCESSBILITY atomically:YES]; [dictionary release];

notify\_post("com.apple.accessibility.defaultrouteforcall");

}

// Required

// How long the toggle takes to toggle， in seconds.

extern "C" float getDelayTime() {

return 0.6f;

}

因为程序的创意来自 Shoghian，所以笔者在发布这个程序时也标注了他的名字（如图 5-18所示）。他很高兴，我们还成为了朋友，偶尔也天南地北地扯上一会儿。Speaker SBSettings Toggle 是笔者发布在Cydia 上的第三个程序，虽然功能简单，也没有作什么宣传，但还是累积了近10000的下载量（如图5-19所示），对此笔者已经很满意了。更重要的是，这个 tweak的制作历经坎坷，看似简单的功能却让笔者花费了九牛二虎之力，无疑给了当时刚刚上路，有些轻飘飘的笔者当头一棒！类似的情况出现过若干次后，笔者才意识到仅仅使用class-dump来做逆向工程是不靠谱的，也间接促使笔者下定决心学习IDA和LLDB，从而迈入了iOS逆向工程的新阶段。

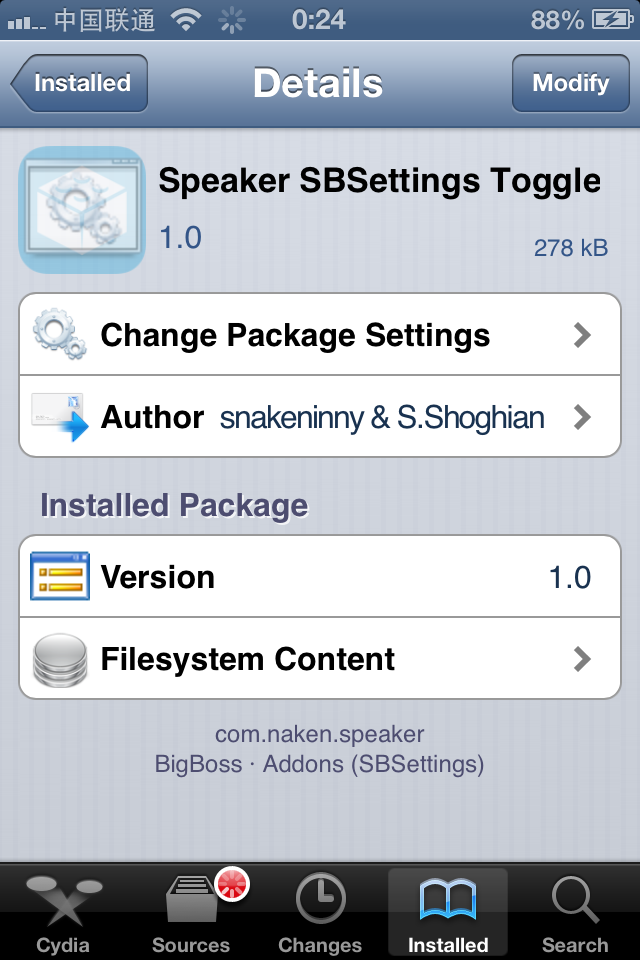


图5- 18 第二作者是Shoghian

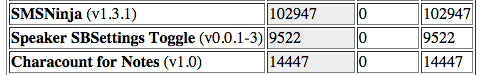


图5- 19 积累了近10000下载量

## 5.4 小结

本章较为完整地介绍了 tweak 的作用原理及编写简单 tweak 的思路和流程，佐以真实案例，能够较好地为初学者提供参考。Objective-C 级别的逆向工程是 iOS 逆向工程的第一关，在没有上手IDA和LLDB之前，对iOS的逆向工程不可能深入到什么地步，也没有什么逻辑可言，相信你从案例里也看出来了，我们对二进制文件的逆向非常力不从心，当问题的关键集中在代码上时，解决问题的主要方式就是猜！虽然刚才编写的代码跟iOS自身的实现差了十万八千里，但因为Objective-C函数名的可读性高，所以即使是猜，也还是能利用class-dump出的函数达到预期效果，给自己带来跟App开发完全不同的感觉，让人耳目一新。

在逆向工程初学阶段，我们的主要目的是熟悉越狱iOS环境，了解前几章讲到的各种逆向知识点，在掌握各种工具用法的同时有意识地培养自己的逆向思维。如果时间比较充裕，强烈建议大家通览class-dump出的头文件，把那些语义明显、自己感兴趣的函数放到iOS上实测一下，这个过程能极大地增加你对iOS底层的熟悉程度，配合后续的IDA与LLDB学习，能达到事半功倍的效果。只要我们多思考、勤练习，就能早日提炼出更适合自己的方法，进一步达到更高的水平。