HZ BOOKS
华章科技

信息安全
技术丛书

iOS应用逆向工程
分析与实战

沙梓社　吴航　刘瑾　著

iOS App Reverse Engineering
Analysis and Practice

- 国内第一本讲解iOS应用逆向工程的实战手册，作者毫无保留地分享了数年来在iOS逆向领域的经验
- 内容系统深入，实战性强，从iOS系统架构、文件类型等逆向理论出发，通过多个实用案例，结合class-dump、Theos、Cycript、Reveal、IDA、GDB等最常用逆向工具阐述了对iOS应用进行逆向分析的思路、方法和技巧

机械工业出版社
China Machine Press

本书结构合理、行文风趣、内容翔实、配图丰富。如果你是中高级iOS开发者，本书一定会让你大开眼界。当你在探究iOS系统底层或别人的App却迷失方向时，本书会告诉你该怎么办，该使用何种工具和策略。逆向爱好者、安全研究人员也能从书中了解到iOS的安全架构和App的攻防策略。iOS开发小白呢，更要入手一本做参考了，不然你开发的App分分钟就被暴露在别人的“显微镜”下了。如果你想缩短和高手之间的差距，想让你的App更安全，读一读此书吧。

——卢明华　友盟 iOS高级工程师

随着移动互联网浪潮的到来，iOS应用开发技术得到了大家广泛的关注，相关的书籍也非常丰富。但是iOS应用安全领域却像一块处女地，鲜有资料涉及。应用安全相关知识是一把双刃剑，黑帽子用它来攻击那些不懂得防范的菜鸟，白帽子用它来保证用户的隐私和维护自己的商业利益。本书从黑帽子攻击的角度，讲解了如何对iOS应用进行逆向工程。通过本书，作者揭开了那些不为人知的iOS应用攻击手段，也将会使得大家更加注意iOS应用的安全问题，最终促进整个业界对于安全的重视。

—— 唐巧　猿题库 iOS高级工程师

逆向工程是一个可以让激情洋溢的程序员快速了解底层结构而成长为高手的快速途径，可惜长久以来，这个领域里的优秀人才都不曾出来向大家分享些学习的经验和体会，直到本书的面世。本书提供了很多贴近现实的实例和平易近人的讲解，帮助你更容易地进入这个领域。如果你想在软件领域有更深的造诣，可以跟着这本书调试书中的实例，一步步地了解iOS逆向工程的基本要领和操作流程。逆向工程多年来一直是我研究各种系统的得力工具，所以我把这本书推荐给大家。想象一下当初一直研习气宗的令狐冲，在知道剑宗存在之后的感觉吧！

—— 邹天矢　360手机卫士前iOS高级工程师

如果你打算编写越狱环境下的iOS插件，或者想从一个全新的角度来审视现有的iOS应用，请从这里开始。本书覆盖范围十分广泛，从class-dump、IDA等常用工具到iOS ARM汇编，囊括了目前相关领域使用的主流方法和技术，完整介绍了iOS应用逆向工程中的各个部分。书中所讲述的实战案例，通过对WhatsApp、Instagram等知名应用进行逆向分析，展现了逆向工程的巨大威力。相信本书将会为广大iOS应用开发者打开一扇新的大门。

—— 沙杰　搜狗号码通iOS高级工程师

投稿热线：(010) 88379604
客服热线：(010) 88378991　88361066
购书热线：(010) 68326294　88379649　68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn

上架指导：计算机/信息安全
ISBN 978-7-111-45072-6
9 787111 450726 >
定价：69.00元

iOS应用逆向工程

分析与实战

iOS App Reverse Engineering
Analysis and Practice

沙梓社　吴航　刘瑾　著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

iOS 应用逆向工程：分析与实战 / 沙梓社，吴航，刘瑾著 . —北京：机械工业出版社，2014.1
（信息安全技术丛书）

ISBN 978-7-111-45072-6

Ⅰ. i…　Ⅱ. ①沙…　②吴…　③刘…　Ⅲ. 移动终端 – 应用程序 – 程序设计　Ⅳ. TN929.53

中国版本图书馆 CIP 数据核字（2013）第 295103 号

版权所有 · 侵权必究
封底无防伪标均为盗版
本书法律顾问　北京市展达律师事务所

本书是 iOS 应用逆向工程方面的权威著作，三位作者都是 iOS 领域内的专家，拥有扎实的理论知识和丰富的实践经验。本书内容以工具 + 代码的形式全面、系统地展开知识点，由浅入深，图文并茂地带着读者一步步探索常规 iOS App 之外的世界。

本书分为四大部分，分别是概念、工具、理论和实战。前三部分介绍 iOS 逆向分析领域的背景、知识体系，以及相应的工具集、理论知识；第四部分则通过 4 个实际案例来将前面的知识以实战的方式展开。第一部分为概念篇，简单介绍 iOS 逆向分析的概念以及 iOS 平台系统架构。第二部分为工具篇，介绍一系列基于 Mac 和 iOS 平台的配套工具，并且重点讲解其中的 class-dump、Theos、Reveal、IDA、GDB 等 5 个工具的使用方法，前 3 个侧重于使用，后 2 个侧重于分析。第三部分为理论篇，主要讲述 iOS 逆向 / 越狱方向的进阶必备理论知识。第四部分为实战篇，通过对 3 个 App Store App 及 1 个系统 App 进行逆向分析的实战操作，让读者能够了解并同步实践已掌握的知识。

机械工业出版社（北京市西城区百万庄大街 22 号　　邮政编码　100037）
责任编辑：罗词亮
藁城市京瑞印刷有限公司印刷
2014 年 1 月第 1 版第 1 次印刷
186mm × 240mm · 18 印张
标准书号：ISBN 978-7-111-45072-6
定　　价：69.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzjsj@hzbook.com

华章计算机

HZBOOKS | Computer Science and Technology

推荐序一

前一段时间跟吴航在微博上私信聊天，他说正在写一本 iOS 方面的书，我让他等书出来后送一本给我。前两天他发私信告诉我书写完了，让我给写个序，我当即表示“压力山大”，但还是欣然答应了。

我认识吴航是在 2011 年 9 月，当时安全管家在找 iOS 开发高手。吴航作为我们安全管家 iOS 开发组的第一个工程师进来了，从零开始搭建 iOS 团队，负责包括安全管家越狱方向在内的 iOS 开发。到了 2012 年年中，我认识到 iOS 自身的安全性非常好，在非越狱的 iOS 上我们能做的关于安全的事情并不多，而越狱行为本身就是一个最大的安全风险，是用户主动选择的结果，跟我们自身的安全理念不符，因此无须投入太多关注，恰好吴航本人也想出去做自己的事情，就支持了他的决定。

在那大半年的接触中，我发现吴航是个难得的技术人才，在技术的专研上有股子狠劲，拥有丰富的开发实战经验，又善于利用各种工具解决问题，因此在他带团队的时候，评估出来的开发进度基本上都能达成。我印象深刻的有两件事，第一件是开发越狱版安全管家。这方面官方公开的资料几乎没有，涉及很多系统底层，因此需要自己摸着石头过河并反复尝试。当时我们制定了一个比较紧的开发周期，希望在较短的时间内开发出越狱版安全管家的原型。吴航的压力不小，他接连几个月都在研究系统底层，向各路高人请教，通过 Google 找寻国外网站上的资料，没日没夜地想办法，后来终于在既定的时间内完成，让我看到了吴航不惧困难、敢于负责任的劲头。另一件事是开发 iOS 版安全管家，有个版本在我的手机上在不同页面间快速切换时有极小的概率会导致安全管家崩溃，我就反馈给吴航。虽然只是个小概率事件，但他亲自反复高强度测试，细致地排查代码，最终揪出了导致这个问题的一个内存指针 bug，足见其严谨的技术态度和对质量高标准的追求。

我虽然不做开发很多年了，但是至今不忘年轻时作为一个工程师，非常渴望与更高水平的人交流，希望聆听高手们的实战经验分享。吴航愿意把他的经验总结成书，是广大 iOS 开发者的福音，这本书能够带给大家实实在在的干货，让大家都能在技术的道路上再攀高峰。

赵岗

安全管家创始人

推荐序二

在生活中，我们经常会忽略许多习以为常的事物。事实上，那些我们每天都与之打交道的东西，往往都蕴含了一种“复杂”的美感：它们由成百上千的微小组件构成，各个微小组件各司其职，在各自的岗位上发挥着不可替代的关键作用。现代生活中，智能手机已经成了我们每天必不可少的工具，通过硬件、软件和固件协同合作，它为我们带来了好玩的游戏、有趣的照片以及便利的沟通渠道——电话和短信。

Core

在一个巴掌大的手机里，各个组件之间的关系错综复杂，互相影响。硬件为固件的运行提供支撑平台，固件掌管软件，而软件又回过头来调度硬件。如果你能控制它们之中的哪怕一个，不就可以让手机听命于你了吗？但 App Store 的插手，给你对它们的控制加上了重重阻力。

本书从独特的角度剖析 iOS 应用，你会从比 App Store App 更低一级的深度去了解软件的各个组件在构造整个软件时起到的作用。你会由此发现你手机的“里世界”，它的能力远不止 App Store 所许可的那样有限，确切地说，它是一台功能齐全的计算机，在它的“里世界”里，一切皆有可能。

少年，从这里开始打破 App Store 的限制，重新认识真正的 iPhone 吧！

Dustin L. Howett

iPhone tweak 开发者

In our lives, we pay very little attention to things that work. Everything we interact with hides a fractal of complexity—hundreds of smaller components, all of which serve a vital role, each disappearing into its destined form and function. Every day, millions of people take to the streets with phones in their hands, and every day hardware, firmware, and software blend into one contiguous mass of games, photographs, phone calls, and text messages.

It holds, then, that each component retains leverage over the others. Hardware owns firmware, firmware loads and reins in software, and software in turn directs hardware. If you could take control of one of them, could you influence a device to enact your own desires?

iOS App Reverse Engineering: Analysis and Practice provides a unique view inside the software running on iOS ™ , the operating system that powers the Apple iPhone® and iPad®. Within, you will learn what makes up application code and how each component fits into the software ecosystem at large. You will explore the hidden second life your phone leads, wherein it is a full-fledged computer and software development platform and there is no practical limit to its functionality.

So, young developer, break free of restricted software and find out exactly what makes your phone tick!

Dustin L. Howett

iPhone Tweak Developer

自　　序

我是一个热爱自助旅游的人。在大学的每个寒暑假，我都会抽出 7 ~ 10 天的时间，挑选国内的一个地方当一次背包客。因为是自助游，没有导游帮忙安排一切，所以在出行前，我和同伴需要花费不少时间制定计划、确认路线、购买车票，考虑路上可能出现的状况，并思考对策。都说旅游能够开阔眼界，尤其是自助游，对此我深有体会：在路上见到的人和事让我增长见识，更重要的是，在开始这段旅程前，我就需要对这段旅程中的点点滴滴有所准备——当身体还站在起点时，我的头脑就已经到达终点了。这种思维方式对全局观的培养是有利的，也让我在思考其他问题时养成了从长计议的习惯。

在 2009 年攻读硕士研究生前，我就曾深入思考过自己想要从事的研究方向。我学习的是计算机专业，而从本科开始，身边绝大部分同学的研究平台是 Windows。作为一名动手能力并不强的普通学生，如果我继续从事 Windows 的研究，有两点好处：

- 这个方向有海量成熟资料，我的学习之路上不会缺少参照；
- 研究人数众多，碰到问题可以请教讨论的人比比皆是。

但是，从另一个侧面来看，这两点“好处”也并非有百利而无一害：

- 参考的资料越多，意味着我会越多地重蹈前人的“覆辙”；
- 研究人数越多，意味着竞争压力更大。

总的来说，如果我从事与 Windows 相关的工作，起步会很顺利，但后续难保不被淹没在人海里；如果另寻他路，入门会很辛苦，但坚持下去或能独辟蹊径。

幸运的是，我的想法与导师的如出一辙。他推荐我选择当时国内“小荷才露尖尖角”的移动开发方向，而我在这之前一直使用的是一款飞利浦蓝屏手机，对智能手机毫无概念，更别说在上面开发软件了。但是，导师是我仔细分析所有硕士生导师特点、与数名学长交换意见之后谨慎挑选的师从对象，他的判断本身就含有我的判断，因此，我相信这个判断，于是开始搜寻移动开发的相关资料。仅仅是了解了一些移动互联网和智能手机的概念，我就隐隐发现，这个行业顺应了人们对计算机和互联网更小、更快、与生活融合更紧的历史发展趋势，一定大有可为，遂将研究方向定为 iOS。

万事开头难，iOS 与我熟悉的 Windows 有着太多太多的不同：类 UNIX、完整生态系统、全封闭、Objective-C 语言，还有影响我最深的“越狱”，这一切的一切在当时几乎找不到完整的参考资料，有半年多近一年的时间，我折腾黑苹果的时间要以星期为单位。我硬着头皮把《Objective-C 基础教程》上宛如天书一般的 Objective-C 代码敲入 Xcode，然后运行模拟器看效果，但代码和画面完全对不上号；对 iOS 上似 UNIX 非 UNIX 的东西（如后台运行）大肆在 Google 上搜索，屡败屡战。当同学们都发表了第一篇小论文时，我甚至没明确自己这个月究竟在干什么，我缺乏太多的基础知识；当同学们周末出去 K 歌、打牙祭时，我一个人闷在宿舍里对着电脑啃代码；当同学们躺在床上睡懒觉时，我一个人一大早爬起来去实验室。一个人是孤独的，但这种孤独换来的是学识的积累，从而转化成内心的笃定，到了最后，因为内在的充实，就不再感到外在的孤独了。男人因孤独而优秀，付出一定会有回报。经过一年多的磨合，在 2011 年 3 月的一天，以前晦涩难懂的代码突然变得平易近人了，每一句的含义、每一行的关系都变得清楚了，零散的知识点在我的脑袋里被连成了线，整个体系的逻辑慢慢清晰了。于是我快马加鞭，在 2011 年 4 月初完成了毕业设计的程序雏形，并得到了当时对此方向并不抱太大希望的导师的高度评价——“从以前自我感觉良好的优越感变成了肚里有货后的真正自信”，这标志着我对 iOS 研究的正式入门。明白自己在做什么之后，就能有的放矢，研究效率呈几何倍率提高。在这两年里，我知道了 Theos 从而“勾搭”上了作者 DHowett，向 Activator 的作者 rpetrich 讨教过问题，跟 TheBigBoss 的管理员 Optimo 发生过争执，他们是我这一路走来帮我解决实际问题最多的朋友；在开发 SMSNinja 的过程中结识了本书另一作者航哥，在不断深入研究的同时认识了一票做人低调、办事高调的高手，意识到自己并不孤单——我们孤胆，我们并肩。

在本书即将出版的时候回望这 4 年，我不禁庆幸自己当初的选择是正确的。在 iOS 方向 4 年的积淀就足够出一本书，这在 Windows 方向是不可想象的，而 Apple、Google 和 Microsoft 三大巨头的不断发力和市场反馈也直接证明了这个行业一定会是互联网下一个十年的绝对主角，能够亲眼见证并参与其中，我三生有幸。人生苦短，必须果敢，所以，少年，不要犹豫了，快到碗里来吧！

在受到航哥的邀请写作本书时，我是有些犹豫的。中国人口众多，各行各业竞争都很激烈，自己走了那么多弯路，碰了 N 鼻子灰才总结提炼出的这些知识，一股脑儿全都交代出去了，会不会有意无意地培养出更多“竞争对手”？这么做是不是把自己的优势拱手相让了？但是纵观越狱 iOS 的发展历史，从基本的 Cydia 和 MobileSubstrate 到 Theos 这样的开发利器，再到 Activator 这样的神级插件，这些对我影响最为深远的软件无一不是开源的，正是因为这些大牛分享了自己的“优势”，我才能博采众长、逐渐成长；rpetrich 牵头的 tweakweek 和 posixninja 牵头的 openjailbreak 也都把宝贵的独门秘籍大白于天下，让更多的爱好者参与越狱 iOS 生态环境的建设。他们是这个圈子里的一线开发者，他们的优势完全没有因为“分享”而减少。我是一个受益于这个分享链条的人，怎么

能在小有所成之后就过河拆桥、断掉我这一环节呢？况且，我是打算在这条路上继续求索的，如果我不停下，我的优势就会一直保持——我的竞争对手只有我自己。相信我们的分享会帮到很多和当年的我一样在门外苦苦徘徊的开发者，集大家的智慧创造出的作品能够更好地让科技服务于人，而且我也能结交更多志同道合的朋友，精神生活得到更大满足。这也聊可算作是从长计议吧。

啰啰唆唆说了很多，有些严肃，但这也正是我对待科学技术的态度。本书的内容适合国内绝大多数不满足于折腾 App Store 的 iOS 爱好者，通篇干货，童叟无欺，比我的硕士毕业论文要实在得多。更多后续的内容还请关注本书的官方微博 @iOS 应用逆向工程。让我们一起提升中国 iOS 开发者在国际上的地位！

在这里，我要感谢母亲对我事业的全力支持，使我在钻研学术之时能尽可能少地因琐事分心。感谢我的爷爷为我的英语启蒙，良好的英语素养是跟国际同行交流的必要条件。感谢我的导师授我以渔，让我在硕士 3 年经历脱胎换骨的成长。感谢 DHowett、rpetrich 和 Optimo 等大牛对我的无私帮助和尖锐批评，让我在快速成长的同时认识到差距巨大，不敢懈怠。感谢 Daniel、gradetwo、zhuliang、大老、Vincent 等前辈对本书的审核与建议，以及对我这个初学者的点拨。感谢我的家人和朋友们，他们的支持与鼓励是我前进下去的不竭动力。还要感谢我未来的女朋友，你 26 年来的缺席让我能一心一意地学习知识，本书稿费啊有我的一半也有你的一半。事业、亲情、友情、爱情是我等凡人的毕生追求，但往往只能求二争三，不可四者兼得，因为这个原因而有意、无意冒犯、伤害过的人，我欠你们一声“对不起”，感谢你们对我的成全。

最后跟大家分享一首我喜爱的诗，啊！人生多么奇妙。

未选之路

罗伯特·弗罗斯特[⊖]

黄色的树林里分出两条路，
可惜我不能同时去涉足，
我在那路口久久伫立，
我向着一条路极目望去，
直到它消失在丛林深处。

但我却选了另外一条路，
它荒草萋萋，十分幽寂；

⊖ Robert Frost（1874—1963），20 世纪美国最受欢迎的诗人之一、四度普利策奖得主。本篇为其代表诗作，原题为“The Road Not Taken”。——编辑注

显得更诱人、更美丽，
虽然在这两条小路上，
都很少留下旅人的足迹。

那天清晨落叶满地，
两条路都未见脚印痕迹。
呵，留下一条路等改日再见！
但我知道路径延绵无尽头，
恐怕我难以再回返。

也许多少年后在某个地方，
我将轻声叹息把往事回顾：
一片树林里分出两条路，
而我选了人迹更少的一条，
从此决定了我一生的道路。

（谨以此书纪念我已仙逝的外祖父刘汉民、祖母吴朝玉）

snakeninny

前　言

为什么要写这本书

两年前我正式从传统网络设备行业转行进入移动互联网行业，当时正是移动应用开发市场最火爆的时候，创业公司如雨后春笋般成立，社交类 App 更是大受追捧，只要有一个不错的构想就可能拿到千万级投资，高价挖人组队的信息更是让人眼花缭乱。那时我已经开发了几个颇具难度的企业应用类 App，对于那些轻量级的普通社交 App 不是太看得上，想着要玩点比较酷的技术，机缘巧合进入了安全管家（北京安管佳科技有限公司），从零开始组建 iOS 团队，负责包括越狱方向在内的 iOS 开发。

其实 iOS 越狱开发就是基于 iOS 的逆向分析工程，那个时候我并没有这方面的经验，面向的是一个完全未知的领域，不过好在有 Google，国内国外的信息多少还是能够搜到点，而且对于 iOS 开发者，越狱开发 / 逆向分析并不是一个完全隔离的世界，虽然被分享出来的都是些零零散散甚至重复度很高的知识，但是只要投入大量精力，把知识归纳总结，慢慢也就整理出一幅完整的图谱。

然而独自一人学习的过程是孤独的，尤其是遇见困难和问题无人交流，让人一筹莫展。每次一个人扛下所有问题的时候，总是感叹：要是有一个水平不错的交流者该是多么幸福？虽然也可以给 Ryan Petrich（@rpetrich）等一线大牛发邮件请教，但很多在我们看来当时解决不了的难题在这类高手眼中可能就是个低级问题，不苦心钻研一番根本不好意思去问。这个阶段大概持续了有大半年，直到 2012 年在微博上遇到本书的另一作者 snakeninny，那时他还是一个面临毕业的研究生，整天“不务正业”地研究 iOS 底层，而且研究得还相当有深度。我曾和他提过：“你看，有多少人都投入到 App 领域捞钱去了，你咋不去呢？”他说：“小弟的目标远大，要玩就朝着国际一线大牛的目标去！”小兄弟，你够狠！

不过，多数时候我们都是自己在折腾，虽然偶尔在网上交流一下问题及解决方法，但往往能分享得到一些有价值的内容。在一起合写本书之前，我们曾经合作逆向分析过陌陌，做了一个插件用于在陌陌 iPhone 版上把美女的位置标注在地图上，当然我们都是善意的开发者，主动将这个漏洞

告诉了陌陌，他们很快就修复了。这次，我们再次合作，将逆向分析这个方向的知识整理出版，呈现给各位读者。

在接触越狱开发 / 逆向分析的这两年，个人感觉最大的收获就是看待 App 时，完全以一种庖丁解牛的眼光去审视：App 如何构成、性能如何间接反映出开发团队水平高低。这些经验知识不仅可用于越狱开发，也可用于传统的 App 开发，至于带来的影响，有正有负吧。我们不能因为所谓正统的 iOS 平台不提倡越狱就否定这个领域的存在，盲目相信那些安全问题不存在不过是掩耳盗铃。

有经验的开发者都明白，知识掌握得越深，越会接触到底层技术。比如 sandbox 保护机制之外到底有什么？ runtime 只用来研究理论知识是不是有点大材小用了？

在 Android 领域，底层技术已经被扩散开，而在 iOS 领域，这个方向展现出来的内容还只是冰山一角。虽然国外也有几本 iOS 安全方向的书籍，比如《Hacking and Securing iOS Applications》、《 iOS Hacker's Handbook》，但是内容太难，多数人读不出所以然，不适合中国国情，即使我们这些有一定经验的开发者，读这些书也像是在读大学教科书。

阳春白雪不为我们这些喜欢实践的技术宅所好，那么就来点下里巴人的，不必遮遮掩掩，直接全面展开这些知识岂不是更痛快？于是就有了我们这本书，书中的内容以工具 + 代码的形式全面、系统地展开知识点，由浅入深，图文并茂，旨在带着读者一步步地探索常规 iOS App 之外的世界。我们不会像一些技术博客那样貌似很高深的独立分析某一片段的代码，而是尽我们所能将一个完整的知识体系呈现给读者，相信读者一定会有所收获。

国内投入在越狱开发这个方向的人并不算少，但都比较低调，而且多集中在大公司中，比如腾讯、360、91、WeiPhone，他们积累的技术非我们这些散兵游勇所能及，不过我们更愿意先将这些知识分享出来，希望能够抛砖引玉。

读者对象

本书主要面向以下读者：

- 对 iOS 系统有浓厚兴趣的读者；
- 中高级 iOS 开发人员，他们在掌握了普通 App 开发之后对 iOS 有更深的渴求；
- 架构师，因为在逆向 App 的整个过程中，有心的技术人员能看到那些经典 App 在结构设计上的经验、优势，并博采众长，有助于整体架构设计水平的提高；
- 在别的系统上从事逆向工程，想要转向 iOS 系统的逆向工程师。

如何阅读本书

本书将分为四大部分，分别是概念、工具、理论和实战。前三部分介绍 iOS 逆向分析这个领域的背景、知识体系，以及相应的工具集、理论知识；第四部分则具体以 4 个实际案例将前面的知识以实战的方式展开，让读者可以实践验证前面学到的知识并进一步更扎实地掌握相应的知识。

如果读者不具备一定的逆向分析经验，建议还是从头开始按顺序阅读，而不要直接跨越到第四部分去模拟实战。虽然实战的成果似乎比较炫，但弄个半懂不懂也没意思对不对？

第一部分为概念篇，将简单介绍 iOS 逆向分析的概念以及 iOS 平台系统架构。

第二部分为工具篇，将会介绍一系列基于 Mac 和 iOS 平台的配套工具，并且重点讲解其中的 class-dump、Theos、Reveal、IDA、GDB 等 5 个工具的使用方法，前 3 个侧重于使用，后 2 个侧重于分析。

第三部分为理论篇，除了介绍 iOS 逆向 / 越狱这个方向的理论知识之外，还顺带聊了一点 iOS 逆向 / 越狱方向的国际一线大牛们的八卦，轻松一下。

第四部分为实战篇，将通过对 3 个 App Store App 及 1 个系统 App 进行逆向分析的实战操作，让读者能够了解并同步实践已掌握的知识。

勘误和支持

由于作者的水平有限，编写的时间也很仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正，欢迎微博联系 @iOS 应用逆向工程，或访问我们的技术博客 http://iosre.com，我们将尽量为读者提供满意的解答。如果你有更多的宝贵意见，也欢迎你通过微博或技术博客与我们联系，我们很期待能够听到你们的真挚反馈。

致谢

首先要感谢 Jay Freeman（@saurik）这样的顶级高手，他奠定了越狱 iOS 的基石。还要感谢 Dustin Howett（@DHowett），是他提供的 Theos 这样强大的开发工具让我得以迈进 iOS 逆向分析的大门。

感谢安全管家，为我进入 iOS 逆向领域提供了一个能充分发挥的环境，虽然我早已离开，但希望它发展得更好。

感谢微博上每一位热心的朋友——唐巧_boy、卢明华、你在瓦西里、isdada、Jagie、

onevcat、戴铭、费西 FISH、xuzhanji、Life 无法 Debug、移动开发小冉、HorseLuke、网络蝎子、hongjiang_wang、月之舞狼、StayNStay、bluesea 哈哈哈、郑州 IOS、青年土豆的烦恼、木土吉吉，以及这个仓促写就的名单之外的更多朋友，感谢他们对我的支持和鼓励。特别感谢唐巧 _boy 的引荐，他的热心帮助促成了本书的合作与出版。

感谢机械工业出版社华章公司的编辑杨绣国老师，感谢她的魄力和远见，在这 3 个月的时间里始终支持我的写作，她的指点和帮助引导我们顺利完成全部书稿。

谨以此书献给我最亲爱的家人，以及众多热爱 iOS 开发的朋友们。

吴航（@hangcom2010）

教师服务登记表

尊敬的老师：

您好！感谢您购买我们出版的_____教材。

机械工业出版社华章公司为了进一步加强与高校教师的联系与沟通，更好地为高校教师服务，特制此表，请您填妥后发回给我们，我们将定期向您寄送华章公司最新的图书出版信息！感谢合作！

个人资料（请用正楷完整填写）

教师姓名		□先生 □女士	出生年月		职务		职称：□教授 □副教授 □讲师 □助教 □其他
学校			学院			系别	
联系电话	办公： 宅电： 移动：			联系地址及邮编			
				E-mail			
学历		毕业院校		国外进修及讲学经历			
研究领域							

主讲课程	现用教材名	作者及出版社	共同授课教师	教材满意度
课程： □专 □本 □研 人数： 学期：□春□秋				□满意 □一般 □不满意 □希望更换
课程： □专 □本 □研 人数： 学期：□春□秋				□满意 □一般 □不满意 □希望更换

样书申请			
已出版著作		已出版译作	
是否愿意从事翻译/著作工作 □是 □否		方向	
意见和建议			

填妥后请选择以下任何一种方式将此表返回：（如方便请赐名片）

地 址：北京市西城区百万庄南街1号 华章公司营销中心 邮编：100037

电 话：(010) 68353079 88378995 传真：(010)68995260

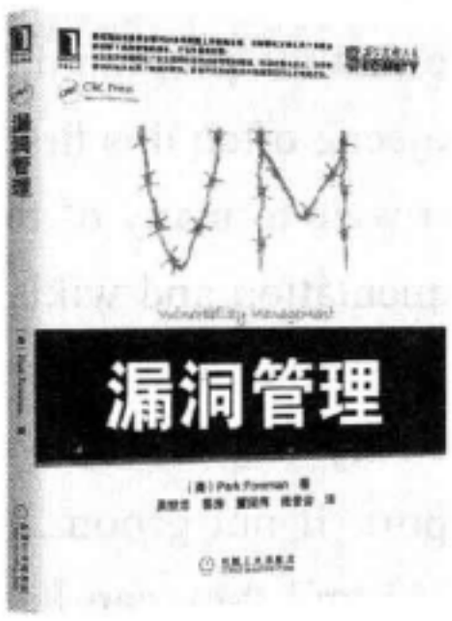
E-mail:hzedu@hzbook.com markerting@hzbook.com 图书详情可登录http://www.hzbook.com网站查询

推荐阅读



Android取证实战：调查、分析与移动安全

作者：Andrew hoog
ISBN：978-7-111-42199-3
定价：69.00元



漏洞管理

作者：Park Foreman
ISBN：978-7-111-40137-7
定价：69.00元



渗透测试实践指南：必知必会的工具与方法战

作者：Patrick Engebretson
ISBN：978-7-111-40141-4
定价：49.00元



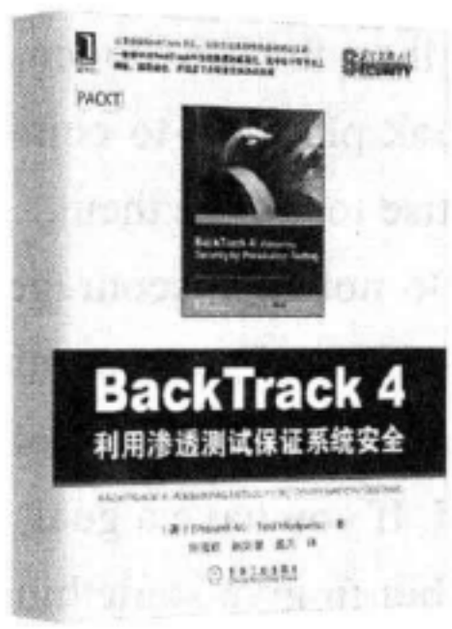
内核漏洞的利用与防范

作者：Enrico Perla 等
ISBN：978-7-111-37429-9
定价：79.00元



网络扫描技术揭秘：原理、实践与扫描器的实现

作者：李瑞民
ISBN：978-7-111-36532-7
定价：79.00元



BackTrack 4: 利用渗透测试保证系统安全

作者：Shakeel Ali
ISBN：978-7-111-36643-0
定价：59.00元

目　录

推荐序一

推荐序二

自序

前言

第一部分　概念篇

第 1 章　iOS 逆向工程简介 …… 2

1.1　iOS 软件逆向工程的要求 …… 2

1.2　iOS 软件逆向工程的作用 …… 2

1.2.1　与安全相关的 iOS 逆向工程 …… 4

1.2.2　与开发相关的 iOS 逆向工程 …… 5

1.3　iOS 软件逆向工程的一般过程 …… 6

1.3.1　系统分析 …… 7

1.3.2　代码分析 …… 7

1.4　iOS 软件逆向工程用到的工具 …… 8

1.4.1　监测工具 …… 8

1.4.2　开发工具 …… 9

1.4.3　反编译器 …… 9

1.4.4　调试器 …… 10

1.5　小结 …… 11

第 2 章　越狱 iOS 平台简介 …… 12

2.1　iOS 系统架构 …… 12

2.1.1　iOS 目录结构 …… 13

2.1.2　iOS 文件权限 …… 15

2.2　iOS 程序类型 …… 16

2.2.1　Application …… 16

2.2.2　Dynamic Library …… 19

2.2.3　Daemon …… 19

2.3　小结 …… 20

第二部分　工具篇

第 3 章　Mac 工具集 …… 22

3.1　class-dump …… 22

3.1.1　class-dump 介绍及下载 …… 22

3.1.2　class-dump 使用演示 …… 23

3.1.3　关于 class-dump 的补充说明 …… 25

3.2　Theos …… 25

3.2.1　Theos 简介 …… 25

3.2.2　Theos 安装及编译 …… 26

3.2.3　Theos 用法简介 …… 28

3.2.4　Theos 开发 tweak 示例 …… 47

3.3 Reveal ····· 49
3.3.1 Reveal 简介 ····· 49
3.3.2 Reveal 安装及功能扩展 ····· 50
3.4 IDA ····· 55
3.4.1 IDA 简介 ····· 55
3.4.2 IDA 使用说明 ····· 56
3.4.3 IDA 分析示例 ····· 65
3.5 其他工具 ····· 68
3.5.1 iTools ····· 68
3.5.2 dyld_decache ····· 69
3.5.3 MesaSQLite ····· 69
3.6 小结 ····· 70

第 4 章 iOS 工具集 ····· 71
4.1 SBSettings ····· 71
4.2 MobileSubstrate ····· 72
4.3 OpenSSH ····· 73
4.4 GDB ····· 74
4.4.1 GDB 简介 ····· 74
4.4.2 GDB 的使用说明 ····· 74
4.5 Cycript ····· 85
4.6 其他常用工具 ····· 88
4.6.1 BigBoss Recommended Tools ····· 88
4.6.2 AppCrackr ····· 88
4.6.3 iFile ····· 89
4.6.4 MobileTerminal ····· 89
4.6.5 Vi IMproved ····· 90
4.6.6 SQLite ····· 90
4.6.7 top ····· 91
4.6.8 syslogd ····· 92
4.7 小结 ····· 92

第三部分 理论篇

第 5 章 Objective-C 相关的 iOS 逆向理论基础 ····· 94
5.1 tweak 的作用原理 ····· 94
5.1.1 Objective-C 语言的特性 ····· 94
5.1.2 MobileSubstrate ····· 96
5.2 tweak 的编写套路 ····· 97
5.2.1 灵感的来源 ····· 98
5.2.2 分析文件，寻找切入点 ····· 99
5.2.3 定位目标函数 ····· 102
5.2.4 测试函数功能 ····· 104
5.2.5 解析函数参数 ····· 105
5.2.6 class-dump 的局限性 ····· 108
5.3 实例演示 ····· 108
5.3.1 得到灵感 ····· 108
5.3.2 分析文件 ····· 109
5.3.3 定位函数 ····· 115
5.3.4 测试函数 ····· 117
5.3.5 编写实例代码 ····· 117
5.4 小结 ····· 119

第 6 章 ARM 汇编相关的 iOS 逆向理论基础 ····· 120
6.1 ARM 汇编基础 ····· 120
6.1.1 基本概念 ····· 121
6.1.2 ARM/THUMB 指令解读 ····· 123
6.1.3 ARM 调用规则 ····· 129
6.2 在 IDA 中分析目标文件 ····· 131
6.2.1 分析函数实现细节 ····· 132
6.2.2 验证分析结果 ····· 139
6.3 在 GDB 中分析目标文件 ····· 142

6.3.1 跟踪代码执行流程 …… 142
6.3.2 动态更改代码 …… 147
6.4 小结 …… 157

第四部分 实战篇

第 7 章 实战 1：去除 OPlayer Lite 的广告 …… 160

7.1 OPlayer Lite 简介 …… 160
7.2 分析源代码结构，并定位广告横幅的代码 …… 161
7.2.1 class-dump 获取 .h 文件 … 161
7.2.2 猜测广告横幅的出现位置 …… 161
7.2.3 分析 PlayView-Controller.h …… 162
7.3 去除广告横幅，节省流量 …… 167
7.3.1 找到“准备”广告的位置 …… 167
7.3.2 追溯到网络连接的起点 … 173
7.4 编写调试代码 …… 175
7.4.1 编写 Tweak_OPlayer-Lite.xm …… 175
7.4.2 编辑 control …… 176
7.4.3 编辑 RemoveOPlayer-Ads.plist …… 176
7.4.4 编辑 Makefile …… 177
7.4.5 编译 + 打包 …… 177
7.4.6 安装 …… 177
7.5 调试去广告插件 …… 177
7.6 代码结果整理 …… 178
7.7 小结 …… 179

第 8 章 实战 2：WhatsApp 消息拦截 …… 180

8.1 WhatsApp 简介 …… 180
8.2 分析源代码结构，并定位监听点 …… 180
8.2.1 class-dump 获取 .h 文件 … 180
8.2.2 导入 Xcode …… 182
8.2.3 找到 WhatsAppApp-Delegate …… 182
8.2.4 找到 ChatManager …… 183
8.3 编写调试代码 …… 184
8.3.1 分析函数列表 …… 184
8.3.2 编写 Tweak_Whats-App.xm …… 185
8.3.3 编辑 control …… 186
8.3.4 编辑 monitor.plist …… 186
8.3.5 编辑 Makefile …… 186
8.3.6 编译 + 打包 …… 187
8.3.7 安装 …… 187
8.4 调试运行监听插件 …… 187
8.4.1 运行 WhatsApp，SSH 连接 …… 187
8.4.2 发送 / 接收消息，监控日志 …… 188
8.4.3 保存数据 …… 189
8.5 代码结果整理 …… 194
8.6 小结 …… 195

第 9 章 实战 3：Instagram 图片保存 …… 196

9.1 Instagram 简介 …… 196

9.2 分析源代码结构，并定位
图片展示点 …… 196
9.2.1 用 class-dump
获取 .h 文件 …… 196
9.2.2 导入 Xcode …… 198
9.2.3 找到 AppDelegate …… 198
9.2.4 注入 Reveal …… 199
9.2.5 寻找图片页相关 class …… 200
9.3 保存图片 …… 205
9.3.1 尝试保存图片 …… 205
9.3.2 对 .h 文件进行瘦身 …… 206
9.3.3 编译测试 …… 208
9.4 弹出提示菜单 …… 208
9.4.1 弹出菜单代码 …… 208
9.4.2 使用 Theos 新增函数 …… 210
9.4.3 使用 runtime 新增函数 …… 211
9.5 代码结果整理 …… 214
9.6 小结 …… 215

第 10 章 实战 4：iOS 电话操作 … 216

10.1 常用电话操作 …… 216
10.2 分析源代码结构，并定位
电话操作的代码 …… 216
10.2.1 class-dump 获取 .h 文件 … 216
10.2.2 寻找拨打电话的函数 …… 217
10.2.3 寻找接听电话的函数 …… 229
10.2.4 寻找挂断电话的函数 …… 234
10.2.5 寻找删除通话记录的
函数 …… 236
10.2.6 寻找拦截电话的方法 …… 238
10.3 编写调试代码 …… 250
10.3.1 编写 Tweak_Phone-
Operation.xm …… 250
10.3.2 编辑 control …… 254
10.3.3 编辑 Phone-
Operation.plist …… 255
10.3.4 编辑 Makefile …… 255
10.3.5 编译 + 打包 + 安装 …… 255
10.4 调试电话操作插件 …… 255
10.5 代码结果整理 …… 257
10.6 小结 …… 257

越狱开发一览 …… 258

沙箱逃脱 …… 263

**编写 tweak——新时代的
hacking …… 265**

第一部分

概 念 篇

软件的逆向工程指的是通过分析一个程序或系统的功能、结构或行为，将它的技术实现或设计细节推导出来的过程。当我们因为工作需要，或是对一个软件的功能很感兴趣，却又拿不到它的源代码时，往往可以通过逆向工程的方式对它进行分析。

对于 iOS 开发者来说，运行在 iOS 平台上的各种软件是我们知晓的最复杂且超奇妙的虚拟物品之一，它们精巧而细致，新颖且创意十足。作为开发人员，在使用这许多经典的 App 的同时，除了会称赞 App 的实现效果之外，也会经常思考：它们的架构是如何搭建的？它们的运行机制又具体是什么样的？

第 1 章

iOS 逆向工程简介

虽然从理论上来说 iOS App 是一个运行于沙箱（sandbox）中的程序，但并不表示我们就对它束手无策，事实上，可以通过逆向工程来一窥其究竟。接下来，就引入 iOS 软件逆向工程的概念。

1.1 iOS 软件逆向工程的要求

iOS 软件逆向工程指的是在软件层面上进行逆向分析的一个过程。读者如果想要达到对 iOS 软件较强的逆向分析能力，最好能非常熟悉 iOS 设备的硬件构成、iOS 系统的运行原理，还要具备丰富的 iOS 开发经验。比如：拿到一个 App 之后能够清晰地推断出这个 App 使用的技术，包括引用了哪些 framework、哪些经典的第三方代码，以及整个 App 工程大致的文件个数、大致的代码行数，另外，还要想象一下如果由自己来组织设计开发，需要投入的人员、时间和精力等。

这要求高吗？其实真的不算高！不过，这些条件都是充分非必要的。如果你目前还不具备这些技术条件，那么有两个必要条件则一定要满足：强烈的好奇心和锲而不舍的精神。因为在 iOS 软件逆向工程中，好奇心会驱动你去钻研经典的 App，而在钻研的过程中会遇到一系列的困难和障碍，各种各样的技术坑，别人埋的、自己挖的，此时就需要锲而不舍的精神来支撑你一步步向前走。但是请相信，在投入大量精力去编写代码、调试程序、分析逻辑之后，你会在不断的试验和错误中感受到逆向工程的艺术之美。

1.2 iOS 软件逆向工程的作用

打个比喻，iOS 逆向工程就像一杆长矛，专门用于刺破目标程序本以为安全的保护盾，而且还是从想象不到的角度来刺破。

对于微信、陌陌、WhatsApp 之类的 IM 软件，交流的信息是它们的核心；对于银行、交

易支付、电商类的软件，交易和客户信息是它们的核心。所有的核心数据都是需要重点保护的，于是，数据库加密、网络连接加密、云存储等，开发人员通过各种手段重重保护，为的就是让使用者能够高枕无忧。

可是 iOS 逆向工程对这些常规 App 的攻击不是来自同一个维度！这些攻击可用一个词来定义：维度攻击（“维度攻击”的概念来自科幻小说《三体》，在此向《三体》致敬）。

从维度的角度来解释，我们看待常规 App 的感觉更像是将 App 的所有数据、UI、逻辑内容都平展地铺开在一张二维的大纸上，外围圈上厚厚的城墙。它的平面图概念如同图 1-1 所示的谷歌地图一般。



图 1-1　飞到二维城市的上空来俯瞰整座城市

但是当我们跳跃到城墙之外，在三维的天空中俯瞰这个 App 的二维城堡，这个城堡的内部结构几乎完全开放在眼前，包括所有的 Objective-C 函数定义、所有的接口数据、甚至所有的函数内执行代码。城墙的防护意义已经荡然无存！此时限制我们的只是这个二维平面城堡到底有多大，里面的内容到底有多丰富，以及排列是整齐有序的还是杂乱无章的。

此时，基于逆向工程的技术能力，可以选择想进入的任意点“高维进入”，监视甚至改变二维平面上的运行逻辑，从而达到获取核心信息或软件设计原理等战术目的，而不是从二维世界打破外围城墙强行攻破。

说得似乎很玄乎，但事实上，就笔者对十余款经典 App 以及 iOS 系统进行逆向的经历过程来看，逆向工程的使用确实无一例外地达到了目标。

上面的比喻虽有些不恰当，但也形象地说明了 iOS 逆向工程的强大能力。现在回到正题，概括起来，iOS 逆向工程主要有两个作用：

- 攻破目标程序，拿到关键信息，可以归类于与安全相关的逆向工程；

❑ 借鉴他人的程序功能来开发自己的软件，可以归类于与开发相关的逆向工程。

接下来就让我们将这些内容逐步展开。

1.2.1 与安全相关的 iOS 逆向工程

从前面的内容介绍已可以大致了解，iOS 逆向工程最突出的应用领域就是与安全相关的。比如：工程师通过逆向一个金融类 App，来评定这个软件的安全等级；安全专家通过逆向 iOS 病毒，来找到杀毒的方法；安全公司通过逆向 iOS 系统电话、短信功能，来构建一个手机防火墙……

1. 评定安全等级

iOS 中那些具有交易功能的 App 一般会先加密敏感数据，然后将加密过的数据存储在本地或通过互联网传输。而如果安全意识不够强，就完全有可能将重要的用户信息（如银行账号和密码）直接用明文保存或传输，安全隐患极大。

假如一个有名望的公司考虑推出一款 App，为了让自己 App 的质量能够对得起用户的信任，该公司请一家安全公司帮忙评估这个 App 的安全性。这家安全公司自然是不会被授权使用这个 App 源代码的，也就是说，正向分析其安全性不可行，所以他们只有通过逆向工程，来尝试“攻击”这个 App，然后依据结果评定其安全等级。

补充说明：前面在介绍“维度攻击”的概念时已经提及，App 虽然可以将低维的攻击防守得如铜墙铁壁一般，但是挡不住高维的逆向攻击。不过不可以据此得出 App 不安全的结论，因为 iOS 逆向工程的使用均来自一个前提：iOS 越狱。普通用户可能不知道，在将 iOS 设备越狱之后，即将整个平台都置于风险之中，此时也就更谈不上 App 的绝对安全了。在这种环境下，我们使用这些逆向工程技术来分析评估目标 App 中可被攻击点有多少，可攻击点越少的自然就是越安全的。

2. 逆向恶意软件

iOS 是智能移动终端操作系统，它同计算机操作系统已没有本质区别。从第一代开始，它就已具备了上网功能，而互联网正是恶意软件传播的最好媒介。2009 年暴露的 Ikee 病毒是 iOS 上公开的第一款蠕虫病毒，它会感染那些已经越狱并且安装了 SSH，但是又没有更改其默认 root 密码“alpine”的 iOS 设备，并且将它们的锁屏背景图改成一个英国歌手的照片。

对于恶意软件的开发者来说，他们通过逆向工程定位系统和软件漏洞，利用漏洞渗透进目标主机，获取敏感数据，甚至为所欲为。

对于杀毒软件的开发者来说，他们通过逆向工程剖析病毒样本，观察病毒行为，尝试杀掉被感染主机上的病毒，并总结出可以防范病毒的方法。

3. 检查软件后门

开源软件的一大好处是其具有较好的安全性。成千上万的程序员浏览并修改开源软件的代码，代码中几乎不可能存在任何后门，软件的安全问题往往在大白于天下之前就能及时得到解决。而对于闭源软件，逆向工程是检查其软件后门的方法之一，比如我们常会在越狱的

iPhone 上安装 360、金山、腾讯等各家手机安全助手，这些服务程序的权限极高，无一例外都拥有超强的能力，可以访问用户的通话记录、短信记录甚至更多，但我们不知道它们会不会收集不该收集的用户信息，并把这些信息上传到自己的服务器。这种“有或没有”的结论一直是个争议，我们得不到可信的官方数据，那么就只能通过逆向工程来检查该软件是否存在类似的后门。

补充说一句，这些手机安全助手的开发者们本身也是逆向工程的高手，善用矛的同时也善用盾，想要达到同台 PK 的水平需要积累锻炼足够长的时间才行。

4. 去除软件使用限制

iOS 开发者通过 App Store 或 Cydia 出售自己的 App，作为他们最主要的经济来源之一。在软件的世界里，收费与盗版永远是同时存在的，而不少 iOS 开发者也在自己的 App 里加入了反盗版的功能。但这场矛与盾的战争永远不会停止，再好的防御也会有被攻破的一天，防止盗版几乎是一项不可能完成的任务。比如，Cydia 上最知名的“共享”源 xsellize 能够在几乎所有收费软件发布后的 3 天内对其完成破解，让他们的用户免费享受到收费软件的功能。

1.2.2 与开发相关的 iOS 逆向工程

对于 iOS 开发者来说，逆向工程是最为实用的技术之一。例如，工程师可以逆向系统调用，在自己的程序里使用一些文档中没有提及的私有功能，还可以逆向一些经典的软件，包括竞争对手的软件，从而研究自己的软件能够借鉴些什么技术。

1. 逆向系统调用

程序员编写的软件能够运行在操作系统中，提供各种各样的功能，是因为操作系统本身已经内嵌了这些功能，软件只是拿来重新组合使用罢了。众所周知，能在 App Store 上架的 App 的功能十分有限，在苹果公司严格的审核制度下，绝大多数 App 的实现都源于公开的开发文档，而不能使用诸如锁屏、关机等文档中不涉及的功能。如果你的程序面向 Cydia，那么不采用非公开功能将导致程序几乎没有竞争力。如果你的程序想拥有打电话、发短信等不公开的功能，最有效的途径就是逆向 iOS 系统调用，还原系统实现相应功能的代码，并应用到自己的程序中。

2. 借鉴别的软件

逆向工程最受欢迎的应用场合就是“借鉴”他人的软件功能。对于来源是 App Store 的 App 来说，大多数技术的实现并不复杂，巧妙的创意和良好的运营才是其成功的关键。如果只是单纯借鉴其功能，那么采用逆向工程，费时费力，性价比不高，从头开发一个功能类似的软件反而要省时省力得多。但是，当我们不知道被借鉴软件中的某个功能是如何实现的时候，逆向工程就能起到关键性的作用。这种情况在大量使用私有函数的 Cydia 软件中尤其常见，比如 2013 年 3 月面世的，号称 iOS 平台上第一款通话录音软件的 Audio Recorder，足够有趣，但我们不知其原理如何，此时使用逆向工程的技术就能够对它了解一二。

有些软件，比如微信，功能经典，结构也非常复杂，所涉及的代码自然十分多。图 1-2

是使用 class-dump 获取微信 4.5 的 .h 文件目录。

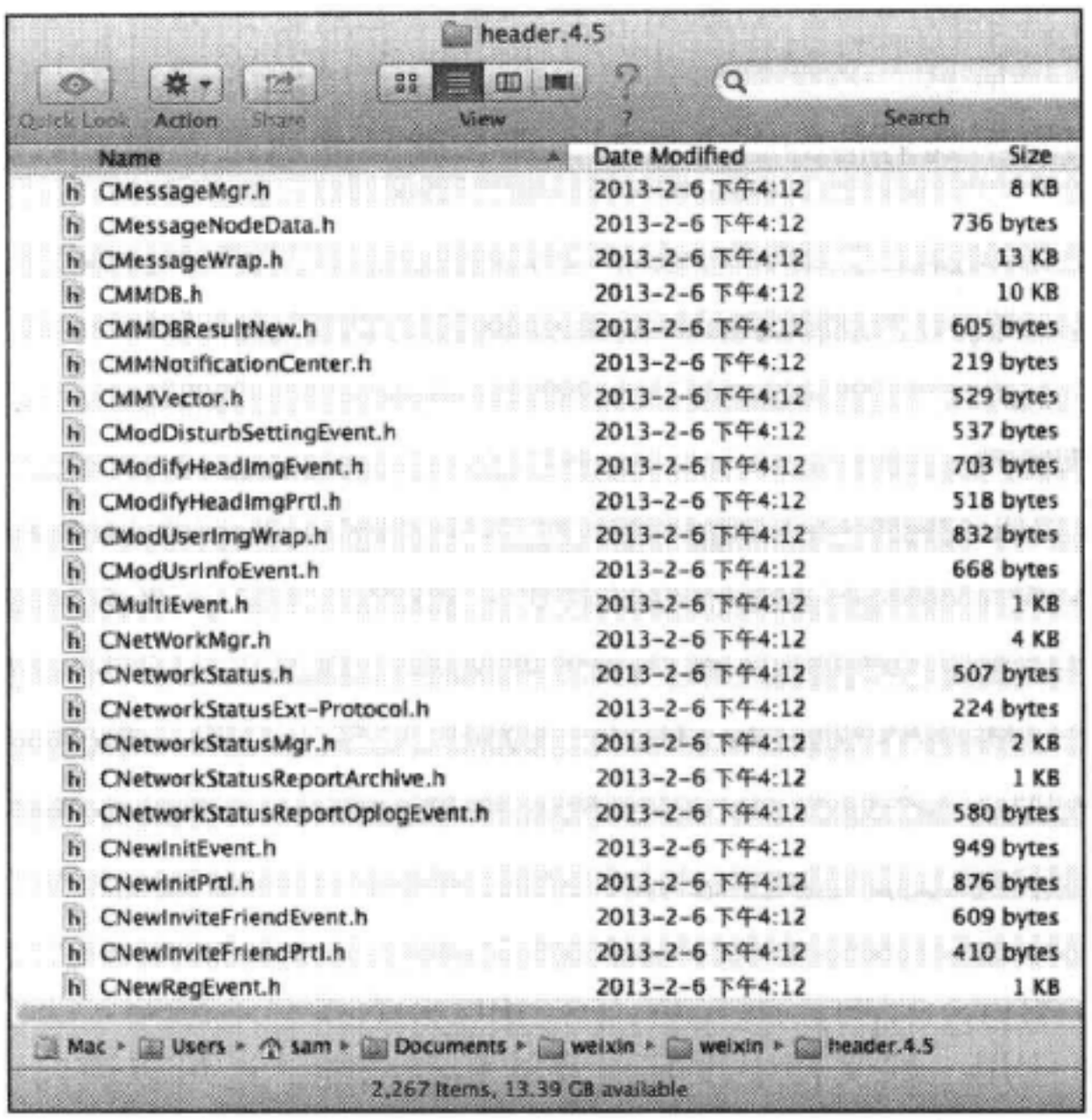


图 1-2 通过 class-dump 获取的微信 4.5 的 .h 文件目录

可以看到，在目录中共有 2267 个 .h 文件，光是将文件名一个个地看下来，天都要黑了。那这种情况下应该怎么办？当然不可能这样傻傻地一个文件一个文件地读过去，事实上，这时便可以通过合适的逆向分析工具加上丰富的开发经验来帮助定位关键信息点。对于微信这样足够经典的目标程序，使用的技术及框架都非常有借鉴意义，所以也值得我们对它进行逆向工程，收集相关的知识并学习，进而一步步地掌握其精髓。

1.3 iOS 软件逆向工程的一般过程

当我们想要逆向一个 iOS 软件时，应该从何入手？应该怎么思考？这本书的初衷就是引导初学者走进 iOS 逆向工程的大门，培养读者从逆向的角度思考问题。

所谓的 iOS 逆向工程初学者，其实至少也是 iOS App 开发的中高级程序员了，理论上都拥有相当丰富的开发经验。所以在后续的过程中，对于一些非常初级的技术原理，可能会讲得非常简单，甚至直接被忽略，毕竟读那些内容对有经验的开发人员来说是在浪费时间。

一般来说，软件逆向工程可以看做系统分析和代码分析两个阶段的有机结合。在系统分析阶段，要从整体上观察目标程序的行为特征、文件的组织架构，从而找到我们感兴趣的地方。一旦能够较为熟练地操作这个程序，并且找到了我们想要逆向的点，就可以进入代码分析阶段，利用各种工具对程序本身的二进制文件进行分析，达到我们的目的。

1.3.1　系统分析

在系统分析阶段，应在不同的条件下运行目标程序，在程序中进行各种各样的操作，观察程序的行为特征，同时寻找我们感兴趣的功能点。比如：选择哪个选项会弹框，按下哪个按钮会发声，输入什么内容屏幕会有什么显示；同时，还可以浏览文件系统，观察程序显示的图片、程序的配置文件存放在哪里，对应的数据库文件中存放了哪些信息、是否加密等。

以新浪微博 App 为例，当我们将微博的目录展开时，会看到如图 1-3 所示的结构。在微博目录下有 Documents、Library 等子目录，这些目录中放着对应的数据库文件以及缓存的图片文件等。

另外，在 Documents 目录下还可以看到 db_36000_3414827754.dat 这样的文件，从文件名上即可识别出它是数据库文件，此时使用 SQLite 工具打开便可看到如图 1-4 所示的数据内容。

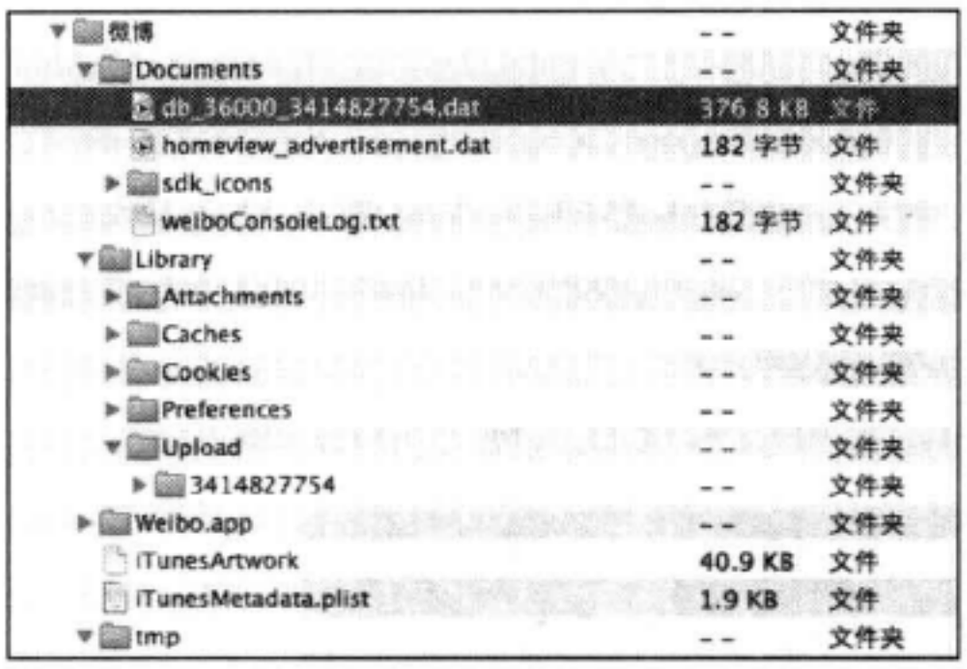


图 1-3　新浪微博 App 的 Documents 目录

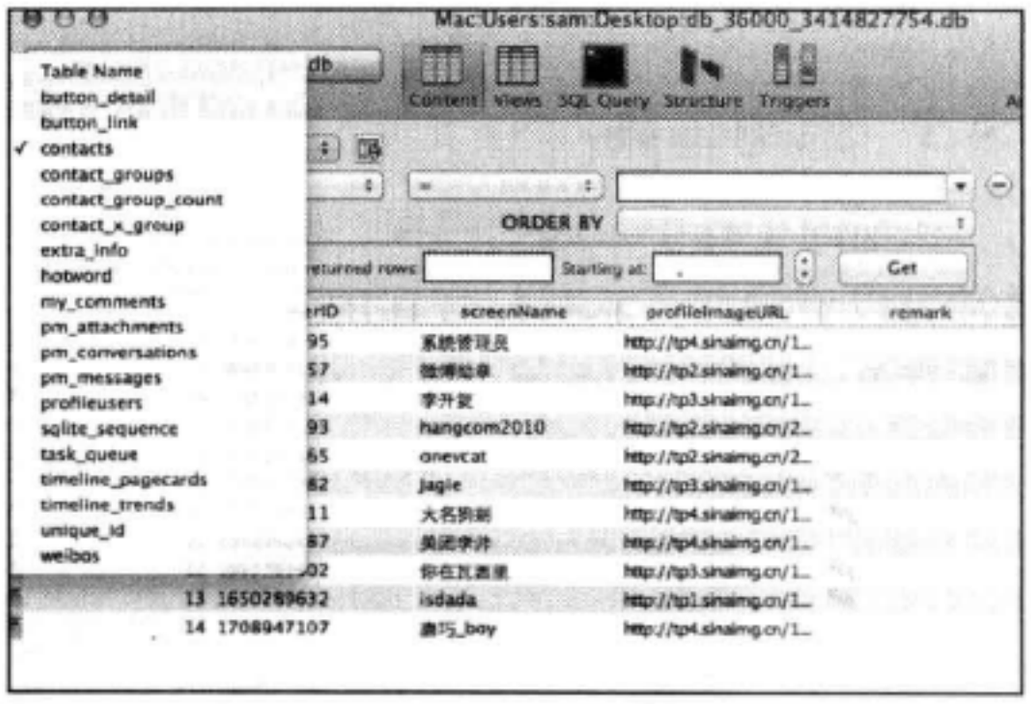


图 1-4　新浪微博 App 的个人数据文件

此时，我们能够看到在新浪微博里的联系人、所发的微博内容等信息。

因为这些信息已经通过 App 自身的目录文件结构呈现在我们面前，所以可以看到，对 iOS 平台以及 App 的理解程度直接决定了我们在逆向工程时所需要花费的时间和精力，以及所获取信息的性价比。

1.3.2　代码分析

在外围的分析完成之后，接下来就是对程序的可执行文件进行代码这一层级的分析了。

使用逆向分析的方法可以倒推出这个程序的设计思路、内部算法和实现细节，这是一个非常复杂的过程，甚至可以说是一种解构的艺术。要想让自己的逆向工程水平达到艺术的高度，需要对软件开发、CPU 工作原理和 iOS 有透彻的理解。而且很多软件的复杂度很高，即使其代码规范、文档详尽，可是一旦开源，很多人还是会因其具有的复杂度而惊叹，比如 Activator 这样的系统级插件程序。一点一点地分析程序的底层指令绝非易事，也不是一本书能够完全阐述清楚的。

本书的目标仅仅是向初学者讲述 iOS 逆向工程入门时所用到的工具和可能的思路，书中

所提及的工具及知识均是笔者长期积累的个人经验之谈，对知识点的覆盖不可能面面俱到，期望读者能结合书中的理论和实例举一反三，形成自己的学习方法。

1.4 iOS 软件逆向工程用到的工具

对于 iOS 的逆向工程，除了掌握理论知识之外，最重点的内容就是使用好各种工具了。整个 iOS 逆向工程所涉及的工具数量达到数十款之多，而且这还未 100% 覆盖到。因为毕竟在这一方向没有一套统一的教学规范来指明哪些工具该用，哪些工具不该用，各个参与者都是根据自己掌握的深度及喜好来选择最称手的工具的。

好的工具，以及对这些工具的熟练使用能够极大地协助我们进行高效的逆向工程。在本阶段的逆向工程中，介绍的工具集主要可以分为四大类：监测工具、开发工具、反汇编器（disassembler）、调试器（debugger）。

1.4.1 监测工具

在逆向工程中，起到嗅探、监测、记录目标程序行为的工具统称为监测工具。这些工具通常可以记录并显示目标程序的某些操作，如网络活动、文件访问等。iOS 逆向常用的监测工具有 Reveal、tcpdump、libNotifyWatch 和 PonyDebugger 等。

图 1-5 中所呈现的是一款监测工具——Reveal，它可用来查看目标 App 的 UI 元素。

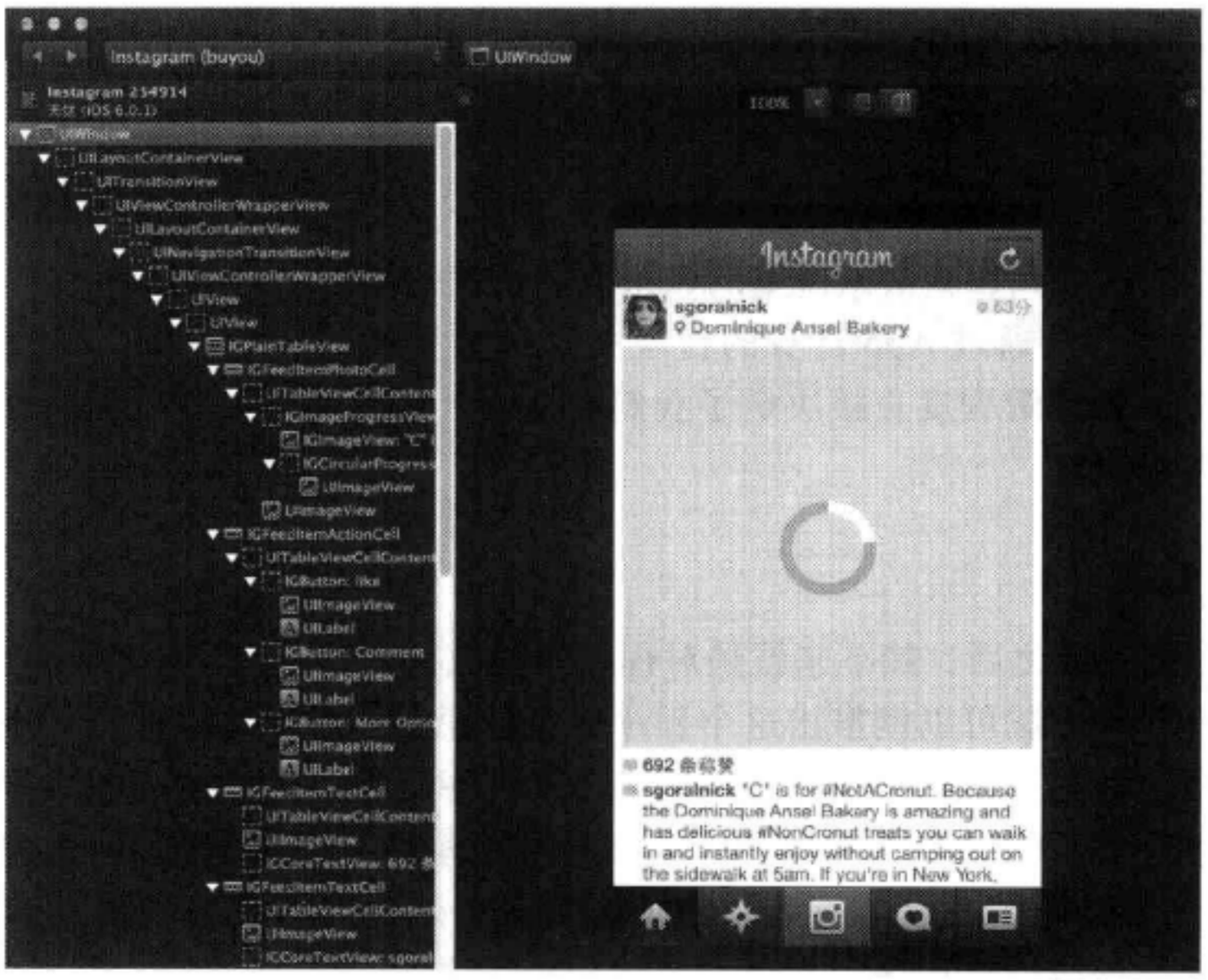


图 1-5 Reveal——用来查看目标 App 的 UI 元素的监测工具

使用 Reveal 可以非常直观地看见 App 的界面上各个 UI 元素之间的从属关系，并且轻

松、快捷地定位到我们想查看的 UI 元素。有了类似这样的工具，很多我们原本以为会非常枯燥的分析工作均可以轻松搞定，达到事半功倍的效果。

1.4.2 开发工具

进入代码分析这一层级，就必须掌握开发工具的使用了。实际上监测工具与开发工具一直都是组合使用的，单纯使用某一类工具根本无法深入。

对于有成熟开发经验的 iOS 程序员来说，Xcode 是最常用的开发工具，但在此之外，还有可以用于开发越狱代码的工具，其中最方便也最有名的一个工具就是 Theos。这是一个非常简单易用的开发工具，但是它能编译生成最具威力的 tweak 插件，直刺 App 心脏，完成我们期望它完成的任务。

图 1-6 是 Theos 的创建工程命令行。

```
sammatoMacBook-Pro:Desktop sam$ cd /opt/theos
sammatoMacBook-Pro:theos sam$ ls
LICENSE         bin             extras          include         lib             mod
Prefix.pch      documentation   framework       include.zip     makefiles       templates
sammatoMacBook-Pro:theos sam$ cd bin
sammatoMacBook-Pro:bin sam$ ./nic.pl
NIC 1.0 - New Instance Creator
------------------------------
  [1.] iphone/application
  [2.] iphone/library
  [3.] iphone/preference_bundle
  [4.] iphone/tool
  [5.] iphone/tweak
Choose a Template (required): 5
Project Name (required): monitor
Package Name [com.yourcompany.monitor]: com.hangcom.monitor
Author/Maintainer Name [sam]:
MobileSubstrate Bundle filter [com.apple.springboard]:
Instantiating iphone/tweak in monitor/...
Done.
sammatoMacBook-Pro:bin sam$
```

图 1-6 Theos——tweak 开发工具

在后面的工具篇中会详细介绍 Theos 的使用方法。仅从个人体验来说，这是让笔者最为兴奋的开发工具，在接触、使用 Theos 之前，笔者感觉自己一直都被限制在普通 App 的低维世界中，直到学会并掌握了 Theos 的使用方法，才觉得真正突破了普通 App 的开发界限，跨越了维度的限制，可以从高维来看待 App 了。

1.4.3 反编译器

随着自己能力的一步步进阶，反编译器就会出现在我们的视野中。在 iOS 逆向工程领域，反编译器主要就是 IDA。

IDA 是逆向工程中最常用的工具之一，横跨 Windows/Linux/Mac 平台，它会把目标程序的可执行文件作为输入，然后输出这个程序的汇编代码，甚至伪代码。

反编译是跟 CPU 架构相关的，但像 IDA 这样强大的反编译器支持多种 CPU 架构，如图 1-7 所示。

IDA 有免费版本，可以输出函数列表及对应的汇编代码。而相较于免费版的 IDA，收费版本所能带来的功能就更为震撼了。图 1-8 是 IDA Pro（收费版）的截图，对于 iOS App，它

能做到的不仅仅是反汇编，甚至连伪代码都能高度接近。

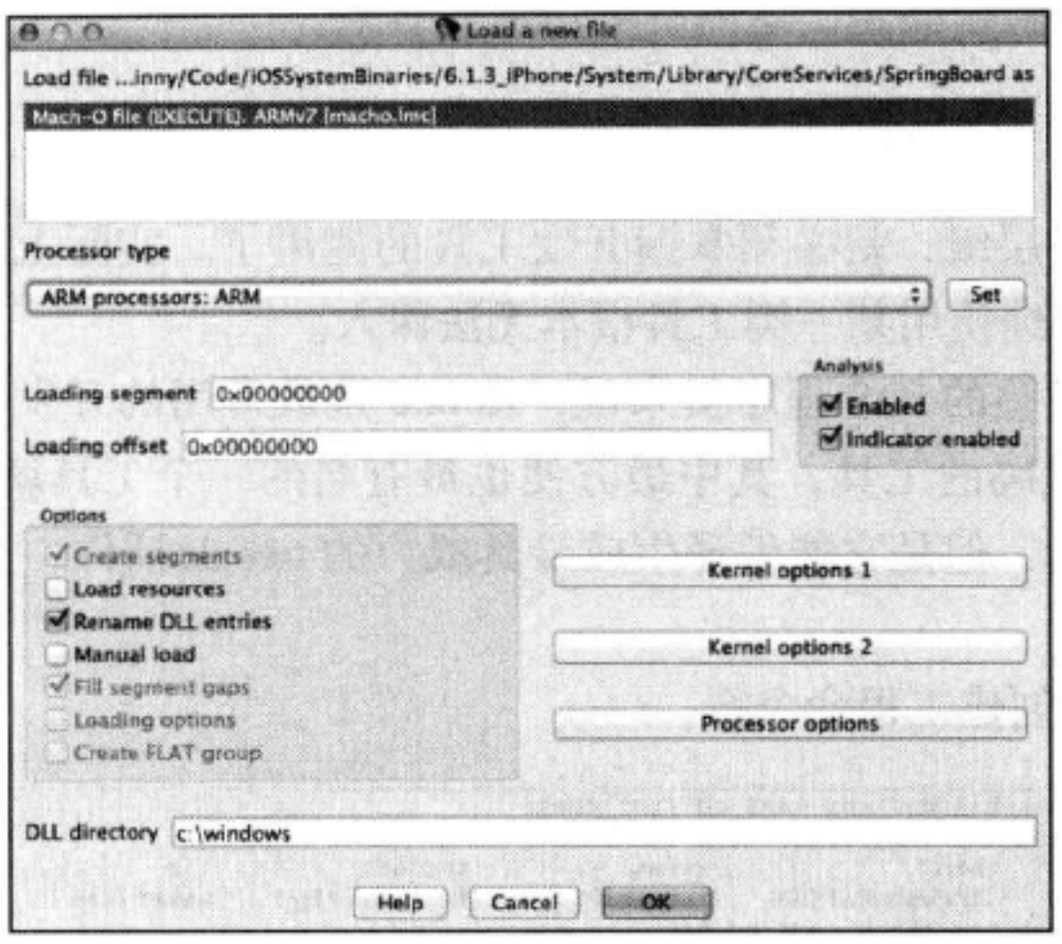


图 1-7 IDA——强大的反编译器



图 1-8 IDA 反编译 App 的伪代码

笔者第一次看到这样的伪代码时，不禁感叹：逆向分析工具都强大到了这一步，还有什么是能藏得住的？

1.4.4 调试器

对程序员来说，调试器是相当熟悉的了，最常见、最熟悉的就是在 Xcode 中的单步调试，程序员可以在某一行代码上设置断点，使调试器能够在那一行代码执行之前暂停程序执行，并且显示程序当前的状态。

因为程序员拥有自己程序的源代码，所以调试器的断点可以指定在源代码的代码行上，单步调试可以以源代码的行为单位执行代码。而熟悉 Xcode 调试的人也都知道，其实所谓

Xcode 的调试器，不过就是把 GDB（GNU Debugger）或 LLDB 做了一层图形化的包装，集成在一起更好用而已。

但是在逆向工程中，我们往往没有目标程序的源代码，而且需要在 iOS 设备上进行调试，此时就需要在 iOS 设备上运行基于命令行的 GDB 了。图 1-9 是使用 GDB 进行调试的示例。

```
275 Social                                F 0x340ec000        dyld Y Y /System/L
ibrary/Frameworks/Social.framework/Social at 0x340ec000 (offset 0x127f000)
                                                              (objfile is) [memory o
bject "/System/Library/Frameworks/Social.framework/Social" at 0x340ec000]
276 libsandbox.1.dylib                    - 0x3a1f5000        dyld Y Y /usr/lib/
libsandbox.1.dylib at 0x3a1f5000 (offset 0x127f000)
                                                              (objfile is) [memory o
bject "/usr/lib/libsandbox.1.dylib" at 0x3a1f5000]
277 libMatch.1.dylib                      - 0x398cd000        dyld Y Y /usr/lib/
libMatch.1.dylib at 0x398cd000 (offset 0x127f000)
                                                              (objfile is) [memory o
bject "/usr/lib/libMatch.1.dylib" at 0x398cd000]
gdb$ b *(0xb5000 + 0x00009218)
Breakpoint 2 at 0xbe218
gdb$ c

Breakpoint 2, 0x000be218 in _mh_execute_header ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ ni
0x000be218 in _mh_execute_header ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
```

图 1-9　GDB——调试利器

当然，我们同样可以利用调试器设置断点、单步调试，来观察甚至更改程序的执行流程，从而达到分析程序的目的，只不过这个过程不如拥有代码调试那么直观、方便。

1.5　小结

本章介绍了 iOS 软件逆向工程的基本概念、作用和一般的逆向过程，以及需要用到的一些有代表意义的工具等，旨在让读者对 iOS 逆向工程有一个概念上的了解，详细的技术内容会在后面的章节中逐一讲解。

第 2 章

越狱 iOS 平台简介

相较于 iOS 应用的高层表象，人们对其底层实现更感兴趣，这也是大家进行逆向工程的原动力。

但是，我们也都知道，原版 iOS 是个对开发者封闭的黑盒子，直到 Redsn0w 等越狱组织对 iOS 进行破解、越狱之后，这个盒子才被揭开，神秘的 iOS 系统得以完全展现在我们面前。

2.1 iOS 系统架构

对于原版（即未越狱）iOS，Apple 官方并未提供直接访问 iOS 文件系统的任何入口，iOS 开发者只需要遵循规定，参考文档即可完成工作，所以，纯粹的 App Store 开发者可能对 iOS 系统的目录结构一无所知。

因为沙箱及权限的关系，从 App Store 下载的 App 不能访问除自己目录以外的绝大多数文件。这时，可以借助 Xcode，通过点击 Xcode → Organizer → Devices 浏览指定 App 的目录结构，如图 2-1 所示。但是，这种方式也仅能访问 /var/mobile/Applications 下的 App 目录，而对系统里的其他目录仍是束手无策。

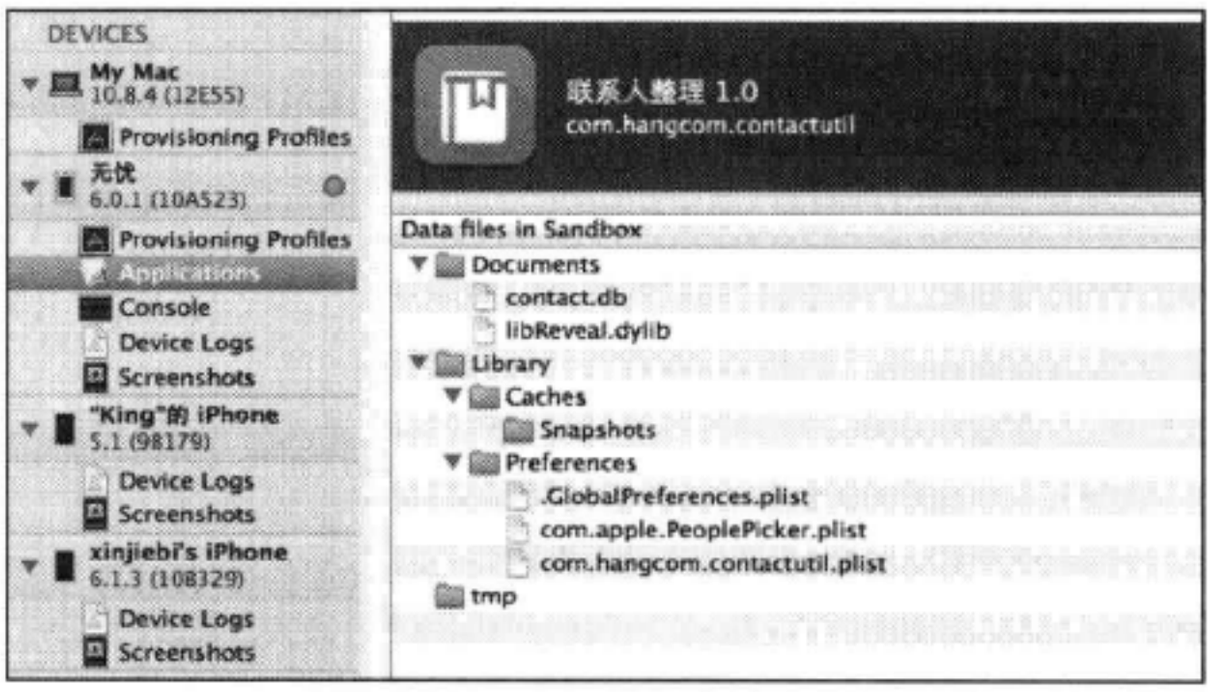


图 2-1　使用 Xcode 查看 App 的目录结构

在越狱 iOS 中，用户可通过安装强大的第三方文件管理工具，如 iFunBox、iExplorer 或 iFile 等，访问 iOS 全系统文件。图 2-2 是用 iFile 查看 iOS 文件系统的界面，它可以直接读取和修改系统中的任何文件，不用再顾虑严格的沙箱与权限限制。



图 2-2　iFile 浏览文件目录

至此，逆向工程的大门已经向我们敞开了，一起来看看吧。

2.1.1　iOS 目录结构

iOS 是由 Mac OS X 演化而来的，而 Mac OS X 则是基于 UNIX 操作系统的，这三者虽然有很大区别，但它们血脉相连。从 Filesystem Hierarchy Standard（文件系统层次标准）和 hier(7) 中，可以一窥 iOS 目录结构设计的参考标准。

Filesystem Hierarchy Standard（以下简称 FHS）为类 UNIX 操作系统的文件目录结构制定了一套标准，制定这套标准的初衷之一是为了能让用户预知文件或目录的存放位置；Mac OS X 在此基础上形成了自己的 hier(7) 框架。类 UNIX 操作系统的常见目录结构如下所示。

- / 根目录，以斜杠表示，其他所有文件和目录在根目录下展开。
- /bin：“binary”的简写，存放提供用户级基础功能的二进制文件，如 ls、ps 等。
- /boot：存放能使系统成功启动的所有文件，这些文件一般在内核用户程序开始执行前得到调用。在 iOS 中此目录为空。
- /dev：“device”的简写，存放 BSD 设备文件。每个文件代表系统的一个块设备或字符设备，一般来说，“块设备”以块为单位传输数据，如硬盘；而“字符设备”以字符为单位传输数据，如调制解调器。
- /sbin：“system binaries”的简写，存放提供系统级基础功能的二进制文件，如 netstat、reboot 等。
- /etc：“et cetera”的简写，存放系统脚本及配置文件，如 passwd、hosts 等。在 iOS 中，/etc 是一个符号链接，实际指向 /private/etc。
- /lib：存放系统库文件、内核模块及设备驱动等。iOS 中此目录为空。
- /mnt：“mount”的简写，存放临时的文件系统挂载点。iOS 中此目录为空。
- /private：存放两个目录，分别是 /private/etc 和 /private/var。
- /tmp：临时目录。在 iOS 中，/tmp 是一个符号链接，实际指向 /private/tmp。
- /usr：包含了大多数用户工具和程序。/usr/bin 包含那些 /bin 和 /sbin 中未出现的基础功能，如 nm、killall 等；/usr/include 包含所有的标准 C 头文件；/usr/lib 存放库文件。
- /var：“variable”的简写，存放一些经常更改的文件，比如日志、用户数据、临时文件等。其中 /var/mobile/Applications 下存放了所有 App Store App，是要重点关注的目录之一。

上述目录中的内容多用于系统底层，逆向难度较大，作为初学者，暂时不用在其中投入太

多精力。从学和练的角度出发，循序渐进，由易到难，从我们熟悉的内容开刀，效率更高。

作为 iOS 开发者，日常操作所对应的功能模块大多来自 iOS 的独有目录，如下所示。

- /Applications：存放所有的系统 App 和来自 Cydia 的 App，不包括 App Store App。越狱的过程把 /Applications 变成了一个符号链接，实际指向 /var/stash/Applications，如图 2-3 所示。这个目录也是我们要重点关注的目录之一。
- /Developer：/Developer 相对没那么重要，它的出现完全是因为我们在 Xcode 连接 iOS 设备时选择了“Use for Development”，这个目录的结构如图 2-4 所示。
- /Library：用来存放系统 App 的数据。其中最需要关注的是 /Library/MobileSubstrate 目录，因为这个目录中存放了所有基于 MobileSubstrate 的插件，其结构如图 2-5 所示。

在 iOS 系统中，MobileSubstrate 是一个提供 hook 功能的基础平台，运行在这个平台上的插件通常被称为 tweak。

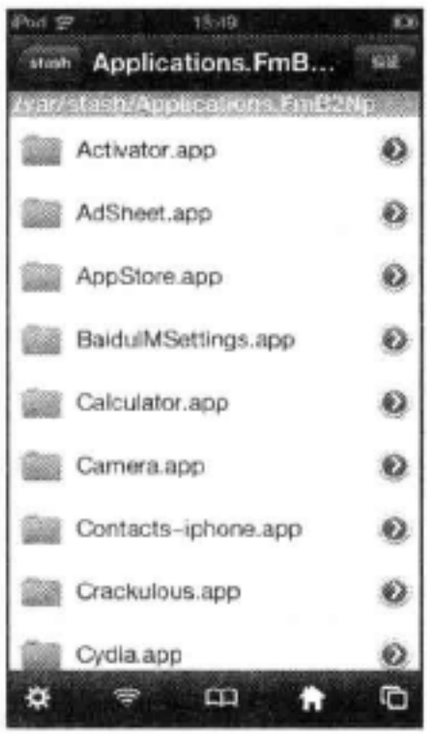


图 2-3 /Applications 目录



图 2-4 /Developer 目录



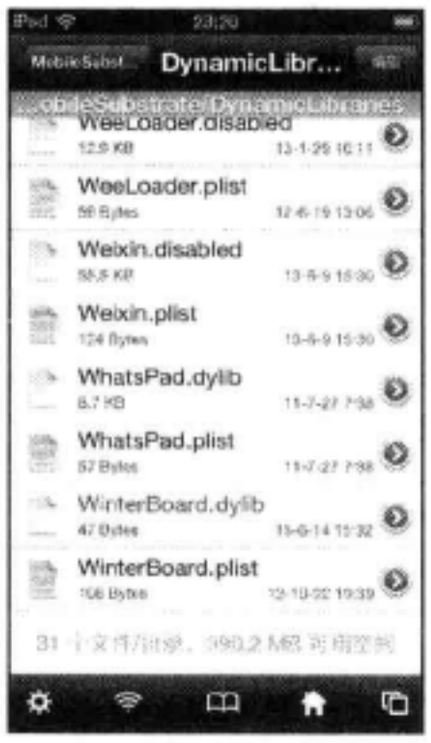


图 2-5 /Library/MobileSubstrate 目录

从图 2-5 中的文件列表中也可看到，/Library/MobileSubstrate 下通常有 3 类文件。

- dylib：即 Dynamic Library，也就是 tweak 插件。
- plist：用于配合 dylib 使用的 filter 文件，指定注入目标，其格式如下（将在第 3 章中详细介绍）：

```
Filter = {
  Bundles = (com.apple.springboard);
};
```

- disabled：被 SBSettings 禁用的 tweak 文件，图 2-5 中的 Weixin.disabled，其实就是把 Weixin.dylib 改了个名字，不让 MobileSubstrate 加载而已。
- /System：iOS 文件系统中最重要的目录之一，包含了大量的系统组件，其目录结构如图 2-6 所示。其中需要重点关注的有：
  - /System/Library/Frameworks 和 /System/Library/PrivateFrameworks：存放 iOS 中的各种 framework，SDK 文档里提及的各种 framework 只是冰山一角，还有数不清的未公开功能等待我们去挖掘。

- /System/Library/CoreServices 里的 SpringBoard.app：也就是桌面管理器，是用户与系统交流的最重要的中介。
- /System/Library/PreferenceBundles：其中的各种 bundle 提供了“设置”中的绝大多数功能，也是 iOS 逆向工程入门阶段练习的好目标。

❑ /User：用户目录，实际指向 /var/mobile，其目录结构如图 2-7 所示。

这个目录里存放了大量的用户数据，比如：

❑ /var/mobile/Media/DCIM：照片目录。
❑ /var/mobile/Library/SMS：短信目录。
❑ /var/mobile/Library/Mail：邮件目录。
❑ /var/wireless/Library/CallHistory：通话记录。这个稍微特殊一点，在 iOS 4 时代，通话记录是存放在 /var/mobile/Library/CallHistory 下的。

另外一个非常重要的子目录就是 /var/mobile/Applications，里面存放的是通过 App Store 下载的 App，如图 2-8 所示。



图 2-6　System/Library 目录



图 2-7　/var/mobile 目录



图 2-8　/var/mobile/Applications 目录

在对 iOS 文件系统的目录结构有了初步了解之后就能够针对感兴趣的功能定位其对应的文件，也可以开展下一步的工作了。

2.1.2　iOS 文件权限

iOS 是一个多用户操作系统。“用户”是一个抽象的概念，它代表对操作系统的所有权和使用权，比如，mobile 用户是无法调用 reboot 命令重启 iOS 系统的，而 root 用户则可以；“组”是用户的一种组织方式，一个组可以包含多个用户，一个用户也可以属于多个组。

iOS 中的每个文件都有一个属主用户和一个属主组，或者说这个用户和这个组拥有这个文件；每个文件都具有一系列权限，简单地说，权限的作用在于说明文件的属主用户能做什么，属主组能做什么，以及其他所有人能做什么。iOS 用 3 bit 来表示文件的使用权，从高位到低位分别是 r（read）权限、w（write）权限以及 x（execute）权限。文件与用户的关系存在

以下三种可能性：

- 此用户是属主用户。
- 此用户不是属主用户，但在属主组里。
- 此用户既不是属主用户，又不在属主组里。

因此，需要 3 × 3 bit 来表示一个文件的权限，其中 1 bit 如果置 1，则代表生效，置 0 代表无效。例如，111101101 代表 rwxr-xr-x，即该文件的属主用户拥有 r、w、x 权限，而属主组和其他所有人只具有 r 和 x 权限；同时，二进制的 111101101 转换成十六进制是 755，也是一种常见的权限表示法。

2.2 iOS 程序类型

越狱 iOS 中最常见的程序有 Application、Dynamic Library 和 Daemon 三类。在当前阶段，逆向工程的目标仍是这三类程序，对它们的了解越深入，逆向工程就会越顺利。这三类是不同类型的程序，分工不同，其目录结构和文件权限也有较大区别。下面一起来看看吧。

2.2.1 Application

Application 就是我们常说的应用程序，简称 App，其意义早已为所有 iOS 开发人员熟知。不过，在本节中还是要对 Application 的概念作一个完整的回顾和补充。

1. bundle

在 iOS 这个体系中，bundle 的概念来源于 NeXTSTEP，其实它就是一个按某种标准结构来组织的目录，其中包含可执行程序以及运行所需的资源。App、framework 和 PreferenceBundle 都是以 bundle 的形式存在的，当你将某个 bundle 确立为逆向目标后，绝大多数逆向资源都可以在 bundle 内找到，大大减少了逆向工程的工作量。

2. App 的结构

在 App Store 中只能发布 App 这一种类型的程序，绝大多数 iOS 开发者对 App 的结构很熟悉；但在越狱 iOS 平台上，还有两类 App 形式的存在：WeeApp（即依附于 Notification Center 的 App）和 PreferenceBundle（即依附于 Settings 的 App），它们的结构与 App 类似，常见于 Cydia 平台，这里也把它们看做是 App。

framework 的结构与 App 类似，但 framework 的 bundle 中存放的是一个动态链接库，而不是可执行程序；相对来说，framework 在逆向工程中的地位比 App 更高，因为一个 App 的绝大多数功能都是通过调用 framework 提供的接口来实现的。

在逆向工程时，主要关注 App 中的以下三个部分。

（1）Info.plist

Info.plist 记录 App 的基本信息，如 bundle identifier（唯一标识符）、可执行文件名、图标文件名等。

这个文件之所以如此重要，是因为后面的开发调试过程中会利用 MobileSubstrate 对 App 进行 hook（钩取、注入），获取正确的 bundle identifier 是成功逆向的第一前提。图 2-9 是 WhatsApp 的 Info.plist。

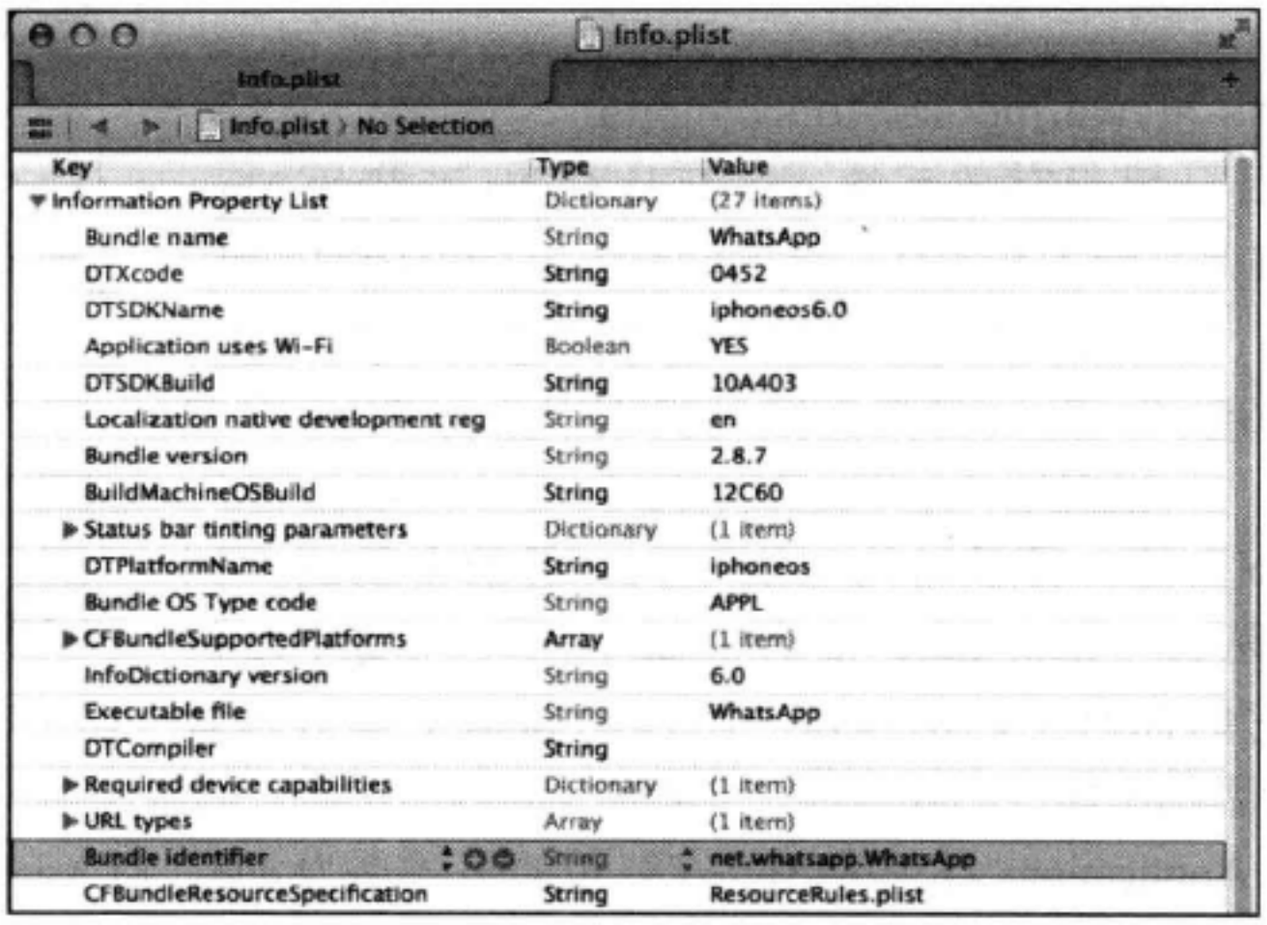


图 2-9　Info.plist 详情

（2）Executable（可执行程序）

App 的执行入口，也是逆向工程最主要的目标之一，得到它之后，马上斧钺加身，class-dump、IDA 还有 GDB 等重磅武器都会攻击它。图 2-10 是笔者开发的一个联系人信息优化 App，打开之后即可在目录结构中看到对应的可执行文件（即 Info.plist 中的“Executable file”）。

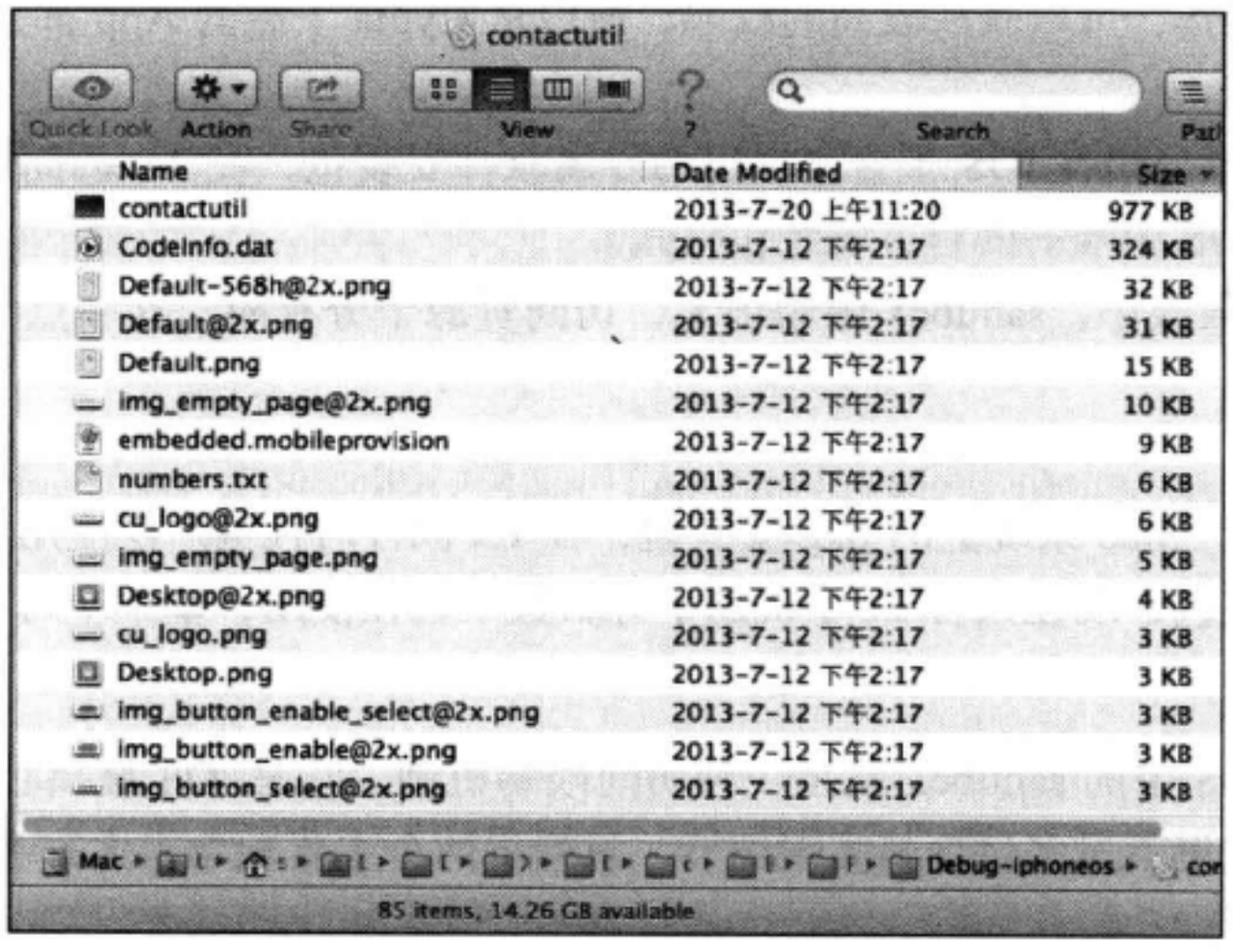


图 2-10　App 的目录内容

（3）Resources（资源文件）

图标、图片、声音、配置文件、nib 文件以及各种 App 运行时用到的资源文件，同样可见于图 2-10。其中，各种本地化字符串（.strings）是定位逆向目标的重要线索。

3. /Applications 和 /var/mobile/Applications

/Applications 目录用于存放系统 App 及从 Cydia 下载的 App，如图 2-11 所示，在此目录下有来自 Cydia 的 iFile.app 和 iOS 自带的 MobileMail.app。



图 2-11 Applications 目录

而 /var/mobile/Applications 目录存放的则是从 App Store 下载的 App。

两者都是 App 文件，目录结构区别不大，但前者的属主用户和属主组一般是 root 和 admin，后者的属主用户和属主组都是 mobile，因而它们拥有不同的权限。不同的越狱工具对两者 sandbox（沙箱）的影响不同，一般来说，前者的 sandbox 比后者的限制要少很多。

顺便提一下，/Applications 目录中 App 安装包的格式多为 deb，而 /var/mobile/Applications 目录中 App 安装包的格式则为 ipa，由此又引申出了 iOS 平台上的不同安装包格式。

（1）deb

deb 格式是 Debian 系统（包含 Debian 和 Ubuntu）专属安装包格式，配合 APT 软件管理系统使用，是当前 Linux 下非常流行的一种安装包格式，是由 Cydia 作者 Jay Freeman(saurik) 移植到 iOS 中的。

由于 UNIX 类系统对权限、所有者、组的要求很严格，而 deb 格式安装包又经常会涉及系统比较底层的操作，可以获取较高的权限，所以从 Cydia 下载的 App 都是 deb 格式的。

（2）ipa

ipa 格式是苹果公司在 iOS 平台上推出的专属软件安装包，在 2.0 的固件发布后才正式投入使用，是目前 iOS 中唯一的官方安装包格式。

ipa 的文件权限很小，sandbox 限制很大，访问资源十分有限，这一点早已为广大开发人员所熟知。

（3）pxl

pxl 格式起源于 Mac 系统上的 pkg 安装包，在 1.x 固件时代被广泛应用，曾经是 iOS 平台上唯一的软件安装包，现在只有 91 手机助手等少数平台在使用，基本上可以忽略它的存在。

4. sandbox

通俗地说，iOS 中的 sandbox 就是一种访问限制机制。它是 iOS 最核心的安全部件之一，其实现很复杂，这里不过多讨论其实现细节，只列举两个主要现象供大家参考。

（1）文件访问

总的来说，sandbox 会将 App 的文件访问限制在这个 App 的 bundle 内部，因此一个 App 不知道其他 App 的存在，更别说访问它们了。

不过，App 可以通过 framework 来访问诸如照片、歌曲、通讯录等系统数据。

（2）硬件调用

App 可以通过 framework 来调用摄像头、GPS 模块等硬件。跟文件访问一样，因为这些操作要通过 framework 来进行，所以 App 的作用范围还是在 iOS 的控制之内。

在初学阶段，我们的目标不是逆向 sandbox，所以知道以上两点就够用了。在 iOS 逆向工程中，越狱操作已经破除了 iOS 的绝大多数安全限制，而这一点往往被我们忽略了。有时会碰到一些看似奇怪的问题，比如某个 tweak 不能写文件，或者通过 objc_getClass 函数获取不到类等，在确保自己的代码没有问题的前提下，这时就要回过头来检查这些问题是不是由对权限或者 sandbox 的误解造成的。

2.2.2 Dynamic Library

大部分的 iOS 开发者可能都没有接触过 Dynamic Library 的概念，因为面向 App Store 的 App 用不到它。但是放眼望去，Cydia 中的 tweak 插件无一不是以 Dynamic Library 的形式工作的，正是这些 tweak 插件的存在让我们能够随意定制自己的 iOS。所以，有必要了解 Dynamic Library 的基本知识。

Static Library 的概念很简单，在一个 App 启动时，系统会把 App 的代码和它所链接的 Static Library 一股脑放进分配好的内存空间里，不过因为一次性加载的内容过多，会造成 App 启动慢。

Dynamic Library 则相对“智能”一些，只有当 App 要用到这个 Library 时，系统才把这个 Library 加载进内存。具体来说，当一个 App 启动时，iOS 内核创建一个新进程，然后把 App 的代码加载到新进程的内存空间里；同时内核还会启动 Dynamic Loader（/usr/lib/dyld），把 App 需要的 Dynamic Library 加载进 App 的内存空间。

值得一提的是，虽然 Dynamic Library 的功能非常强大，但其本身并不是可执行程序，而是以类似插件的形式存活在一个 App 中的，是这个 App 的一部分。所以，Dynamic Library 的权限及内存空间是由加载它的那个 App 决定的，比如你写了一个 Instagram 的 tweak 插件，用来将你喜欢的图片保存在本地，如果你的保存目录是 /var/mobile/Applications/InstagramGUID/Documents，那么这样操作没有问题，你的 tweak 插件能够非常好地工作；如果你想保存在 /Developer 下，而你又没注意或忘记了这个 tweak 插件是存活在一个 App Store App 里，那么可能在兴高采烈地保存了一大堆美图，准备回头细细品味时，却发现 /Developer 里啥也没有——你的写操作都被 sandbox 给禁掉了。

2.2.3 Daemon

相信本书的绝大部分读者从接触 iOS 开发的第一天起，就不断被 Apple 灌输这样一个观念——iOS 中没有真正的后台多任务，你的 App 在后台将被大大限制[⊖]。如果你是一个纯粹的 App Store 开发者，坚信并坚守这个观念，那么它将是你的 App 通过 Apple 审核的助推剂；

⊖ 从 iOS 7 开始，官方已经支持真正的后台多任务了。

但既然你阅读了这本书，想要了解一些官方文档没有阐述的事实，那么你就要保持冷静，谨慎思考，让我们一起回想一下 iPhone 上的一些现象。

1）当我们正在用 iPhone 上网或刷微博，接到一个电话，所有其他操作会立即中断，系统第一时间将接听电话的界面呈现在用户面前。如果 iOS 中没有后台多任务，系统是如何迅速响应这个来电的呢?

2）相信很多越狱手机上都装了 360、QQ 手机助手等类似的软件，用来拦截垃圾短信、骚扰电话。出于对巨头们的不完全信任，以及同台竞技的小小需求，笔者也做了一款这样的手机助手软件——SMSNinja。如果 iOS 中没有后台多任务，SMSNinja 这类助手软件如何保证自己能及时处理收到的短信呢?

3）Backgrounder 是一款帮助 App 实现真正后台运行的强大插件，有了它，再也不用担心因为 push 功能的不给力而漏收即时信息啦！如果 iOS 中没有真正的后台多任务，Backgrounder 怎么会存在呢?

这些现象无一不说明 iOS 中是存在真正的后台多任务的，那么难道是 Apple 说错了？并不是！对于 App Store App 来说，当用户按下 home 键，程序就进入后台了，大多数功能都会被暂停；也就是说，对于遵纪守法的 App Store 开发者来说，可以把 iOS 看做是没有真正后台多任务的系统，因为你唯一能干的事不支持后台多任务。但 iOS 是基于 Mac OS X 的，后者又跟所有类 UNIX 操作系统一样，有 daemon（即守护进程，Windows 称 Service）的概念，随着越狱 iOS 全文件系统的开放，daemon 也展现在我们面前。

daemon 存在的意义就是后台运行，为用户使用操作系统提供各种“守护”，如 imagent 保障 iMessage 的正确收发，mediaserverd 处理几乎所有的音频、视频，syslogd 记录系统日志等。

iOS 系统中的 daemon 主要由一个可执行文件和一个 plist 文件构成。iOS 的根进程是 /sbin/launchd，会在开机或接到命令时检查 /System/Library/LaunchDaemons 和 /Library/Daemons 下所有符合格式规定的 plist 文件，然后按需启动对应的 daemon。这里的 plist 文件与 App 中的 Info.plist 文件作用类似，记录了 daemon 的基本信息。

相对于 App，daemon 提供的功能要底层得多，逆向难度也要大得多，如果对某个 daemon 的认识不够成熟，随意更改它可能导致系统崩溃。当你在 iOS 逆向工程上有了一定的积累后，就可以挑战这些 daemon 了，虽然相比 App，花费的精力会更多，但它们也会给你更丰厚的回报。例如，前段时间刚刚发布的“iOS 第一款电话录音软件”Audio Recorder 就是通过逆向 /usr/sbin/mediaserverd 这个 daemon 实现的。

2.3 小结

本章介绍了 iOS 系统架构和 iOS 中的常见程序类型，App Store 开发者一般不需要了解也接触不到这些知识，容易形成概念盲区。本章旨在科普那些在逆向工程中非常重要但 Apple 官方闭口不提的 iOS 系统知识点，为 App Store 开发者打开 iOS 系统中的另一扇门。

就本章所涉及的知识点，将其内容扩充成一整本书都不为过，但作为逆向工程的初学者，了解这些内容就已足够为我们的逆向之路开一个好头了。

第二部分

工 具 篇

前一篇介绍了逆向工程的基础概念，从本篇开始，将介绍在逆向分析的过程中会使用到的一系列开发、调试工具。

相对于常规 App Store App 开发，iOS 逆向工程中用到的工具其最大特点就是“杂”。在 App Store App 开发中，一个 Xcode 就可以完成绝大部分开发工作，而且因为 Xcode 是 Apple 的嫡系出身，它的下载、安装和使用都非常方便。至于其他的一些插件、工具，所提供的只是一些锦上添花的功能，在开发中并不是必需的。

而在不那么常规的 iOS 逆向工程中，我们却不得不面对一长串叫起来都嫌绕口的工具。这就如同面前摆着两张桌子，一张很整洁，上面只放着一个高贵大方的 Xcode；而另一张桌子上横七竖八地堆满了一二十个工具，其中的几个大块头分别叫 Theos、Reveal、IDA……

这些工具相互之间并没有紧密耦合、互相依赖的关系，整合度远没有 Xcode 高，因此在使用过程中得根据需要把它们手动组合起来。本篇并没有涵盖全部主流工具，比如强大的 PonyDebugger，不过相信读者在完全吃透本书内容之后，会具备举一反三的能力，可根据自己的需求安装对应的工具。

另外，因为需要介绍的工具颇多，略显杂乱，所以本篇的内容分成两章，分别是 Mac 工具集和 iOS 工具集。

第 3 章

Mac 工具集

在 iOS 逆向工程中通常会使用一系列的工具，它们的用处各不相同，而在 Mac 上完成的主要是开发和调试工作（iOS 干这个活儿有些吃力，毕竟屏幕尺寸在那摆着呢）。

本章主要介绍的工具有 4 个：class-dump、Theos、Reveal、IDA，其他的都是配套使用的辅助工具。

3.1 class-dump

3.1.1 class-dump 介绍及下载

class-dump，顾名思义，就是用来 dump 目标对象的 class 信息的工具。它利用 Objective-C 语言的 runtime 特性，将存储在 Mach-O 文件中的 @interface 和 @protocol 信息提取出来，并生成对应的 .h 文件。

关于 class-dump 更详尽的信息及安装文件，请参见 http://stevenygard.com/projects/class-dump，图 3-1 是对应的下载信息。

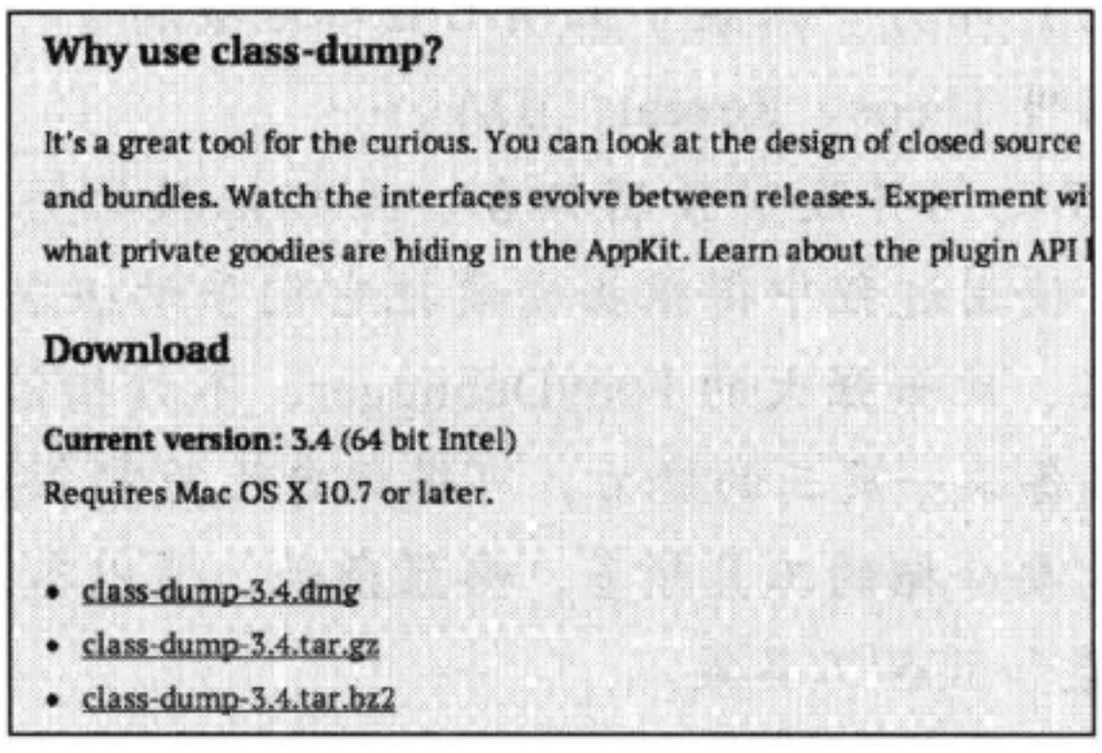


图 3-1　class-dump 下载页面

选择 class-dump-3.4.dmg 下载后，将 dmg 文件中的 class-dump 复制到 /usr/bin 目录，并在 Terminal 中执行“ sudo chmod 777 /usr/bin/class-dump”命令赋予其执行权限，然后运行 class-dump，即可看到如图 3-2 所示的信息。

```
class-dump 3.4 (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, x
        -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.platfor
ion>.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)
```

图 3-2 class-dump 运行参数

class-dump 是一个相对简单易用的工具，本章之所以第一个介绍它，是因为它能够导出一个闭源程序的 @interface 信息，并将其整理成开发者习惯的阅读方式，让浏览者一目了然。因此，本节内容作为逆向工程的基础参考资料非常合适。也正是有了这个基础，后续的逆向工作才能更好地进行。

3.1.2 class-dump 使用演示

本书的读者多是经验丰富的 iOS 开发者，肯定也都知道 Xcode 生成的 App 实际上只是一个压缩包，而 class-dump 的对象，App 的可执行文件就身藏此包中。在此用笔者自己的一个 App 来演示一下，如图 3-3 所示。

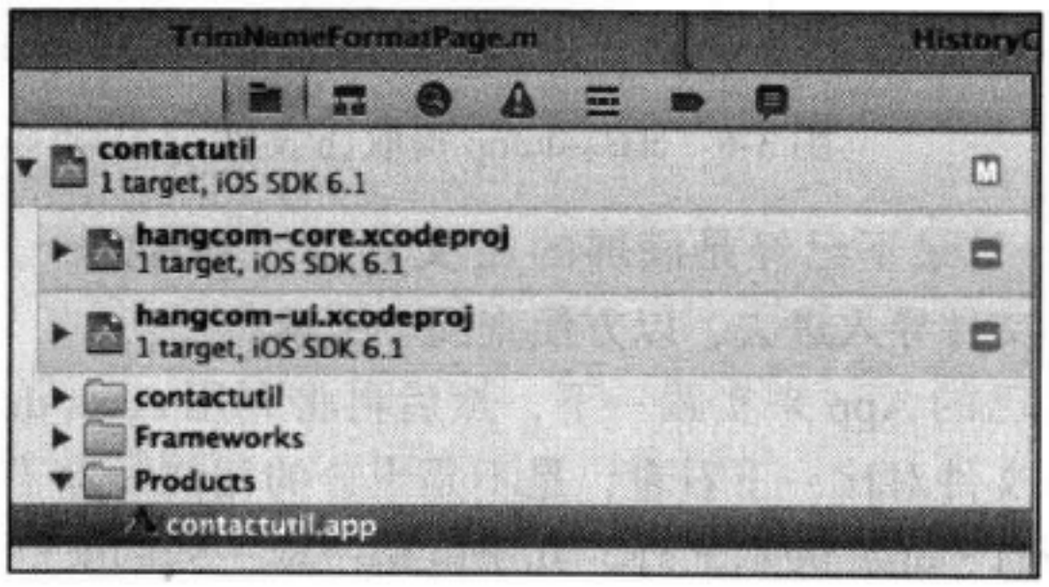


图 3-3 Xcode 中对应的 App 文件

可以看到，图 3-3 中的 contactutil.app 就是一个压缩包，将此 contactutil.app 展开，如图 3-4 所示。

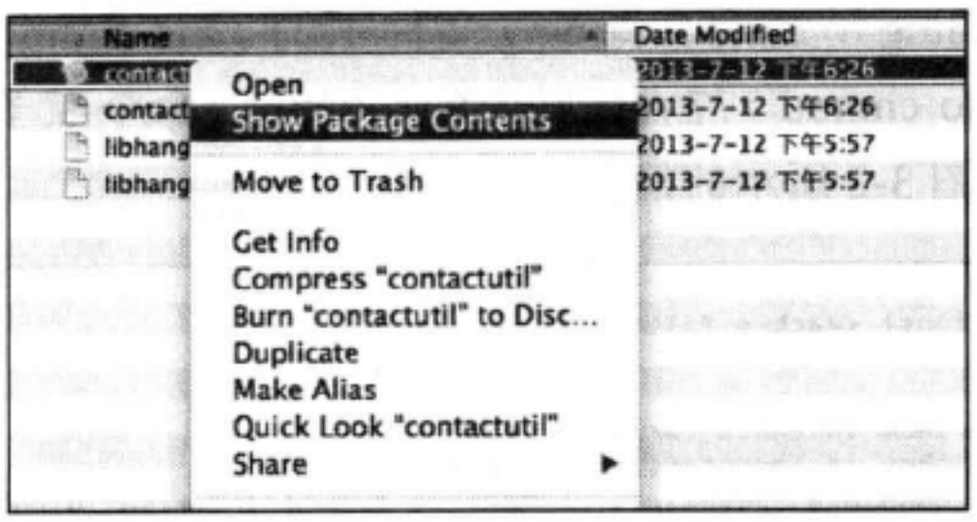


图 3-4 右键展开 App 压缩包内容

这时，有经验的人可以凭经验来判断 contactutil 就是我们的目标，如图 3-5 所示。

Name	Date Modified
contactutil	2013-7-12 下午6:26
CodeInfo.dat	2013-7-12 下午5:57
Default-568h@2x.png	2013-7-12 下午5:57
Default@2x.png	2013-7-12 下午5:57
Default.png	2013-7-12 下午5:57
img_empty_page@2x.png	2013-7-12 下午5:57
embedded.mobileprovision	2013-7-12 下午6:26

图 3-5 App 压缩包的内容展示

如果是新手，可以打开 Info.plist，来查看“ Executable file”字段对应的文件名，即目标文件。

获取此目标文件后，将其复制到任何一个目录下，然后再用 class-dump 来提取我们想要的信息，如图 3-6 所示。

```
sammatoMacBook-Pro:Desktop sam$ class-dump -H contactutil -o contactutil.h
2013-07-15 14:08:21.646 class-dump[23171:707] Unknown load command: 0x00000025
2013-07-15 14:08:21.648 class-dump[23171:707] Unknown load command: 0x00000026
2013-07-15 14:08:21.649 class-dump[23171:707] Unknown load command: 0x00000029
sammatoMacBook-Pro:Desktop sam$
sammatoMacBook-Pro:Desktop sam$
sammatoMacBook-Pro:Desktop sam$ ls contactutil.h/
AboutController.h                                   NSCopying-Protocol.h
AppDelegate.h                                       NSData-JSONKitDeserializing.h
BadgeView.h                                         NSDictionary-JSONKitSerializing.h
```

图 3-6 class-dump 提取 .h 文件

此时的 contactutil.h 目录下已经是满屏的 .h 文件了，笔者的个人习惯是再建一个空的 Xcode 工程，将这些 .h 文件导入进去，以方便查阅、检索。

各位读者可以用自己的 App 来实践一下，然后再将利用 class-dump 导出的 .h 文件和原有的 Xcode 工程中的源文件对比一下看看，是不是十分的相似？除了一些参数定义上的类型用 id，以及参数名用 arg1、arg2 表示之外，几乎就是一模一样的吧？甚至由 class-dump 导出的文本格式比我们原有的还要标准。

而更现实的意义是：同样是闭源程序，class-dump 既然能提取其自身 App 中的 .h 文件，那么同样也能提取别人 App 里的 .h 文件！

对于经验丰富的开发人员，有了这些 .h 文件，就已经可以初步看清楚目标 App 的程序结构了；不过现在的 App 工程越来越大，而且还在不停地引用第三方源代码，所以经常会发现通过 class-dump 导出成百上千个 .h 文件，这种情况让人非常头疼，靠人工及经验来一点点地定位是很好的练习方式，但过程实在非常复杂，所以后续会讲到如何使用 Reveal 来精确定位程序功能入口。

3.1.3 关于 class-dump 的补充说明

class-dump 虽然非常有用，但有时我们会发现 class-dump 执行失败，无法得到想要的 .h 文件，或 .h 文件的内容是加密的密文。笔者就曾经碰到过这种问题，被折腾得上蹿下跳，一度以为 class-dump 的 bug 被我们“幸运”地碰到了。后来查询了更多资料才明白，出现这种情况基本上都是因为 class-dump 的作用对象必须是未经加密的可执行文件，而从 App Store 下载的 App 都是经过签名加密的，可执行文件被加上了一层“壳”，class-dump 应付不了这样的文件，所以看上去“失效”了。原版的 App Store 中的 App 就像是一颗硬硬的核桃，我们试图用 class-dump 这样一个镊子来取肉，别说解馋，没把镊子夹坏就算不错啦！所以，要想吃核桃，还得先用别的工具来把壳砸开才行。

下一章会介绍如何用 AppCrackr 来自动砸壳，大部分情况下壳都能砸开。退一步说，即使真的碰到那些做了反逆向保护而不好砸的，也能在一些网站上找到对应 App 的越狱安装版，这些版本的壳已经被手动砸开了，请各位尽情享用；在积累了一定的逆向工程基础后，就可以尝试反逆向，自己动手砸壳，丰衣足食！

3.2 Theos

3.2.1 Theos 简介

Theos 是一个越狱开发工具包，由大神 Dustin Howett（@DHowett）开发并分享到 GitHub 上。Theos 与其他越狱开发工具相比，最大的特点就是简单：下载安装简单、Logos 语法简单、编译发布简单，可以让使用者把精力都放在开发工作上去。

在 Theos 之前，曾经有一个 toolchain（还有其他工具，比如 iOSOpenDev，不过笔者没使用过），可笔者在安装它的过程中就遇到了不少麻烦，经历惨痛，实在是不堪回首，现在回想起来还一直庆幸遇见了 Theos，遇到了命中注定的那个“它”。

这里插播一个关于 DHowett 的小段子。DHowett 的全名叫 Dustin L. Howett，他是个很有个性的少年，出生在美国宾夕法尼亚州的郊区，从小痴迷计算机。大学读了不到一年，觉得老师讲得没意思，自己不好好听也跟不上，更重要的是，他跟一个姑娘展开了疯狂的异地恋，于是就干脆辍学，搬到了那个姑娘的所在地加州，并求职进了 Saurik 的公司 SaurikIT（DHowett 的早期作品 CyDelete 以 Cy 开头，而这种命名方式是 Saurik 御用的，说明 DHowett 的作品得到了 Saurik 的认可，也足见 DHowett 与 Saurik 关系之好）。但遗憾的是，在 Dustin 辍学后，他和女朋友之间开始出现问题，最后分道扬镳了。之后 Dustin 离开了 SaurikIT，进

入了另一家创业公司 DailyBooth，但这家公司经营不善，没多久就倒闭了，他就回家待业了。过了没多久，Dustin 爱上了另一个女孩，于是他又为了这个姑娘搬到旧金山，并且在当地一家不错的公司 Airbnb 找到了一份新工作。在笔者眼里，Dustin 敢想敢干，敢爱敢恨，敢做敢当，真是“让我们红尘作伴活得潇潇洒洒”，可以说是一位风一般的男子，令人十分崇拜。

好了，回到正文。Theos 是一个非常轻量级的开发工具，全部的动作均可在命令行下完成。有时笔者在做一些小小的逆向工程时，连 Xcode 都不需要打开，直接在 Terminal 中用 vim 复制和编辑代码，然后就可以打包和测试了。

3.2.2 Theos 安装及编译

在安装 Theos 之前，需要具备以下环境。

1. iOS SDK

一般来说，iOS 开发者的计算机上都会装有 iOS SDK。简单来说，就是要在下载 Xcode 后进一步安装 CLT（Command Line Tools，命令行工具），其在 Xcode 中的路径是 Xcode → Preferences → Downloads，如图 3-7 所示。

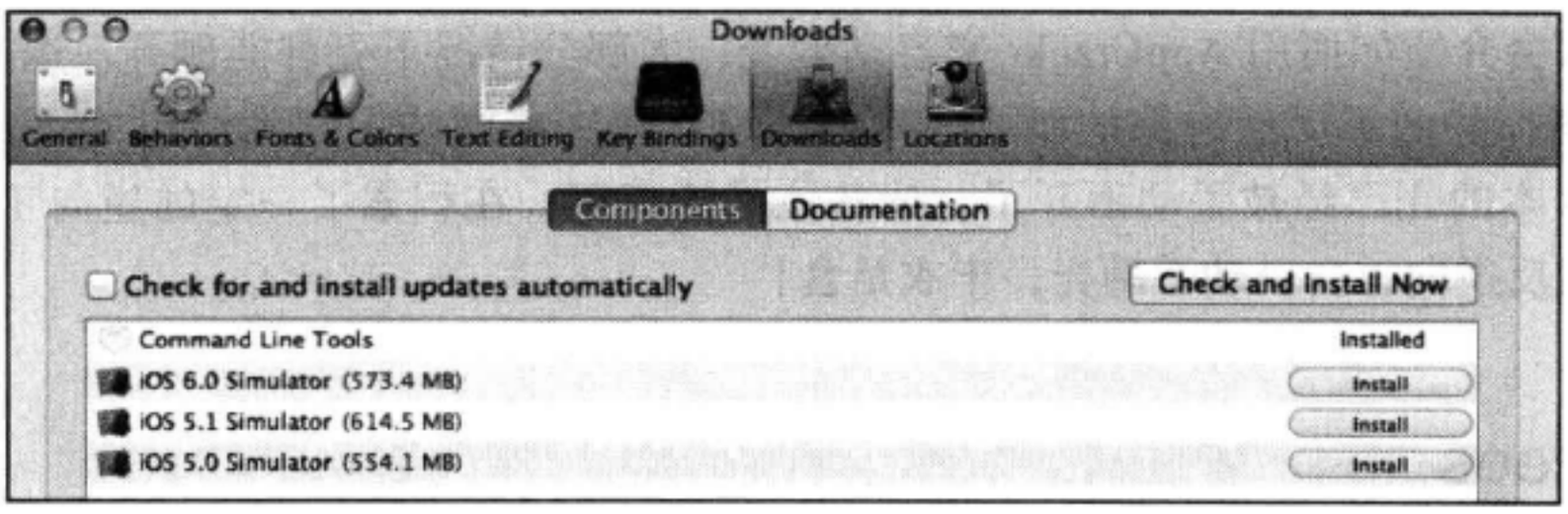


图 3-7 CLT 安装界面

需要注意的是，Xcode 5 之后，CLT 被整合到 Xcode 中，无须额外下载安装。

2. 配置环境变量

在配置环境变量时，使用如下命令：

```
export THEOS=/opt/theos
```

通过上述命令来设置安装路径，这个 /opt/theos 是个推荐路径，笔者电脑上也是这么设置的，使用者也可以根据自己的喜好来自定义路径。或者进入 Terminal/vi ~/.profile 加上这个路径。

3. 获取 Theos

在 Terminal 中使用以下命令来获取 Theos：

```
sudo git clone git://github.com/DHowett/theos.git $THEOS
```

或者打开 https://github.com/DHowett/theos 直接下载解压至 $THEOS，一样能用。当然前者升级更加方便，推荐！

4. 安装 ldid

ldid 是专门用来签名 iOS 文件的工具，用以取代 Xcode 自带的 codesign。ldid 可以从 https://github.com/downloads/rpetrich/ldid/ldid.zip 下载，并将解压所得到的“ldid”文件放在 /opt/theos/bin（$THEOS/bin）下。

顺便提一句：ldid 是由 @Saurik 开发，rpetrich 深加工的。此乃两位高手呕心沥血的作品，其功能强大得一塌糊涂。如果你对 ldid 的工作原理感兴趣，可以阅读一下这篇文章 http://www.saurik.com/id/8，也可以去 https://github.com/rpetrich/ldid 看看源码，保证受益匪浅。

5. 配置 MobileSubstrate 环境

在 Terminal 中使用以下命令来配置开发时需要的 MobileSubstrate 环境：

```
sudo $THEOS/bin/bootstrap.sh substrate
```

此处会遇到 Theos 的一个 bug，它无法生成一个有效的 libsubstrate.dylib 文件，需要我们手动将其替换成有效文件。解决方法很简单：用 iFunBox 或 iTools 等软件（使用方法见 3.5 节）将 iOS 设备上的 /Library/Frameworks/CydiaSubstrate.framework/CydiaSubstrate 复制到 Mac 中，然后运行以下命令，替换无效的 libsubstrate.dylib：

```
sudo mv -f /path/to/CydiaSubstrate $THEOS/lib/libsubstrate.dylib
```

6. 安装 dpkg

dpkg 是一个专门用来制作 deb（Debian package）的工具，我们用 Theos 开发出来的插件都将会以 deb 格式发布。笔者推荐读者采用 MacPorts 这一简单方法安装 dpkg，具体步骤如下：

- 前往 http://www.macports.org/install.php 下载对应 OSX 版本的 pkg 安装包；
- 安装完成后，在 Terminal 中运行：

```
sudo port selfupdate
```

确保 MacPorts 升级到了最新版本；

- 运行：

```
sudo port install dpkg
```

因为这个过程要安装 dpkg 及不少依赖工具，所以显得有些漫长。站起来走两步，活动活动，或者看看 @iOS 应用逆向工程有没有更新微博，稍稍放松一下吧。

安装成功后可以看到 dpkg 的用法，如图 3-8 所示。

从上面这些参数也可以看到，dpkg 的功能其实很强大，包括打包、安装、卸载等，而且它是 Mac 和 iOS 上都需要的工具：除了 Theos 开发要依赖它之外，还可以在 iOS 的命令行中执行“dpkg -i xxx.deb”来安装某个 deb 文件，如下所示：

```
dpkg -i com.hangcom.monitor_0.0.1-40_iphoneos-arm.deb
```

不过对于笔者这样的懒人来说，图形界面的可操作性比命令行要强得多，所以，在第 4

章介绍 iFile 时也会说明如何通过 iFile 来安装 deb 文件。

```
sammatoMacBook-Pro:~ sam$ dpkg --help
Usage:
  dpkg -i|--install      <.deb file name> ... | -R|--recursive <dir> ...
  dpkg --unpack          <.deb file name> ... | -R|--recursive <dir> ...
  dpkg -A|--record-avail <.deb file name> ... | -R|--recursive <dir> ...
  dpkg --configure               <package name> ... | -a|--pending
  dpkg -r|--remove | -P|--purge <package name> ... | -a|--pending
  dpkg --get-selections [<pattern> ...]     get list of selections to stdout
  dpkg --set-selections                     set package selections from stdin
  dpkg --update-avail <Packages-file>       replace available packages info
  dpkg --merge-avail <Packages-file>        merge with info from file
  dpkg --clear-avail                        erase existing available info
  dpkg --forget-old-unavail                 forget uninstalled unavailable pkgs
  dpkg -s|--status <package-name> ...       display package status details
  dpkg -p|--print-avail <package-name> ... display available version details
  dpkg -L|--listfiles <package-name> ...    list files `owned' by package(s)
  dpkg -l|--list [<pattern> ...]            list packages concisely
  dpkg -S|--search <pattern> ...            find package(s) owning file(s)
  dpkg -C|--audit                           check for broken package(s)
  dpkg --print-architecture                 print target architecture (uses GCC)
  dpkg --print-gnu-build-architecture       print GNU version of target arch
  dpkg --print-installation-architecture    print host architecture (for inst'n)
  dpkg --compare-versions <a> <rel> <b>     compare version numbers - see below
  dpkg --help | --version                   show this help / version number
  dpkg --force-help | -Dh|--debug=help      help on forcing resp. debugging
  dpkg --licence                            print copyright licensing terms
```

图 3-8 dpkg 运行参数

7. 安装 Theos NIC templates

Theos NIC templates 内置了 5 种 Theos 工程类型的模板，方便我们创建多样的 Theos 工程。我们还可以从 https://github.com/DHowett/theos-nic-templates/archive/master.zip 获取额外的 5 种模板，下载后将解压得到的 5 个 .tar 文件复制到 $THEOS/templates/iphone 下即可。

3.2.3 Theos 用法简介

1. 创建工程

使用 Theos 创建工程的工作需要在 Terminal 中完成，虽然都是在命令行操作，但实际非常简单，下面来逐步讲解。

（1）更改工作目录至常用的 iOS 工程目录（如笔者的是“ /Users/snakeninny/Code”），然后输入“$THEOS/bin/nic.pl”，启动 NIC（New Instance Creator），如图 3-9 所示。

可以看到共有 10 种模板可供选择，其中 1、4、6、8、9 是 Theos 的自带模板，2、3、5、7、10 是 要另外下载的。在逆向工程的初级阶段，所开发程序的主要类型是 tweak，其他模板的介绍可关注我们的官方微博@iOS 应用逆向工程。

（2）输入“9”，按 Enter 键，此时 Theos 要求我们输入工程名，如图 3-10 所示。

（3）输入一个工程名称，按 Enter 键，

```
[illegible]-MacBook-Pro: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):
```

图 3-9 Theos 运行参数

此时 Theos 要求我们输入 deb 包的名字（类似于 bundle identifier），如图 3-11 所示。

```
Choose a Template (required): 9
Project Name (required):
```

图 3-10 输入 Theos 工程名

```
Choose a Template (required): 9
Project Name (required): iOSRE
Package Name [com.yourcompany.iosre]:
```

图 3-11 输入 Theos 工程的 bundle identifier

（4）输入 deb 包的名字，按 Enter 键，Theos 要求我们输入 tweak 作者的名字，如图 3-12 所示。

```
Choose a Template (required): 9
Project Name (required): iOSRE
Package Name [com.yourcompany.iosre]: com.naken.iosre
Author/Maintainer Name [snakeninny]:
```

图 3-12 输入作者名

（5）输入名字后，按 Enter 键，Theos 要求我们输入 MobileSubstrate Bundle filter，也就是 tweak 作用对象的 bundle identifier，如图 3-13 所示。

```
Choose a Template (required): 9
Project Name (required): iOSRE
Package Name [com.yourcompany.iosre]: com.naken.iosre
Author/Maintainer Name [snakeninny]: snakeninny
[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]:
```

图 3-13 输入作用对象的 bundle identifier

（6）输入 bundle filter，Theos 要求我们指定 tweak 安装完成后需要重启的应用，仍以 bundle identifier 表示。输入完成后按下回车，搞定（如图 3-14 所示）。比 Xcode 简单吧？

```
Choose a Template (required): 9
Project Name (required): iOSRE
Package Name [com.yourcompany.iosre]: com.naken.iosre
Author/Maintainer Name [snakeninny]: snakeninny
[iphone/tweak] MobileSubstrate Bundle filter [com.apple.springboard]:
com.apple.mobilephone
[iphone/tweak] List of applications to terminate upon installation (sp
ace-separated, '-' for none) [SpringBoard]: com.apple.mobilephone
Instantiating iphone/tweak in iosre/...
Done.
```

图 3-14 创建 Theos 工程完毕

以上的步骤完成之后，一个名为 iosre 的文件夹就在当前目录下生成了，文件夹中就是刚创建的 tweak 工程，你可以把这个文件夹挪到任意位置。

2. 定制工程文件

用 Theos 创建 tweak 工程非常方便，但简洁的工程框架下是粗糙的内容，因此我们需要进一步加工相关文件。先来看看刚刚生成的工程目录（如图 3-15 所示），除去一个指向 Theos 目录的符号链接外，只有 4 个文件，从工程复杂度来说完全不会吓跑初学者，反而会让我们跃跃欲试（再次赞美一下 @DHowett）。

古语云：“一粒米中藏世界，半边锅内煮乾坤。”4 根顶梁柱就足以撑起 tweak 的毛坯房，

但漂亮的 tweak 离不开我们的精装修，这 4 个文件的内容可是大有玄机！

Name	Date Modified	Size	Kind
theos	Today 12:10 AM	10 bytes	Alias
control	Today 12:10 AM	209 bytes	Document
iOSRE.plist	Today 12:10 AM	58 bytes	Property List
Makefile	Today 12:10 AM	114 bytes	Document
Tweak.xm	Today 12:10 AM	1 KB	MacVi...ument

图 3-15 Theos 工程目录

（1）control

control 文件记录了 deb 包管理系统所需的基本信息，会被打包进 deb 包中。之前创建的 tweak 工程中的 control 文件内容如下所示。

```
Package: com.naken.iosre
Name: iOSRE
Depends: mobilesubstrate
Version: 0.0.1
Architecture: iphoneos-arm
Description: An awesome MobileSubstrate tweak!
Maintainer: snakeninny
Author: snakeninny
Section: Tweaks
```

其中：

- Package 字段用于描述这个 deb 包的名字，采用的命名方式同 bundle identifier 类似，均为反向 DNS 格式，可以按需更改；
- Name 字段用于描述这个工程的名字，可以按需更改；
- Depends 字段用于描述这个 deb 包的“依赖”。“依赖”指的是这个程序运行的基本条件，可以填写固件版本或其他程序，如果当前 iOS 不满足“依赖”中定义的条件，则此 tweak 无法正常运行。如

```
Depends: mobilesubstrate, firmware (>= 6.0)
```

表示当前 iOS 版本必须在 6.0 以上，且必须安装 MobileSubstrate，才能正常运行这个 tweak，可以按需更改；

- Version 字段用于描述这个 deb 包的版本号，可以按需更改；
- Architecture 字段用于描述 deb 包安装的目标设备架构，不要更改；
- Description 字段是 deb 包的简单介绍，可以按需更改；
- Maintainer 字段用于描述 deb 包的维护人，即 deb 包的制作者而非 tweak 的作者，例如 BigBoss 源中所有 deb 包的维护人均为 BigBoss，而非软件作者，可以按需更改；
- Author 字段用于描述 tweak 的作者（注意与 Maintainer 的区别），可以按需更改；
- Section 字段用于描述 deb 包所属的程序类别，不要更改。

control 文件中可以自定义的字段还有很多，但上面这些信息就已经足够了。更全面的说明可以参阅 debian 的官方网站：http://www.debian.org/doc/debian-policy/ch-controlfields.html，

或者留意其他 deb 包中的 control 文件。值得留意的是，Theos 在打包 deb 时会对 control 文件作进一步处理，上面的 control 文件在得到处理后内容如下。

```
Package: com.naken.iosre
Name: iOSRE
Depends: mobilesubstrate
Version: 0.0.1-2
Architecture: iphoneos-arm
Description: An awesome MobileSubstrate tweak!
Maintainer: snakeninny
Author: snakeninny
Section: Tweaks
Installed-Size: 16
```

这里更改了 Version 字段，标示 Theos 的打包次数，方便管理；增加了一个 Installed-Size 字段，用以描述 deb 包安装后的估算大小，与实际大小可能有偏差，不要更改。

control 文件中的很多信息直接体现在 Cydia 中，如图 3-16 所示，读者可以对比其内容。



图 3-16 control 文件的展示效果

（2）iOSRE.plist

这个 plist 文件的作用和 App 中的 Info.plist 类似，都记录一些配置信息，它描述了 tweak 的作用范围。这个 plist 文件支持 3 种格式，分别是 XML、二进制文件和 OpenStep（即常见的 property list 格式），这里着重讲解最为常见的 OpenStep 格式，可以用 Xcode 打开。

iOSRE.plist 的最外层是一个 Dictionary，只有一个名为“Filter”的键，如图 3-17 所示。

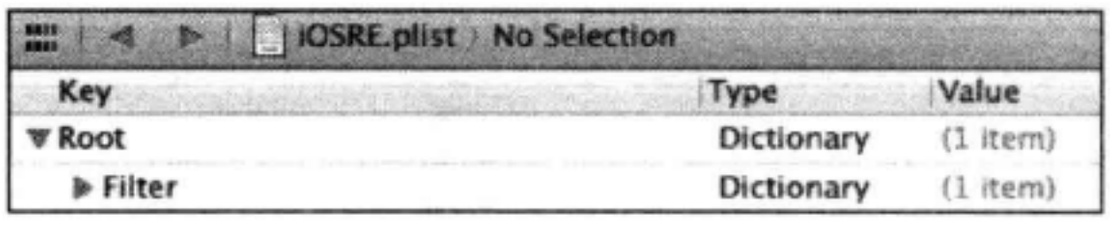


图 3-17 iOSRE.plist 中的 Filter 参数

Filter 下是一系列 array，可以分为以下三类。

❑ Bundles，指定若干 bundle identifier 为 tweak 的作用对象，如图 3-18 所示。

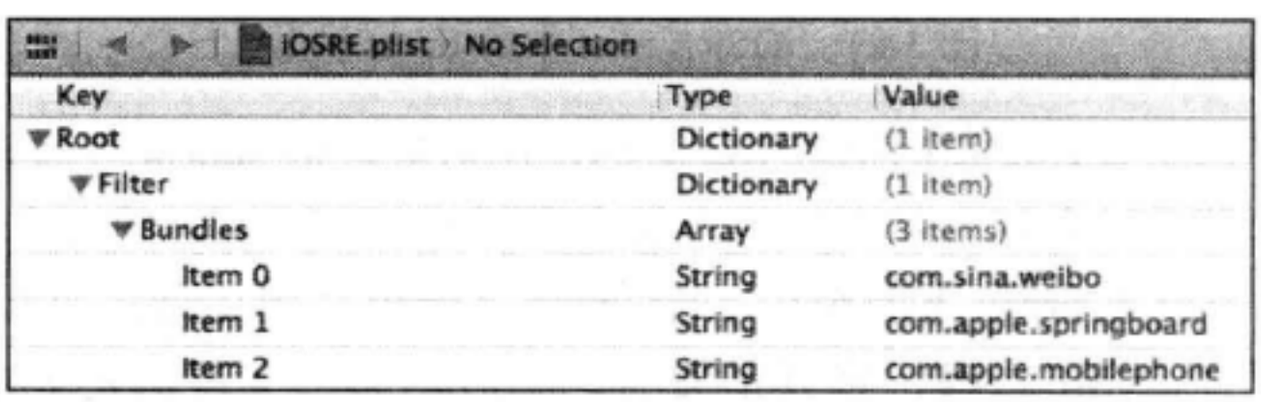


图 3-18 iOSRE.plist 中的 Filter 参数，指定多个 bundle

按照图 3-18 中的配置，tweak 的作用对象是三个 App，即新浪微博客户端、SpringBoard

和电话程序。

❑ Classes，指定若干 class 为 tweak 的作用对象，如图 3-19 所示。

Key	Type	Value
▼ Root	Dictionary	(1 item)
▼ Filter	Dictionary	(1 item)
▼ Classes	Array	(3 items)
Item 0	String	SBAwayController
Item 1	String	NSString
Item 2	String	UIAlertView

图 3-19 iOSRE.plist 中的 Filter 参数，指定多个 class

按照图 3-19 的配置，tweak 的作用对象是三个 class，即 SBAwayController、NSString 和 UIAlertView。

❑ Executables，指定若干可执行文件为 tweak 的作用对象，如图 3-20 所示。

Key	Type	Value
▼ Root	Dictionary	(1 item)
▼ Filter	Dictionary	(1 item)
▼ Executables	Array	(3 items)
Item 0	String	lsd
Item 1	String	mediaserverd
Item 2	String	imagent

图 3-20 iOSRE.plist 中的 Filter 参数，指定多个可执行文件

按照图 3-20 中的配置，tweak 的作用对象是三个可执行文件，即 lsd、mediaserverd 和 imagent。

这三类 array 可以混合使用，如图 3-21 所示。

iOSRE.plist › No Selection

Key	Type	Value
▼ Root	Dictionary	(1 item)
▼ Filter	Dictionary	(3 items)
▼ Executables	Array	(1 item)
Item 0	String	imagent
▼ Bundles	Array	(1 item)
Item 0	String	com.apple.springboard
▼ Classes	Array	(1 item)
Item 0	String	SBIconModel

图 3-21 iOSRE.plist 中的 Filter 参数

这样的配置方式能够方便我们指定 tweak 的作用对象，但 saurik 在设计 bundle filter 时脑子一热，犯了个小错误（大神也是人，是人就会犯错）：按图 3-21 的混合配置方式，一个文件只有满足 Filter 中所有 array 下的至少一个条件，tweak 才能生效。这样显然是不合理的，所以 saurik 添加了一个 Mode 键，将其值设置为 Any，那么文件满足 Filter 中的任一条件，就能成为 tweak 的作用对象，如图 3-22 所示。

当 Filter 下的 array 只有一类时，不需要添加 Mode 和 Any 键值对。

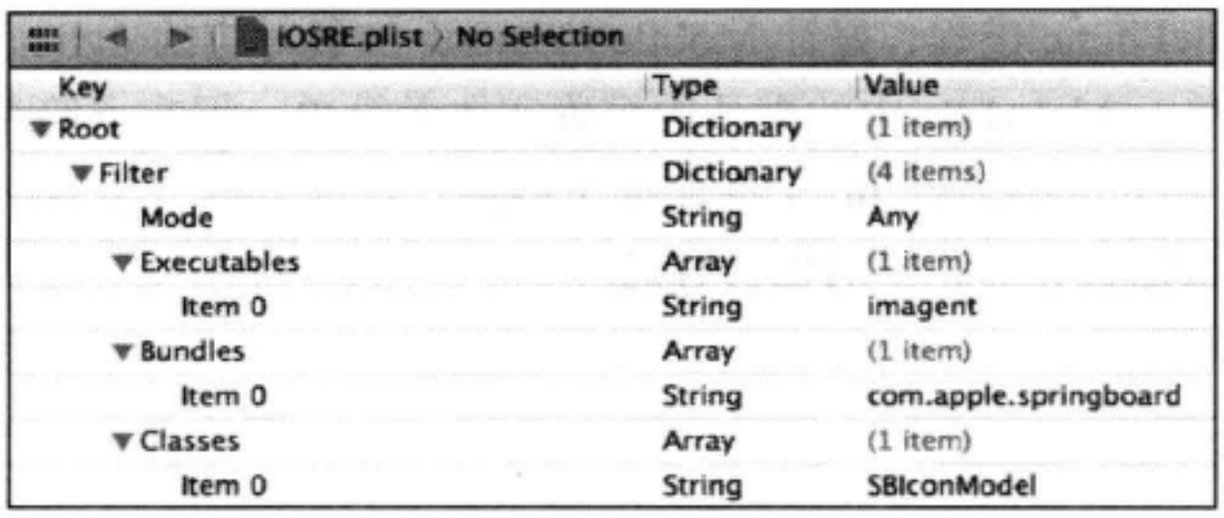


图 3-22 iOSRE.plist 中的 Filter 参数，指定 Mode

（3）Makefile

Makefile 文件指定编译和链接所涉及的文件、框架、库等信息，将整个过程自动化。iOSRE 工程的 MakeFile 内容如下：

```
include theos/makefiles/common.mk

TWEAK_NAME = iOSRE
iOSRE_FILES = Tweak.xm

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 MobilePhone"
```

下面来逐行解读：

```
include theos/makefiles/common.mk
```

固定写法，不要更改。

```
TWEAK_NAME = iOSRE
```

tweak 的名字，即用 Theos 创建工程时指定的“Project Name”，与 control 文件中的“Name”字段对应，不要更改。

```
iOSRE_FILES = Tweak.xm
```

tweak 包含的源文件（不包括头文件），多个文件间以空格分隔，如“Hook.xm New.x ObjC.m ObjC++.mm”，可以按需更改。

```
include $(THEOS_MAKE_PATH)/tweak.mk
```

根据不同的 Theos 工程类型，include 不同的 .mk 文件。在逆向工程初级阶段，我们开发的一般是 Application、Tweak 和 Tool 三种类型的程序，它们对应的 .mk 文件分别是 application.mk、tweak.mk 和 tool.mk，可以按需更改。

```
after-install::
    install.exec "killall -9 MobilePhone"
```

从字面意思读者朋友应该就能对这两行的作用猜个八九不离十，这里默认的是在 tweak

安装之后杀掉 MobilePhone 进程，好让 MobileSubstrate 加载对应的 dylib 。

是不是非常简单？ MakeFile 中的默认内容确实非常简单，但有点简单过头了。如何指定 SDK 版本？怎么导入 framework？lib 文件在哪里链接？作为 iOS 开发者的你一定会提出这些问题。别急别急，面包会有的，牛奶也会有的。

❑ 指定 SDK 版本

```
TARGET = iphone:SDK 版本
```

例如

```
TARGET = iphone:6.0
```

上面语句是在告诉 Theos 应该工作在哪个 SDK 版本下。需要注意的是，最新版的 Xcode 附带两个版本的 linker，一个用于 gcc，另一个用于 clang。其中，gcc 版的 linker 在 armv6 上会崩溃，而 clang 版本的 linker 已不再支持 armv6 架构。所以，6.0 以上版本的 SDK 已不再支持 4.3 版本以下的 iOS 或 armv6 架构，如果像上面那样笼统定义，会导致 link 出错，如图 3-23 所示。

```
[redacted]-MacBook-Pro:iosre snakeninny$ make
Making all for tweak iOSRE...
 Linking tweak iOSRE...
collect2: ld terminated with signal 6 [Abort trap: 6]
ld(7759,0x7fff794a6180) malloc: *** error for object 0x7fad59c0ba80: pointer bei
ng freed was not allocated
*** set a breakpoint in malloc_error_break to debug
collect2: ld terminated with signal 6 [Abort trap: 6]
ld(7763,0x7fff794a6180) malloc: *** error for object 0x7fad91c0ba80: pointer bei
ng freed was not allocated
*** set a breakpoint in malloc_error_break to debug
lipo: can't open input file: /var/folders/vh/hsyhn8fs6zncqltbgpfpt6k40000gn/T//c
cfsAPyu.out (No such file or directory)
make[2]: *** [obj/iOSRE.dylib.ba964c90.unsigned] Error 1
make[1]: *** [internal-library-all_] Error 2
make: *** [iOSRE.all.tweak.variables] Error 2
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-23 TARGET 指定不明确导致编译报错

此时，有两种选择可以避免出现这个错误。如果确定要采用 6.0 以上版本的 SDK，可以在 MakeFile 文件的最前面加上这样的语句：

```
ARCHS = armv7
TARGET = iphone:6.1:4.3
```

即指定开发架构为 armv7，采用 6.1 版本的 SDK（即 Xcode 中的 Base SDK），且发布对象为 iOS 4.3 及以上版本（即 Xcode 中的 Development Target）。如果想采用低于 6.0 版本的 SDK 以针对多种架构或兼容早于 4.3 版本的 iOS，则可加上这样的语句：

```
ARCHS = armv6 armv7
TARGET = iphone:低于 6.0 版本的 SDK:高于 3.0 版本的 iOS
```

即指定开发架构为 armv6 和 armv7，采用低于 6.0 版本的 SDK，且发布对象可以是

iOS 4.3 以下版本。一般来说，采用上面一种方案即可满足需求。

❑ 导入 framework

```
iOSRE_FRAMEWORKS = framework名
```

例如

```
iOSRE_FRAMEWORKS = UIKit CoreTelephony CoreAudio
```

这个功能没什么可多说的，但既然是 tweak 开发，很多朋友关注的应该是如何导入 private framework 吧？很简单，用下面的语句即可：

```
iOSRE_PRIVATE_FRAMEWORKS = private framework名
```

例如

```
iOSRE_PRIVATE_FRAMEWORKS = AppSupport ChatKit IMCore
```

虽然只是多了个 PRIVATE，但有一点要交代：private framework 是 App Store App 开发所不允许使用的，它的内容变化无常，在导入之前，一定要确定导入的 private framework 确实存在。举一个例子，如果你的 tweak 打算兼容 iOS 5 和 iOS 6 两个版本，那么 MakeFile 写成：

```
ARCHS = armv7
TARGET = iphone:6.1:4.3
include theos/makefiles/common.mk

TWEAK_NAME = iOSRE
iOSRE_FILES = Tweak.xm
iOSRE_PRIVATE_FRAMEWORKS = AccountsDaemon

include $(THEOS_MAKE_PATH)/tweak.mk
```

可以成功编译和链接，并不会报错。但是，因为 AccountsDaemon 这个 private framework 只存在于 6.0 及以上版本的 SDK 中，在 iOS 5 中是没有的，所以这个 tweak 在 iOS 5 中会因找不到 framework 而无法正常工作。这种情况可以通过弱链接（搜索“ weak linking”）或 dlopen()、dlsym() 和 dlclose() 系列函数动态调用 private framework 来解决。

❑ 链接 lib

```
iOSRE_LDFLAGS = -lx
```

Theos 采用 GNU Linker 来链接 lib。在 Terminal 中输入“ man ld”，定位到“ -lx”部分，如图 3-24 所示。

```
Options that control libraries
  -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.
```

图 3-24 lx 参数

图 3-24 中的大致意思是说，-lx 代表链接 libx.a 或 libx.dylib，即为 x 加上 lib 的前缀，和“.a”或“.dylib”的后缀；如果 x 是“y.o”的形式，则直接链接 y.o，不加任何前缀或后缀。由图 3-25 可知，iOS 支持链接的库全是以“libx.dylib”和“y.o”形式命名的，完全兼容 GNU Linker。

这样，链接 lib 就很方便了。例如，要链接 libsqlite3.0.dylib、libz.dylib 和 dylib1.o，像下面这么写就可以了：

```
iOSRE_LDFLAGS = -lz -lsqlite3.0 -dylib1.o
```

稍后还有一个字段需要介绍，但一般来说，MakeFile 中定义了以上字段就已经完全够用了。更详细的 MakeFile 介绍，可以参阅 http://www.gnu.org/software/make/manual/html_node/Makefiles.html。

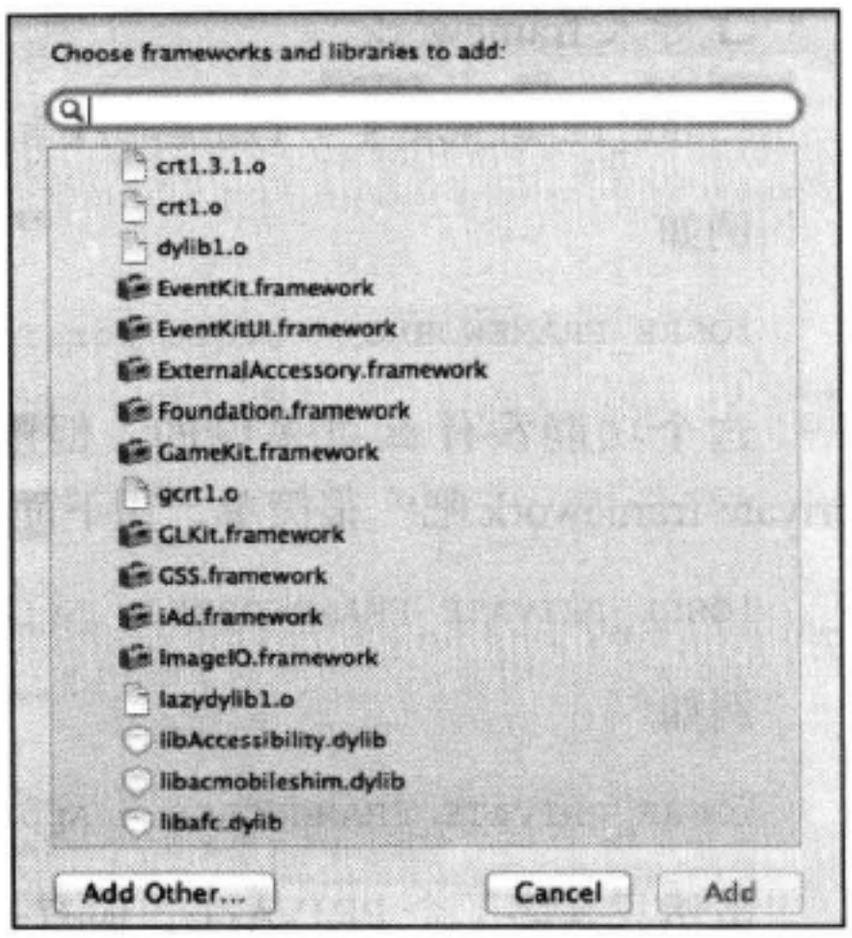


图 3-25 Xcode 中导入的 library 界面

（4）Tweak.xm

用 Theos 创建 tweak 工程，默认生成的模板源文件是 Tweak.xm。“xm”中的“x”代表这个文件支持 Logos 语法，如果后缀名是单独一个 x，说明源文件支持 Logos 和 C 语法；如果后缀名是 xm，说明源文件支持 Logos 和 C/C++ 语法，与 m 和 mm 的区别类似。Tweak.xm 的内容如下：

```
/* 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
*/
```

这就是最基本的 Logos 语法，包含 %hook、%log、%orig 这 3 个预处理指令，它们的作用分别如下。

❑ %hook

指定需要 hook 的 class，必须以 %end 结尾，如：

```
%hook SpringBoard
-(void)menuButtonDown:(id)down
{
    NSLog(@"你按下了 home 键！ ");
    %orig; // 调用 menuButtonDown: 的原始函数
}
%end
```

这段代码的意思是，hook 住 SpringBoard 类中的 menuButtonDown: 函数，先将“你按下了 home 键!”写入 syslog，再执行函数的原始操作。

❑ %log

该指令在 %hook 内部使用，将函数的类名、参数等信息写入 syslog，可以按 %log([(<type>)<expr>, ...]) 的格式追加其他打印信息，如：

```
%hook SpringBoard
-(void)menuButtonDown:(id)down
{
    %log((NSString *)@"iOSRE", (NSString *)@"Debug");
    %orig; // 调用 menuButtonDown: 的原始函数
}
%end
```

打印结果如图 3-26 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Aug  8 14:21:43 Touch-1 SpringBoard[15536]: MS:Notice: Loading: /Library/MobileS
ubstrate/DynamicLibraries/iOSRE.dylib
Aug  8 14:22:45 Touch-1 SpringBoard[15536]: -[<SpringBoard: 0x1f597b10> menuButt
onDown:<GSEvent 0x1f5b1eb0>{type = 0, windowLoc = (0, 0)}]: iOSRE, Debug
Touch-1:~ root#
```

图 3-26 %log 输出的日志信息

❑ %orig

该指令在 %hook 内部使用，执行被 hook 的函数的原始代码，如：

```
%hook SpringBoard
-(BOOL)launchApplicationWithIdentifier:(id)identifier suspended:(BOOL)suspended
{
    NSLog(@"启动 %@", identifier);
    return %orig;
}
%end
```

如果去掉 %orig，那么原始函数不会得到执行，例如：

```
%hook SpringBoard
-(void)menuButtonDown:(id)down
{
    // 按下 home 键没反应
}
%end
```

还可以利用 %orig 更改原始函数的参数，例如：

```
%hook SpringBoard
-(BOOL)launchApplicationWithIdentifier:(id)identifier suspended:(BOOL)suspended
{
    NSLog(@"启动 MobilePhone");
    return %orig(@"com.apple.mobilephone", suspended);
}
```

这样一来，只要 SpringBoard 调用 launchApplicationWithIdentifier: suspended: 函数来启动某个应用程序，实际启动的都是电话程序，用来恶作剧一下再好不过了，哈哈！

除了 %hook、%log、%orig，Logos 常用的预处理指令还有 %group、%init、%ctor、%new、%c，我们继续逐一介绍。

❑ %group

该指令用于将 %hook 分组，便于代码管理以及按条件初始化分组（含义稍后有详细解释），必须以 %end 结尾；一个 %group 可以包含多个 %hook，所有不属于某个自定义组的 %hook 被隐式归类到 %group _ungrouped 中。%group 用法如下：

```
%group iOS5Hook
%hook CKConversationList
-(id)conversationForRecipients:(id)recipients create:(BOOL)create service:(id)
service
{
    id result = %orig;
    NSLog(@"此函数存在于 iOS 5 中，iOS 6 中没有");
    return result;
}
%end
```

```
%end // iOS5Hook

%group iOS6Hook
%hook CKConversationList
-(id)conversationForRecipients:(id)recipients create:(BOOL)create
{
    id result = %orig;
    NSLog(@"此函数存在于iOS 6中,iOS 5中没有");
    return result;
}
%end
%end // iOS6Hook

%hook SpringBoard
-(void)powerDown
{
    %orig;
}
%end
```

这段代码的意思是，在%group iOS5Hook中钩住CKConversationList类的conversationForRecipients:create:service:函数，在%group iOS6Hook中钩住CKConversationList类的conversationForRecipients:create:函数，而在%group _ungrouped中钩住SpringBoard类的powerDown函数。

需要注意的是，%group必须配合下面的%init使用才能生效。

❑ %init

该指令用于初始化某个%group，必须在%hook或%ctor内调用。如果带参数，则初始化指定的是组，如果不带参数，则初始化_ungrouped，如：

```
%hook SpringBoard
-(void)applicationDidFinishLaunching:(id)application
{
    %orig;

    %init; // 等同于%init(_ungrouped)
     if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber_iOS_5_0
&& kCFCoreFoundationVersionNumber <= kCFCoreFoundationVersionNumber_iOS_5_1)
        %init(iOS5Hook);
     if (kCFCoreFoundationVersionNumber > kCFCoreFoundationVersionNumber_iOS_5_1)
        %init(iOS6Hook);
}
%end
```

只有调用了%init，对应的%group才能起作用，切记切记!

❑ %ctor

tweak的构造函数完成初始化工作；如果不显式定义，则Theos会自动生成一个%ctor，并在其中调用%init(_ungrouped)。因此，程序

```
%hook SpringBoard
-(void)reboot
{
    NSLog(@"打雷啦，下雨，收衣服啦！ ");
    %orig;
}
%end
```

可以成功生效，这是因为 Theos 帮我们隐式定义了

```
%ctor
{
    %init(_ungrouped);
}
```

而程序

```
%hook SpringBoard
-(void)reboot
{
    NSLog(@"打雷啦，下雨，收衣服啦！ ");
    %orig;
}
%end

%ctor
{
    // 需要显式调用 %init！
}
```

中的 %hook 无法生效，因为我们显式定义了 %ctor，却没有显式调用 %init，%group(_ungrouped) 不起作用。%ctor 一般可以用来初始化 %group 以及进行 MSHookFunction 等操作，如：

```
%ctor
{
    %init;
     if (kCFCoreFoundationVersionNumber >= kCFCoreFoundationVersionNumber_iOS_5_0
&& kCFCoreFoundationVersionNumber <= kCFCoreFoundationVersionNumber_iOS_5_1)
        %init(iOS5Hook);
    if (kCFCoreFoundationVersionNumber > kCFCoreFoundationVersionNumber_iOS_5_1)
        %init(iOS6Hook);
    MSHookFunction((void *)&AudioServicesPlaySystemSound,
        (void *)&replaced_AudioServicesPlaySystemSound,
        (void **)&original_AudioServicesPlaySystemSound);
}
```

注意，%ctor 不需要以 %end 结尾。

❑ %new

在 %hook 内部使用，为一个现有 class 添加新函数，功能与 class_addMethod 相同。它的

用法是：

```
%hook SpringBoard
%new
-(void)namespaceNewMethod
{
    NSLog(@"新添加了一个SpringBoard函数！");
}
%end
```

有的读者可能会问，Objective-C 的 category 语法也可以为现有 class 添加新函数，为什么还需要 %new 呢？其实原因就在于 category 与 class_addMethod 的区别，前者是静态的，而后者是动态的。那么在这种情况下，是静态还是动态，有什么关系呢？当然有关系，尤其是当 class 来自某个可执行文件的时候。举个例子，上面的代码为 SpringBoard 类添加了一个新函数，如果使用 category，代码应该是这样：

```
@interface SpringBoard (iOSRE)
-(void)namespaceNewMethod;
@end

@implementation SpringBoard (iOSRE)
-(void)namespaceNewMethod
{
    NSLog(@"新添加了一个SpringBoard函数！");
}
@end
```

如果你尝试编译上面的代码，会得到“error: cannot find interface declaration for ‘SpringBoard’”的报错信息，即编译器找不到SpringBoard类的定义。我们可以给出SpringBoard的定义：

```
@interface SpringBoard : NSObject
@end

@interface SpringBoard (iOSRE)
-(void)namespaceNewMethod;
@end

@implementation SpringBoard (iOSRE)
-(void)namespaceNewMethod
{
    NSLog(@"新添加了一个SpringBoard函数！");
}
@end
```

重新编译，仍然会报如图 3-27 所示的错。

ld 找不到 SpringBoard 的定义。一般来说，iOS 程序员在碰到这个错误时的第一反应是：“是不是忘了导入哪个 framework？”，但是转念一想，SpringBoard 类是 SpringBoard 这个

App 里的一个类，而不是一个 framework，要怎么导入呢？现在你是不是觉得 %new 非常可爱了呢？

```
Undefined symbols for architecture armv7:
  "_OBJC_CLASS_$_SpringBoard", referenced from:
      l_OBJC_$_CATEGORY_SpringBoard_$_category in Tweak.xm.20b682e7.o
      objc-class-ref in Tweak.xm.20b682e7.o
ld: symbol(s) not found for architecture armv7
collect2: ld returned 1 exit status
```

图 3-27 编译报错

❑ %c

该指令的作用等同于 objc_getClass()，即动态获取一个类的定义，在 %hook 或 %ctor 内使用。

Logos 的预处理指令还有 %subclass 和 %config，但笔者到现在也没有用过，感兴趣的读者可以移步 http://iphonedevwiki.net/index.php/Logos 一探究竟。

3. 编译 + 打包 + 安装

完成了 Theos 的安装，使用 NIC 创建了第一个 tweak 工程，还逐一解读了工程的组成文件，那么现在就剩下一步——编译了，完成这一步，就可以尝尝鲜了：把 tweak 安装到我们的 iDevice 上，从此开始周而复始的“安全模式”之旅，是不是很期待呢？

（1）编译

Theos 采用与 debian 相同的 make 命令来编译 Theos 工程。顺便提一句，如果 Terminal 提示“command not found”，那就说明没有安装 CLT（Command Line Tools）。

运行 make 命令，如图 3-28 所示。

从输出的信息看，Theos 完成了预处理、编译、签名等一系列动作，此时你会发现当前目录下多了一个新的 obj 文件夹，里面有一个 .dylib 文件，这就是 tweak 最核心的部分。

```
[redacted]-MacBook-Pro:iosre snakeninny$ make
Making all for tweak iOSRE...
 Preprocessing Tweak.xm...
 Compiling Tweak.xm...
 Linking tweak iOSRE...
 Stripping iOSRE...
 Signing iOSRE...
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-28 运行 make 命令

（2）打包

打包使用的 make package 命令来自 Theos 本身，其实就是先 make 再 dpkg-deb，运行结果如图 3-29 所示。

```
[redacted]-MacBook-Pro:iosre snakeninny$ make package
Making all for tweak iOSRE...
 Preprocessing Tweak.xm...
 Compiling Tweak.xm...
 Linking tweak iOSRE...
 Stripping iOSRE...
 Signing iOSRE...
Making stage for tweak iOSRE...
dpkg-deb: building package `com.naken.iosre' in `./com.naken.iosre_0.0.1-8_iphon
eos-arm.deb'.
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-29 运行 make package 命令

生成了一个名为“com.naken.iosre_0.0.1-8_iphoneos-arm.deb”的文件，这就是可以最终发布的安装包。

注意，OSX 10.9 默认不再自带 gnutar，在运行“make package”时，可能会报错“dpkg-deb (subprocess): failed to exec tar -cf: No such file or directory”。解决办法是下载 https://github.com/DHowett/dm.pl/raw/master/dm.pl，将其重命名为 dpkg-deb，运行“sudo chmod a+x dpkg-deb”赋予其执行权限，再将其放到 $THEOS/bin 下即可。

make package 命令还有一个很重要的功能。在执行完 make package 之后，除了 obj 文件夹外，你会发现 tweak 工程目录下还生成了一个“_”的文件夹，如图 3-30 所示。

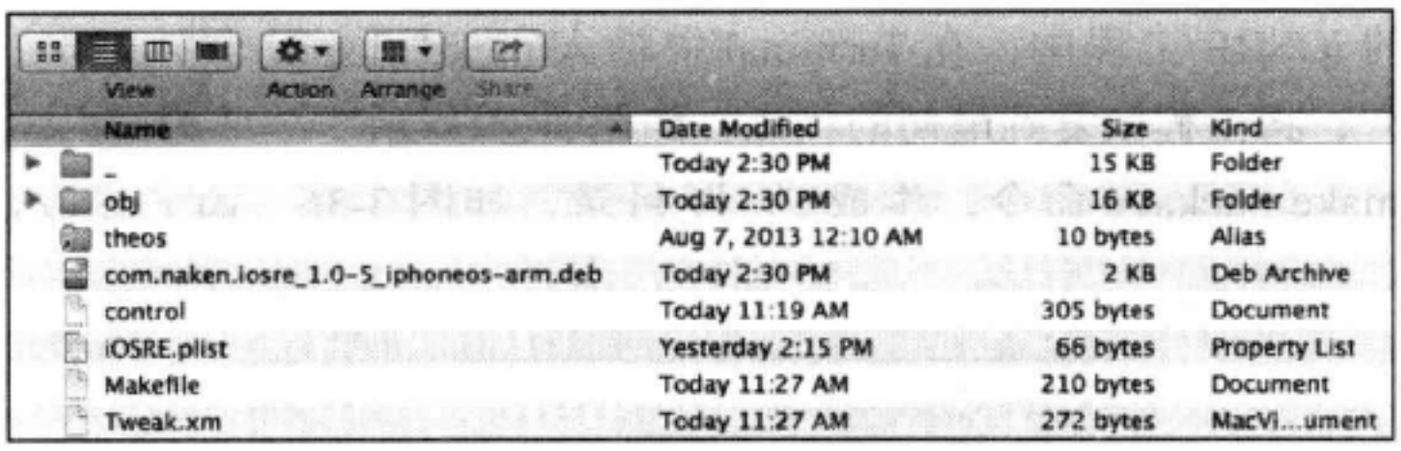


图 3-30 工程目录展示

这个文件夹是干什么的？打开它，可以看到两个文件夹，分别是 DEBIAN 和 Library，如图 3-31 所示。

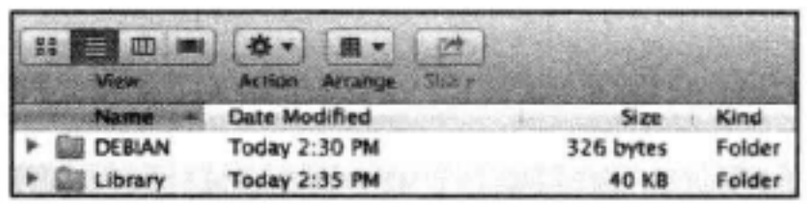


图 3-31 DEBIAN 和 Library 目录

其中 DEBIAN 下只有 tweak 工程中的 control 文件，Theos 在编译过程中稍稍增加了几个字段而已。Library 的目录结构如图 3-32 所示。

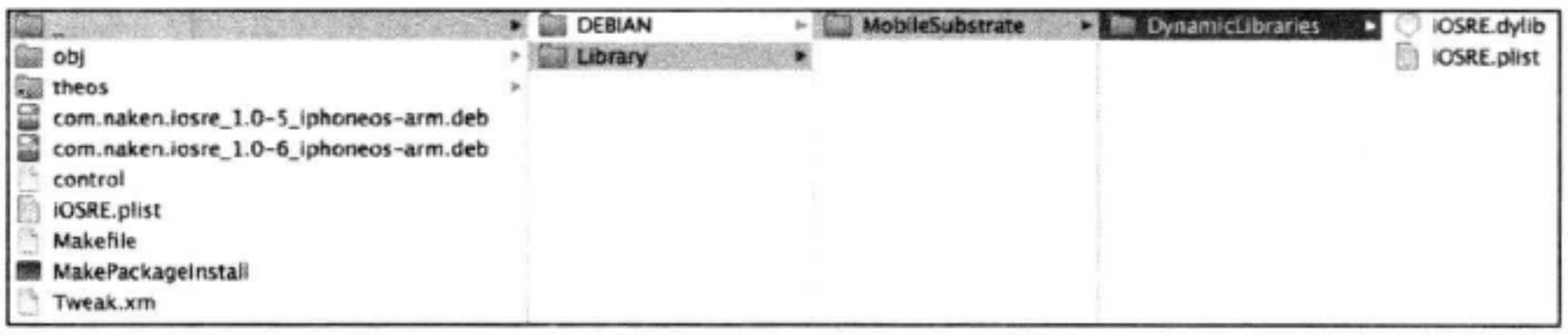


图 3-32 Library 目录详情 1

对比生成 deb 包的内容，如图 3-33 所示。

```
[redacted]-MacBook-Pro:iosre snakeninny$ dpkg-deb --contents /Users/snakeninny/Code/iosre/com.naken.iosre_1.0-5_iphoneos-arm.deb
drwxr-xr-x root/wheel         0 2013-08-09 14:30 ./
drwxr-xr-x root/wheel         0 2013-08-09 14:30 ./Library/
drwxr-xr-x root/wheel         0 2013-08-09 14:30 ./Library/MobileSubstrate/
drwxr-xr-x root/wheel         0 2013-08-09 14:30 ./Library/MobileSubstrate/DynamiclLibraries/
-rwxr-xr-x root/wheel      8880 2013-08-09 14:30 ./Library/MobileSubstrate/DynamiclLibraries/iOSRE.dylib
-rw-r--r-- root/wheel        66 2013-08-09 14:30 ./Library/MobileSubstrate/DynamiclLibraries/iOSRE.plist
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-33 Library 目录详情 2

在 Cydia 中 iOSRE 的文件系统如图 3-34 所示。

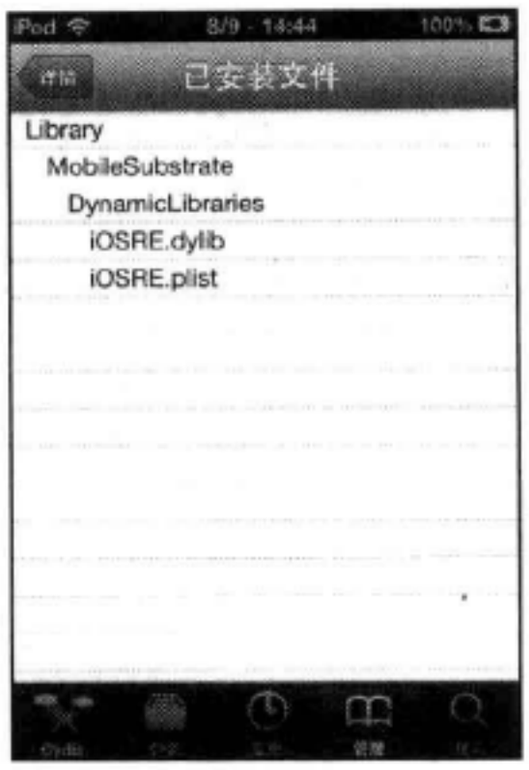


图 3-34 Cydia 中的文件详情

三者是完全相同的。看到这里，你可能也猜到了，这个 deb 包其实就是由 DEBIAN 提供的 debian 信息，Library 提供实际文件的简单组合。事实上，我们可以在这个“_”目录中添加任意子目录，然后将“_”重命名为 layout，重新生成 deb 包并安装到 iOS 中。如此一来，layout 中除 DEBIAN 以外的文件会被解包到 iOS 文件系统的相同位置（这里的 layout 相当于 iOS 中的“/”），极大扩充了 deb 包的作用范围。下面用一个小例子佐以说明：

回到刚才的 iOSRE 工程中，在 Terminal 中输入 make clean 以及“rm *.deb”，将工程恢复到最初的状态，如图 3-35 所示。

然后运行 make package 命令，生成“_”目录，如图 3-36 所示。

```
[redacted]-MacBook-Pro:iosre snakeninny$ make clean
rm -rf ./obj
rm -rf "/Users/snakeninny/Code/iosre/_"
[redacted]-MacBook-Pro:iosre snakeninny$ rm *.deb
[redacted]-MacBook-Pro:iosre snakeninny$ ls -a
.                .theos           control
..               Makefile         iOSRE.plist
.DS_Store        Tweak.xm         theos
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-35 运行 make clean 命令

```
[redacted]-MacBook-Pro:iosre snakeninny$ make package
Making all for tweak iOSRE...
 Preprocessing Tweak.xm...
 Compiling Tweak.xm...
 Linking tweak iOSRE...
 Stripping iOSRE...
 Signing iOSRE...
Making stage for tweak iOSRE...
dpkg-deb: building package `com.naken.iosre' in `./com.naken.iosre_1.0-9_iphone
os-arm.deb'.
[redacted]-MacBook-Pro:iosre snakeninny$ ls -a
.
..
.DS_Store
.theos
Makefile
Tweak.xm
_
com.naken.iosre_1.0-9_iphoneos-arm.deb
control
iOSRE.plist
obj
theos
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-36 运行 make package 命令

继续输入 vi，按下 i，切换到插入模式；然后输入 iOSRE，再按下 esc，切换到编辑模式；再按下“shift + ;”，切换到命令模式，输入“wq /Users/snakeninny/Code/iosre/_/1.test”，如图 3-37 所示。

执行完成后，“/Users/snakeninny/Code/iosre/_”下增加了一个 1.test，如图 3-38 所示。

```
  1 iOSRE
~
~
~
:wq /Users/snakeninny/Code/iosre/_/1.test
```

图 3-37 编辑 1.test

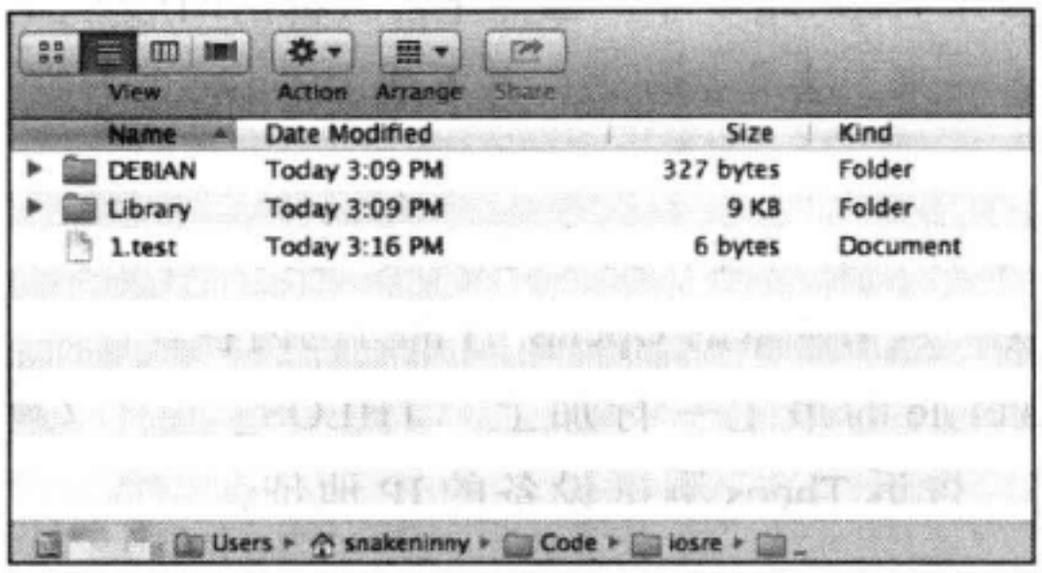


图 3-38 工程目录展示

下面把 1.test 散落到 iOS 的各个角落，如图 3-39 所示。

```
-MacBook-Pro:iosre snakeninny$ mkdir -p /Users/snakeninny/Code/iosre/_/sn
akeninny
-MacBook-Pro:iosre snakeninny$ mkdir -p /Users/snakeninny/Code/iosre/_/va
r/mobile/Library/iOSRE
-MacBook-Pro:iosre snakeninny$ mkdir -p /Users/snakeninny/Code/iosre/_/De
veloper
-MacBook-Pro:iosre snakeninny$ mkdir -p /Users/snakeninny/Code/iosre/_/Sy
stem/Library/AccessibilityBundles/iPodUI.axbundle
-MacBook-Pro:iosre snakeninny$ cp /Users/snakeninny/Code/iosre/_/1.test /
Users/snakeninny/Code/iosre/_/snakeninny/2.test
-MacBook-Pro:iosre snakeninny$ cp /Users/snakeninny/Code/iosre/_/1.test /
Users/snakeninny/Code/iosre/_/var/mobile/Library/iOSRE/3.test
-MacBook-Pro:iosre snakeninny$ cp /Users/snakeninny/Code/iosre/_/1.test /
Users/snakeninny/Code/iosre/_/Developer/4.test
-MacBook-Pro:iosre snakeninny$ cp /Users/snakeninny/Code/iosre/_/1.test /
Users/snakeninny/Code/iosre/_/System/Library/AccessibilityBundles/iPodUI.axbundl
e/5.test
-MacBook-Pro:iosre snakeninny$
```

图 3-39 将 1.test 散落各个子目录

再调用“mv /Users/snakeninny/Code/iosre/_ /Users/snakeninny/Code/iosre/layout”把“_”重命名为“layout”。最后用“make package install”命令来安装，并在 Cydia 中查看 iOSRE 的文件系统，如图 3-40 所示。

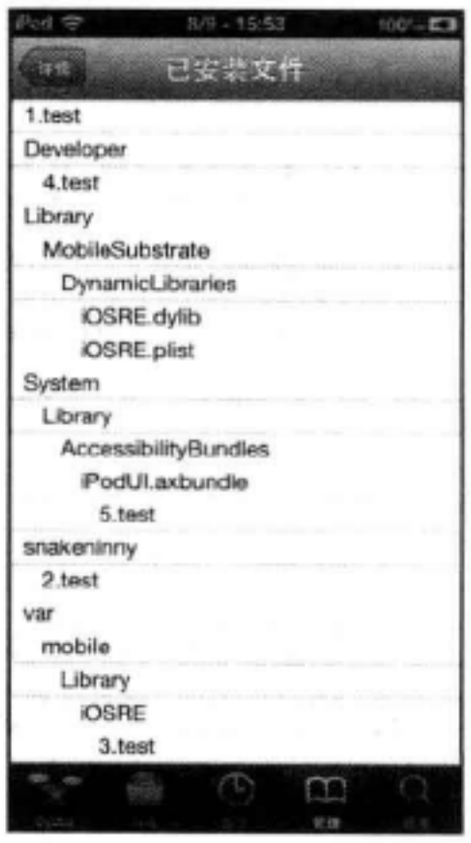


图 3-40 Cydia 中的文件展示

除 DEBIAN 以外的所有文件都被解压到 iOS 文件系统的相同位置，本来不存在的中间文件夹也被自动创建。deb 包的玄机还有很多，笔者也只是管中窥豹，更全面的介绍请登录 http://www.debian.org/doc/debian-policy，官方文档一直是我们最好的学习资料。

（3）安装

最后，要把这个 deb 文件安装到 iOS 中去。安装的方法多

种多样，这里介绍两种最具代表性的：图形界面安装法和命令行安装法。大多数人的第一直觉是图形界面一定比命令行简单，那好，咱们先介绍图形界面安装法。

图形界面安装法

这个方法确实简单：通过 iFunBox、iTools 等软件把 deb 拖到 iOS 中去，然后用 iFile 安装它，最后用 SBSettings 来 respring 一下。虽然全过程都由图形界面操作，但人机交互太多，又要动电脑又要滑手机，一来二去非常烦琐，并不适用于 tweak 开发。

命令行安装法

这个方法要用到简单的 SSH 命令，要求越狱 iOS 安装 OpenSSH，如果你对这部分知识不了解，建议先快速浏览一遍第 4 章的 OpenSSH 部分的内容。

首先，需要在 MakeFile 的最上一行加上“THEOS_DEVICE_IP = 本机 IP”，告诉 Theos 测试设备的 IP 地址，如图 3-41 所示。

```
1 THEOS_DEVICE_IP = 192.168.1.108
2 ARCHS = armv7
3 TARGET = iphone:6.1:4.3
```

图 3-41 指定 THEOS_DEVICE_IP

然后调用“make package install”命令完成编译打包安装一条龙服务，如图 3-42 所示。

```
[redacted]-MacBook-Pro:iosre snakeninny$ make package install
Making all for tweak iOSRE...
 Preprocessing Tweak.xm...
 Compiling Tweak.xm...
 Linking tweak iOSRE...
 Stripping iOSRE...
 Signing iOSRE...
Making stage for tweak iOSRE...
dpkg-deb: building package `com.naken.iosre' in `./com.naken.iosre_0.0.1-9_iphon
eos-arm.deb'.
install.copyFile "./com.naken.iosre_0.0.1-9_iphoneos-arm.deb" "com.naken.iosre_0
.0.1-9_iphoneos-arm.deb"
root@192.168.1.108's password:
com.naken.iosre_0.0.1-9_iphoneos-arm.deb      100% 2148     2.1KB/s   00:00
install.exec "dpkg -i com.naken.iosre_0.0.1-9_iphoneos-arm.deb"
root@192.168.1.108's password:
(Reading database ... 5746 files and directories currently installed.)
Preparing to replace com.naken.iosre 0.0.1-7 (using com.naken.iosre_0.0.1-9_ipho
neos-arm.deb) ...
Unpacking replacement com.naken.iosre ...
Setting up com.naken.iosre (0.0.1-9) ...
install.exec "killall -9 SpringBoard"
root@192.168.1.108's password:
[redacted]-MacBook-Pro:iosre snakeninny$
```

图 3-42 运行 make package install 命令

从以上信息中可以看到，Theos 在把 deb 文件复制到 iOS 中、安装 deb 文件和 respring 时各要求我们输入一次 root 密码。虽然多次输入密码给人很安全的感觉，但实在是太麻烦了。好在通过设置 iOS 的 authorized_keys 可以省略 SSH 输密码的步骤，让 make package install 真正地从“一只多脚虫”变成“一条飞天龙”，具体步骤如下（记得将下面内容中的 snakeninny 改成你的对应目录）：

❑ 删除“/Users/snakeninny/.ssh/known_hosts”中本机 IP 的条目

具体做法是在 Terminal 中输入“vi /Users/snakeninny/.ssh/known_hosts”，将光标移动到本机 IP 那一行，按下 v 键，再按下“shift + 4”组合键，选中这一行，如图 3-43 所示。

然后按下 c 键删掉这一行，再依次按下 esc、“shift + ;”，输入 wq 并按 Enter 键，保存文件。当然也可以在 Finder 中完成上述操作，两种方案，由你选择。

```
5 192.168.1.108 ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD9N9mpJTUgLYkqGTM/NT0RSz
  nW4nePd/vDzwAC4qLl06DSHyjfGxxggdotjjKzJkqn6xvIp3pSTZdDiFDgM0czDU63VxKZi0tZAn
  /DsTfY9imCsTAf8C9v0FEsnUIS0j83Z4UnDjujazgLzzLm0FkD2oy5ZsTG2chLXPWrkiSOOEbBqD
  TbdhDLGuG2Y4vp1AHDKweq/jYPkxFlMCj0WY+/eU+bXMuJoiDhoCfr+FeD4pFvoiq4/55JHXllE7
  6GGTpvfihI4gC6DuqJuS7kGay30d0qliblд0m7/4RU5hkE7lE2J0bAw42MpVllEEyitaIPolwTqt
  4cSlIIMXOn000d
```

图 3-43 修改 known_hosts

❑ 生成 authorized_keys

在 Terminal 中输入“ ssh-keygen -t rsa”，执行完成后输入“ cp /Users/snakeninny/.ssh/id_rsa.pub ~/authorized_keys”，在用户目录下生成 authorized_keys。

❑ 配置 iOS

以 root 身份远程登录到 iOS 中，输入“ ssh-keygen”，生成“ /var/root/.ssh”目录；然后输入 exit 退出远程登录，再输入“ scp ~/authorized_keys root@192.168.1.108:/var/root/.ssh”，如图 3-44 所示。

```
[redacted]-MacBook-Pro:~ snakeninny$ scp ~/authorized_keys root@192.168.1.108:/var/
root/.ssh
root@192.168.1.108's password:
authorized_keys                               100%  418     0.4KB/s   00:00
[redacted]-MacBook-Pro:~ snakeninny$
```

图 3-44 上传 authorized_keys

输入一次 root 密码，结束战斗。重新远程登录到 iOS，还需要输密码吗？我们的 make package install 真正变成了一次配置，一键安装，一劳永逸！

（4）清理

Theos 还提供了方便的工程清理命令 make clean，实际作用就是依次执行“ rm -rf ./obj”和“ rm -rf /Users/snakeninny/Code/iosre/_”两个命令，删除 make 和 make package 命令生成的文件夹。

3.2.4 Theos 开发 tweak 示例

前几节完整地介绍了 Theos 的安装和使用方法，虽然并没有涵盖 Theos 的所有功能，但对于逆向工程初学者来说已经完全够用了。讲了这么多内容却还没有涉及一行真实的代码，是不是有些意犹未尽啊？

接下来我们以一个最简单的 tweak 为例来进行讲解。安装了该程序之后，每次重启 SpringBoard 都将会弹出一个对话框。

1. 编辑 control

```
Package: com.naken.iosre
Name: iOSRE
Depends: mobilesubstrate, firmware (>= 5.0)
Version: 1.0
Architecture: iphoneos-arm
Description: 在 SpringBoard 启动后弹出 UIAlertView！
```

```
Maintainer: snakeninny
Author: snakeninny
Homepage: http://iosre.com
Section: Tweaks
```

相比默认生成的 control 文件，这里在 Depends 字段加上了对 iOS 版本的要求，包括将 Version 改为 1.0，更改 tweak 的 Description（描述），还增加了官方网站信息等。

2. 编辑 iOSRE.plist

打开 iOSRE.plist 后，指定 tweak 的作用对象是 SpringBoard，如图 3-45 所示。

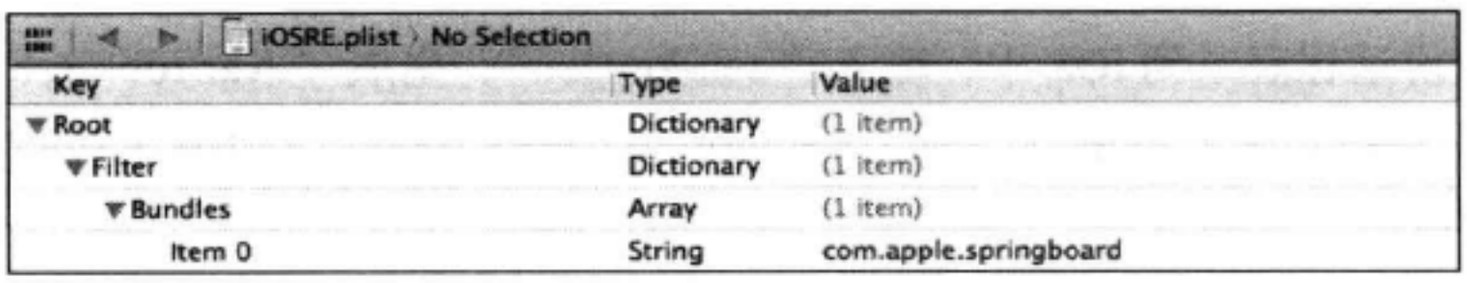


图 3-45 指定 Filter

3. 编辑 MakeFile

```
THEOS_DEVICE_IP = 192.168.1.108
ARCHS = armv7
TARGET = iphone:6.1:4.3

include theos/makefiles/common.mk

TWEAK_NAME = iOSRE
iOSRE_FILES = Tweak.xm
iOSRE_FRAMEWORKS = UIKit

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 SpringBoard"
```

4. 编辑 Tweak.xm

```
%hook SpringBoard

-(void)applicationDidFinishLaunching:(id)application
{
    %orig;

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Hello, Tweak!"
    message:nil delegate:self cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alert show];
    [alert release];
}

%end
```

以上代码非常简单，当 SpringBoard 的 applicationDidFinishLaunching: 函数得到调用时，代表 SpringBoard 的启动过程已经结束。我们钩住这个函数，首先调用 %orig 完成它的原始操作，然后弹出一个自定义的 UIAlertView，这样一来，每次重启 SpringBoard 都会弹出一个对话框。你看懂了吗?

准备就绪，在 Terminal 中键入“make package install”，待 SpringBoard 重启之后会看到图 3-46 所示的结果，有没有感到非常轻松愉快?

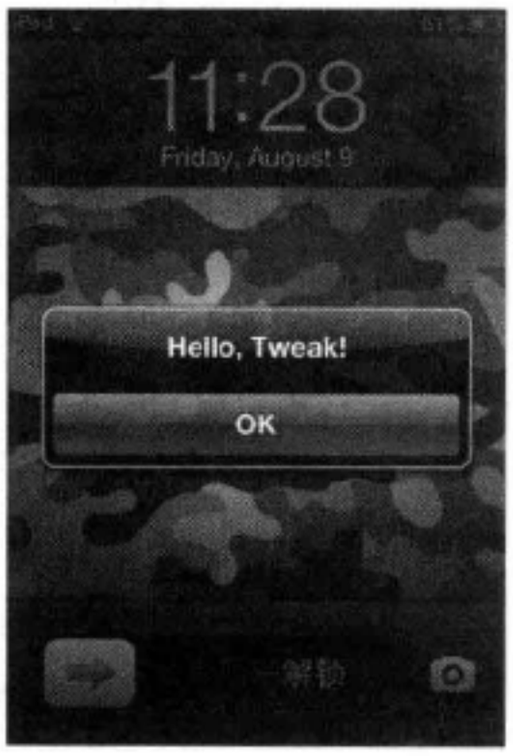


图 3-46 tweak 效果展示

是的，仅仅是这样一些小小的改动，就已经可以改变 App 的行为了。此时，封闭的 iOS 已经向我们打开了大门……

因为有 Theos 这样的开发工具存在，修改闭源的 iOS 程序变得前所未有的简便。不过在前面也提到了，现在的 App 工程量越来越大，通过 class-dump 得到的 .h 文件也越来越多，从浩如烟海的函数名中筛选出我们感兴趣的目标，比确定目标后编写代码要难得多。面对成千上万行代码，如果没有其他工具辅助分析，逆向工程简直会变成一种噩梦，让人一筹莫展。那么接下来，就轮到这些辅助分析工具隆重登场了。

3.3 Reveal

3.3.1 Reveal 简介

Reveal 是一个很强大的 UI 分析工具，与其他几个功能相近的工具（如 PonyDebugger）相比，其最大的特点就是非常直观，用来查看 App 的 UI 布局非常方便。

图 3-47 是 用 Reveal 抓 取 Instagram 的效果。可从图中左侧看到，界面元素全部以树形方式展现出来。优异的交互能够让我们非常快速地定位到各个 UI 元素，同样也就能看到各个类名，比如 IGFeedItemPhotoCell。有了这个名字，再打开那些用 class-dump 导出来的 .h 文件来看看吧!

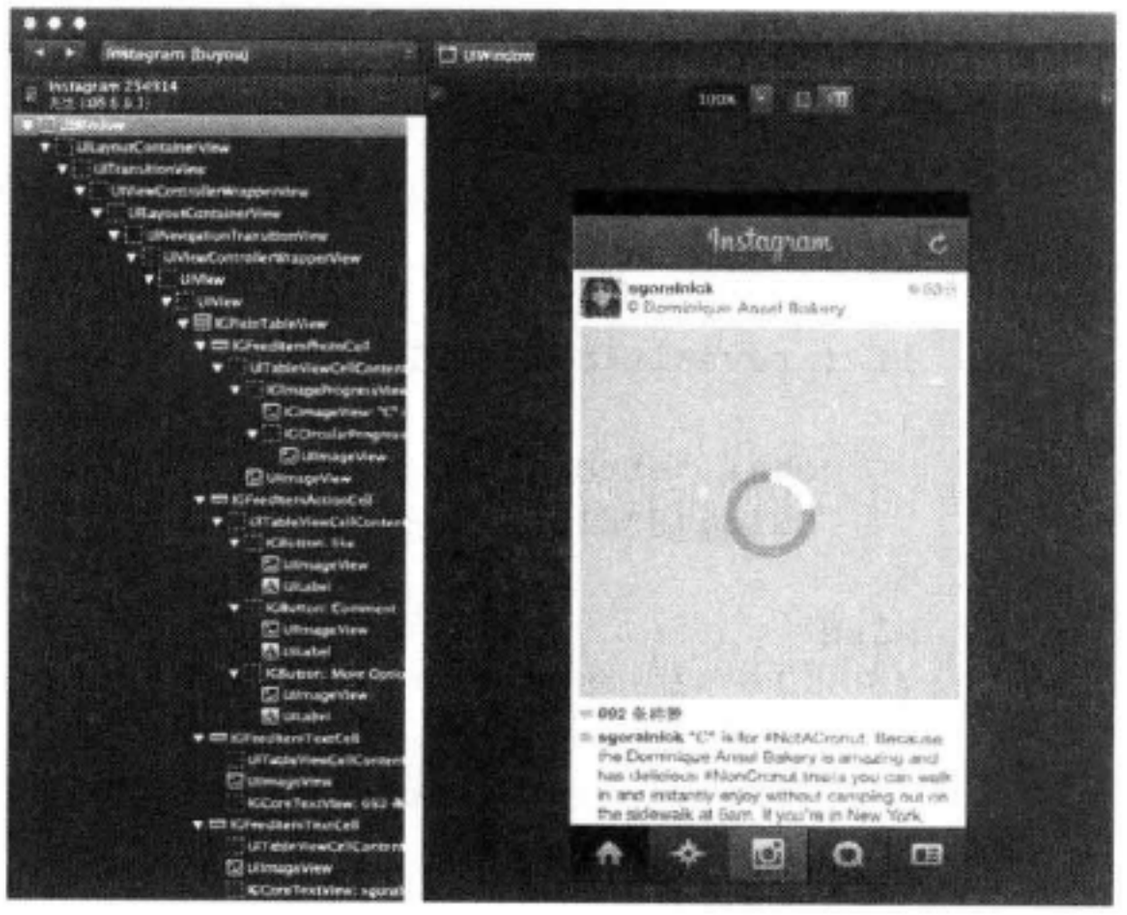


图 3-47 用 Reveal 剖析 App 的效果展示

顺便说明一下，Reveal 官网提供的方法只能监测、调试自己的 App，第三方的 App 无法监测，也就是说，想要实现图 3-47 的效果，需要对 Reveal 做一下扩展。

3.3.2 Reveal 安装及功能扩展

读者可从 http://revealapp.com/ 下载 Reveal（如图 3-48 所示），笔者当前使用的 0.8.3 Beta 版可以免费试用，如果正式版推出之后可能就不免费了。如果 Beta 版已过期而读者又很想继续使用 Reveal，除了购买之外，还可以将电脑的时间调整提前。

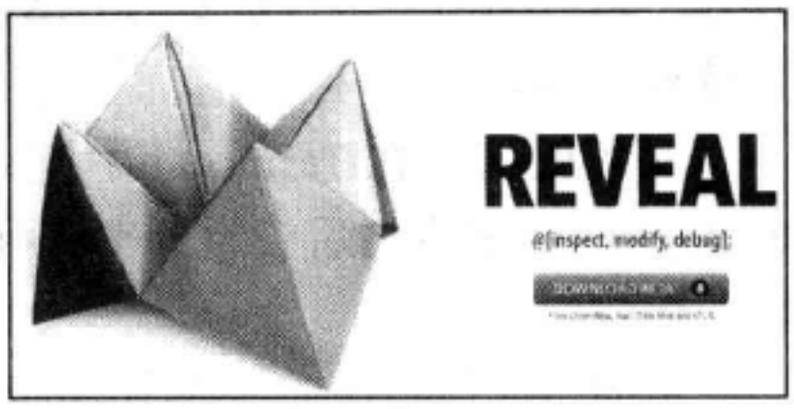


图 3-48 Reveal 网站展示

从它的网站上看，能看到 3 个关键字：inspect、modify、debug。不过在这里只用到了 insepct，至于 debug，等 Reveal 更强大一些再说吧。

Reveal 的安装非常简单，环境要求仅仅是 Mac OSX 10.8 和 iOS 6 以上即可。不过，笔者曾经犯过一个错，在 iOS 5 上来回折腾，可就是出不来，然后大骂 Reveal……

Reveal 有两种使用方法，分别是 Static Library Integration 和 Dynamic Library Integration，详情可参见 http://support.revealapp.com/kb/getting-started/integrating-reveal-with-your-ios-app。前者通过 Static Library 集成到 App 中，这个不是咱的菜，略过。接下来看看 Dynamic Library 方法。

Reveal 官网上介绍的方法就是通过 dlopen 函数来加载 libReveal.dylib 并达到 Dynamic Library Integration 效果的。

```
- (void)startReveal
{
    NSString *revealLibName = @"libReveal.dylib";
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *dyLibPath = [NSString stringWithFormat:@"%@/%@", documentsDirectory,
revealLibName];

    _revealLib = NULL;
    _revealLib = dlopen([dyLibPath cStringUsingEncoding:NSUTF8StringEncoding],
RTLD_NOW);

    if (_revealLib == NULL)
    {
        char *error = dlerror();
        NSLog(@"dlopen error: %s", error);
    }
    else
    {
        // Post a notification to signal Reveal to start the service.
        [[NSNotificationCenter defaultCenter] postNotificationName:@"IBARevealReq
uestStart" object:self];
    }
}
```

等等！要说明一下，网站上介绍的这个 Dynamic Library 方法也只是把 dylib 注入到使用者已有代码的 App 中。可这种情况下，代码我都有了，还要你干啥？即使做得再炫，也不过是个可有可无的鸡肋，似乎用 Xcode 自带的 Instruments 还更直接一些。

我们的目标既然是用 Reveal 来查看那些我们没有源代码的 App，而 Reveal 的标准模式并不支持这功能，那么就需要想想有什么方法能够达到这个目标了。

事实上，Reveal 的 Dynamic Library 及其调用方法本身就意味着能够扩展，它所使用的代码如下：

```
    _revealLib = dlopen([dyLibPath cStringUsingEncoding:NSUTF8StringEncoding],
RTLD_NOW);
```

也就是说，只要能够让目标 App 执行这样的代码，就可以让 Dynamic Library 在目标 App 进程中运行起来了。而这么做所依赖的知识就是前两节提到的 class-dump 和 Theos。

接下来就开始一步步地对 Reveal 进行功能扩展吧！

1. 获取 libReveal.dylib

点击菜单 Help / Show Reveal Library in Finder，获取 libReveal.dylib，如图 3-49 所示。

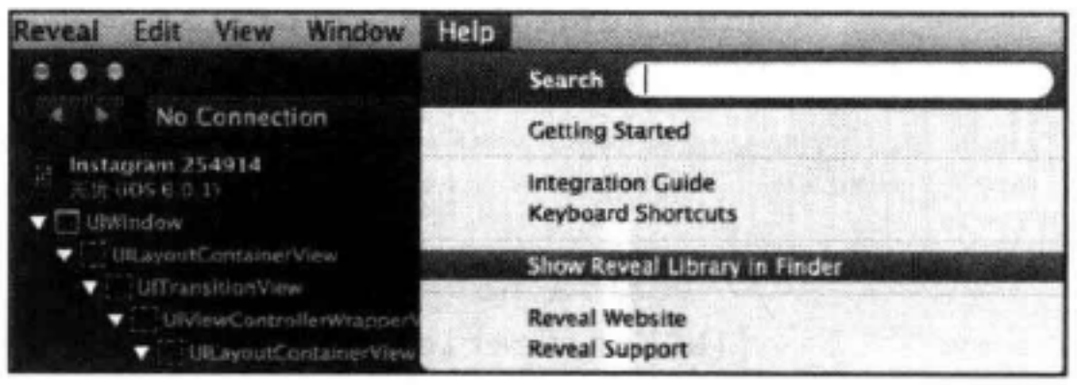


图 3-49 如何获取 libReveal.dylib

2. 将 libReveal.dylib 导入目标 App 的目录

可以通过 SSH 的方式将文件上传，不过更方便的是使用 iTools（后面也会提到），如图 3-50 所示。

按目前的代码约定，libReveal.dylib 需要放在 Documents 目录下，其实这也是相对的。

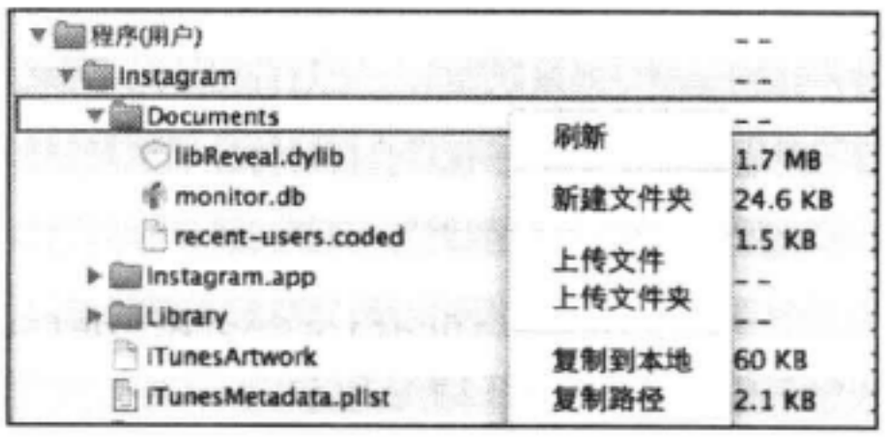


图 3-50 使用 iTools 将 libReveal.dylib 上传至 App 的 Documents 目录

3. 创建 RevealUtil

RevealUtil 的代码其实来源于 Reveal 网站，这里被笔者合并成了一个单独的 Util 文件，以备后面使用，如下所示：

```
//
//  RevealUtil.m
//  contactutil
//
//  Created by sam on 13-6-4.
//  Copyright (c) 2013年 sam. All rights reserved.
//

#import "RevealUtil.h"
```

```
#import <dlfcn.h>

@implementation RevealUtil

- (void)startReveal
{
    NSString *revealLibName = @"libReveal.dylib";
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *dyLibPath = [NSString stringWithFormat:@"%@/%@", documentsDirectory,
revealLibName];

    _revealLib = NULL;
    _revealLib = dlopen([dyLibPath cStringUsingEncoding:NSUTF8StringEncoding],
RTLD_NOW);

    if (_revealLib == NULL)
    {
        char *error = dlerror();
        NSLog(@"dlopen error: %s", error);
    }
    else
    {
        // Post a notification to signal Reveal to start the service.
        [[NSNotificationCenter defaultCenter] postNotificationName:@"IBARevealReq
uestStart" object:self];
    }
}

- (void)stopReveal
{
    if (_revealLib)
    {
        [[NSNotificationCenter defaultCenter] postNotificationName:@"IBARevealReq
uestStop" object:self];
        if (dlclose(_revealLib) == 0)
        {
            _revealLib = NULL;
        }
        else
        {
            char *error = dlerror();
            NSLog(@"Reveal library could not be unloaded: %s", error);
        }
    }
}

@end
```

再对应地把 .h 文件配好，如下所示：

```
//
//  RevealUtil.h
//  contactutil
//
//  Created by sam on 13-6-4.
//  Copyright (c) 2013年 sam. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface RevealUtil : NSObject {
    void *_revealLib;
}

- (void)startReveal;
- (void)stopReveal;

@end
```

之所以将官网介绍的代码封装进 RevealUtil，是为了供后面的 tweak 代码调用。因为从模块划分的角度，tweak 只完成了一个 hook 的功能，而主要的执行代码还是放在对应的 .m 文件中的。

4. 使用 Theos

可以创建一个新的 Theos 工程，不过直接沿用前面提到的 monitor tweak 工程更省事，只需要做一点修改就可以将我们的工作继续进行下去。

1）首先，修改 monitor.plist，指定 Filter，如下所示：

```
{ Filter = { Bundles = ( "com.burbn.instagram" ); }; }
```

2）然后创建一个 Tweak_Instagram.xm，如下所示：

```
#import "RevealUtil.h"

%hook AppDelegate

- (BOOL)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2
{
    %orig;

    RevealUtil *ru = [[RevealUtil alloc] init];
    [ru startReveal];

    return YES;
}

%end
```

每个 App 都有自己不同的 AppDelegate 文件，比如 WhatsApp 的 delegate 入口文件是

WhatsAppAppDelegate，而 Instagram 的 delegate 入口文件是 AppDelegate。它们都需要执行 didFinishLaunchingWithOptions，有经验的 iOS 程序员都能理解。至于 AppDelegate 文件名的定位，则是根据 class-dump 或 Cycript 得到的，具体的细节会在实战案例中讲到。

3）修改 Makefile。

增加了 Tweak_Instagram.xm 和 RevealUtil.m 之后，需要更新一下 Makefile 文件，如下所示：

```
include theos/makefiles/common.mk

TWEAK_NAME = monitor
monitor_FILES = Tweak.xm Tweak_Instagram.xm RevealUtil.m
monitor_FRAMEWORKS = UIKit CFNetwork
monitor_PRIVATE_FRAMEWORKS = AppSupport
include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 Instagram "
```

Makefile 中不仅仅增加了 Tweak_Instagram.xm 和 RevealUtil.m，并且还在 monitor_FRAMEWORKS 中增加了 CFNetwork，在 monitor_PRIVATE_FRAMEWORKS 中增加 AppSupport，这两个参数是针对 Reveal 添加的，有了它们，Reveal 才能正常起作用。

4）使用 make/make package。

之前已经提过 make/make package 的用法，这两步执行完就会生成插件对应的一个 deb 文件。还有一个 make install 命令，不过由于该命令要求 iOS 设备必须打开 SSH，需要多次输入密码，甚至还要重启 SpringBoard，所以为了方便起见，这次采用 iTools 安装。

5. 安装 monitor 插件

在代码开发完成并打包生成 deb 之后，就需要把这个 deb 安装到设备上去试试了。

（1）上传 monitor.deb

生成的 deb 名称格式类似于 com.hangcom.monitor_0.0.1-1_iphoneos-arm.deb，连接上 iOS 设备后，上传至 /var/root，这个路径是个默认值（如图 3-51 所示），也可以自己选一个看着顺眼的。



图 3-51 使用 iTools 上传 deb

（2）安装 monitor.deb

打开 iOS 设备上的 iFile，找到上传的 monitor.deb，点击选择“安装程序”，如图 3-52 所示。此时如果 Instagram 还开着，需要人为关闭一下。

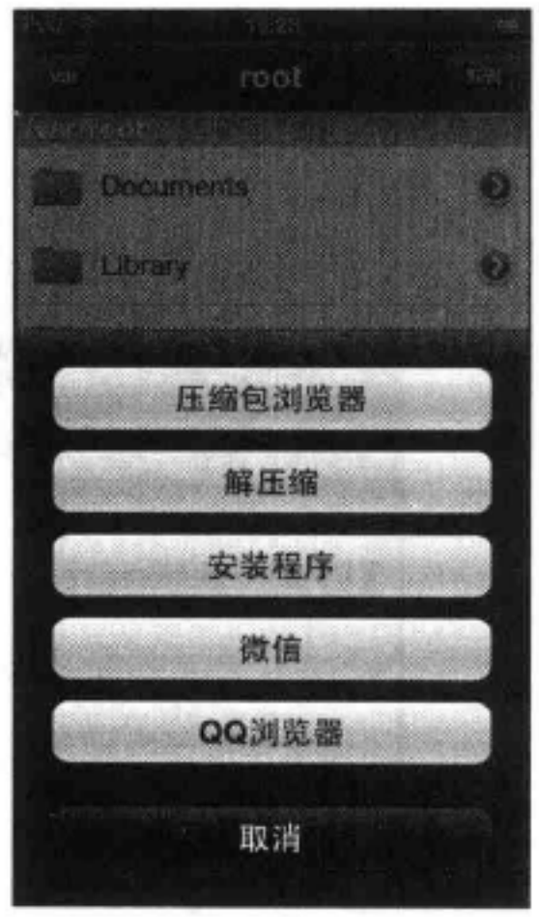


图 3-52 使用 iFile 安装 deb

6. 体验 Reveal 的扩展功能

好了，现在准备工作已经完成，该体验一下 Reveal 有多么亲切可爱了。

首先要保证的是 Mac 和 iOS 在同一个无线网内，也就是连接的是同一个 Wi-Fi，至于 Mac 和 iOS 设备是否连着数据线已经不重要了。

然后启动 Reveal，从界面左上角选择要连接的 App，进入不同的页面之后还可以点击右上角的刷新按钮来刷新监测的页面信息，如图 3-53 所示。

以此类推，用 Reveal + Theos 的组合可以监控任意你想看的 App，它们的 UI 结构从此在你眼前一目了然，由此就可以向更深处挖掘。



图 3-53 利用 Reveal 剖析 Instagram 的效果展示

3.4 IDA

3.4.1 IDA 简介

即使你以前没有从事过逆向工程相关的工作，IDA（The Interactive Disassembler）在业界

的鼎鼎大名你也一定听说过，而对于大多数 IT 从业者来说，IDA 三个字则是如雷贯耳，它乃逆向工程中最负盛名的神器之一，如图 3-54 所示。如果说 class-dump 能够帮我们罗列出要分析的点，那 IDA 就能进一步帮我们把这些点连成线，形成面。

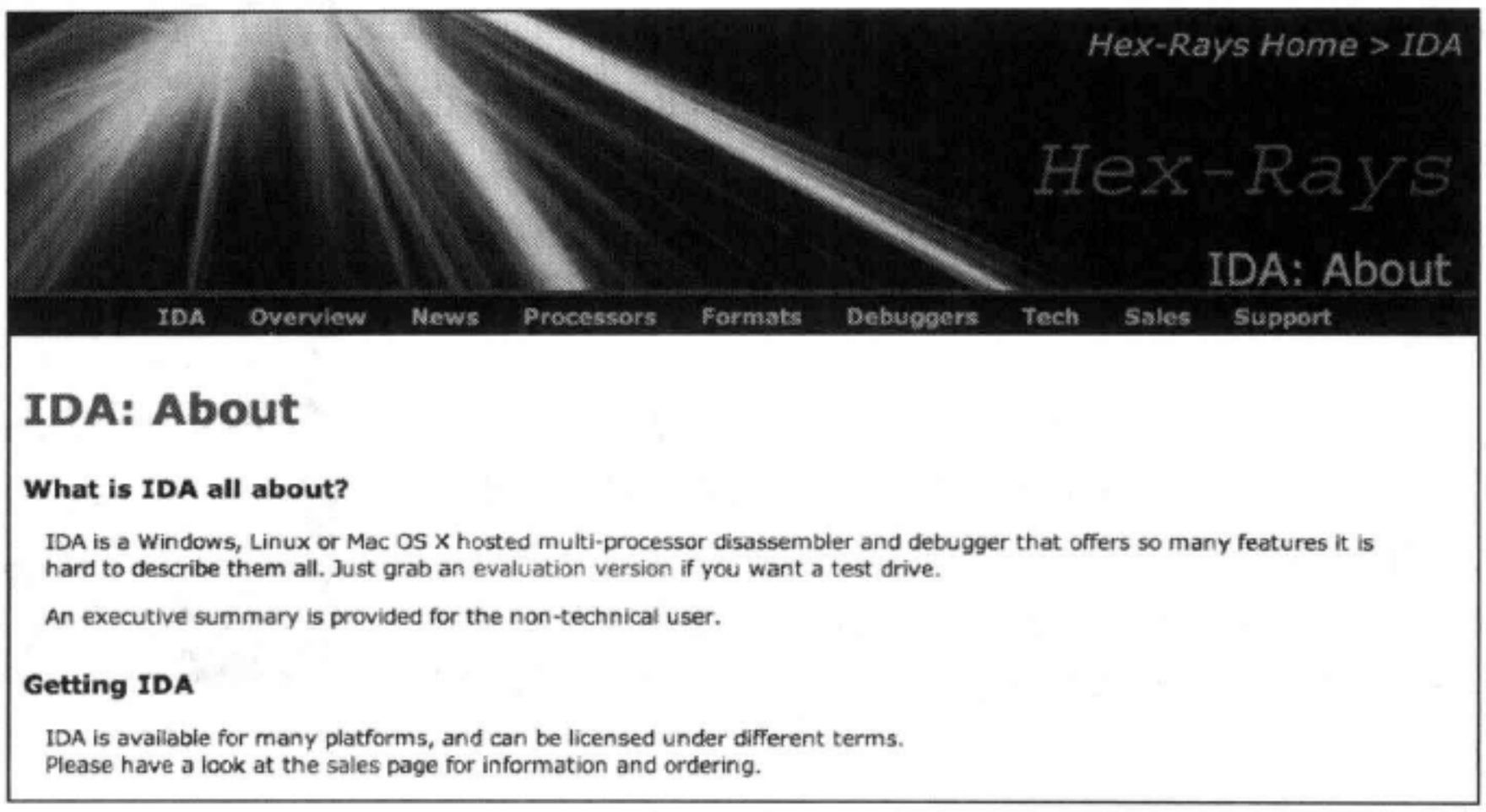


图 3-54 IDA 的官网页面

笼统地说，IDA 是一个支持 Windows、Linux 和 Mac OS X 的多平台反汇编器 / 调试器，它的功能非常强大，以至于连官方都不能给出一个详尽的功能列表。

IDA 的正式版是收费的，但其作者也是程序员出身，深知我们生活不易，所以慷慨地提供了一个免费的试用版，对于逆向工程初学者来说，已完全够用了。IDA 的下载和安装十分方便，具体可参考：https://www.hex-rays.com/products/ida/index.shtml，本书不再赘述。

3.4.2 IDA 使用说明

IDA 启动时会短暂地显示如图 3-55 所示的窗口。

这时可以单击 OK 按钮，或等上几秒，它会自动关闭，之后就会看到 IDA 的主界面，如图 3-56 所示。

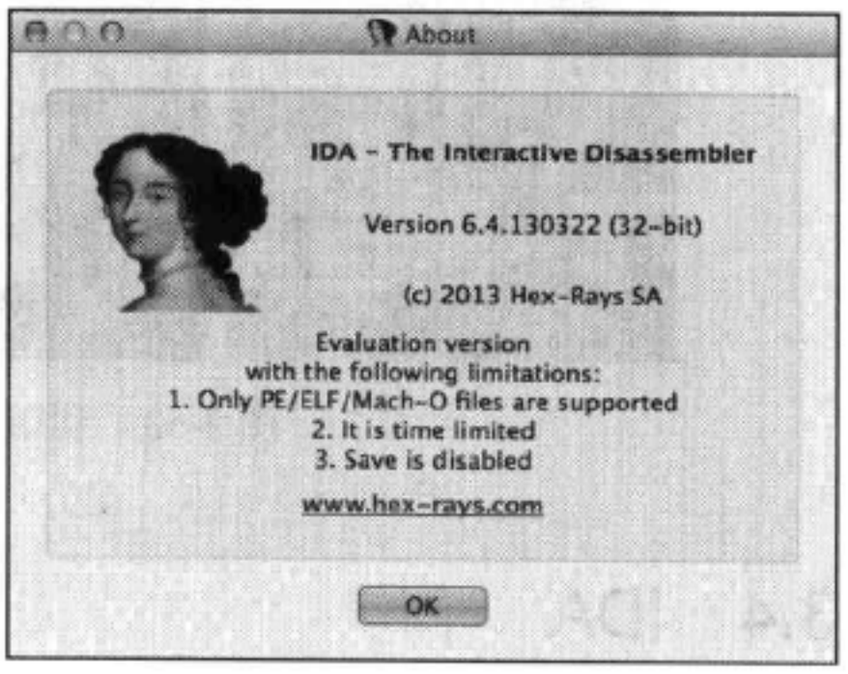


图 3-55 IDA 的 About 窗口

在该界面中，不用烦琐地在菜单中点击“打开文件”，然后一个目录一个目录地去翻找，只需把要分析的文件拖进 IDA 的灰色区域就行了。

打开文件后，还需要我们进行一些基本的配置，如图 3-57 所示。

1）因为我们要逆向的是 iOS 上的文件，而运行 iOS 的设备均采用 ARM 架构的处理器，所以在“ Process type”一栏，我们选择“ ARM

processors: ARM”。

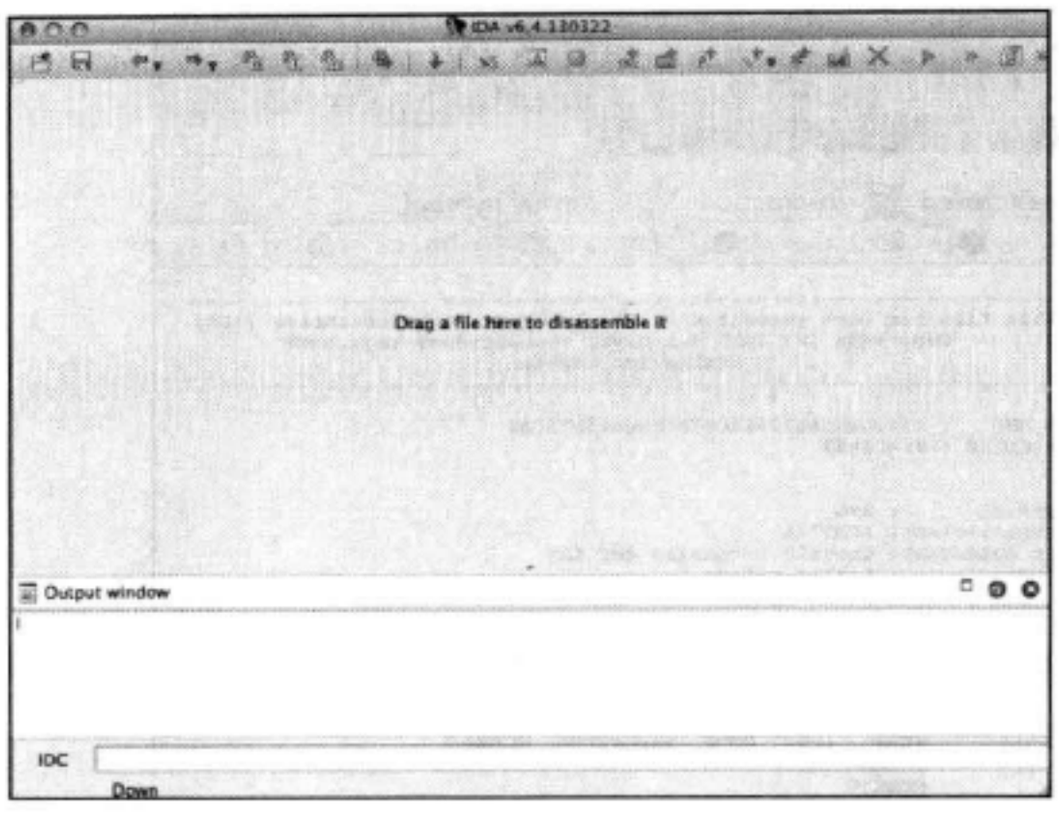


图 3-56 IDA 的主界面

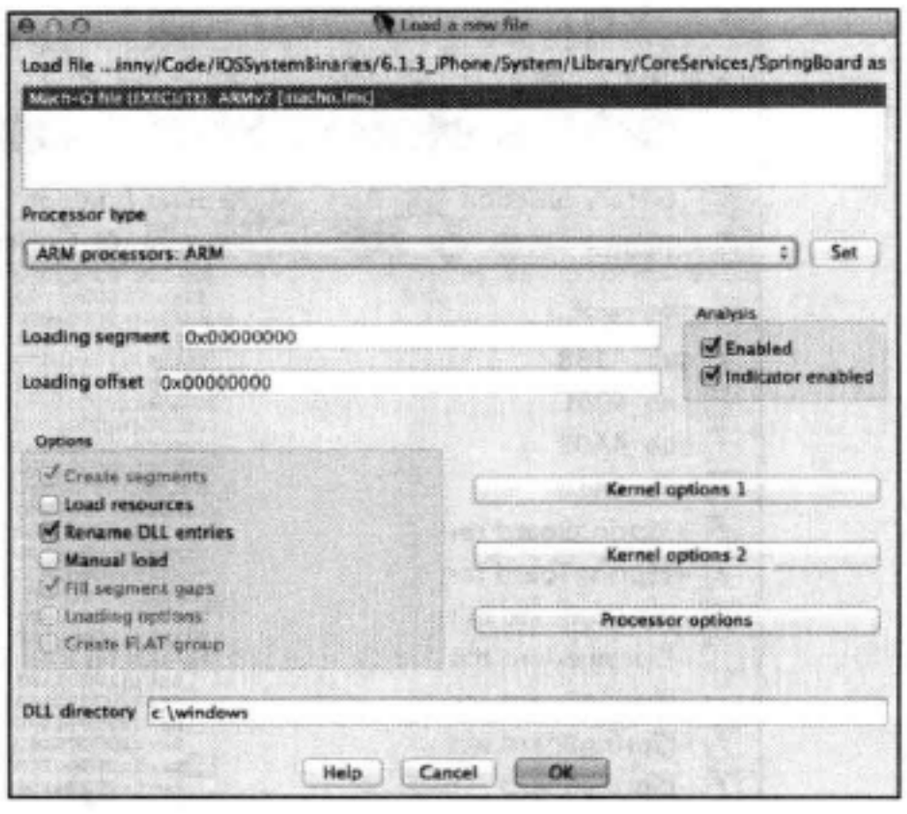


图 3-57 IDA 打开文件的参数配置

2）对于一些 fat binary（为了兼容不同架构的处理器，而把多种指令集糅合进一个 binary）来说，最上面的白框内还可能出现多个 Mach-O 文件供我们选择。一般来说，选择自己设备上 CPU 对应的那一个版本就可以了（如 iPhone 4 是 ARMv7，iPhone 5 是 ARMv7s）。接下来点击 OK 按钮，此时，会连续弹出好几个窗口，一直点击 YES 和 OK 按钮就可以了，如图 3-58 ～图 3-60 所示）。

3）因为试用版的 IDA 无法保存，所以即使在这些窗口上勾选“Don’t display this message again”，下次打开 IDA 还是会出现这些窗口。虽然有些麻烦，但这么强大的工具都让我们免费使用了，麻烦就麻烦点吧。

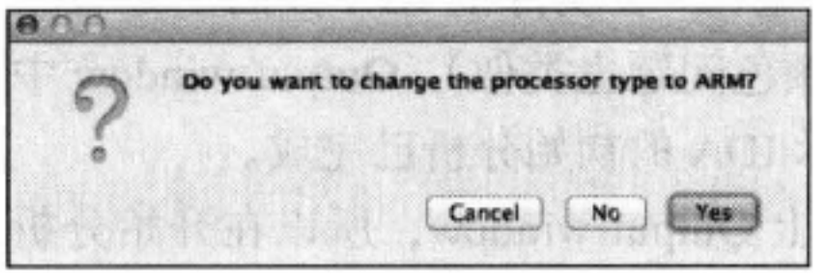


图 3-58 IDA 加载选项 1

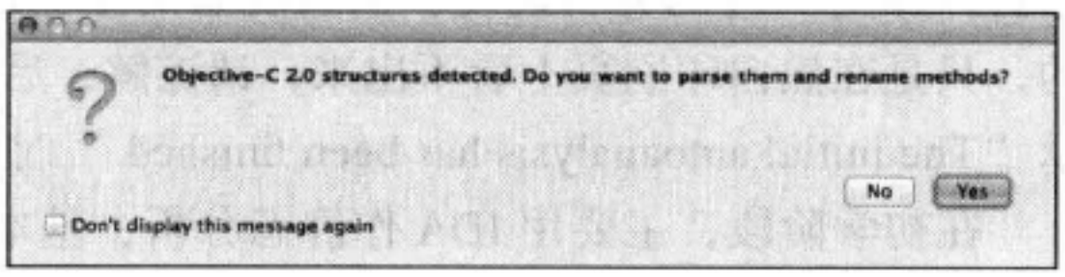


图 3-59 IDA 加载选项 2

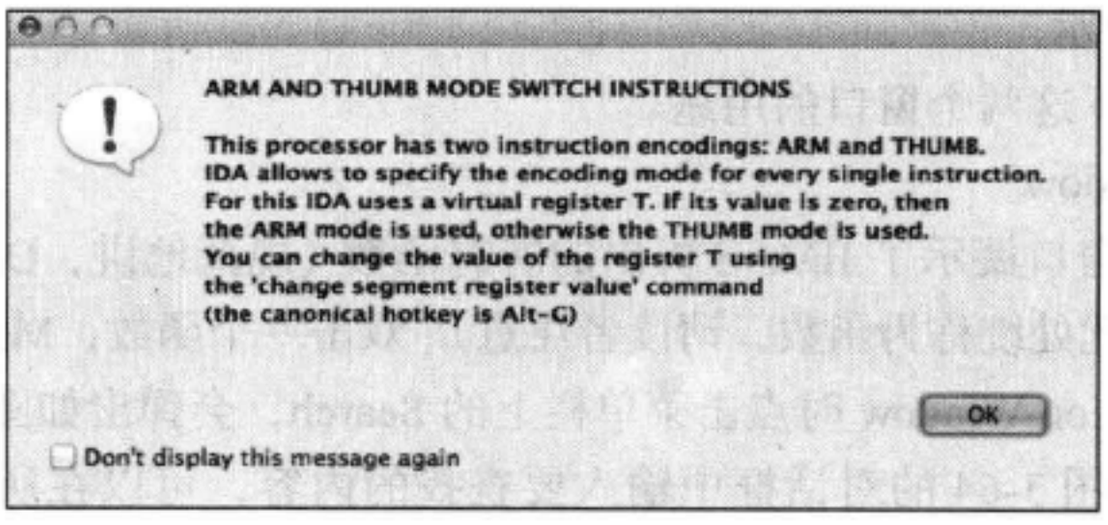


图 3-60 IDA 加载选项 3

连续操作后，内容丰富的主界面再次进入我们眼帘，如图 3-61 所示。

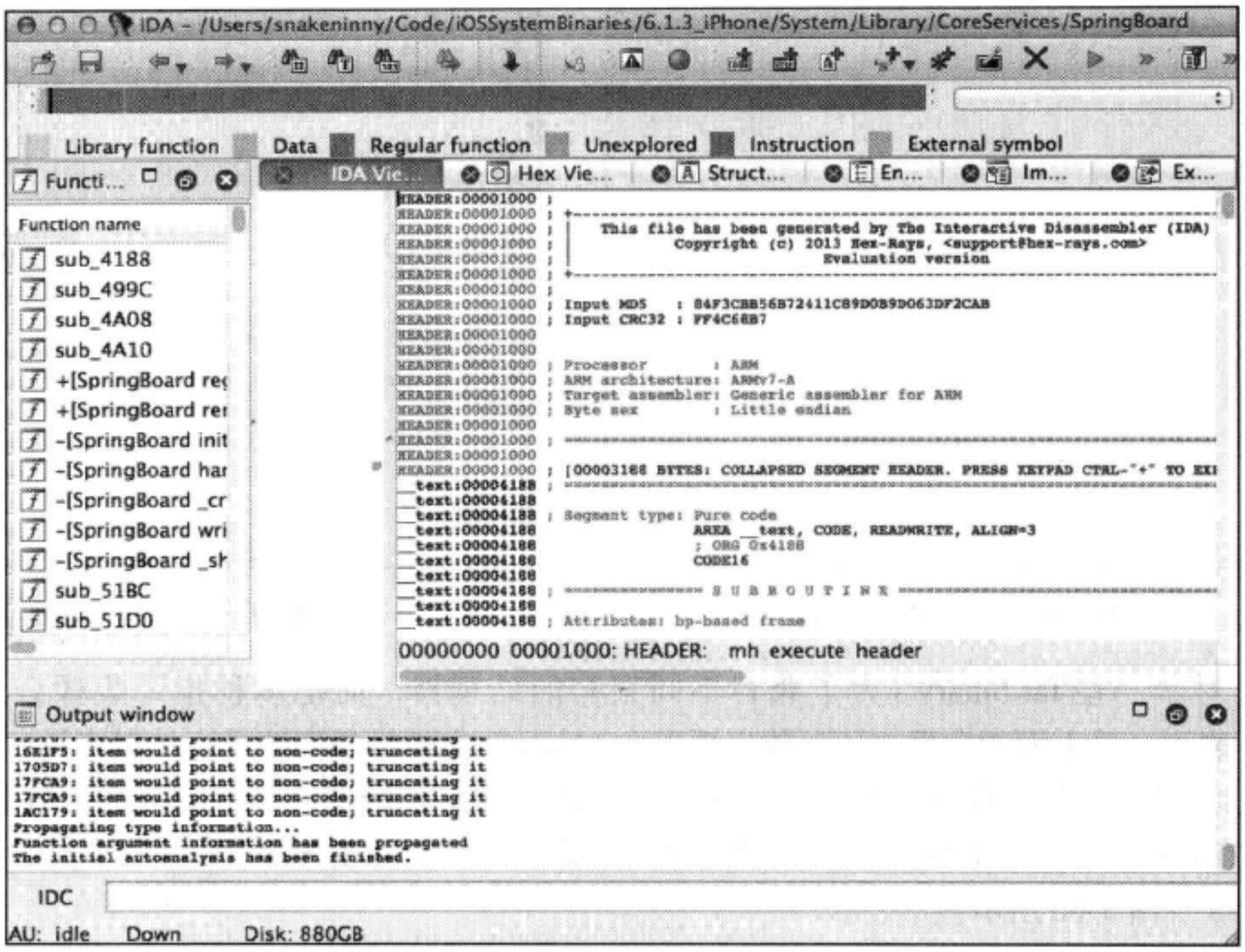


图 3-61 IDA 打开文件页面 1

在刚进入图 3-61 所示的界面时，上方的进度条还在不断滚动，下方的 Output window 也会打印出对文件的分析进度。当进度条的主色调变成蓝色（在实际的 IDA 界面上是有颜色的，只是在黑白印刷页上看不出来，请谅解，后续的颜色问题也类似），Output window 中显示“The initial autoanalysis has been finished.”时，表示 IDA 的初始分析已完成。

在初学阶段，主要用 IDA 作静态分析，基本用不上 Output window，所以在开始分析之前，可以先关掉 Output window。

现在我们看到的两个大窗口分别是左侧小部分的 Function window 和右侧大部分的 Main window，如图 3-62 所示。

下面分别介绍一下这两个窗口的用途。

（1）Function window

顾名思义，这个窗口展示了 IDA 分析出的所有函数（规范地讲，ObjC 的 function 应该称为 method，即方法，此处统称为函数，请读者注意），双击一个函数，Main window 会显示它的函数体。在选中 Function Window 时点击菜单栏上的 Search，会弹出如图 3-63 所示的子菜单。

选择 Search 并在图 3-64 的对话框中输入要查找的内容，可以在所有函数名中查找指定的字符串，十分方便。当要查找的内容出现在多个函数名里时，还可以点击 Search again 来遍历这些函数名。当然，上面的操作也都可以用快捷键完成。

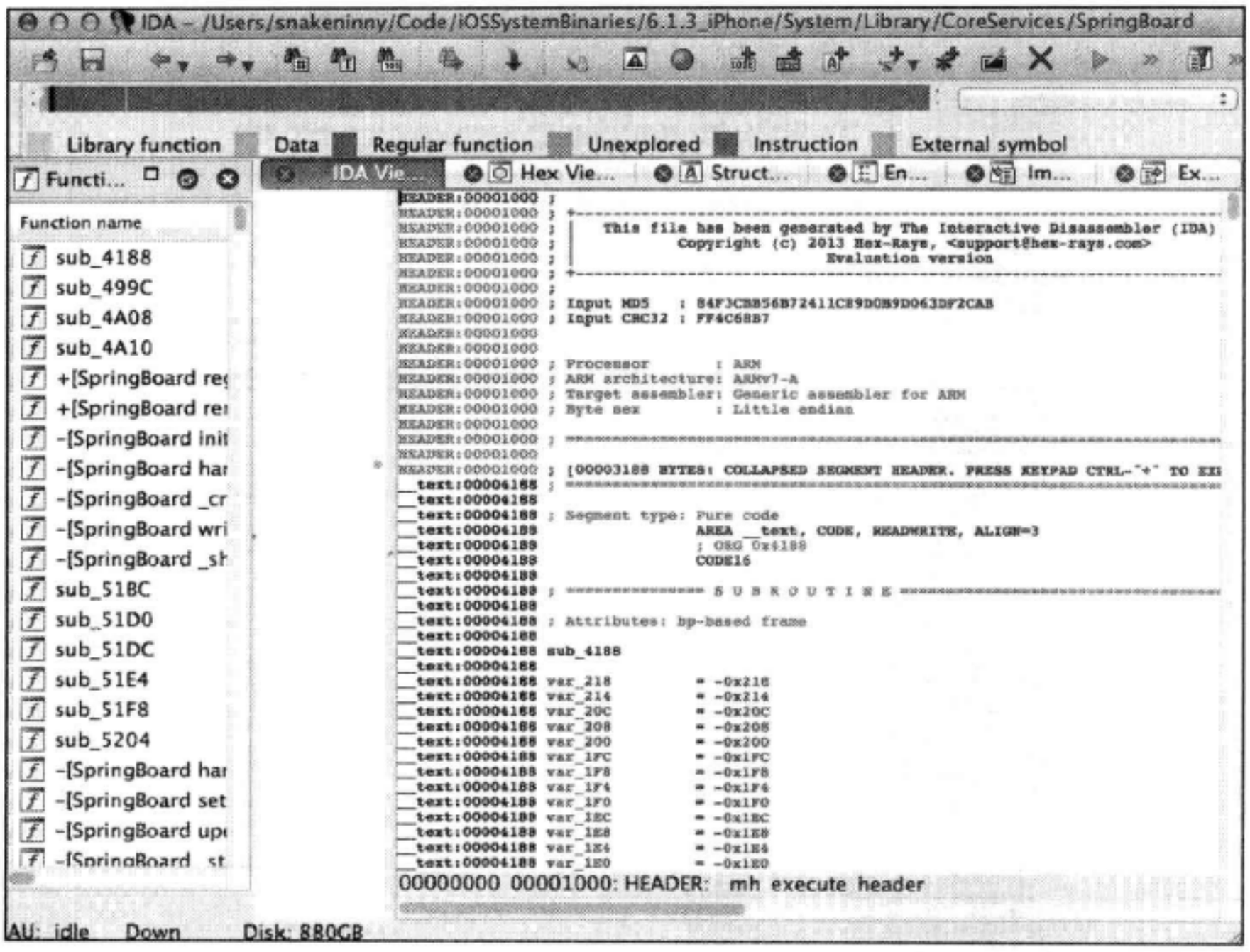


图 3-62 IDA 打开文件页面 2

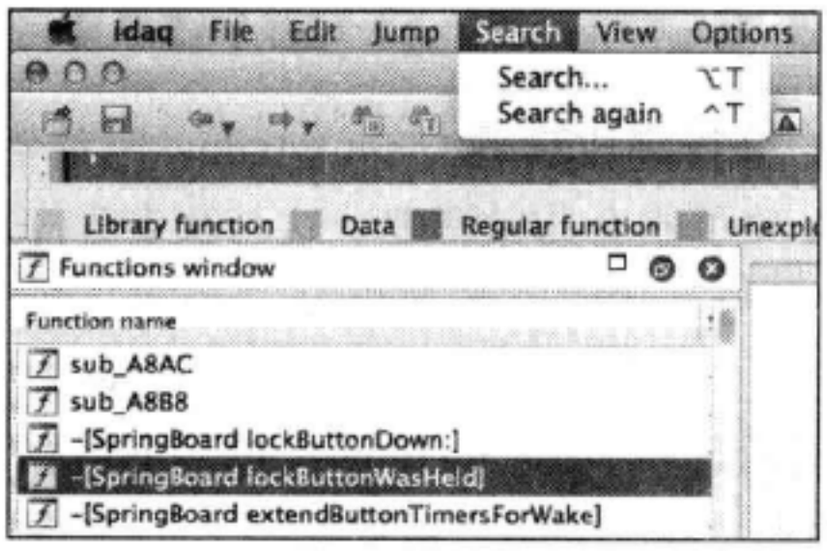


图 3-63 IDA search 1

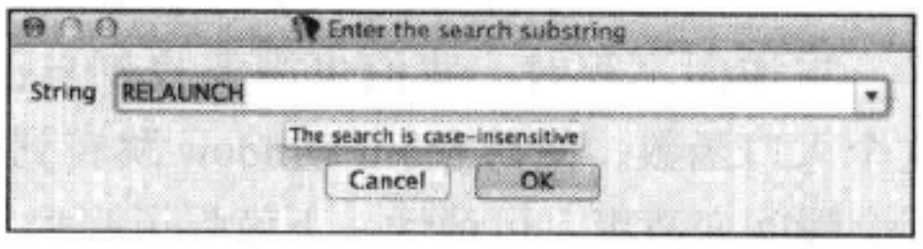


图 3-64 IDA search 2

Function window 中的 ObjC 函数跟 class-dump 导出的内容吻合。除了 ObjC 函数外，IDA 还将所有的 subroutine 罗列了出来，这是 class-dump 所做不到的。class-dump 导出的内容都是 ObjC 函数名，可读性高，容易上手，是 iOS 逆向工程初学者的乐园；subroutine 的名称只是一个代号，没有明显含义，分析难度大，大多数初学者看到这里就打了退堂鼓。但是，iOS 的底层是用 C/C++ 实现的，class-dump 拿它没辙，只能使用 IDA 这样的工具。要想深层次挖掘 iOS 中最有趣的部分，掌握 IDA 的用法是我们的必经之路。

（2）Main window

初始分析完成后的 Main window 如图 3-65 所示。

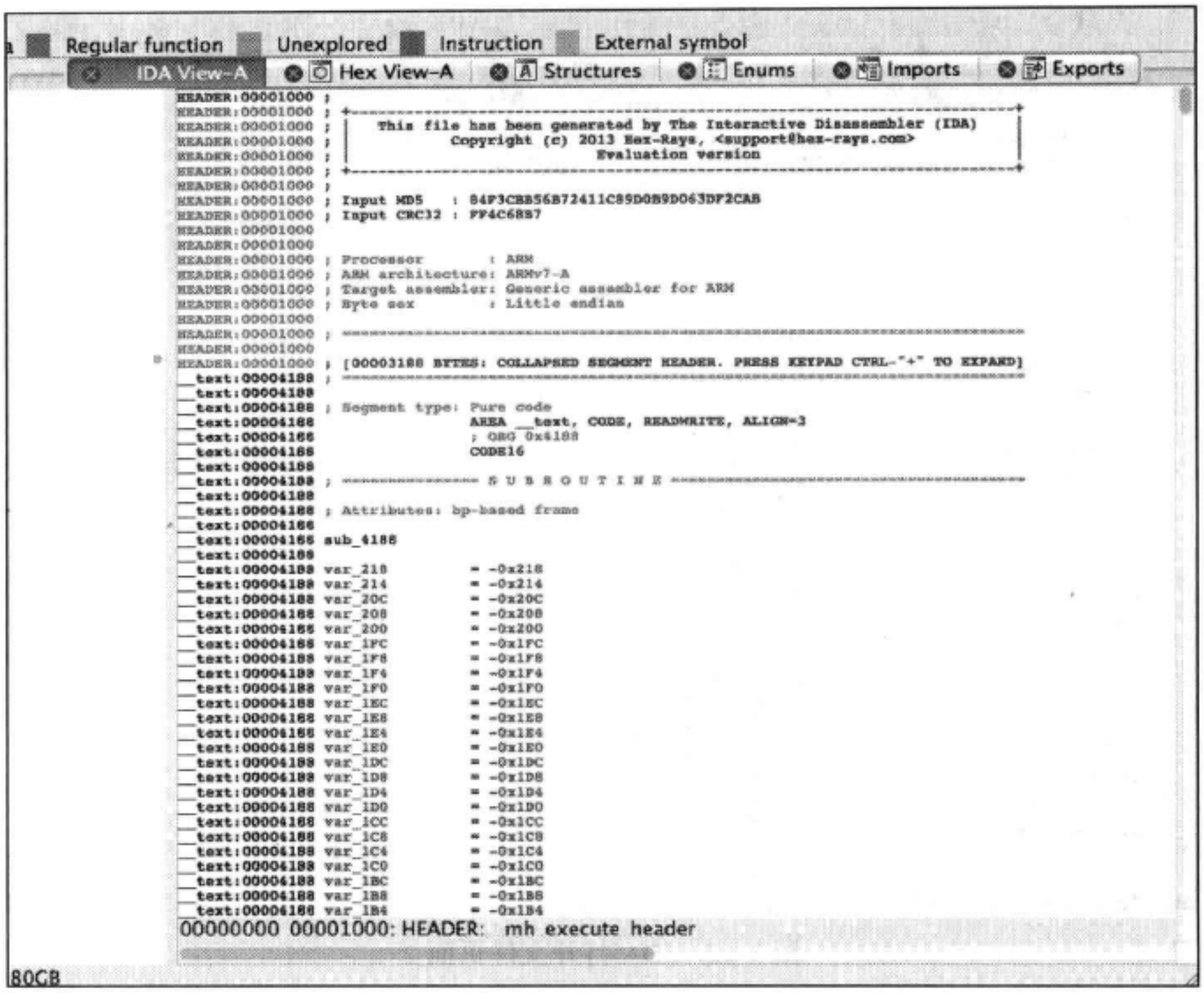


图 3-65 IDA Main Window

绝大多数程序员，包括笔者，在第一次看见初始分析完成后的 Main window 时都懵了，这都是哪儿跟哪儿啊？这是人类文字吗？还是把 IDA 关了刷会儿微博吧。这与很多程序员在写第一行代码时不知如何下手的感觉很类似。其实，与写代码时需要定义一个入口函数一样，在逆向工程时，我们也需要找到自己感兴趣的入口函数。在 Function window 中双击这个入口函数，使得 Main window 跳转到函数体，然后选中 Main window，按一下空格，界面会瞬间变清爽，可读性一下就强了起来，让人感觉到“我的天空，星星都亮了”（如图 3-66 所示）。

Main window 有两种显示模式，分别是 Graph view 和 Text view。Graph view 把被分析的文件逻辑用方块的形式表现出来，便于我们分析程序各个模块之间的关系。各个方块之间的执行顺序用带箭头的线表示，当执行出现分支时，满足判断条件分支的线是绿色的，否则是红色的；当执行没有分支时，线是蓝色的，比如在图 3-67 中，在最上面的方块执行完后，会判断 R0 和 1 的关系，如果 R0 – 1 != 0，则满足判断条件（BNE，即 Branch if Not Equal to zero），走绿色的那条路接着执行最下面的方块；否则走红色的路执行中间的方块，而中间的方块执行完后顺着蓝色的路执行最下面的方块。此处是 IDA 的难点之一，后面的章节中会再次讲解。

```
; SpringBoard - (void)reboot
; Attributes: bp-based frame

; void __cdecl -[SpringBoard reboot](struct SpringBoard *self, SEL)
__SpringBoard_reboot_

var_C= -0xC

PUSH            {R4,R5,R7,LR}
ADD             R7, SP, #8
STR.W           R8, [SP,#8+var_C]!
SUB             SP, SP, #8
MOV             R0, #(_UIApp_ptr - 0xAC3A) ; _UIApp_ptr
MOVW            R1, #(:lower16:(selRef_beginIgnoringInteractionEvents - 0xAC42))
ADD             R0, PC ; _UIApp_ptr
MOVT.W          R1, #(:upper16:(selRef_beginIgnoringInteractionEvents - 0xAC42))
LDR             R0, [R0] ; _UIApp
ADD             R1, PC ; selRef_beginIgnoringInteractionEvents
LDR             R1, [R1] ; "beginIgnoringInteractionEvents"
LDR             R0, [R0]
BLX             _objc_msgSend
MOV             R0, #(selRef_sharedApplication - 0xAC5C) ; selRef_sharedApplication
MOV             R5, #(classRef_UIApplication - 0xAC5E) ; classRef_UIApplication
ADD             R0, PC ; selRef_sharedApplication
ADD             R5, PC ; classRef_UIApplication
LDR             R4, [R0] ; "sharedApplication"
LDR             R0, [R5] ; _OBJC_CLASS_$_UIApplication
MOV             R1, R4
BLX             _objc_msgSend
MOV             R1, #(selRef_hideSpringBoardStatusBar - 0xAC72) ; selRef_hideSpringBoardStatusBar
ADD             R1, PC ; selRef_hideSpringBoardStatusBar
LDR             R1, [R1] ; "hideSpringBoardStatusBar"
BLX             _objc_msgSend
BL              sub_236FC
MOVS            R0, #0
MOV.W           R8, #0
BL              sub_2430C
MOVS            R0, #1
BL              sub_24350
LDR             R0, [R5] ; _OBJC_CLASS_$_UIApplication
MOV             R1, R4
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_performSelector_withObject_afterDelay_ - 0xACAA))
MOVW            R9, #0
MOVT.W          R1, #(:upper16:(selRef_performSelector_withObject_afterDelay_ - 0xACAA))
MOV             R2, #(selRef__rebootNow - 0xACAC) ; selRef__rebootNow
ADD             R1, PC ; selRef_performSelector_withObject_afterDelay_
ADD             R2, PC ; selRef__rebootNow
LDR             R1, [R1] ; "performSelector:withObject:afterDelay:"
MOVT.W          R9, #0x4010
LDR             R2, [R2] ; "_rebootNow"
MOVS            R3, #0
STRD.W          R8, R9, [SP]
BLX             _objc_msgSend
ADD             SP, SP, #8
LDR.W           R8, [SP+0xC+var_C],#4
POP             {R4,R5,R7,PC}
; End of function -[SpringBoard reboot]
```

图 3-66 IDA 函数展开

```
; SpringBoard - (void)lockButtonUp:(struct __GSEvent *)
; Attributes: bp-based frame

; void __cdecl -[SpringBoard lockButtonUp:](struct SpringBoard *self, SEL, struct __GSEvent *)
__SpringBoard_lockButtonUp__

var_1C= -0x1C

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
PUSH.W          {R8,R10,R11}
STR.W           R2, [SP,#0x18+var_1C]!
MOV             R4, R0
MOV             R0, #(off_211DF0 - 0xAF1A) ; off_211DF0
ADD             R0, PC ; off_211DF0
LDR.W           R10, [R0] ; byte_26CBF0
LDRB.W          R0, [R10]
CMP             R0, #1
BNE             loc_AF3C
```

```
MOV             R0, #(cfstr_SLockbuttonup - 0xAF38) ; "%s lockButtonUp"
MOV             R1, #(aSpringboardL_0 - 0xAF3A) ; "-[SpringBoard lockButtonUp:]"
ADD             R0, PC  ; "%s lockButtonUp"
ADD             R1, PC  ; "-[SpringBoard lockButtonUp:]"
BL              sub_1A6BB4
```

```
loc_AF3C
MOV             R0, #(byte_26CBC0 - 0xAF48) ; byte_26CBC0
ADD             R0, PC ; byte_26CBC0
LDRB            R0, [R0]
CMP             R0, #0
BNE             loc_B030
```

图 3-67 IDA 函数跳转

细心的读者也一定注意到了，IDA 中的字体色彩缤纷，不同颜色的字体表示的含义也各不相同，如图 3-68 所示。

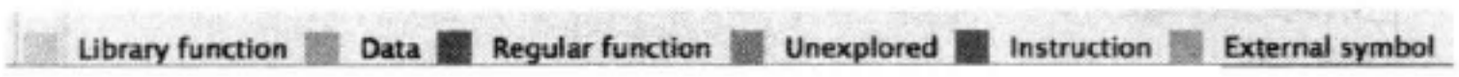


图 3-68 IDA 中的彩色工具栏

当你选中一个符号时，相同的符号都会用黄色高亮显示，方便我们跟踪这个符号的轨迹，如图 3-69 所示。

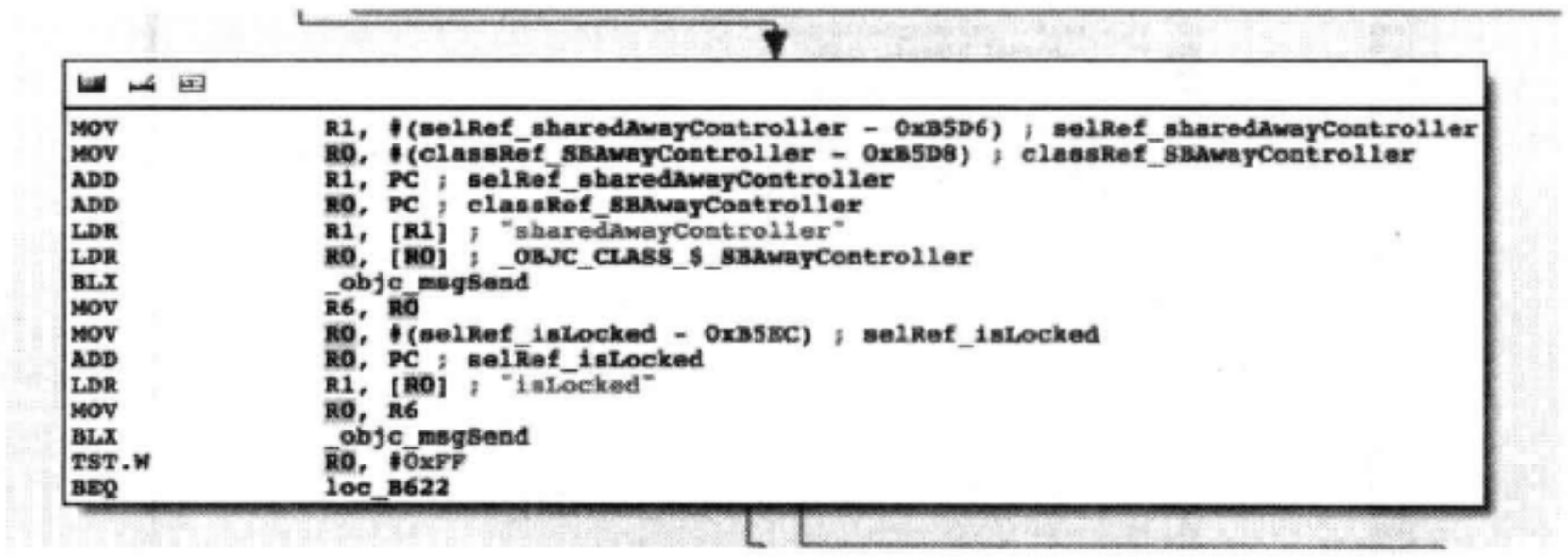


图 3-69 IDA 函数展示

在任意符号上点击鼠标右键，会弹出如图 3-70 所示的菜单。

其中，常用的功能有“Jump to xref to operand...”，点击后出现的窗口罗列了这个文件中所有显式引用这个符号的信息，如图 3-71 所示。

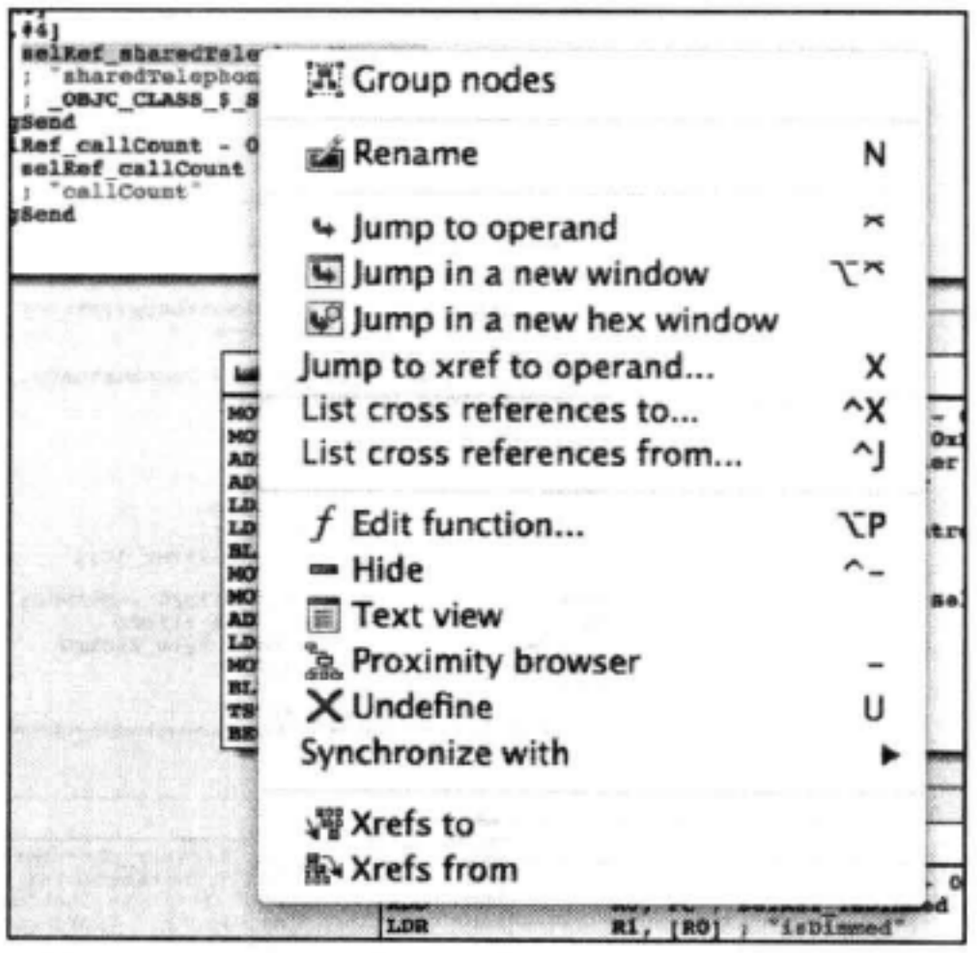


图 3-70 在符号上右击弹出的菜单

如果你觉得这种表示方法不够直观，更喜欢 Main window 中的 Graph view 形式，可以选择菜单下面的“Xrefs to”，但如果运气不好，这个符号被引用过多，就会看到类似图 3-72 这样的一团乱麻的界面，真是剪不断，理还乱。

在图 3-72 中，SBAssistantController 这个类中的 uiPluginWantsActivation:forEvent:revealMode:completion: 函数显式使用了 isLocked 这个符号，图中间黑色的部分都是由一根根直线构成的，两侧基本已经黑成了一片，可以看出这个符号被大量引用，说明该符号在文件中应该起到了很关键的作用。

相对地，Xrefs from 则会显示所有这个符号显式引用的符号，如图 3-73 所示。

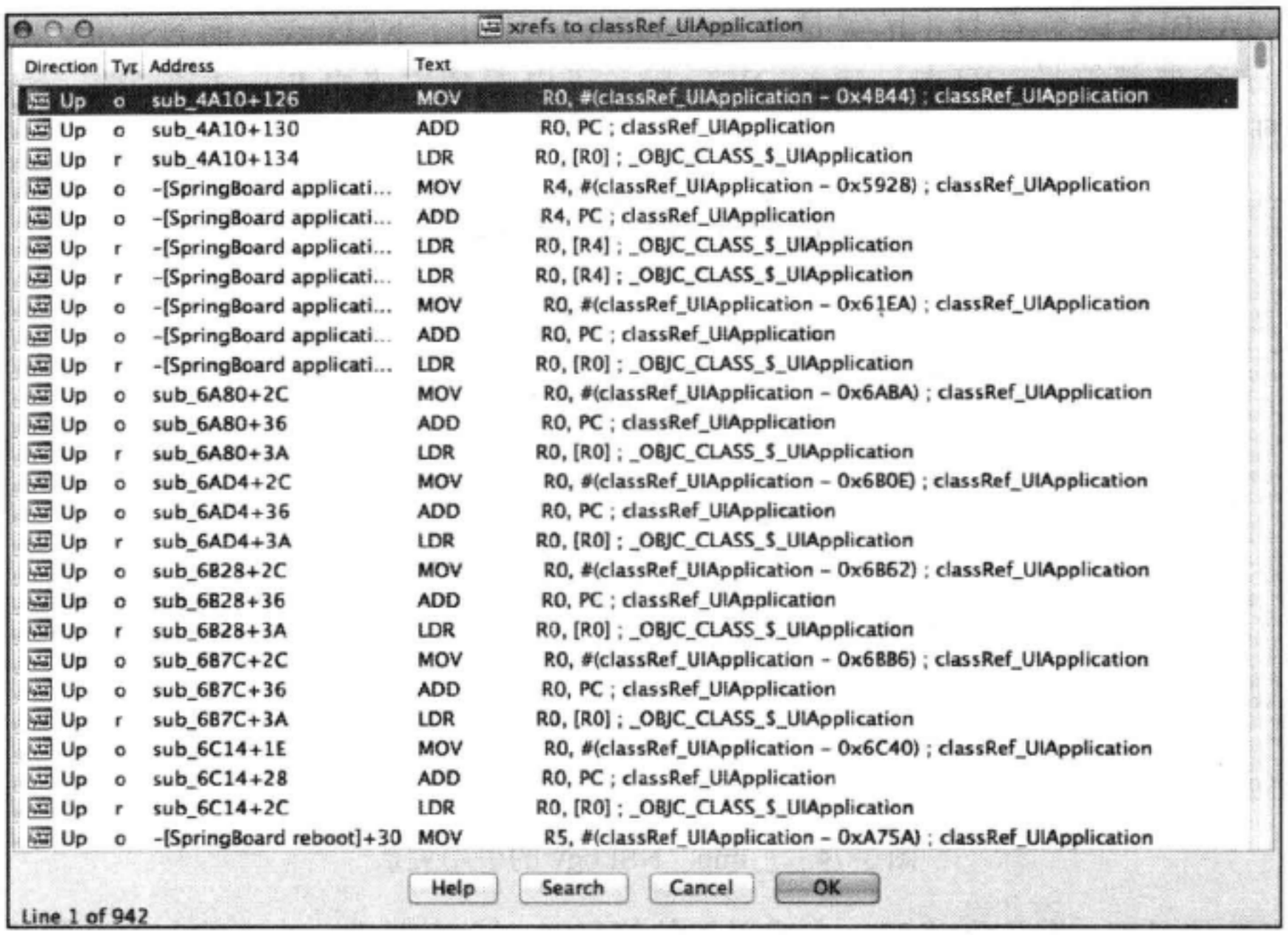


图 3-71 符号表信息

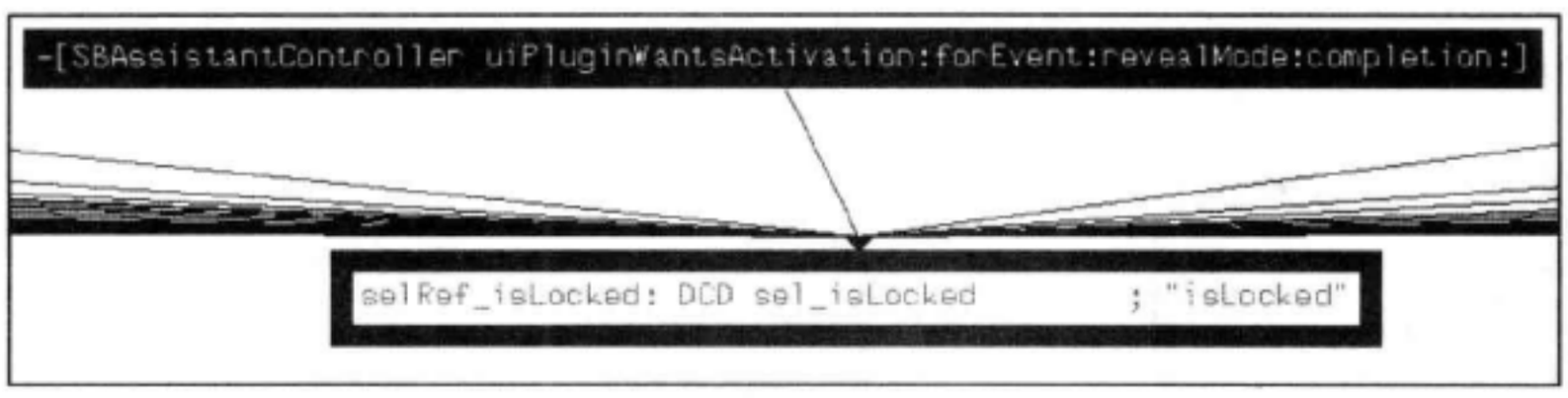


图 3-72 符号引用的混乱界面



图 3-73 符号的显示引用关系

sub_1A6BB4 这个符号（其实是个 subroutine）引用了 NSLogv，而 NSLogv 又引用了 NSLogv 这个外部符号。双击 __imp__NSLogv，可以看到它来自 Foundation.framework，如图 3-74 所示。

```
UNDEF:002A1318 ;
UNDEF:002A1318 ; Imports from /System/Library/Frameworks/Foundation.framework/Foundation
UNDEF:002A1318 ;
UNDEF:002A1318                 IMPORT __imp__NSClassFromString
UNDEF:002A1318                                         ; CODE XREF: _NSClassFromStringij
UNDEF:002A1318                                         ; DATA XREF: _NSClassFromStringio ...
UNDEF:002A131C                 IMPORT _NSCocoaErrorDomain ; DATA XREF: sub_D2778+14Aio
UNDEF:002A131C                                         ; -[SBIconModelPropertyListFileStore _load:error:]+96io ...
UNDEF:002A1320                 IMPORT __imp__NSHomeDirectory ; CODE XREF: _NSHomeDirectoryij
UNDEF:002A1320                                         ; DATA XREF: _NSHomeDirectoryio ...
UNDEF:002A1324                 IMPORT _NSLocalizedDescriptionKey
UNDEF:002A1324                                         ; DATA XREF: -[SBPasscodeController _passwordEntered:]+12Aio
UNDEF:002A1324                                         ; __nl_symbol_ptr:_NSLocalizedDescriptionKey_ptrio
UNDEF:002A1328 IMPORT __imp__NSLog     ; CODE XREF: _NSLogij
UNDEF:002A1328                         ; DATA XREF: _NSLogio ...
UNDEF:002A132C IMPORT __imp__NSLogv    ; CODE XREF: _NSLogvij
UNDEF:002A132C                         ; DATA XREF: _NSLogvio ...
UNDEF:002A1330                 IMPORT __imp__NSSearchPathForDirectoriesInDomains
UNDEF:002A1330                                         ; CODE XREF: _NSSearchPathForDirectoriesInDomainsij
UNDEF:002A1330                                         ; DATA XREF: _NSSearchPathForDirectoriesInDomainsio ...
UNDEF:002A1334                 IMPORT __imp__NSStringFromBOOL
UNDEF:002A1334                                         ; CODE XREF: _NSStringFromBOOLij
UNDEF:002A1334                                         ; DATA XREF: _NSStringFromBOOLio ...
UNDEF:002A1338                 IMPORT __imp__NSStringFromClass
UNDEF:002A1338                                         ; CODE XREF: _NSStringFromClassij
UNDEF:002A1338                                         ; DATA XREF: _NSStringFromClassio ...
UNDEF:002A133C                 IMPORT __imp__NSStringFromRange
UNDEF:002A133C                                         ; CODE XREF: _NSStringFromRangeij
UNDEF:002A133C                                         ; DATA XREF: _NSStringFromRangeio ...
```

图 3-74 __imp__NSLogv 的引用关系

在多数情况下，我们在找到一个感兴趣的符号时，会想要进一步查找与这个符号相关的所有线索。一种笨拙但有效的方法是在选中 Main Window 时点击菜单栏上的 Search，此时会弹出如图 3-75 所示的子菜单。

然后选择“text...”，会弹出如图 3-76 所示的窗口。

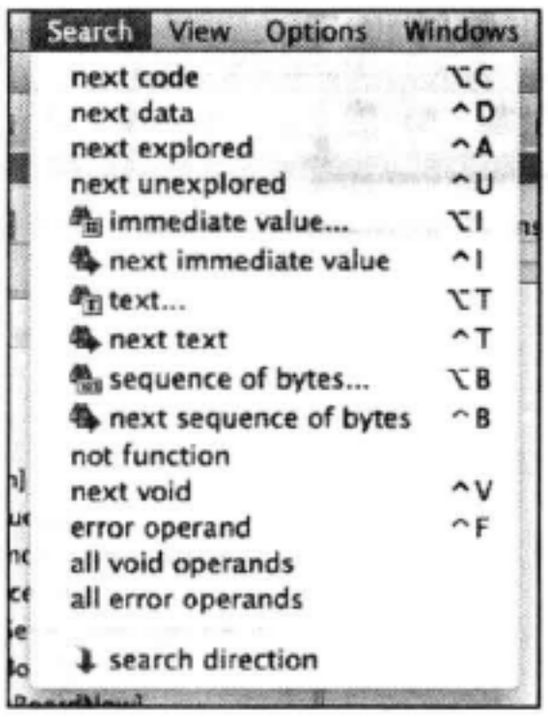


图 3-75 IDA 的 Search 菜单

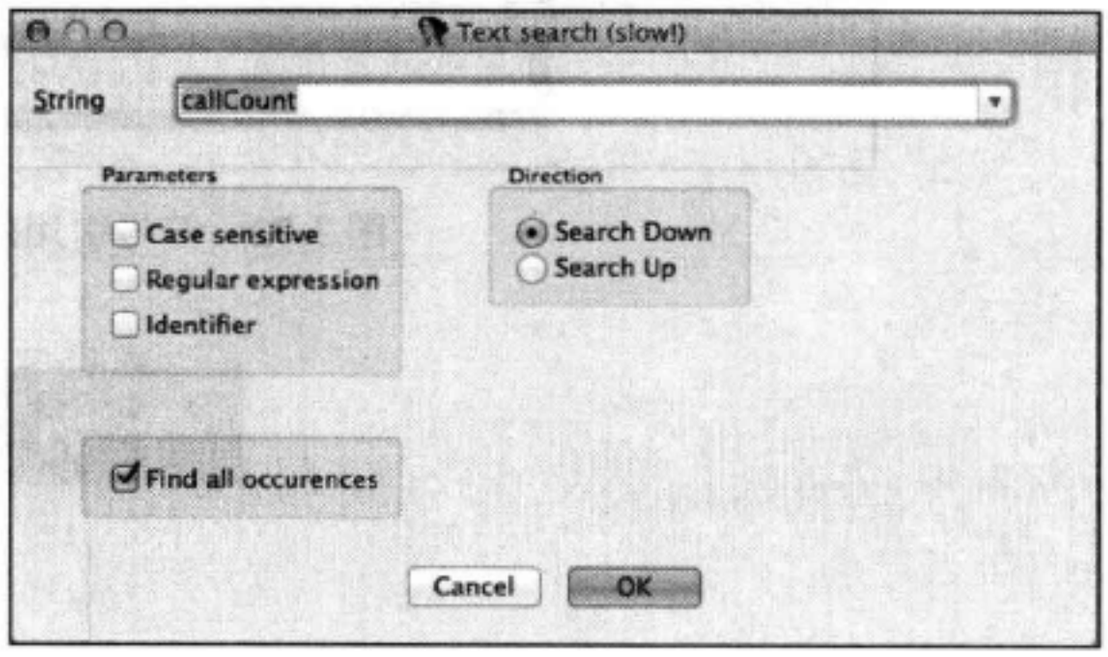


图 3-76 输入关键字

可以根据自己的情况选择搜索是否对大小写敏感，搜索格式是不是正则表达式等，然后勾选“Find all occurences”，点击 OK 按钮，IDA 会将文件中所有满足搜索条件的符号列出，供我们一一查看。

Graph view 提供的功能非常多，以上只是简单介绍了几个常用的功能，熟练地使用它们

是我们从事更深入研究的保障。Graph view 的界面比较简洁，各代码块间逻辑清晰，适合肉眼观看。相对来说，现阶段切换到 Text view 的机会比较少，一般是在配合 GDB 进行动态调试时，才会在 Text view 中对界面左侧罗列的符号地址特别关注，如图 3-77 所示。

```
__text:0000BC3C
__text:0000BC3C                 PUSH            {R4-R7,LR}
__text:0000BC3E                 ADD             R7, SP, #0xC
__text:0000BC40                 PUSH.W          {R8,R10,R11}
__text:0000BC44                 SUB             SP, SP, #0xC4
__text:0000BC46                 STR             R0, [SP,#0xDC+var_D8]
__text:0000BC48                 MOV             R0, #(___stack_chk_guard_ptr - 0xBC54) ; ___stack_chk_guard_ptr
__text:0000BC50                 ADD             R0, PC ; ___stack_chk_guard_ptr
__text:0000BC52                 LDR             R0, [R0] ; ___stack_chk_guard
__text:0000BC54                 STR             R0, [SP,#0xDC+var_D4]
__text:0000BC56                 LDR             R2, [R0]
__text:0000BC58                 MOV             R1, #(selRef_sharedInstance - 0xBC6C) ; selRef_sharedInstance
__text:0000BC60                 MOV             R0, #(classRef_SBPlatformController - 0xBC6E) ; classRef_SBPlatformController
__text:0000BC68                 ADD             R1, PC ; selRef_sharedInstance
__text:0000BC6A                 ADD             R0, PC ; classRef_SBPlatformController
__text:0000BC6C                 STR             R2, [SP,#0xDC+var_1C]
__text:0000BC6E                 LDR             R1, [R1] ; "sharedInstance"
__text:0000BC70                 LDR             R0, [R0] ; _OBJC_CLASS_$_SBPlatformController
__text:0000BC72                 STR             R1, [SP,#0xDC+var_C4]
__text:0000BC74                 BLX             _objc_msgSend
__text:0000BC78                 STR             R0, [SP,#0xDC+var_C0]
__text:0000BC7A                 MOV             R1, #(selRef_sharedConnection - 0xBC8E) ; selRef_sharedConnection
__text:0000BC82                 MOV             R0, #(classRef_MCProfileConnection - 0xBC90) ; classRef_MCProfileConnection
__text:0000BC8A                 ADD             R1, PC ; selRef_sharedConnection
__text:0000BC8C                 ADD             R0, PC ; classRef_MCProfileConnection
__text:0000BC8E                 LDR             R1, [R1] ; "sharedConnection"
__text:0000BC90                 LDR             R0, [R0] ; _OBJC_CLASS_$_MCProfileConnection
__text:0000BC92                 BLX             _objc_msgSend
__text:0000BC96                 MOV             R8, R0
__text:0000BC98                 MOV             R0, #(selRef_array - 0xBCAC) ; selRef_array
__text:0000BCA0                 MOV             R5, #(classRef_NSMutableArray - 0xBCAE) ; classRef_NSMutableArray
__text:0000BCA8                 ADD             R0, PC ; selRef_array
__text:0000BCAA                 ADD             R5, PC ; classRef_NSMutableArray
__text:0000BCAC                 LDR             R4, [R0] ; "array"
__text:0000BCAE                 LDR             R0, [R5] ; _OBJC_CLASS_$_NSMutableArray
__text:0000BCB0                 MOV             R1, R4
__text:0000BCB2                 BLX             _objc_msgSend
__text:0000BCB6                 MOVW            R1, #(:lower16:(selRef_dictionary - 0xBCCC))
__text:0000BCBA                 MOV             R11, R0
__text:0000BCBC                 MOVT.W          R1, #(:upper16:(selRef_dictionary - 0xBCCC))
__text:0000BCC0                 MOV             R0, #(classRef_NSMutableDictionary - 0xBCCE) ; classRef_NSMutableDictionary
__text:0000BCC8                 ADD             R1, PC ; selRef_dictionary
__text:0000BCCA                 ADD             R0, PC ; classRef_NSMutableDictionary
__text:0000BCCC                 LDR             R1, [R1] ; "dictionary"
__text:0000BCCE                 LDR             R0, [R0] ; _OBJC_CLASS_$_NSMutableDictionary
__text:0000BCD0                 BLX             _objc_msgSend
__text:0000BCD4                 MOV             R10, R0
__text:0000BCD6                 LDR             R0, [R5] ; _OBJC_CLASS_$_NSMutableArray
__text:0000BCD8                 MOV             R1, R4
__text:0000BCDA                 BLX             _objc_msgSend
```

图 3-77 Text view 列表

IDA 中有的 bug 会导致 Graph view 的末端显示不全，当你对某一个 Graph view 块产生明显怀疑时，就切换到 Text view 看看 Graph view 是不是漏显示了某些代码。

3.4.3 IDA 分析示例

说了 IDA 的这么多使用方法，下面以一个简单的示例来向大家演示 IDA 的威力。越狱 iOS 的用户都知道，在 Cydia 中安装完一个 tweak 后，Cydia 会建议我们 Restart SpringBoard，那么这个 respring 的操作是如何实现的呢？可在 IDA 中打开 SDK 中附带的 SpringBoard 可执行文件（路径是 /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOSX.X.sdk/System/Library/CoreServices/SpringBoard.app/SpringBoard），然后在 Function window 中搜索 relaunchSpringBoard，定位到这个函数，双击跳转到它的实现代码中，如图 3-78 所示。

可以看到，这个函数的实现流程既简单又清晰。根据从上到下的执行顺序，首先调用 beginIgnoringInteractionEvents，在 respring 之前不再处理任何用户交互；然后调用 sharedApplication，取得当前程序的单例对象；进而调用 hideSpringBoardStatusBar 来隐藏状态栏；接着连续执行两个 subroutine，分别是 sub_242C8 和 sub_236FC。首先我们双击

sub_242C8，看看它做了什么，如图 3-79 所示。

```
; SpringBoard - (void)relaunchSpringBoard
; Attributes: bp-based frame

; void __cdecl -[SpringBoard relaunchSpringBoard](struct SpringBoard *self, SEL)
__SpringBoard_relaunchSpringBoard_

var_10= -0x10
var_C= -0xC

PUSH            {R4,R5,R7,LR}
ADD             R7, SP, #8
SUB             SP, SP, #8
MOV             R0, #(_UIApp_ptr - 0xAE6E) ; _UIApp_ptr
MOVW            R1, #(:lower16:(selRef_beginIgnoringInteractionEvents - 0xAE76))
ADD             R0, PC ; _UIApp_ptr
MOVT.W          R1, #(:upper16:(selRef_beginIgnoringInteractionEvents - 0xAE76))
LDR             R0, [R0] ; _UIApp
ADD             R1, PC ; selRef_beginIgnoringInteractionEvents
LDR             R1, [R1] ; "beginIgnoringInteractionEvents"
LDR             R0, [R0]
BLX             _objc_msgSend
MOV             R0, #(selRef_sharedApplication - 0xAE90) ; selRef_sharedApplication
MOV             R5, #(classRef_UIApplication - 0xAE92) ; classRef_UIApplication
ADD             R0, PC ; selRef_sharedApplication
ADD             R5, PC ; classRef_UIApplication
LDR             R4, [R0] ; "sharedApplication"
LDR             R0, [R5] ; _OBJC_CLASS_$_UIApplication
MOV             R1, R4
BLX             _objc_msgSend
MOV             R1, #(selRef_hideSpringBoardStatusBar - 0xAEA6) ; selRef_hideSpringBoardStatusBar
ADD             R1, PC ; selRef_hideSpringBoardStatusBar
LDR             R1, [R1] ; "hideSpringBoardStatusBar"
BLX             _objc_msgSend
MOVS            R0, #1
BL              sub_242C8
BL              sub_236FC
LDR             R0, [R5] ; _OBJC_CLASS_$_UIApplication
MOV             R1, R4
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_performSelector_withObject_afterDelay_ - 0xAED2))
MOVS            R3, #0
MOVT.W          R1, #(:upper16:(selRef_performSelector_withObject_afterDelay_ - 0xAED2))
MOV             R2, #(selRef__relaunchSpringBoardNow - 0xAED4) ; selRef__relaunchSpringBoardNow
ADD             R1, PC ; selRef_performSelector_withObject_afterDelay_
ADD             R2, PC ; selRef__relaunchSpringBoardNow
LDR             R1, [R1] ; "performSelector:withObject:afterDelay:"
MOVS            R4, #0
LDR             R2, [R2] ; "_relaunchSpringBoardNow"
MOVT.W          R3, #0x4010
STR             R4, [SP,#0x10+var_10]
STR             R3, [SP,#0x10+var_C]
MOVS            R3, #0
BLX             _objc_msgSend
ADD             SP, SP, #8
POP             {R4,R5,R7,PC}
; End of function -[SpringBoard relaunchSpringBoard]
```

图 3-78 relaunchSpringBoard 函数的实现

```
; Attributes: bp-based frame

sub_242C8
PUSH            {R4,R7,LR}
MOV             R4, R0
MOV             R0, #(off_211F4C - 0x242DA) ; off_211F4C
ADD             R7, SP, #4
ADD             R0, PC ; off_211F4C
LDR             R0, [R0] ; dword_26CB8C
LDR             R0, [R0]
CBZ             R0, loc_242FE
```

```
MOV             R0, R4
BLX             _NSStringFromBOOL
MOV             R2, R0
MOV             R0, #(cfstr_S_1 - 0x242FA) ; "%s() %@"
MOV             R1, #(aSbworkspacere1 - 0x242FC) ; "SBWorkspaceRelaunchWhenFinishedTerminat"...
ADD             R0, PC  ; "%s() %@"
ADD             R1, PC  ; "SBWorkspaceRelaunchWhenFinishedTerminat"...
BL              sub_1A5514
```

```
loc_242FE
MOV             R0, #(byte_26CBCC - 0x2430A) ; byte_26CBCC
ADD             R0, PC ; byte_26CBCC
STRB            R4, [R0]
POP             {R4,R7,PC}
; End of function sub_242C8
```

图 3-79 subroutine 跳转 1

可以看到，除了中间的 sub_1A5514 外，sub_242C8 只做了一些简单的寄存器操作，可以忽略；双击 sub_1A5514，跳转到它的实现代码处，如图 3-80 所示。

这里除了 sub_1A55D0 外，其余的也都是一些简单的寄存器操作。双击 sub_1A55D0，如图 3-81 所示。

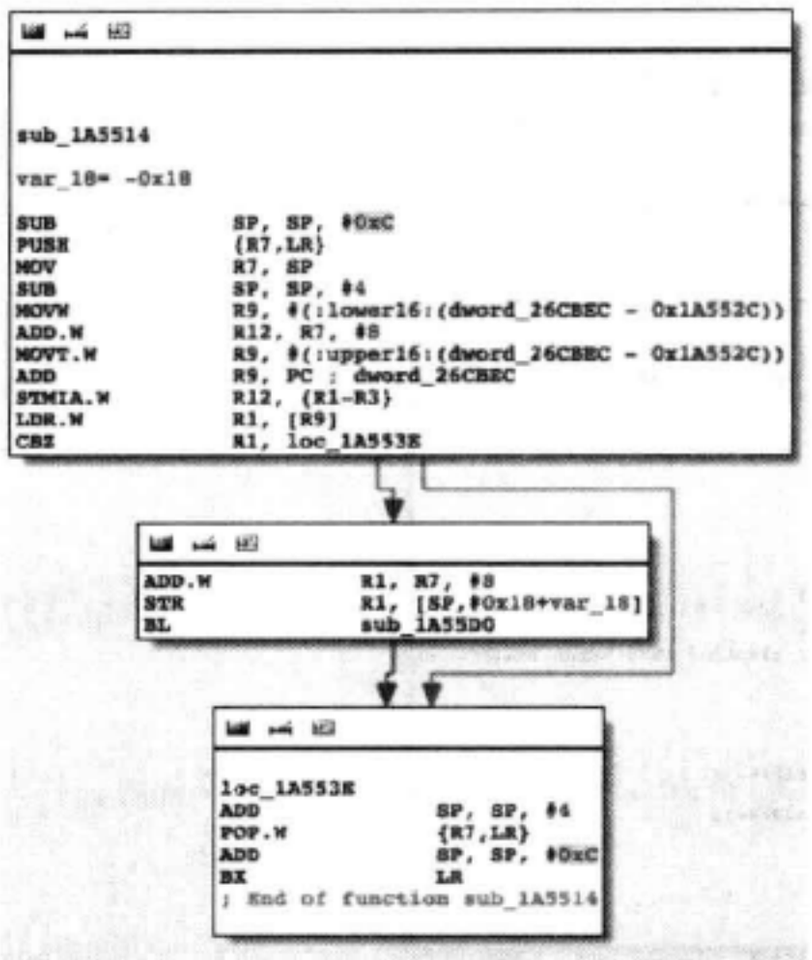


图 3-80 subroutine 跳转 2

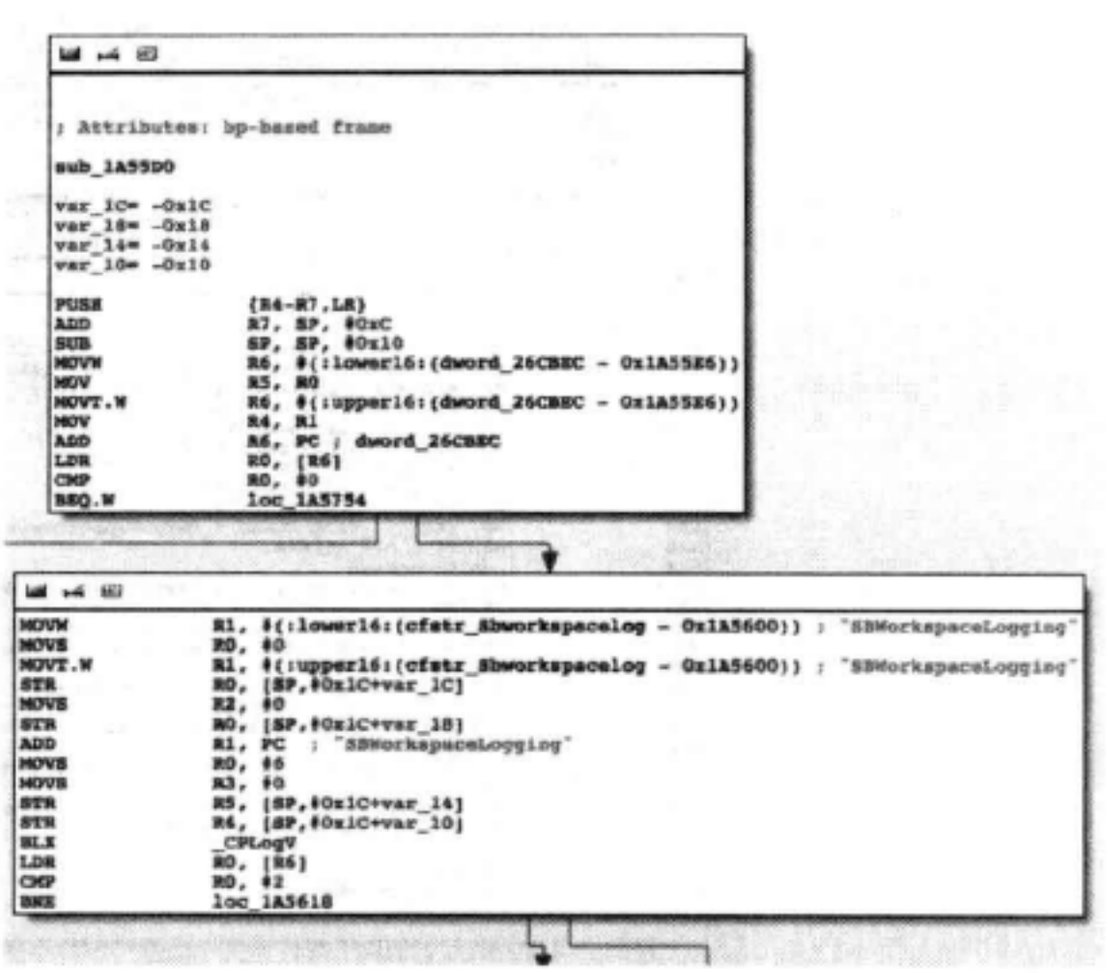


图 3-81 查找到 SBWorkspaceLogging

我们一眼就能看到 SBWorkspaceLogging 的关键词，稍微懂一点英语的读者都能猜到，sub_1A55D0 的作用是将 respring 的一些操作记录下来，与 respring 的主体功能无关。点击 IDA 菜单栏上的蓝色后退按钮，如图 3-82 所示，或直接按 esc 键，回退到 relaunchSpringBoard 函数体中，继续往下分析。



图 3-82 导航栏

双击 sub_236FC，跳转到如图 3-83 所示的界面。

只需要大致浏览，结合一些常识就可以猜到这个 subroutine 的作用是 TerminateAllApps，具体做法是把 TerminateAllApps 的任务加到一个执行队列中，然后执行队列中的任务，从而关掉所有的 App。sub_1A5514 再次出现，前面已经分析过，它只对系统行为做一些记录。再次回退到 relaunchSpringBoard 函数体中，到这里，我们的分析已经接近尾声了：函数调用 _relaunchSpringBoardNow，完成 respring 操作。

不需要了解汇编代码，也不用熟悉调用规则，我们以几乎零基础的水平成功完成了这次逆向工程，不是吗？当然，这不能说明我们的水平有多么高深，而是提醒我们应该为拥有 IDA 这样强大且免费的神器而感到幸运，从而激励我们加倍努力。在绝大多数情况下，我们对 IDA 的使用与上面的示例没有本质区别，你只需要耐着性子，仔细咀嚼 IDA 呈现出的每一行代码，要不了多久，你就会深切感受到逆向工程的艺术之美。



图 3-83 subroute 跳转

3.5 其他工具

除了以上 4 个比较重要的工具之外，还有一些辅助性的工具需要了解一下，比如 iTools、dyld_decache、MesaSQLite。

3.5.1 iTools

iTools 是国内的创想天空（ThinkSky）出品的一款 iOS 设备的同步管理软件，用户可以使用 iTools 非常方便地完成对 iOS 设备的信息查看、音乐 / 铃声 / 照片 / 书籍 / 文件管理、软件安装等操作。

在这里主要用 iTools 来做文件传输，我们常用的目录包括 /var/mobile/Applications 和 /var/root。

从图 3-84 中可以看到，“程序（用户）”其实就是对应了

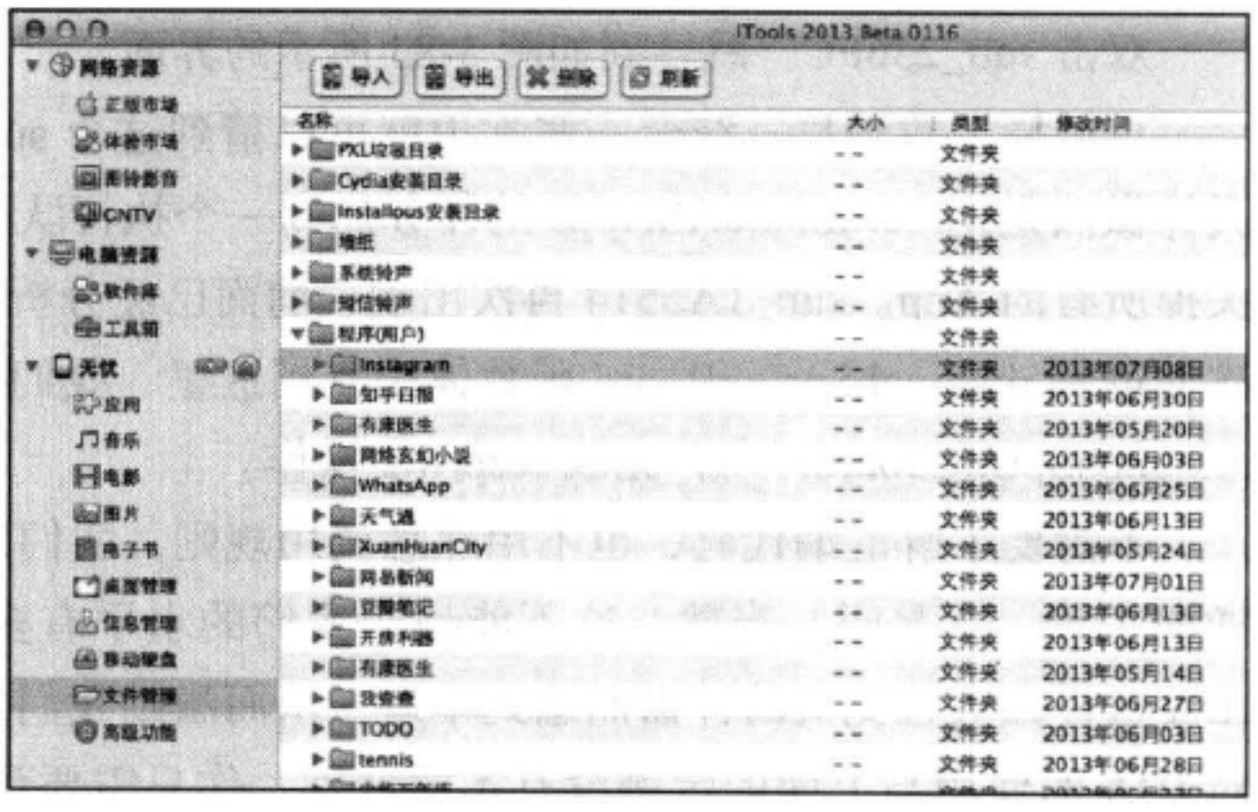


图 3-84 iTools 显示 iOS 设备结构

/var/mobile/Applications，只是 iTools 更方便地为我们把 App 名称显示出来了。

3.5.2 dyld_decache

相信不少读者在浏览 iOS 文件系统时会发现一个问题：/System/Library/Frameworks 和 PrivateFrameworks 目录下的 bundle 中，怎么没有可执行文件？实际上，从 iOS 3.1 开始，包括 frameworks 在内的许多库文件被存进了一个大 cache 中，这个 cache 文件位于 /System/Library/Caches/com.apple.dyld 下，可以使用 KennyTM 开发的 dyld_decache 将其中的库文件提取出来。这样做的好处主要是确保逆向分析的目标文件与 iOS 中的对应文件完全相同，避免出现驴唇不对马嘴的低级错误。有关这个 cache 的进一步介绍，可以参阅 DHowett 的博客文章 http://blog.howett.net/2009/09/cache-or-check/。

在使用 dyld_decache 之前，要将 /System/Library/Caches/com.apple.dyld 下的 dyld_shared_cache_armv6（或 dyld_shared_cache_armv7）从 iOS 复制到 Mac OS X 中。注意，iOS 4.3 以后的系统支持 ASLR（Address Space Layout Randomization），如果复制的动作是由 iOS 中开启了 ASLR 的进程完成的，这个 cache 的内容会发生变化，导致 dyld_decache 提取失败。一个简单的解决方法是使用基于 AFC（Apple File Connection）服务的软件，如 iFunBox、iExplorer 或 iTools 来复制这个 cache。

dyld_decache 可从 https://github.com/kennytm/Miscellaneous/downloads 下载。准备工作完成后，在 Terminal 中输入命令“dyld_decache -o path/to/destination/folder path/to/dyld_shared_cache_armvX”即可进行提取操作，如图 3-85 所示。

```
[redacted]-iMac:~ snakeninny$ dyld_decache -o /Users/snakeninny/Code/iOSSystemBinar
ies/ExtractedBinaries /Users/snakeninny/Code/iOSSystemBinaries/dyld_shared_cache
_armv7
  0/565: Dumping '/System/Library/BulletinBoardPlugins/CMASBBPlugin.bundle/CMASB
BPlugin'...
  1/565: Dumping '/System/Library/BulletinBoardPlugins/SMSBBPlugin.bundle/SMSBBP
lugin'...
  2/565: Dumping '/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib'...
  3/565: Dumping '/System/Library/CoreServices/Encodings/libArabicConverter.dyli
b'...
  4/565: Dumping '/System/Library/CoreServices/Encodings/libCyrillicConverter.dy
lib'...
  5/565: Dumping '/System/Library/CoreServices/Encodings/libGreekConverter.dylib
'...
  6/565: Dumping '/System/Library/CoreServices/Encodings/libHebrewConverter.dyli
b'...
  7/565: Dumping '/System/Library/CoreServices/Encodings/libJapaneseConverter.dy
lib'...
  8/565: Dumping '/System/Library/CoreServices/Encodings/libKoreanConverter.dyli
b'...
  9/565: Dumping '/System/Library/CoreServices/Encodings/libLatin2Converter.dyli
b'...
 10/565: Dumping '/System/Library/CoreServices/Encodings/libLatin5Converter.dyli
```

图 3-85 执行 dyld_decache 命令行

3.5.3 MesaSQLite

MesaSQLite 是用来查看 SQLite 数据库的工具，因为在后续的调试代码中，我们会把包括内容和调试信息的代码输出到 SQLite 中保存，而此时导出到 Mac 中查看会比较方便一些，

如图 3-86 所示。

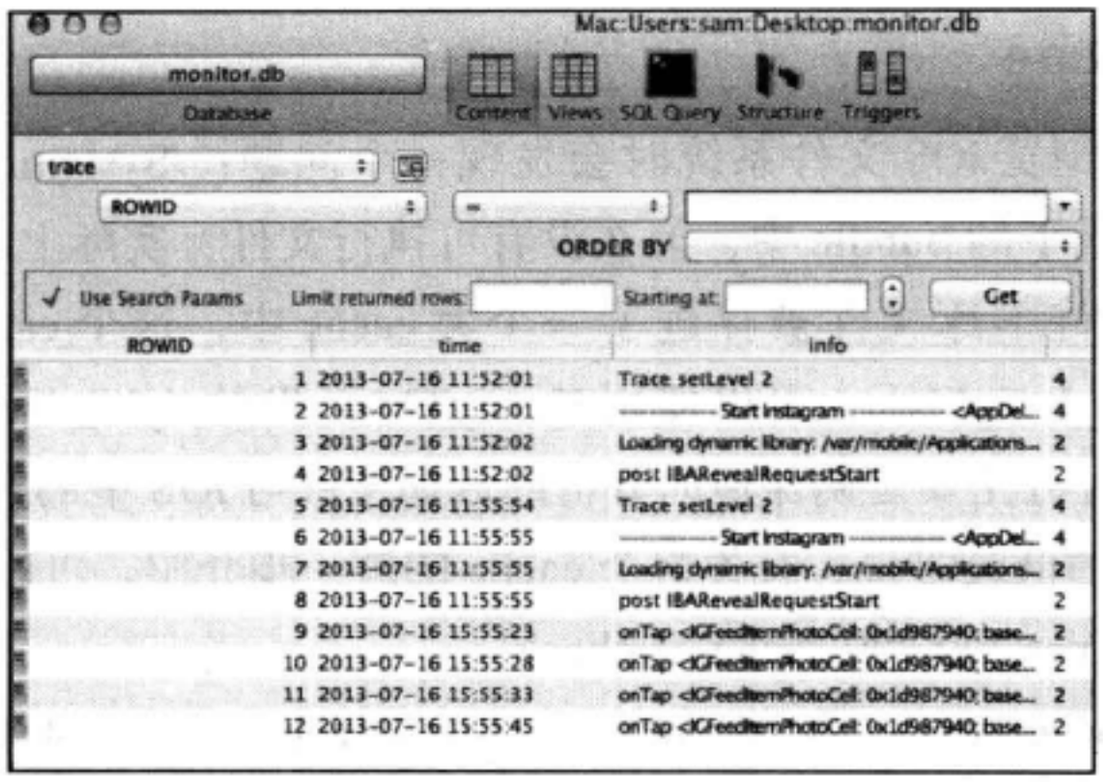


图 3-86 MesaSQLite 查看 SQLite 数据库文件

3.6 小结

本章重点介绍了 4 个工具：class-dump、Theos、Reveal、IDA，从这 4 个工具的作用也可以看出，了解这些工具，可一步步由浅入深地掌握逆向分析的内容。

第4章

iOS工具集

第3章是围绕Mac平台的开发、调试工具展开的，但是，为了配合正常完成iOS的逆向工程，我们需要针对iOS平台安装、配置一系列的工具，将两个平台联动起来，才能正常地完成整个逆向工程。

逆向工程是基于越狱平台来进行的，所以，对iOS设备的越狱完成之后，就得挨个安装工具来搭建这个调试平台了。不过，由于对于逆向工程并没有一个标准的模板来定义应该如何搭配使用工具，所以这一章的内容均是笔者根据经验筛选出来的，只求达到目标即可，并未覆盖所有主流工具。

说明 本章中所有的工具均是使用Cydia来安装的，如图4-1所示，不包括其他渠道的工具。



图4-1 Cydia启动界面

4.1 SBSettings

在iOS越狱平台上，首先需要知道的一个工具是SBSettings，对越狱玩家来说应该是不陌生的。它是一个iOS系统增强工具，前身是BigBoss出品的BossPrefs，主要功能有：

- 清理系统内存。
- 快速注销、重启、关机以及进入安全模式等。
- 隐藏桌面图标（比如不常用的系统自带软件）。
- 自定义飞行模式、蓝牙、Wi-Fi、EDGE、自动横屏、亮度调节等开关的设置。

SBSettings来自Cydia的自带BigBoss源，由BigBoss亲自开发，如图4-2所示。



图4-2 SBSettings安装详情

在使用SBSettings之后，很多原本需要进入“设置”中完成的操作可以直接在顶部状态栏滑动一下呼出，如图4-3所示。

SBSettings界面出现之后，可以直接查看IP地址、开关3G、

Wi-Fi 等功能，再不用一层层点击，进入到“设置”状态实现操作。并且，还可以在 SBSettings 的设置界面中控制 iOS 设备上的 MobileSubstrate 插件（即 tweak）开关，如图 4-4 所示。



图 4-3 SBSettings 在桌面被“呼出”

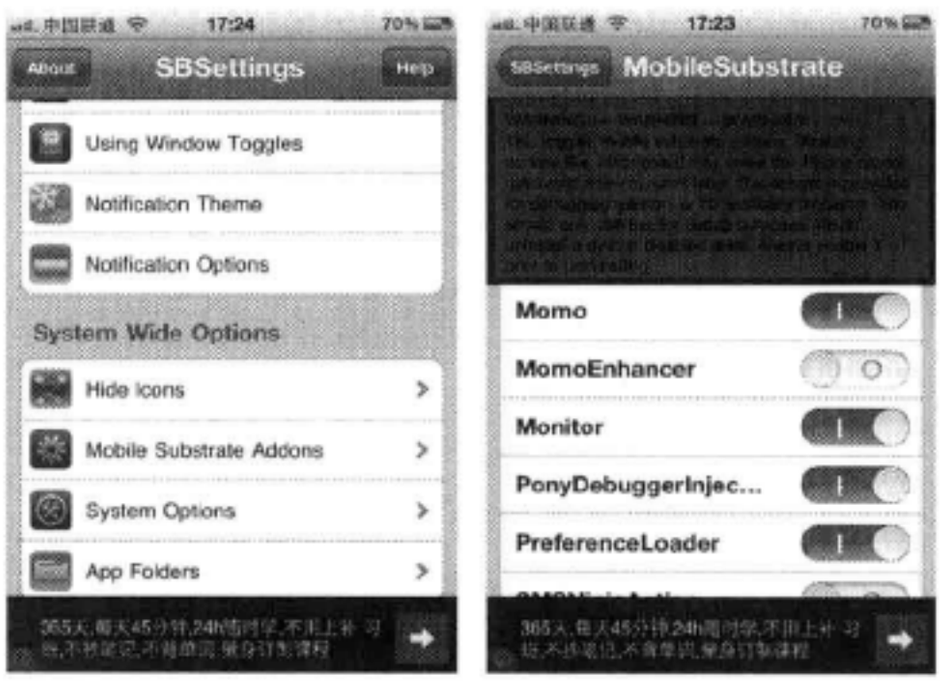


图 4-4 SBSettings 管理插件

有了这样一个便捷、强大的平台管理工具，也总算是让我们体会到了一点越狱之后的好处。

4.2 MobileSubstrate

第 1 章已经提到过，MobileSubstrate 是现有越狱插件运行的基础。

MobileSubstrate 由 3 部分组成：

- MobileHooker，前面介绍的 Theos 就是基于这个 hooker 实现的；
- MobileLoader，用于加载第三方开发的 tweak 代码；
- Safe mode，用于保护系统，一旦第三方 tweak 代码导致 SpringBoard 异常，能让整个系统进入安全模式，在这种模式下，所有第三方 tweak 代码都将失效，给我们机会来修复系统。

MobileSubstrate 的安装源是 Cydia/Telesphoreo(http://apt.saurik.com)，如图 4-5 所示。

通过 iTools 连接 iOS 设备可以看到，在 /Library/MobileSubstrate/DynamicLibraries 目录下放了很多 dylib 文件，这些文件也就是在 iOS 设备中安装的所有 tweak 插件在目录中的文件存在形式，如图 4-6 所示。



图 4-5 MobileSubstrate 安装详情

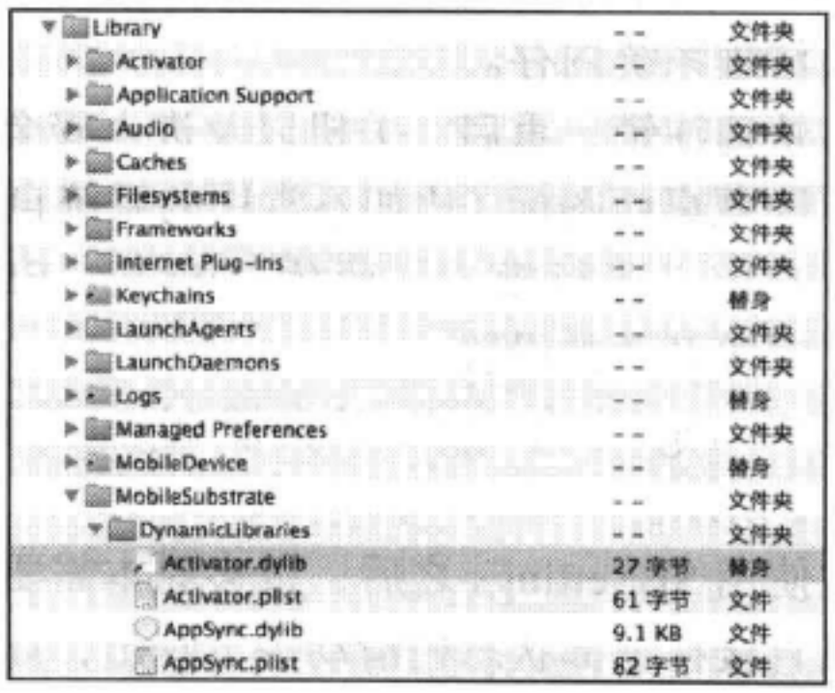


图 4-6 MobileSubstrate 对应的目录结构

4.3 OpenSSH

OpenSSH，顾名思义，就是提供了一个 SSH 服务，安装之后即给外界提供了一个通过 SSH 接入 iOS 设备的途径。有了这个 SSH 服务，就可以不连接数据线、以无线的方式远程登录到 iOS 设备上了，而且想干什么就可以干什么。

不过，其实主要也就使用了两个命令：ssh 和 scp，前者用于远程登录，后者用于远程复制文件。

虽然多数时候用 iTools 来传输文件，但偶尔也会使用 scp 来上传或下载文件，操作的命令如图 4-7 所示。

```
sammatoMacBook-Pro:monitor sam$ scp com.hangcom.monitor_0.0.1
-4_iphoneos-arm.deb root@192.168.1.115:/var/root
root@192.168.1.115's password:
com.hangcom.monitor_0.0.1- 100% 3490     3.4KB/s   00:00
sammatoMacBook-Pro:monitor sam$
```

图 4-7 scp 运行命令

OpenSSH 的安装源是 Cydia/Telesphoreo（http://apt.saurik.com），如图 4-8 所示。SSH 可以通过 SBSettings 来管理，其界面如图 4-9 所示。



图 4-8 OpenSSH 安装详情

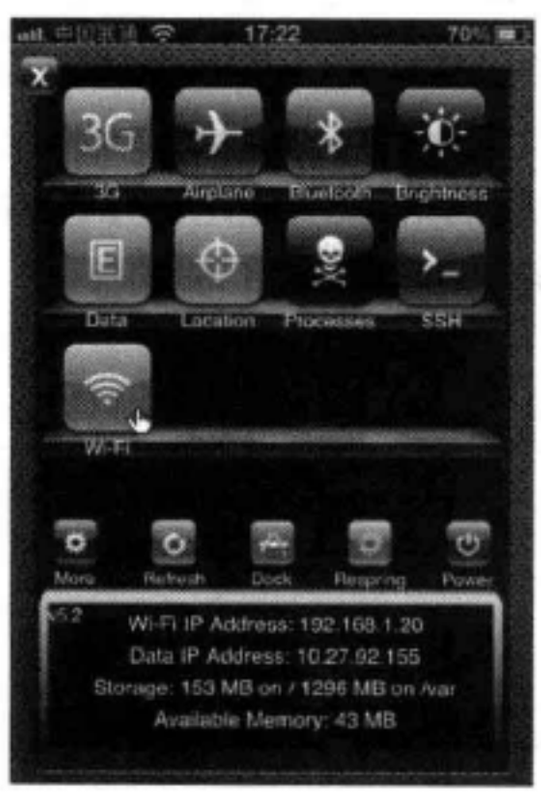


图 4-9 通过 SBSettings 开 / 关 SSH 服务

注意，OpenSSH 的默认密码是 alpine，安装之后，要立即登录并修改密码，注意，需要修改密码的账号有两个：root 和 mobile，修改 root 密码的动作如图 4-10 所示。

```
sammatoMacBook-Pro:~ sam$ ssh root@192.168.1.115
root@192.168.1.115's password:
buyou:~ root# passwd
Changing password for root.
New password:
Retype new password:
buyou:~ root#
```

图 4-10 SSH 登录并修改密码

如果没有修改默认密码，蠕虫病毒（如 Ikee）或其他人就能够通过这个默认密码登录 iOS 设备，并且会导致严重后果：iOS 设备上的所有数据，包括短信、电话记录、联系人，甚至微信聊天信息、微博私信都可以被他人一览无余。一旦密码被篡改，再想恢复就只能重装这个服务了。

4.4 GDB

4.4.1 GDB 简介

如果说 IDA 是倚天剑，GDB 就是屠龙刀，两者在逆向工程中的地位不相上下，难分伯仲；GDB 的全名叫 The GNU Project Debugger，是类 UNIX 操作系统中最强大的动态调试工具之一（也可运行在 Windows 中）。GDB 的功能可以概括为以下四点：

- ❑ 在你指定的条件下启动程序。
- ❑ 在你指定的条件下停止程序。
- ❑ 在程序停止的时候检查程序内部发生的事。
- ❑ 在程序停止的时候对程序进行改动，观察程序的执行过程有什么变化。



图 4-11 GDB 安装详情

GDB 没有图形界面，使用时看着 Terminal 中黑压压的一片文字，很容易就吓跑初学者，但一旦掌握其基本用法，配合 IDA 双管齐下，就能解决很多难倒大片初学者的问题，投资收益比极高。iOS 中的 GDB 需要从 radare 源（http://cydia.radare.org/）下载，如图 4-11 所示。

在 GDB 安装完成后，推荐安装由 @osxreverser 编写的 gdbinit。gdbinit 是 GDB 初始化脚本，在 GDB 启动过程中执行，对 GDB 有一定的优化作用。@osxreverser 的 gdbinit 可以从 https://github.com/gdbinit/Gdbinit/archive/master.zip 下载，下载后将解压得到的 gdbinit 复制到 iOS 的 /var/root 目录下，并重命名为 .gdbinit 即可，具体命令如图 4-12 所示。

```
[redacted]-iMac:~ snakeninny$ scp /Users/snakeninny/Downloads/Gdbinit-master/gdbini
t root@192.168.2.2:/var/root/.gdbinit
root@192.168.2.2's password:
gdbinit                                       100%  103KB 103.5KB/s   00:00
[redacted]-iMac:~ snakeninny$
```

图 4-12 gdbinit 的运行参数

4.4.2 GDB 的使用说明

在调试阶段，一般情况下，可通过 ssh 方式使用 GDB 来实现调试（当然在少数特殊情况下也可用 MobileTerminal 调试，但不推荐）。在 Terminal 中输入“gdb”，进入之后提示符变为“gdb$”，如图 4-13 所示（不同版本的 GDB 输出可能不同，以下同理）。

```
[redacted]-FunMaker-5:~ root# gdb
/usr/bin/gdb: line 55: awk: command not found
warning: unrecognized host cpusubtype , defaulting to host==armv7.
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4) (
Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
gdb$
```

图 4-13 进入 GDB

虽然GDB能够启动程序，但是当前阶段的逆向目标绝大多数是各种App和daemon，前者是由一个可执行程序组成的，仅以GDB启动这个可执行程序，App无法显示界面；daemon是开机自启动的，根本轮不到GDB来启动。因此，一般用到的是GDB的attach命令，将GDB直接注入此进程用到的命令则是“attach PID”或者“attach ProcessName”，如图4-14所示。

```
Touch-1:~ root# gdb
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4) (
Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
gdb$ attach SpringBoard
Attaching to process 150.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ..........................................
..............................................................................
..............................................................................
```

图4-14 attach SpringBoard

attach命令完成后，程序停止，等待我们输入进一步的命令。

注意，所有GDB命令涉及的正确地址必须来自本机二进制文件，可通过第3章介绍的dyld_decache获得；从SDK中得到的二进制文件与本机通常不同，两者地址不能混用。

常用的GDB命令如下。

（1）break

break命令的作用是设置断点，可以简写为b。在逆向工程中，没有调试程序的源代码，所以一般用到的是“b function”或“b *address”命令，前者在函数的起始位置设置断点，如图4-15所示。

后者在地址处设置断点，如图4-16所示。

```
gdb$ b object_getClass
Breakpoint 3 at 0x3a0203d8
gdb$ b notify_post
Breakpoint 4 at 0x3a559462
gdb$ b AudioServicesPlaySystemSound
Breakpoint 5 at 0x31e5274a
gdb$
```

图4-15 在函数名上设置断点

```
gdb$ b *(0x59000 + 0x00009218)
Breakpoint 9 at 0x62218
gdb$ c
```

图4-16 在内存地址上设置断点

当进程执行到断点时，断点所在的那一行代码并未得到执行。有一点需要注意：因为调试对象在编译时存在差异，GDB不一定总能从调试对象中获取足够的调试信息，所以“b function”的方式并不总能成功，如图4-17所示。

因此，本书在设置GDB断点时，大多采用“b *address”的方式。那么，应该如何得出某一行代码的地址呢？因为ASLR（Address Space Layout Randomization，也可称为“地址空间随机布局”，

```
gdb$ b -[NString init]
No symbol table is loaded.  Use the "file" command.
gdb$ b +[NSBundle mainBundle]
No symbol table is loaded.  Use the "file" command.
gdb$
```

图4-17 对函数名设置断点未必能成功

详见 http://theiphonewiki.com/wiki/ASLR）的关系，同一进程在每次启动时在内存中的位置都会发生一定偏移：

某行代码的地址 = ASLR 造成的偏移 + 本行代码在文件中的地址

因此要分别获取等式右边的两个数据。

ASLR 对某一进程造成的偏移在这个进程存活的时间里是不变的，所以在附上一个进程时，它的 ASLR 偏移已经确定，可以通过“info share”或“info sh”命令查看，如图 4-18 所示。

```
gdb$ info sh
The DYLD shared library state has not yet been initialized.
                                                Requested State Current Stat
e
Num Basename                            Type Address         Reason I I Source

  I I                                     I I                    I I I I

  1 SpringBoard                         - 0xd2000           exec Y Y /System/Li
brary/CoreServices/SpringBoard.app/SpringBoard at 0xd2000 (offset 0xd1000)
  2 dyld                                - 0x2fefb000        dyld Y Y /usr/lib/d
yld at 0x2fefb000 (offset 0xfb000) with prefix "__dyld_"
  3 SubstrateBootstrap.dylib            - 0x37b000          dyld Y Y /Library/F
rameworks/CydiaSubstrate.framework/Libraries/SubstrateBootstrap.dylib at 0x37b00
0 (offset 0x37b000)
  4 StoreServices                       F 0x36a76000        dyld Y Y /System/Li
brary/PrivateFrameworks/StoreServices.framework/StoreServices at 0x36a76000 (off
set 0x3c2000)
                                                          (objfile is) [memory ob
ject "/System/Library/PrivateFrameworks/StoreServices.framework/StoreServices" a
t 0x36a76000]
  5 AirTraffic                          F 0x33c4b000        dyld Y Y /System/Li
brary/PrivateFrameworks/AirTraffic.framework/AirTraffic at 0x33c4b000 (offset 0x
3c2000)
```

图 4-18 info sh 查 ASLR

每一条目最后的 offset 字样即标注了此二进制文件在内存中的 ASLR 偏移，如 SpringBoard 是 0xd1000，dyld 是 0xfb000，SubstrateBootstrap.dylib 是 0x37b000，等等。要得到等式左边的最终结果，还需要进一步用 IDA 得到目标代码的地址。上一章已经介绍过如何用 IDA 定位我们感兴趣的函数，下面以 - [SpringBoard menuButtonDown:] 为例，来定位它的实现代码，如图 4-19 所示。

```
; SpringBoard - (void)menuButtonDown:(struct __GSEvent *)
; Attributes: bp-based frame

; void __cdecl -[SpringBoard menuButtonDown:](struct SpringBoard *self, SEL, struct __GSEvent *)
__SpringBoard_menuButtonDown__

var_18= -0x18
var_14= -0x14
var_10= -0x10

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
STR.W           R8, [SP,#0xC+var_10]!
SUB             SP, SP, #0x10
MOVW            R1, #(:lower16:(selRef_standardUserDefaults - 0x9238))
MOV             R4, R0
MOVT.W          R1, #(:upper16:(selRef_standardUserDefaults - 0x9238))
MOV             R0, #(classRef_NSUserDefaults - 0x923A) ; classRef_NSUserDefaults
ADD             R1, PC ; selRef_standardUserDefaults
ADD             R0, PC ; classRef_NSUserDefaults
LDR             R1, [R1] ; "standardUserDefaults"
LDR             R0, [R0] ; _OBJC_CLASS_$_NSUserDefaults
BLX             _objc_msgSend
MOV             R1, #(selRef_boolForKey_ - 0x9250) ; selRef_boolForKey_
MOVW            R2, #(:lower16:(cfstr_Sbdisablehomeb - 0x9256)) ; "SBDisableHomeButton"
ADD             R1, PC ; selRef_boolForKey_
MOVT.W          R2, #(:upper16:(cfstr_Sbdisablehomeb - 0x9256)) ; "SBDisableHomeButton"
ADD             R2, PC ; "SBDisableHomeButton"
LDR             R1, [R1] ; "boolForKey:"
BLX             _objc_msgSend
MOV             R1, #(off_214E14 - 0x9266) ; off_214E14
ADD             R1, PC ; off_214E14
LDR             R2, [R1] ; unk_2705E8
MOV             R1, #(byte_2705C8 - 0x9272) ; byte_2705C8
ADD             R1, PC ; byte_2705C8
LDRB            R2, [R2]
LDRB            R1, [R1]
ORRS            R0, R2
ORRS            R0, R1
TST.W           R0, #0xFF
BEQ             loc_92A2
```

图 4-19 使用 IDA 配合使用

假设要在第二行“ ADD R7, SP, #0xC”下断点，将鼠标焦点放在这一行代码上，按下空格键切换到 Text view，如图 4-20 所示。

```
__text:00009218 var_10          = -0x10
__text:00009218
__text:00009218                 PUSH            {R4-R7,LR}
__text:0000921A                 ADD             R7, SP, #0xC
__text:0000921C                 STR.W           R8, [SP,#0xC+var_10]!
__text:00009220                 SUB             SP, SP, #0x10
__text:00009222                 MOVW            R1, #(:lower16:(selRef_standardUserDefaults - 0x9238))
__text:00009226                 MOV             R4, R0
__text:00009228                 MOVT.W          R1, #(:upper16:(selRef_standardUserDefaults - 0x9238))
__text:0000922C                 MOV             R0, #(classRef_NSUserDefaults - 0x923A) ; classRef_NSUserDefaults
__text:00009234                 ADD             R1, PC ; selRef_standardUserDefaults
__text:00009236                 ADD             R0, PC ; classRef_NSUserDefaults
__text:00009238                 LDR             R1, [R1] ; "standardUserDefaults"
__text:0000923A                 LDR             R0, [R0] ; _OBJC_CLASS_$_NSUserDefaults
__text:0000923C                 BLX             _objc_msgSend
__text:00009240                 MOV             R1, #(selRef_boolForKey_ - 0x9250) ; selRef_boolForKey_
__text:00009248                 MOVW            R2, #(:lower16:(cfstr_Sbdisablehomeb - 0x9256)) ; "SBDisableHomeButton"
__text:0000924C                 ADD             R1, PC ; selRef_boolForKey_
__text:0000924E                 MOVT.W          R2, #(:upper16:(cfstr_Sbdisablehomeb - 0x9256)) ; "SBDisableHomeButton"
__text:00009252                 ADD             R2, PC  ; "SBDisableHomeButton"
__text:00009254                 LDR             R1, [R1] ; "boolForKey:"
__text:00009256                 BLX             _objc_msgSend
__text:0000925A                 MOV             R1, #(off_214E14 - 0x9266) ; off_214E14
__text:00009262                 ADD             R1, PC ; off_214E14
__text:00009264                 LDR             R2, [R1] ; unk_2705E8
__text:00009266                 MOV             R1, #(byte_2705C8 - 0x9272) ; byte_2705C8
__text:0000926E                 ADD             R1, PC ; byte_2705C8
__text:00009270                 LDRB            R2, [R2]
__text:00009272                 LDRB            R1, [R1]
__text:00009274                 ORRS            R0, R2
__text:00009276                 ORRS            R0, R1
__text:00009278                 TST.W           R0, #0xFF
__text:0000927C                 BEQ             loc_92A2
__text:0000927E                 MOV             R0, #(off_214E10 - 0x928A) ; off_214E10
__text:00009286                 ADD             R0, PC ; off_214E10
__text:00009288                 LDR             R0, [R0] ; byte_2705F8
__text:0000928A                 LDRB            R0, [R0]
__text:0000928C                 CMP             R0, #1
__text:0000928E                 BNE.W           loc_948E
__text:00009292                 MOV             R0, #(cfstr_IgnoringMenuBu - 0x929E) ; "Ignoring menu button events"
__text:0000929A                 ADD             R0, PC  ; "Ignoring menu button events"
__text:0000929C                 BL              sub_1A8EF8
__text:000092A0                 B               loc_948E
__text:000092A2 ; ---------------------------------------------------------------------------
```

图 4-20 切换到 Text View 模式

Main window 左侧的一列十六进制数字，即是每一行代码在 SpringBoard 二进制文件中的地址，显然“ ADD R7, SP, #0xC”对应的是 0x0000921A。至此，成功获得了两个关键数据，得出“ ADD R7, SP, #0xC”的实际地址是 0x6221A，即 0xd1000+ 0x921A。在 GDB 中输入“ b *0xda21a”或“ b *(0xd1000+ 0x921a)”，即在 0xda21a 处设下断点，如图 4-21 所示。

```
gdb$ b *(0xd1000 + 0x0000921A)
Breakpoint 1 at 0xda21a
```

图 4-21 设置断点

需要特别注意的是，iOS 上的 GDB 还存在不少 bug，进程执行到断点时偶尔会出现 SIGTRAP、SIGINT 等异常，如图 4-22 所示。

这时可以采取重启调试进程，将断点往前或往后设置若干行代码等方式来回避这类 bug。例如，此处把断点前移一行代码至“ PUSH {R4-R7,LR}”，断点即可正常工作，如图 4-23 所示。

```
gdb$ b *(0xd1000 + 0x0000921A)
Breakpoint 1 at 0xda21a
gdb$ c

Program received signal SIGTRAP, Trace/breakpoint trap.
0x000da218 in _mh_execute_header ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
```

图 4-22 GDB 有时会出现 SIGTRAP、SIGINT 等异常

```
gdb$ b *(0xd1000 + 0x00009218)
Breakpoint 3 at 0xda218
gdb$ c

Breakpoint 3, 0x000da218 in _mh_execute_header ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
```

图 4-23 断点前移

在图 4-23 中，设置一个断点后，GDB 会显示当前断点编号，也可以通过 info break 或

info b 命令来查看所有的断点信息，如图 4-24 所示。

```
gdb$ info b
Num Type           Disp Enb Address    What
5   breakpoint     keep y   0x000da218 <_mh_execute_header+33304>
gdb$
```

图 4-24 查看断点列表

还可通过 disable breakpointNumber 命令来禁用某个断点，即将断点的 Enb 位置 n，如 disable 3 命令会禁用 3 号断点；disable 命令可禁用所有断点。enable 命令与 disable 命令的用法类似，但功能完全相反。要彻底删除断点，可使用 delete 或 d 命令；delete breakpointNumber 命令的作用是删除某个断点。当进程停在断点处时，可以通过 continue 或 c 命令来使进程继续运行。另外，部分进程如 SpringBoard 在挂起一段时间后会自动重启，需要用 GDB 重新附上才能继续调试，且 ASLR 偏移一般会发生变化，请读者在实践过程中注意。建议在 GDB 调试一个进程（尤其是 SpringBoard）时，切换窗口进行别的操作前，最好是禁用或删除所有的断点，然后继续进程，防止 SpringBoard 自动重启从而导致所有 GDB 断点丢失的惨剧发生。

（2）nexti

nexti 的作用是执行下一条机器指令，且将一个函数看做一条指令，不进入函数体；nexti 可以简写成 ni，是配合断点使用最多的指令之一。很多读者可能对“将一个函数看做一条指令，不进入函数体”的解释有些模糊，这里举个例子来说明，如图 4-25 所示。

```
__text:000032BC ; RootViewController - (void)buttonPressed:(id)
__text:000032BC ; Attributes: bp-based frame
__text:000032BC
__text:000032BC ; void __cdecl -[RootViewController buttonPressed:](struct RootViewController *self, SEL, id)
__text:000032BC __RootViewController_buttonPressed__     ; DATA XREF: __objc_const:000043C4↓o
__text:000032BC
__text:000032BC var_14           = -0x14
__text:000032BC var_10           = -0x10
__text:000032BC var_C            = -0xC
__text:000032BC
__text:000032BC                  PUSH            {R4,R5,R7,LR}
__text:000032BE                  ADD             R7, SP, #8
__text:000032C0                  SUB             SP, SP, #0xC
__text:000032C2                  MOV             R5, R0
__text:000032C4                  MOV             R0, #(cfstr_ID - 0x32D2) ; "i = %d"
__text:000032CC                  MOVS            R1, #1
__text:000032CE                  ADD             R0, PC  ; "i = %d"
__text:000032D0                  MOV             R4, R2
__text:000032D2                  BLX             _NSLog
__text:000032D6                  MOVW            R0, #(:lower16:(selRef_stringWithFormat_ - 0x32EC))
__text:000032DA                  MOV             R3, R4
```

图 4-25 IDA 中查看 buttonPressed 函数

该例中，在地址 0x32d2 处，NSLog 得到调用。已知 ASLR 偏移是 0x17000，那么在 0x1A2D0（0x32d0 + 0x17000）处下断点，并使进程执行到此断点处。输入命令 ni，执行下一条指令，进程会停在 0x1A2D2（0x32d2 + 0x17000）处，NSLog 尚未得到执行；继续按下 Enter 键（不输入任何命令，直接按下 Enter 键，则会重复执行上一条命令），NSLog 得到执行，进程会停在 0x1A2d6（0x32d6 + 0x17000）处，NSLog 函数被当做一条指令，ni 命令并未进入 NSLog 的实现代码中去，如图 4-26 所示。

这就是“将一个函数看做一条指令，不进入函数体”的意思，请读者注意跟下面的

“stepi”命令区分。

```
gdb$ b *(0x00003200 + 0x17000)
Breakpoint 1 at 0x1a200
gdb$ c

Breakpoint 1, 0x0001a200 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0001a202 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0001a206 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0001a20a in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 4-26 使用 ni 和 c 调试

（3）stepi

stepi 的作用是执行下一条机器指令，且进入函数体；stepi 可以简写成 si，也是配合断点使用最多的指令之一。对比上面的 nexti，相信读者已经能够理解“进入函数体”的意思，下面再举一个例子加深读者的印象。在图 4-27 中，双击 0x32d2 处的 NSLog，进入 NSLog 的“实现代码”，如图 4-27 所示。

```
__symbolstub1:00003FF4 ; ================ S U B R O U T I N E =======================================
__symbolstub1:00003FF4
__symbolstub1:00003FF4 ; Attributes: thunk
__symbolstub1:00003FF4
__symbolstub1:00003FF4 _NSLog                                  ; CODE XREF: -[RootViewController buttonPressed:]+16ip
__symbolstub1:00003FF4                 LDR             PC, =__imp__NSLog
__symbolstub1:00003FF4 ; End of function _NSLog
```

图 4-27 IDA 查看 NSLog 的“实现代码”

imp 前缀告诉我们 NSLog 是从外部 import 来的，所以此处的“实现代码”可以理解成是一个指针，指向 NSLog 真正实现代码的地址。NSLog“实现代码”第一条指令的地址是 0x1aff4（0x3ff4 + 0x17000）。当进程停在 0x1a2d2（0x32d2 + 0x17000）处，NSLog 尚未得到执行时，输入 si，GDB 会停在 NSLog“实现代码”的第一条指令（即 0x1aff4）处，如图 4-28 所示。

那么继续执行 si，0x31d8e218 这个地址又是怎么来的？双击图 4-27 中 0x3ff4 处的 __imp__NSLog，跳转到其“实现代码”处，如图 4-29 所示。

可以看到，NSLog 是从 Foundation.framework 中 import 来的，佐证了刚才的说法。那么，0x31d8e218 这个地址就应该是 Foundation 的 ALSR 偏移加上 NSLog 函数在 Foundation 二进制文件中的地址，我

```
Breakpoint 2, 0x0001a200 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0001a202 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ si
0x0001aff4 in dyld_stub_NSLog ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x31d8e218 in NSLog ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x31d8e21a in NSLog ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 4-28 使用 si

们来验证一下：在 GDB 中按下“control + C”，输入“info sh”，定位到 Foundation，如图 4-30 所示。

```
UNDEF:00005C4C ; Imports from /System/Library/Frameworks/Foundation.framework/Foundation
UNDEF:00005C4C ;
UNDEF:00005C4C                 IMPORT __imp__NSLog      ; CODE XREF: _NSLogij
UNDEF:00005C4C                                          ; DATA XREF: _NSLogio ...
UNDEF:00005C50                 IMPORT _OBJC_CLASS_$_NSAutoreleasePool ; DATA XREF: _main+1Eio
UNDEF:00005C50                                          ; __objc_classrefs:classRef_NSAutoreleasePoolio
UNDEF:00005C54                 IMPORT _OBJC_CLASS_$_NSString
UNDEF:00005C54                                          ; DATA XREF: -[RootViewController buttonPressed:]+32io
UNDEF:00005C54                                          ; __objc_classrefs:classRef_NSStringio
UNDEF:00005C58 ;
```

图 4-29 IDA 查看 __imp__NSLog

```
/lib/libobjc.A.dylib" at 0x38493000]
  5 Foundation                  F 0x31d5c000        dyld Y Y /System/Library/Fram
eworks/Foundation.framework/Foundation at 0x31d5c000 (offset 0xba7000)
                                                    (objfile is) [memory object "/Sys
tem/Library/Frameworks/Foundation.framework/Foundation" at 0x31d5c000]
  6 CoreFoundation              F 0x39380000        dyld Y Y /System/Library/Fram
eworks/CoreFoundation.framework/CoreFoundation at 0x39380000 (offset 0xba7000)
                                                    (objfile is) [memory object "/Sys
tem/Library/Frameworks/CoreFoundation.framework/CoreFoundation" at 0x39380000]
  7 UIKit                       F 0x32c4f000        dyld Y Y /System/Library/Fram
eworks/UIKit.framework/UIKit at 0x32c4f000 (offset 0xba7000)
                                                    (objfile is) [memory object "/Sys
tem/Library/Frameworks/UIKit.framework/UIKit" at 0x32c4f000]
  8 CoreGraphics                F 0x3122a000        dyld Y Y /System/Library/Fram
eworks/CoreGraphics.framework/CoreGraphics at 0x3122a000 (offset 0xba7000)
                                                    (objfile is) [memory object "/Sys
tem/Library/Frameworks/CoreGraphics.framework/CoreGraphics" at 0x3122a000]
  9 libstdc++.6.dylib           - 0x39ddc000        dyld Y Y /usr/lib/libstdc++.6
.dylib at 0x39ddc000 (offset 0xba7000)
                                                    (objfile is) [memory object "/usr
/lib/libstdc++.6.dylib" at 0x39ddc000]
 10 libgcc_s.1.dylib            - 0x325eb000        dyld Y Y /usr/lib/libgcc_s.1.
dylib at 0x325eb000 (offset 0x325eb000)
```

图 4-30 info sh 可能看到 Foundation

Foundation 的 ASLR 偏移是 0xba7000，依据之前的公式，NSLog 函数在 Foundation 二进制文件中的地址应该是 0x31d8e218 减去 0xba7000，即 0x311e7218。打开 IDA，把 Foundation 二进制文件拖进去，定位到 NSLog 函数，如图 4-31 所示。

```
__text:311E7218 ; ================ S U B R O U T I N E =======================================
__text:311E7218
__text:311E7218
__text:311E7218                 EXPORT _NSLog
__text:311E7218 _NSLog                                  ; CODE XREF: -[NSRecursiveLock unlock]+9Cip
__text:311E7218                                         ; -[NSAutoreleasePool release]+2Eip ...
__text:311E7218
__text:311E7218 var_18          = -0x18
__text:311E7218
__text:311E7218                 SUB             SP, SP, #0xC
__text:311E721A                 PUSH            {R7,LR}
__text:311E721C                 MOV             R7, SP
__text:311E721E                 SUB             SP, SP, #4
__text:311E7220                 ADD.W           R9, R7, #8
__text:311E7224                 STMIA.W         R9, {R1-R3}
__text:311E7228                 ADD.W           R1, R7, #8
__text:311E722C                 STR             R1, [SP,#0x18+var_18]
__text:311E722E                 BL              _NSLogv
__text:311E7232                 ADD             SP, SP, #4
__text:311E7234                 POP.W           {R7,LR}
__text:311E7238                 ADD             SP, SP, #0xC
__text:311E723A                 BX              LR
__text:311E723A ; End of function _NSLog
```

图 4-31 IDA 进行 NSLog 的定位

NSLog 的第一行代码地址是 0x311e7218，与我们的计算结果吻合。看到这里，想必读者

已经能够理解 nexti 和 stepi 的相同点和不同点了吧！

再次提醒，因为 GDB 的 bug，以及断点位置的关系等原因，ni 和 si 命令有时会继续程序，有时会让人不知所终，这种情况可以通过更改断点位置来予以回避。

（4）print

GDB 的主要功能之一是“在程序停止的时候检查程序内部发生的事”，而这个功能正是通过 print 命令完成的；print（或简写为 p）命令的功能是打印某处的值，如“p $r0”是打印寄存器 r0 中存放的值。下面举一个简单的例子来演示 print 命令的作用，如图 4-32 所示。

```
__text:000032BC
__text:000032BC                 PUSH    {R4,R5,R7,LR}
__text:000032BE                 ADD     R7, SP, #8
__text:000032C0                 SUB     SP, SP, #0xC
__text:000032C2                 MOV     R5, R0
__text:000032C4                 MOV     R0, #(cfstr_ID - 0x32D2) ; "i = %d"
__text:000032CC                 MOVS    R1, #1
__text:000032CE                 ADD     R0, PC  ; "i = %d"
__text:000032D0                 MOV     R4, R2
__text:000032D2                 BLX     _NSLog
__text:000032D6                 MOVW    R0, #(:lower16:(selRef_stringWithFormat_ - 0x32EC))
__text:000032DA                 MOV     R3, R4
```

图 4-32 IDA 查看“MOVS R1, #1”

已知 IDA 中显示二进制文件的 ASLR 偏移为 0xd7000。注意 0x32cc 处的“MOVS R1, #1”指令，我们在下一个断点，待断点被触发后，输入“p $r1”查看 r1 的值，然后输入 ni 执行此条指令，再输入“p $r1”查看 r1 的值，如图 4-33 所示。

```
gdb$ b *(0x000032CC + 0xd7000)
Breakpoint 3 at 0xda2cc
gdb$ c

Breakpoint 3, 0x000da2cc in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ p $r1
$5 = 0x3311099e
gdb$ ni
0x000da2ce in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ p $r1
$6 = 0x1
gdb$ c
```

图 4-33 p $r1

可以看到，在“MOVS R1, #1”执行之前，r1 的值是 0x3311099e；执行之后，r1 的值是 0x1，print 命令将其正确打印了出来。

ptype（或简写为 pt）命令的功能是打印某处的数据类型，如“pt $r1”是打印 r1 存放值的数据类型。在上例中，执行完“MOVS R1, #1”之后，因为 r1 是 32 位寄存器，所以 r1 的值应该是一个 32 位整型的 1。输入“pt $r1”来验证一下，如图 4-34 所示。

r1 值的类型是 int32_t，没有问题。需要注意的是，pt 无法打印 Objective-C 类型，要实现这个功能，需要用到 print-object（或简写为 po）命令。在图 4-34 中，当 0x3318 处的 BLX _objc_msgSend 得到调用时，r0 中存放的应该是一个 UIAlertView 类型的值。

```
gdb$ p $r1
$1 = 0x1
gdb$ pt $r1
type = int32_t
gdb$ c
```

图 4-34 pt $r1

```
__text:000032FE                 MOV     R4, R0
__text:00003300                 MOV     R0, #(selRef_alloc - 0x3314) ; selRef_alloc
__text:00003308                 MOV     R2, #(classRef_UIAlertView - 0x3316) ; classRef_UIAlertView
__text:00003310                 ADD     R0, PC ; selRef_alloc
__text:00003312                 ADD     R2, PC ; classRef_UIAlertView
__text:00003314                 LDR     R1, [R0] ; "alloc"
__text:00003316                 LDR     R0, [R2] ; _OBJC_CLASS_$_UIAlertView
__text:00003318                 BLX     _objc_msgSend
```

图 4-35 IDA 中查看 r0

我们在 0x3318 下一个断点，在断点被触发后用“ po $r0”命令打印出 r0 中存放值的类型，从而验证上面的猜测，如图 4-36 所示。

```
gdb$ b *(0x89000 + 0x00003318)
Breakpoint 1 at 0x8c318
gdb$ c

Breakpoint 1, 0x0008c318 in main ()
Error while running hook_stop:
No symbol 'disassembly" in current context.
gdb$ po $r0
Error while running hook_stop:
No symbol 'disassembly" in current context.
Error while running hook_stop:
No symbol 'disassembly" in current context.
UIAlertView
```

图 4-36 po $r0

r0 中存放的值被成功识别出来。但是 GDB 中的 po 命令似乎很不稳定，在很多时候并不能正确识别 Objective-C 类型，在图 4-37 和图 4-38 中，0x32f0 处的指令“ MOV R2, #(cfstr_YouHitAnObject - 0x32FC) ;”得到执行后，r2 中的值应该是 NSString 类型，但“ po $r2”却显示“0x13cc does not appear to point to a valid object.”，识别失败。

```
__text:000032D6  MOVW    R0, #(:lower16:(selRef_stringWithFormat_ - 0x32EC))
__text:000032DA  MOV     R3, R4
__text:000032DC  MOVT.W  R0, #(:upper16:(selRef_stringWithFormat_ - 0x32EC))
__text:000032E0  MOV     R2, #(classRef_NSString - 0x32EE) ; classRef_NSString
__text:000032E8  ADD     R0, PC ; selRef_stringWithFormat_
__text:000032EA  ADD     R2, PC ; classRef_NSString
__text:000032EC  LDR     R1, [R0] ; "stringWithFormat:"
__text:000032EE  LDR     R0, [R2] ; _OBJC_CLASS_$_NSString
__text:000032F0  MOV     R2, #(cfstr_YouHitAnObject - 0x32FC) ; "You hit an object of %@"
__text:000032F8  ADD     R2, PC  ; "You hit an object of %@"
__text:000032FA  BLX     _objc_msgSend
```

图 4-37 r2 中是 NSString 类型的值

另外，po 命令有时还会导致被调试的程序出现 segmentation fault 错误而崩溃，导致所有断点在 1 秒钟丢失……总的来说，po 命令的稳定性并不好，建议初学者慎用。

```
gdb$ b *(0x000032F0 + 0x89000)
Breakpoint 2 at 0x8c2f0
gdb$ c

Breakpoint 2, 0x0008c2f0 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0008c2f4 in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ po $r2
Error while running hook_stop:
No symbol "disassembly" in current context.
0x13cc does not appear to point to a valid object.
gdb$ c
```

图 4-38 po $r2 却失败了

前面提到，print 命令可以打印某处的值，那如果此处存放的是一个指针，想要查看指针指向数据的值要怎么办呢？这种情况就要用到 examine（或简写为 x）命令。在 IDA 截屏图中，当 0x32be 处的 BLX _NSLog 得到调用时，r1 中存放了一个指针，指 0, 如图 4-39 所示。

在 0x32be 处断下，我们看看“p $r1”和“x $r1”的输出分别是什么，如图 4-40 所示。

显而易见，r1 中存放了一个地址，其值是 0x2fdbe6d8 ；这个地址存放的值是 0x0。笔者感觉 GDB 的 print 命令还不是很稳定，经常会出现各种 bug，有些烦人，请读者在碰到类似问题的时候一定要淡定。

```
__text:0000329C                PUSH            {R4-R7,LR}
__text:0000329E                ADD             R7, SP, #0xC
__text:000032A0                STR.W           R8, [SP,#0xC+var_10]!
__text:000032A4                SUB             SP, SP, #0x34
__text:000032A6                MOV             R5, R0
__text:000032A8                MOVW            R0, #(:lower16:(cfstr_IS - 0x32BC)) ; "i = %s"
__text:000032AC                MOVS            R6, #0
__text:000032AE                MOVT.W          R0, #(:upper16:(cfstr_IS - 0x32BC)) ; "i = %s"
__text:000032B2                STR             R6, [SP,#0x44+var_38]
__text:000032B4                ADD.W           R8, SP, #0x44+var_38
__text:000032B8                ADD             R0, PC  ; "i = %s"
__text:000032BA                MOV             R4, R2
__text:000032BC                MOV             R1, R8
__text:000032BE                BLX             _NSLog
```

图 4-39 BLX _NSLog

```
gdb$ b *(0x40000 + 0x000032BE)
Breakpoint 1 at 0x432be
gdb$ c

Breakpoint 1, 0x000432be in main ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ p $r1
$1 = 0x2fdbe6d8
gdb$ x $r1
0x2fdbe6d8:     0x00000000
gdb$ c
```

图 4-40 p $r1 和 x $r1

（5）set

set 命令的目的很单纯，设置某个值，可以用来改变程序走向，看图 4-41 的例子（取自 -[SpringBoard menuButtonDown:]）：

图 4-41 中，上面那个方块最后两行代码的意思是：将 r0 的值同 0xff 进行“与”操作，得出的结果如果是 0，走接下来这个代码分支，否则走别的分支。在“TST.W R0, #0xFF”处设置断点，待断点触发后，通过“p $r0”查看 r0 的值跟踪程序走向，如图 4-42 所示。

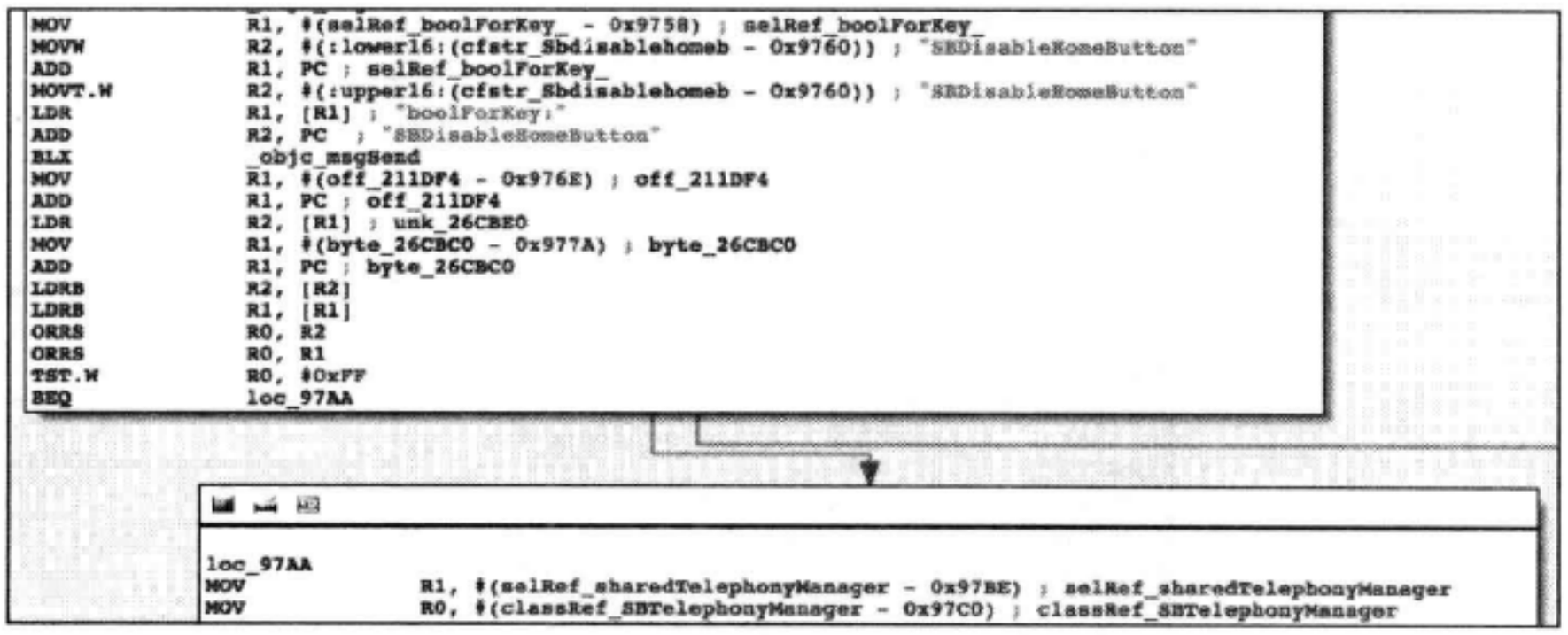


图 4-41 IDA 中的 MOV r0

因为“ TST.W R0, #0xFF”执行前 r0 的值是 0，所以与操作的结果是 0，程序执行到了 0x6d7bc（即 0x64000 + 0x97bc），如图 4-43 所示。

再次触发断点，然后通过 set 命令更改 r0 的值为 1，观察程序的流向是否因此改变，如图 4-44 所示。

```
gdb$ b *(0x64000 + 0x00009780)
Breakpoint 3 at 0x6d780
gdb$ c

Breakpoint 3, 0x0006d780 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ p $r0
$3 = 0x0
gdb$ ni
0x0006d784 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d7a8 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d7bc in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d7bc in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 4-42 p $r0

```
__text:000097AA loc_97AA                                ; CODE XREF: -[SpringBoard menuButtonDown:]+64↑j
__text:000097AA                 MOV     R1, #(selRef_sharedTelephonyManager - 0x97BE) ; selRef_sha
__text:000097B2                 MOV     R0, #(classRef_SBTelephonyManager - 0x97C0) ; classRef_SBT
__text:000097BA                 ADD     R1, PC ; selRef_sharedTelephonyManager
__text:000097BC                 ADD     R0, PC ; classRef_SBTelephonyManager
__text:000097BE                 LDR     R1, [R1] ; "sharedTelephonyManager"
__text:000097C0                 LDR     R0, [R0] ; _OBJC_CLASS_$_SBTelephonyManager
__text:000097C2                 BLX     _objc_msgSend
```

图 4-43 IDA 中的 ADD R1

```
gdb$ b *(0x64000 + 0x00009780)
Breakpoint 4 at 0x6d780
gdb$ c

Breakpoint 4, 0x0006d780 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ p $r0
$4 = 0x0
gdb$ set $r0 = 0x1
gdb$ ni
0x0006d784 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d784 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d790 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d790 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d794 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0006d794 in _mh_execute_header ()
Error while running hook_stop:
```

图 4-44 set $r0

可以看到，程序执行到了0x6d794（即0x64000 + 0x9794），走了右边的线，如图4-45所示。

```
__text:0000977E                ORRS            R0, R1
__text:00009780                TST.W           R0, #0xFF
__text:00009784                BEQ             loc_97AA
__text:00009786                MOV             R0, #(off_211DF0 - 0x9792) ; off_211DF0
__text:0000978E                ADD             R0, PC ; off_211DF0
__text:00009790                LDR             R0, [R0] ; byte_26CBF0
__text:00009792                LDRB            R0, [R0]
__text:00009794                CMP             R0, #1
__text:00009796                BNE.W           loc_9996
__text:0000979A                MOV             R0, #(cfstr_IgnoringMenuBu - 0x97A6) ; "Ignoring menu button events"
__text:000097A2                ADD             R0, PC  ; "Ignoring menu button events"
__text:000097A4                BL              sub_1A6BB4
__text:000097A8                B               loc_9996
```

图4-45 IDA中的CMP R0

GDB的命令还有很多种，这里只列举目前笔者最常用的五种，希望读者能够窥一斑而见全豹，感受到GDB的强大功能（更多功能可参考http://www.arsc.edu/files/arsc/gdb-refcard.pdf，网上也能搜索到很多GDB教程，推荐Peter's GDB tutorial和RMS's gdb Debugger Tutorial）。IDA宜静，GDB宜动，熟练地使用这两个工具是你成为逆向高手的必经之路！

4.5 Cycript

Cycript是由saurik推出的一款脚本语言，可以看做是Objective-JavaScript。很多读者可能对JavaScript并不了解，因此会在潜意识里认为Cycript很晦涩。笔者也不懂JavaScript，所以在知道Cycript的很长一段时间里有意地忽略了它（主要还是因为懒！），直到有一次在公司的无聊会议中把玩MobileTerminal，在Cycript中完成了对几个函数的功能测试，节省了不少时间，才重新认识这门语法简单而功能强大的语言。其实对于程序员来说，脚本语言不难上手，只要克服自己的定式思维，就一定能快速掌握它，Cycript当然也不例外。Cycript具备脚本语言的便利，可以直接用来写App，但saurik自己都说，“This isn't quite ready for primetime”；笔者认为，Cycript最为贴心和实用的功能是它可以帮助我们轻松测试函数效果，整个过程安全无副作用，效果十分显著，实乃业界良心！因此，本书只对此功能作简单介绍，更多详细资料可参阅http://www.cycript.org/、http://iphonedevwiki.net/index.php/Cycript以及http://iphonedevwiki.net/index.php/Cycript_Tricks。



图4-46 Cycript安装详情

Cycript可从Cydia自带源Cydia/Telesphoreo下载，如图4-46所示。

安装完毕，可通过在MobileTerminal中执行或ssh登录到iOS设备上，输入cycript，出现“cy#”提示符，说明已成功启动Cycript，如图4-47所示。

```
Touch-1:~ root# cycript
cy#
```

图4-47 运行Cycript

在启动 Cycript 之后，就可以开始编写 App 了。因为我们主要用它测试函数的功能，而不是用它来写 App，所以需要把代码注入到一个现成的进程中，使代码能运行起来。按下“control + D”组合键，先退出 Cycript。一般来说，选择注入哪个进程，要依测试的具体函数而定——这个函数所属的类存在于哪些进程，则注入这些进程，从而保证调用函数的对象是存在的。这句话的含义有些难以理解，举例说明如下。

假如现在要测试 PhoneApplication 类的 + sharedNumberFormatter 函数功能及其返回值，则必须注入 MobilePhone 这个进程，因为 PhoneApplication 类只存在于 MobilePhone 进程中；同理，如果要测试 SBUIController 类的 - lockFromSource: 函数功能，则必须注入 SpringBoard 这个进程；当然，如果我们要测试 NSString 类的 - length 函数功能及其返回，则可注入任一进程。因为需要用 Cycript 测试的一般都是私有函数，所以一个总的原则是，你从哪个进程逆向出的函数，就注入这个进程来测试；你从哪个库逆向出的函数，就注入调用这个库的进程来测试。

通过进程注入方式调用 Cycript 测试函数的步骤很简单，以 SpringBoard 为例，首先找到进程名或 PID，如图 4-48 所示。

```
-FunMaker-5:~ root# ps ax | grep SpringBoard
15127   ??  Ss     0:16.74 /System/Library/CoreServices/SpringBoard.app/SpringBo
ard
16236 s000  R+     0:00.01 grep SpringBoard
-FunMaker-5:~ root#
```

图 4-48 查找 SpringBoard 进程

SpringBoard 进程的 PID 是 15127。接下来输入“cycript -p 15127”或“cycript -p SpringBoard”，把 Cycript 钩到 SpringBoard 上，这时，Cycript 就已经运行在 SpringBoard 进程里，可以开始测试了。

我们都知道，UIAlertView 是 iOS 中使用最多的弹框类。使用 Objective-C 语言，弹出一个对话框只需要 3 行代码：

```
UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"iOSRE"
message:@"snakeninny" delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil];
[alertView show];
[alertView release];
```

Cycript 是脚本语言，所有对象都声明为 var，其他调用跟 Objective-C 语言比较类似，用几次就可以学会，容易上手。将上面的 Objective-C 语言转化成 Cycript 非常简单，如图 4-49 所示。

```
-FunMaker-5:~ root# cycript -p SpringBoard
cy# var alertView = [[UIAlertView alloc] initWithTitle:@"iOSRE" message:@"snaken
inny" delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil]
@"<UIAlertView: 0x1fa54fc0; frame = (0 0; 0 0); opaque = NO; layer = <CALayer: 0
x1faec930>>"
cy# [alertView show]
cy# [alertView release]
cy#
```

图 4-49 用 Cycript 执行 Obj-C 代码

注意，由于 Cycript 的语句不以引号作为结尾，因此每次只能输入一条语句；其他没什么好交代的，相信大家一看就会了。执行上面的语句后，SpringBoard 会弹出对话框，如图 4-50 所示。



图 4-50 Cycript 执行代码的效果

测试私有函数的方法与以上描述别无二致，下面以登录 iMessage 的 Apple ID 为例，用 Cycript 测试并实现这个功能，如图 4-51 所示。

以上代码的作用如下。

第一行代码获取掌管 iMessage 账号的 CNFRegController 实例，Cycript 的输出表示我们成功获取到了一个 CNFRegController 实例。

第二行代码登录 Apple ID，通常情况下，如果正常登录，会显示如图 4-52 所示的界面，而此时的登录代码会执行登录动作但不出现这个界面。

第三行代码选择收发 iMessage 的可用地址，即按图 4-53 中的 next 按钮之后的步骤，Cycript 的输出表示选择成功。

```
-FunMaker-5:~ root# cycript -p SpringBoard
cy# var controller = [objc_getClass("CNFRegController") controllerForServiceTyp
e:1]
@"<CNFRegController: 0x1ff799b0>"
cy# var account = [controller beginAccountSetupWithLogin:@"snakeninny@gmail.com
" password:@"              " foundExisting:NO]
@"IMAccount: 0x1e54d310 [ID: 26B3EC90-7838-4DEC-82CF-F58FBBB22363 Service: IMSe
rvice[iMessage] Login: E:snakeninny@gmail.com Active: YES LoginStatus: Offline]
"
cy# [controller setAliases:[NSArray arrayWithObject:@"snakeninny@gmail.com"] on
Account:account]
1
cy# [account CNFRegSignInComplete]
1
cy#
```

图 4-51 用 cycript 来登录 iMessage

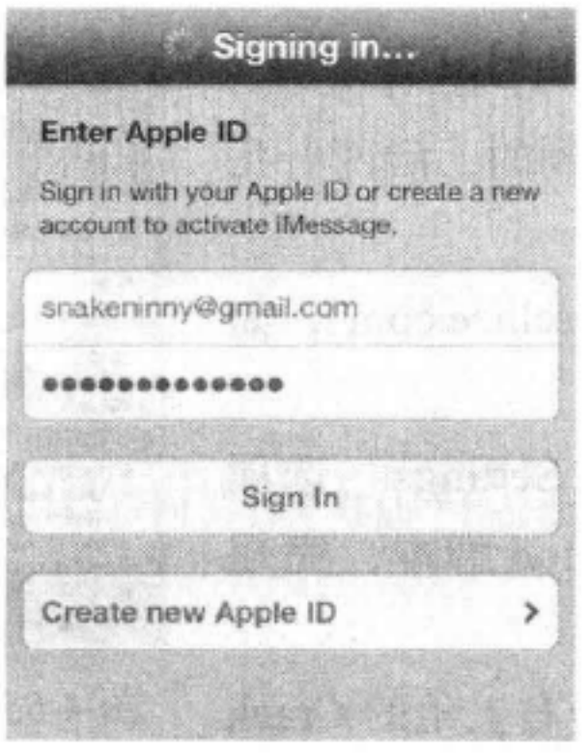


图 4-52 Cycript 操作对应的实际操作 1

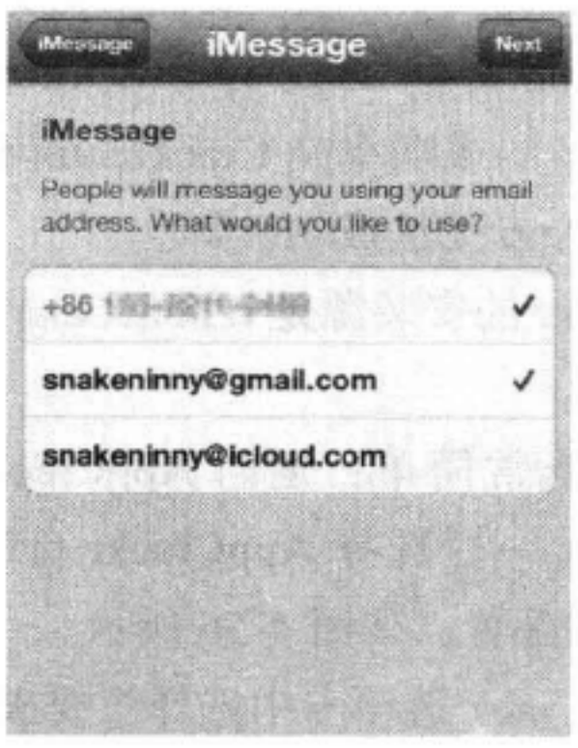


图 4-53 Cycript 操作对应的实际操作 2

第四行代码用于检查此账号是否完成了登录过程，Cycript 的输出表示登录成功。

很简单吧？不用我再解释什么了吧？下面由你把刚才登录 iMessage 的 Cycript 语言翻译成 Objective-C 语言，并编写一个 tweak 来验证你的翻译是否正确，作为本节的练习，好好体会一下 Cycript 的用法。注意，Apple ID 的用户名和密码要改成你自己的哦！

4.6 其他常用工具

前 5 节集中介绍了 SBSettings、MobileSubstrate、OpenSSH、GDB、Cycript 这 5 个工具，它们构成了 iOS 越狱平台的基础。下面还会介绍一些很重要但不是必须掌握的工具。相信这些工具会满意读者的一些好奇心。

4.6.1 BigBoss Recommended Tools

BigBoss 是越狱 iOS 的主力推手之一，BigBoss 源是 Cydia 的内置源，足以说明 BigBoss 在 iOS 界的地位。BigBoss Recommended Tools 是“一系列黑客工具”，能够“让命令行变得有用”，既然是由著名黑客“大老板”推荐的，我们就应该捧个场。来看看它的安装方法，在 Cydia 里搜索“BigBoss Recommended Tools”，然后安装即可，如图 4-54 所示。



图 4-54 BigBoss 推荐工具的安装详情

4.6.2 AppCrackr

前面在介绍 class-dump 时提到过，从 AppStore 获取的 IPA 文件无法直接通过 class-dump 获取 .h 文件，需要用工具对 IPA 做去壳的动作。不错，这种情况下就需要 AppCrackr 上场了。

AppCrackr 是一款用来对 App 破解去壳的工具，功能非常强大，在笔者使用的过程中，几乎能破解绝大多数指定的 App。

与它功能类似的还有 Crackulous、Clutch，不过相对来说，AppCrackr 的成功率是 Crackulous 和 Clutch 无法达到的，后两者几乎有一半的 App 无法成功破解。

AppCrackr 的安装源是 iPhoneCake 源（http://cydia.xsellize.com），如图 4-55 所示。



图 4-55 AppCrackr 安装详情

其使用非常简单：运行 AppCrackr，首先单击“Settings”选项卡，在 Script 一栏选择 AppCrackr 自带的 poedCrackMod 脚本，完成 AppCrackr 的配置，如图 4-56 所示。

然后在 Crack 选项卡中选择想要破解的 App，点击右上角的 Crack 即可开始破解，屏幕下方会显示破解进度，如图 4-57 所示。

在Files选项卡中可以查看所有已成功破解的App（存放在/var/mobile/Documents/Cracked下），如图4-58所示。



图4-56 AppCrackr的设置页面

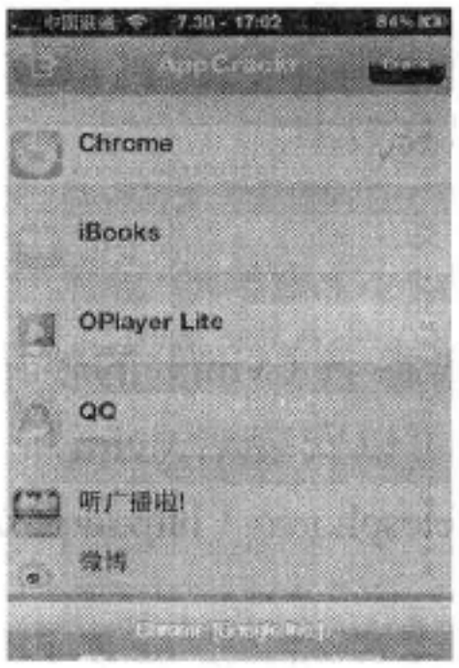


图4-57 AppCrackr开始破解

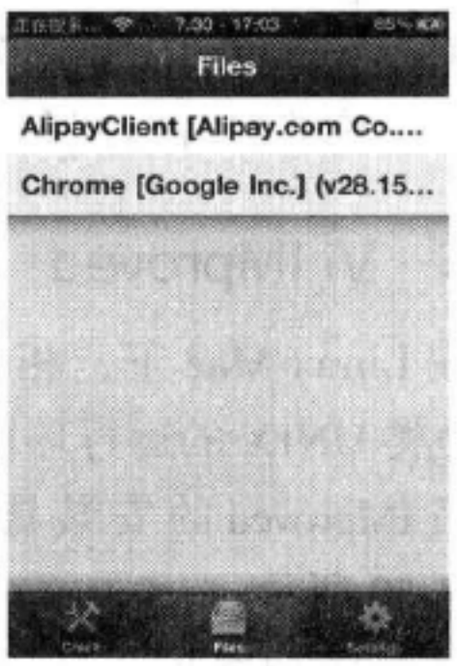


图4-58 破解成功的App列表

4.6.3 iFile

iFile是运行于iOS设备上的一款非常强大的文件管理软件，它能实现对文件的各类操作，从最简单的浏览文件夹，到基础的编辑、剪贴、复制，到强大的属性设置，还有内置的播放器自动关联相关文件，最主要的是：可以安装deb插件！真是一款十分称手的工具软件！

iFile的安装源是BigBoss（http://apt.thebigboss.org/repofiles/cydia/），如图4-59所示。

安装之后运行，即可浏览、查看iOS设备上的所有文件，并且还可以安装deb文件，如图4-60所示。



图4-59 iFile的安装详情



图4-60 iFile浏览文件、安装deb程序

4.6.4 MobileTerminal

MobileTerminal是开源的iOS版Terminal，基本功能一应俱全，可以从BigBoss源下载，

如图 4-61 所示。

其具体用法与 Mac 版 Terminal 区别不大，这个命令行工具已包含在 BigBoss Recommended Tools 里。笔者认为，MobileTerminal 最实用的场景是在没有计算机的环境下结合 Cycript 进行代码测试，它可帮助你将碎片时间利用起来，提高效率。

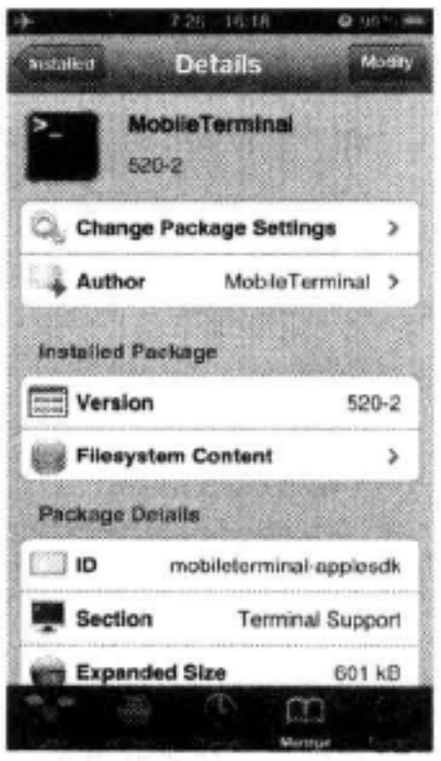


图 4-61 MobileTerminal 的安装详情

4.6.5 Vi IMproved

在 Linux/Mac 下，相信大家都熟悉 vi（vim）的使用方法，而 iOS 是类 UNIX 的运行环境，所以也可以安装 vi（vim）。

Vi IMproved 的安装源是 Cydia/Telesphoreo（http://apt.saurik.com），如图 4-62 所示。

安装成功之后，可以通过 SSH 登录到 iOS 设备上，运行 vim 即可查看、编辑文件，如图 4-63 所示。



图 4-62 Vi IMProved 的安装详情

```
Sams-iPhone:~ root#
Sams-iPhone:~ root#
Sams-iPhone:~ root#
Sams-iPhone:~ root#
Sams-iPhone:~ root# vim /etc/clutch.conf
Sams-iPhone:~ root#
Sams-iPhone:~ root#
Sams-iPhone:~ root#
Sams-iPhone:~ root#
```

图 4-63 运行 vim

4.6.6 SQLite

SQLite 是运行于 iOS 设备上的 SQLite 数据库工具，安装之后，即可打开对应的 SQLite 数据库文件。有时我们会把数据文件通过 iFile 等方式复制到 Mac 平台上，因为这样会看得更清晰；但有的时候也需要直接对 iOS 设备上的数据文件进行查看、更新，此时 SQLite 命

令就非常重要了。

SQLite 的安装源是 Cydia/Telesphoreo（http://apt.saurik.com），如图 4-64 所示。

图 4-65 是在 iOS 设备上运行 SQLite 的示例。



图 4-64 SQLite 的安装详情

```
Sams-iPhone:/var/mobile/Applications/226704B1-74A4-486E-93AD-75A4E252EFB0/
Documents root# /usr/bin/sqlite3 monitor.db
SQLite version 3.7.7
Enter ".help" for instructions
sqlite> .tables
command  momo     setting  trace    weixin
sqlite> select * from trace;
1|2013-06-26 00:05:28|Trace setLevel 4|4|
2|2013-06-26 00:05:28|---------- Start Instagram ---------- <AppDelegate:
0x46e270>, 4|4|
```

图 4-65 运行 SQLite

4.6.7 top

top 是运行于 iOS 平台上的查看当前内存使用情况的工具，使用起来和传统 Unix 平台上的 top 一样。

top 的安装源是 Cydia/Telesphoreo（http://apt.saurik.com），如图 4-66 所示。

top 只是用来看内存使用情况的一个工具，有时被用来分辨 App 是否启动，占用内存多少。但是 Top 也存在另一个使用场景：我们虽然开发了 App，但多数时候从来不去注意它们在计算机上的内存占用情况，以至于不少 App 轻易达到 100MB 以上。而绝大多数时候，这些内存都是可以优化的。如图 4-67 所示，这个内存占用达到 102MB 的是一个电商 App，如果是程序优化得很好的情况下，它的内存占用应该控制在 60MB 左右。



图 4-66 top 的安装详情

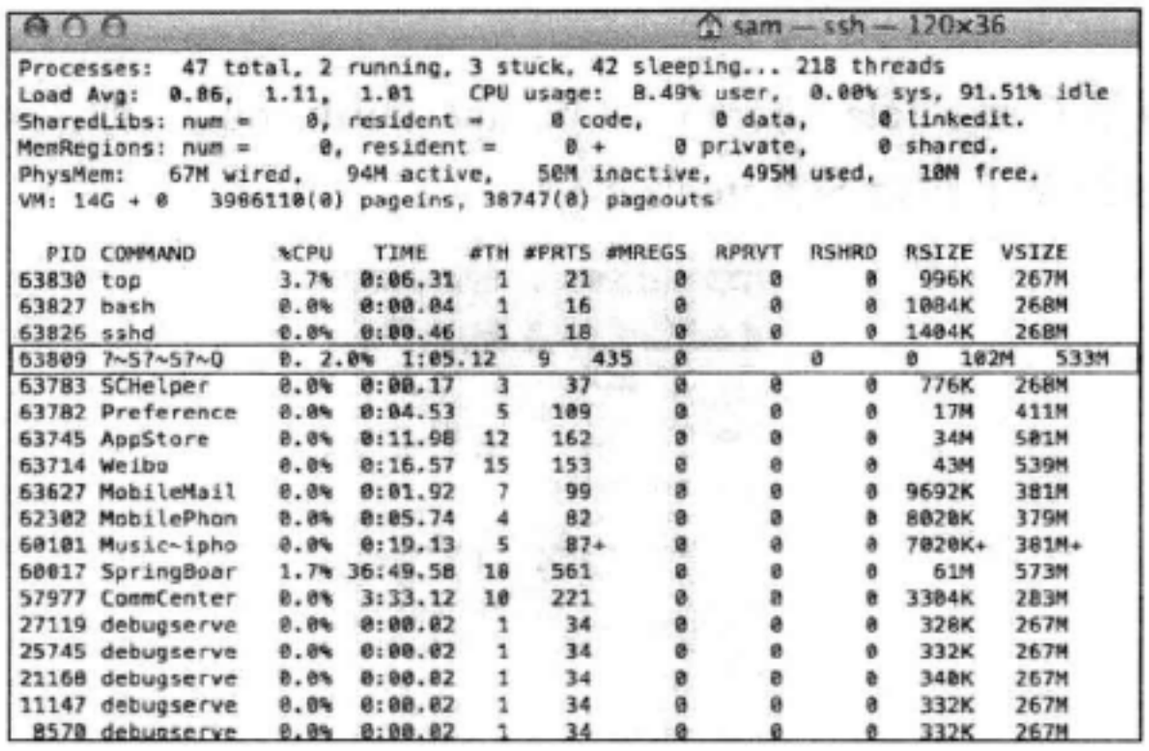


图 4-67 运行 top 查看内存情况

4.6.8 syslogd

syslogd 是用于记录类 UNIX 中系统日志的守护进程，我们可以通过 /var/log/syslog/ 看到对应的 log 输出。

syslogd 的安装源是 Cydia/Telesphoreo（http://apt.saurik.com），如图 4-68 所示。



图 4-68 syslogd 的安装详情

4.7 小结

本章重点介绍了 5 个工具：SBSettings、MobileSubstrate、OpenSSH、GDB、iFile，尤其是 OpenSSH、GDB、iFile 这 3 个工具，是配合 Mac 对 iOS 设备进行管理、调试的必不可少的利器。此外，还介绍了 AppCrackr、Cycript 等在兵器谱上都能排得上名的独门“暗器”。

正是由于这些 Mac 和 iOS 平台的工具的存在，才构成一个完整的 iOS 逆向工程兵器谱。好好准备，然后就可以披挂上阵了！

第三部分

理 论 篇

当我们了解了 iOS 应用逆向工程的基本概念，并亲手把玩过一些逆向工程必备的工具之后，就可以尝试自己动手、逆向工程了。而当你跟着书本实践过一两个 App 之后，接下来可能会有一种无从下手的感觉。是的，逆向工程是一门需要动手的学问，从哪里动手、该怎么动手，其实也是有着相应的方法的。第 5 章和第 6 章分别尝试从高层和底层语言出发，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。

so，这就开始吧！

第 5 章

Objective-C 相关的 iOS 逆向理论基础

Objective-C 语言是一门面向对象的高级语言，本书的读者一般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 开发者进阶为逆向工程师。幸运地是，iOS 采用的文件格式 Mach-O 中包含了足够多的 metadata，让我们能够用 class-dump 等工具还原类的 @interface 部分；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 的作用原理

在第 3 章中介绍 Theos 时，已经引入了 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可执行程序功能的动态链接库，是越狱 iOS 的重要组成部分。



图 5-1 Cydia 中的 tweak 图标

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 App Store 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、KuaiDial、Velox 等。纵观 Cydia，其中 99% 的 tweak 都是基于 MobileSubstrate 编写的，那么 MobileSubstrate 是如何工作的呢？

5.1.1 Objective-C 语言的特性

Objective-C 是一门典型的面向对象语言。iOS 系统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 桌面上的每个图标、解锁滑块和提醒横幅都是对象；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 Apple 的服务器通信，有的对象负责读写文件。另外，一个对象还可以拥有其他对象，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，程序员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”。如果你对 Objective-C 语言有所了解，就会发现，object、function 和 method 的关系原来如此简单。

对象具备了功能，程序员就可以发出指令“[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，可能有读者会说，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？你说得没错，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 语言里，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，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台计算机抬到楼下”。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”也不知道该通过什么来实现方法；“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，“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字典，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”。

笔者的另一个短信管理工具 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，而这是因为笔者更改了 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 MobileSubstrate 进行 hook 操作来实现的。可见，为了更好地进行逆向工程，有必要



图 5-2　LowPowerBanner



图 5-3　SMSNinja

了解 MobileSubstrate 的相关知识。

MobileSubstrate 是一个能够让 iOS 开发者方便地进行 hook 操作的 framework，它由 iOS 界最有名望的顶级高手之一 Jay Freeman（@saurik）开发，是越狱 iOS 的奠基石之一。

笔者虽与 saurik 没有接触过，但笔者的另一位益友良师 Ryan Petrich（@rpetrich）则视 saurik 为榜样，因为他认为 saurik 的技术造诣很高。笔者对 Ryan 就比较了解了，他是加拿大人，7 岁开始编程，现在 25 岁，标准宅男一个，几乎所有的时间都花在 iOS 上了，研究非常深入，他说自己卖软件赚的钱“足够过上第一世界国家的中上层阶级生活”，但他对物质要求着实不高，很多钱都花在买 iOS 设备上了（他拥有每一款 iOS 设备）。

笔者对 Ryan 的技术水平造诣之深可以说佩服得五体投地，因为他从来没有被什么问题难倒过，似乎什么都懂；而他视 saurik 为更高的目标，可见 saurik 的技术功力有多么高深莫测。除了技术牛，saurik 的责任心也很强：在 2011 年年中，有阵子 Cydia 在 Amazon 的服务器因为停电宕机了，saurik 两天时间连续作战，睡了不到两小时，抢修 Cydia。当服务器恢复正常，他在 twitter 上说终于能“ sleep > 1.5 hrs”时，笔者仿佛看见了现代人身上久违的责任感，遂也下定决心，潜心钻研，努力工作，做个对社会有用的人……

5.1.2 MobileSubstrate

MobileSubstrate 由 3 个部分组成，分别是 MobileHooker、MobileLoader 和 Safe mode。

- MobileHooker

MobileHooker 的作用是替换系统函数，也就是所谓的 hook 技术，它主要包含两个函数：

```
IMP MSHookMessage(Class class, SEL selector, IMP replacement,
const char* prefix); // prefix should be NULL.
void MSHookMessageEx(Class class, SEL selector, IMP replacement, IMP *result);
void MSHookFunction(void* function, void* replacement, void** p_original);
```

其中，因为 MSHookMessage () 不是线程安全（thread-safe）的，所以被弃用了，改用 MSHookMessageEx()。MSHookMessageEx () 用于 Objective-C 函数，通过调用 IMP method_setImplementation(Method method, IMP imp) 来将 [class selector] 的实现改为 replacement。

MSHookFunction() 用于 C/C++ 函数，通过编写汇编指令，在程序执行到被 hook 的位置时跳转到 replacement 处，并分配一块内存来保存被 hook 而没有得到执行的指令和其返回地址，从而保证程序能够在执行完 replacement 后继续运行。

值得一提的是，第 3 章中提到的 Theos 只是对 MobileHooker 作了一层封装，增强了它的易用性，其底层实现仍完全基于 MobileHooker。

下面是 MobileHooker 的示例，读者仅需了解即可，推荐使用 Theos 配套的更加简洁的 Logos 语法。

```
// Hook Objective-C method
NSString *(*oldDescription)(id self, SEL _cmd);

// implicit self and _cmd are explicit with IMP ABI
```

```
NSString *newDescription(id self, SEL _cmd) {
    NSString *description = (*oldDescription)(self, _cmd);
    description = [description stringByAppendingString:@"!"];
    return description;
}

MSHookMessageEx(
    [NSObject class], @selector(description),
    &newDescription, &oldDescription
);

// Hook C/C++ function
static const uint32_t canary = 0xdeadbeef;

void *(*oldMalloc)(size_t);

void *newMalloc(size_t size) {
    void *data = (*oldMalloc)(size + 4);
    * (uint32_t *) ((char *) data + size) = canary;
    return data;
}

MSHookFunction(&malloc, &newMalloc, &oldMalloc);
```

❑ MobileLoader

MobileLoader的作用是加载第三方动态链接库，也就是tweak。在iOS启动时，由launchd将MobileLoader载入内存，然后MobileLoader会dlopen所有/Library/MobileSubstrate/DynamicLibraries/目录下的动态链接库；此外，还应该编写一个与自己的动态链接库名字相同的配置文件，以plist作为扩展名，指定动态链接库的作用范围。这个配置文件的格式已在Theos部分详细讲解，此处不再赘述。

❑ Safe mode

iOS程序员的水平参差不齐，程序崩溃在所难免。但因tweak是动态链接库，寄生在别的可执行程序里，一旦出错，可能会导致整个程序崩溃，所以MobileSubstrate引入了Safe mode，它会捕获SIGTRAP、SIGABRT、SIGILL、SIGBUS、SIGSEGV、SIGSYS这6种信号，然后进入安全模式；在安全模式里，所有第三方插件均被禁用，便于用户查错并修复程序。

5.2 tweak的编写套路

只有理解了tweak的工作原理，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，该从何下手，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，而且随着你对系统的了解愈加深入，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程序员使用最多的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 灵感的来源

可能部分 iOS 程序员读到这里时，已经能够结合前几章的知识开始 tweak 的开发了，但是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？下面将结合 @DHowett、@rpetrich 给笔者的建议，以及笔者自己的亲身经历来说说如何获取灵感，一般情况下，有如下方式。

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，别光顾着刷微博。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，笔者经常把长短信的内容存成记事本，但发短信是有字数限制的，于是就做了一个这样的 tweak，用来统计记事本每页的字数，从而控制短信的长度。（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。）

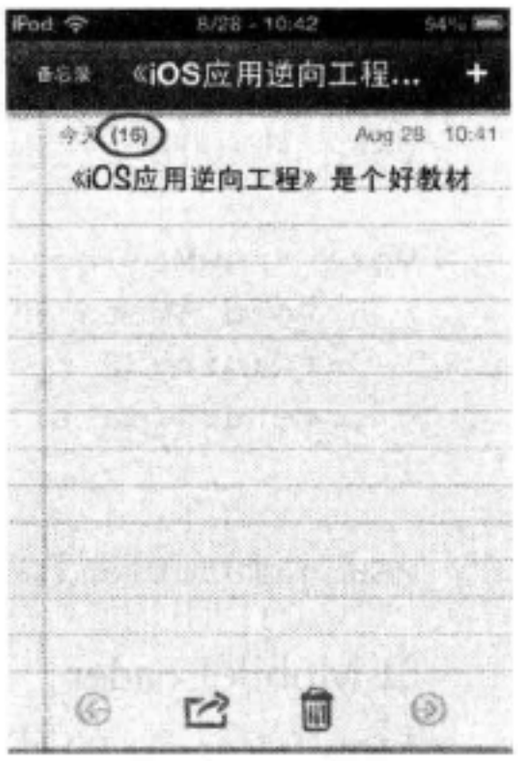


图 5-4 Characount for Note

2. 倾听用户的声音

每个人使用 iOS 的方式都不同，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，你的付出不会没有回报。而 LowPowerBanner 这个软件就是笔者听取用户 PrimeCode 的建议编写的，完成第 1 版仅用时约 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！

Downloads for **snakeninny**

App	Total Count	Installer	Cydia
SMSNinja (v1.2)	22783	0	22783
LowPowerBanner (v0.0.1-42)	35486	0	35486

图 5-5 LowPowerBanner 第 1 版下载量

3. 解剖 iOS

当你的能力越大时，你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。据笔者所知，rpetrich 所有的业余时间都用来浏览 class-dump 出的函数名，然后用 IDA 等工具分析自己感兴趣的功能，把它们的来龙去脉

理顺；DHowett 则稍有些不同，除了“ do random hacks”，他还偶尔恋恋爱、分分手，生活比 rpetrich 稍微丰富一点。

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 已经有 7 年的历史了，但通话录音的功能直到今年才由这位希腊开发者实现。有这个想法的人肯定不在少数，但为什么只有 limneos 成功了？因为他对 iOS 的解剖比别人更彻底！说起来就是这么简单。

5.2.2　分析文件，寻找切入点

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四种。

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 MobileSubstrate 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 App Store App 全部位于 /var/mobile/Applications 下，framework 全部位于 /System/Library/Frameworks 或 /System/Library/PrivateFrameworks 下；其他类型的 App，如 PreferenceBundle 和 WeeApp 的位置可通过 Cydia 记录定位，下面会有详细介绍。
- iOS 中每个 daemon 的作用只要在网上稍加搜索即可查到[⊖]，如图 5-6 所示。

一般后台进程的启动程序都存放在"/System/Library/LaunchDaemon"目录下，就是那些".plist"文件，要不启动这些进程的最简单的方法就是删除相应的".plist"文件、但最好在删除之前先要备份，以防万一。

1、com.apple.DumpPanic.plist这是苹果公司用来评估系统崩溃的，完全没必要运行。

2、com.apple.DumpBasebandCrash.plist这是苹果公司用来苹果基带崩溃的，也没必要运行。

3、ReportCrash有5个带"ReportCrash"的".plist"文件，都是用来收集系统或系统程序崩溃的信息，没什么用处。

4、com.apple.CrashHouseKeeping.plist也是和程序崩溃相关的，可以删除。

5、com.apple.powerlog.plist用来监测第三方不兼容的充电器，可以删。

6、com.apple.tcpdump.server.plist似乎是用来转存网络数据的，具体不是很清楚，但删除后不影响设备。

7、com.apple.chud.chum.plist这个进程应该和苹果的CHUD工具相关(计算机硬件协议开发/ComputerHardware Understanding Developer)，如果你不是一个软件开发者，应该不用启动这个进程。

8、com.apple.chud.pilotfish.plist着也是和苹果CHUD工具相关的一个进程，如果你不开发苹果软件，可以删除这个。要稍微谨慎一点处理的进程，有些用户可以禁用以下几个进程。

以下进程可选择性删除：

1、com.apple.AddressBook.plist如果不启动这个进程，那么电话中的通讯录的载入速度会稍微变慢一些，若不介意这一点，则可以删除这个进程。

2、com.apple.accessoryd.plist不启动这个进程就会禁用一些辅助设备功能，例如: FM收音机、iPhone座充、AV数据线。当然，即便禁用了这个进程，用iPhone座充还是能充，但也只能有充电的功能

图 5-6　daemon 功能的大致介绍

⊖ http://bbs.weiphone.com/read-htm-tid-5250171.html。

要定位这个daemon的可执行程序，可以根据plist的文件名，在/System/Library/LaunchDaemons、/Library/LaunchDaemons或/Library/LaunchAgents中找到这个文件，打开后查看其“ProgramArguments”字段，即是其可执行文件的绝对路径，如图5-7所示。

Key	Type	Value
▼ Root	Dictionary	(9 items)
UserName	String	mobile
HighPriorityIO	Boolean	YES
POSIXSpawnType	String	Interactive
▶ MachServices	Dictionary	(11 items)
EmbeddedPrivilegeDispensation	Boolean	YES
RunAtLoad	Boolean	YES
KeepAlive	Boolean	YES
▼ ProgramArguments	Array	(1 item)
Item 0	String	/usr/libexec/backboardd
Label	String	com.apple.backboardd

图5-7 daemon的配置文件

2.Cydia记录

通过dpkg命令安装的软件，其软件包里的内容会被Cydia如实记录，查看方法是在Cydia的“管理”项中选择“软件包”，如图5-8所示。

在“已安装”界面中选择目标软件，进入软件的“详情”界面，如图5-9所示。

然后选择“文件系统”，即可浏览软件包里的所有文件，如图5-10所示。

已安装文件用树形结构显示，一目了然。

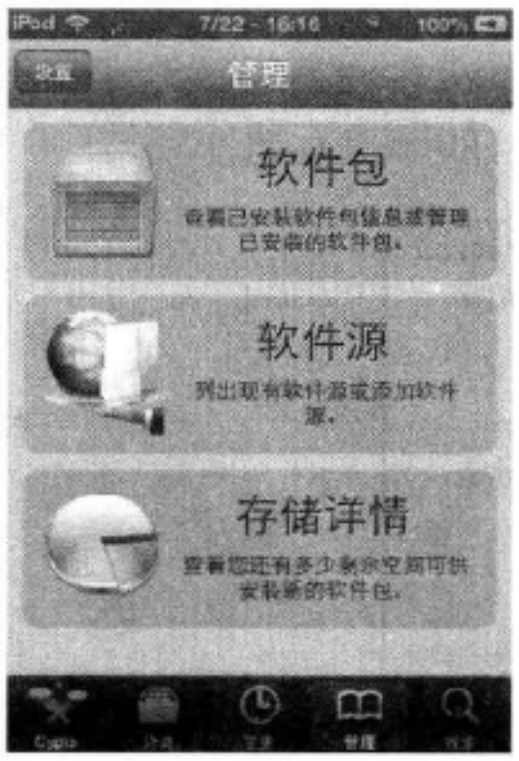


图5-8 Cydia管理界面



图5-9 Cydia“详情”界面

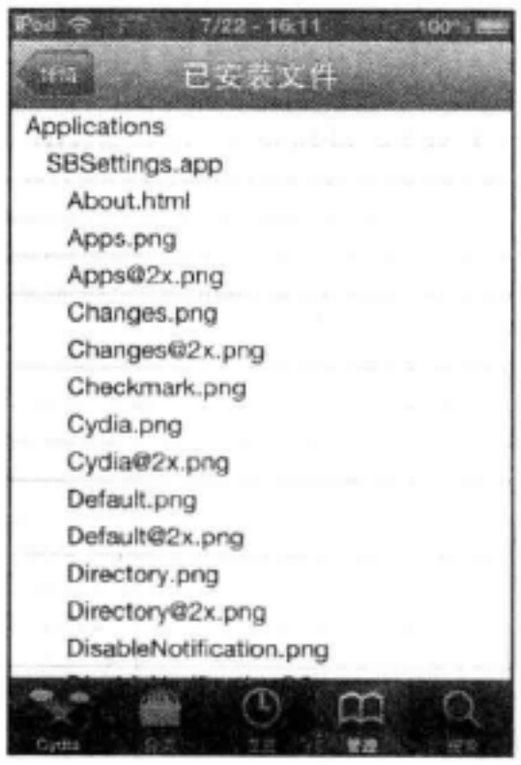


图5-10 Cydia已安装文件界面

3. Preference specifier plist

对于PreferenceBundle和WeeApp（如图5-11所示）来说，它们分别是寄生在“设置”程序和“通知中心”里的，界面可以由具有一定格式的plist文件构造（具体可参考http://iphonedevwiki.net/index.php/Preferences_specifier_plist，此处不再赘述），可执行文件中仅需保存执行实际功能的代码。

当我们在逆向此类程序时，如果程序界面中的元素完全来自preference specifier plist中

的标准组件，如 KuaiDial 设置界面（如图 5-12 所示）则需注意分辨此软件的界面是否由 plist 绘制而成，以及界面 plist 和可执行程序间的关系，从中找到切入点。可能会有读者没有碰到过这种情况，对前面的介绍一知半解，不要紧，稍后本章会以一个实例来讲解这种情况的应对方法。

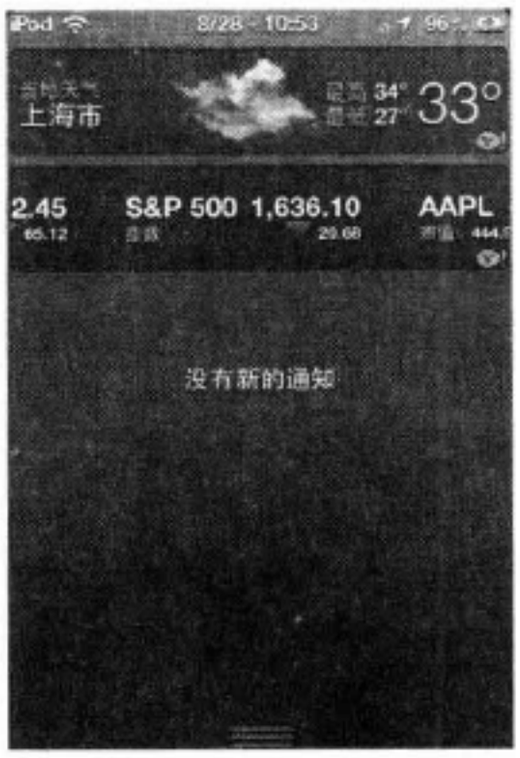


图 5-11　WeeApp 示例



图 5-12　KuaiDial 设置界面

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。这个命令在寻找一个可执行程序和 framework 之间的关系时尤其适用——当我们需要了解某个类被哪些程序引用，或定位一个 framework 中某个函数的调用者时，grep 能够快速缩小查找范围。例如，想知道某个 IMDAccount 实例在哪里初始化，即其 initWithAccountID:defaults:service: 函数是被谁调用时，可以先 ssh 到 iOS，然后在终端中输入“ grep -r initWithAccountID:defaults:service: /System/Library”命令，进行第一次筛选，如图 5-13 所示。

```
Touch-1:~ root# grep -r initWithAccountID:defaults:service: /System/Library
Binary file /System/Library/Caches/com.apple.dyld/dyld_shared_cache_armv7 match
es
grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libFontStream
s.dylib: No such file or directory
grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libJBIG2.dyli
b: No such file or directory
grep: /System/Library/Frameworks/CoreGraphics.framework/Resources/libRIP.dylib:
 No such file or directory
grep: /System/Library/Frameworks/IOKit.framework/Resources: No such file or dir
ectory
grep: /System/Library/Frameworks/System.framework/System: No such file or direc
tory
grep: /System/Library/PrivateFrameworks/GPUSupport.framework/GPUSupport: No suc
h file or directory
Binary file /System/Library/PrivateFrameworks/IMCore.framework/imagent.app/imag
ent matches
Touch-1:~ root#
```

图 5-13　grep 命令

从图 5-13 得知，要查找的函数在 dyld_shared_cache_armv7 和 imagent 中出现了。接着对

decache 过的 dyld_shared_cache_armv7 进行第二次筛选，如图 5-14 所示。

```
[redacted]-MacBook-Pro:~ snakeninny$ grep -r initWithAccountID:defaults:serv
ice: /Users/snakeninny/Code/iOSSystemBinaries/6.1.3_iPod
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.1.3_iPod/System/Li
brary/PrivateFrameworks/IMCore.framework/imagent matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.1.3_iPod/System/Li
brary/PrivateFrameworks/IMDaemonCore.framework/IMDaemonCore matches
[redacted]-MacBook-Pro:~ snakeninny$
```

图 5-14 grep 命令

可以看到，在 /System/Library 目录下，initWithAccountID:defaults:service: 函数被 imagent 和 IMDaemonCore 调用过，这时，只需对这两个二进制文件进行分析即可。

5.2.3 定位目标函数

在找到含有自己目标功能的二进制文件之后，可以通过 class-dump 导出其中的 Objective-C 函数定义，然后在这一堆头文件中寻找自己感兴趣的函数，具体做法可分为以下两种，都比较简单，读者可以自行选择。

1. Mac OS X 自带的搜索功能

不得不承认，Mac OS X 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；不论搜目录还是搜全盘，速度都非常快。利用这一便利条件，可以在大量文件中快速定位我们的目标文件，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 出头文件的文件夹，然后在右上角的搜索栏中输入 proximity（大小写不限），Finder 会询问你是搜文件名还是文件内容，如图 5-15 所示。

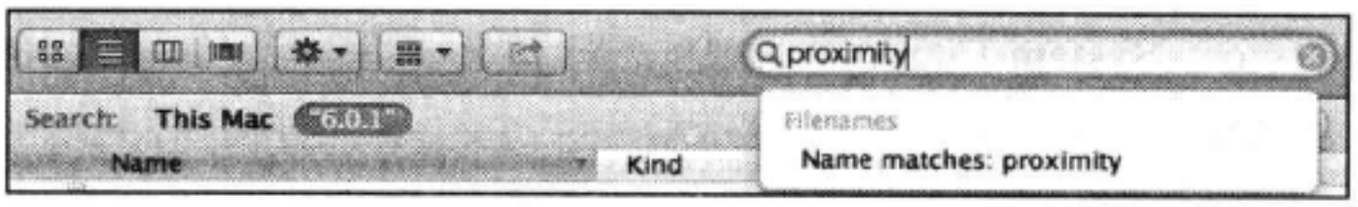


图 5-15 Finder 搜索

显然我们对文件内容更感兴趣，所以不用理会它，Finder 默认会把所有包含 proximity 字眼的头文件罗列出来，如图 5-16 所示。

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符号，当然能轻松解决文本文件里的字符串。在 Terminal 中输入“ grep proximity /Path/To/Headers”，其效果如图 5-17 所示。

grep 显示出的结果虽然全，但看起来有些乱；笔者推荐使用 Finder 的搜索功能，毕竟图形界面比命令行界面看起来舒服。众口难调，读者可根据个人喜好自行定夺。

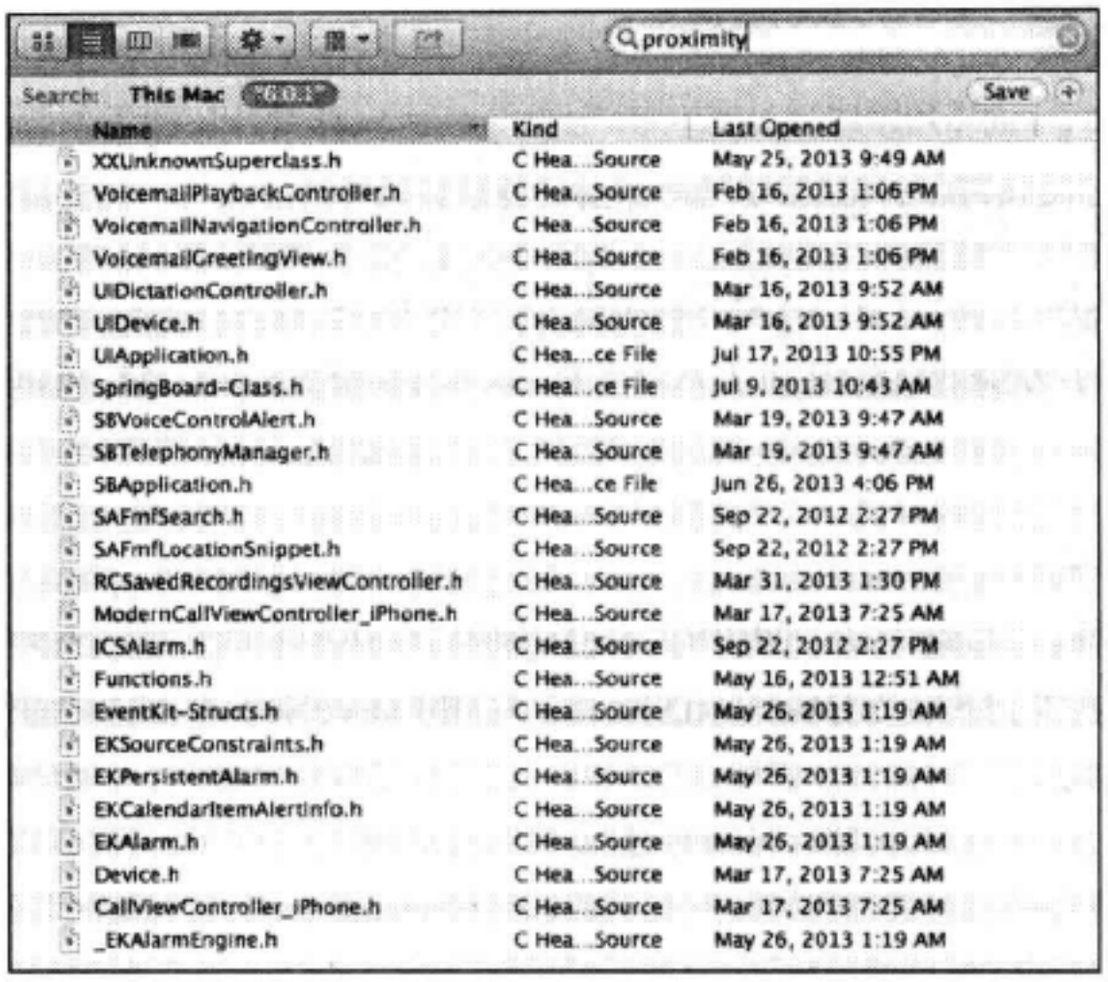


图 5-16 Finder 搜索

```
[redacted]-iMac:~ snakeninny$ grep -r proximity /Users/snakeninny/Code/iOSPrivateHe
aders/6.0.1
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/BackBoardServices/Functions.h:/*
Decide if proximity sensor is on or off. 1 for active (e.g. in a call), 0 for in
active. */
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/BackBoardServices/Functions.h:/*
Decide the mode of proximity sensor. 3 for in call, 0 for not. */
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/_EKAlarmEngine.h:-(void)
_proximityAlertTriggered:(id)triggered entered:(BOOL)entered;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKAlarm.h:@property(assi
gn, nonatomic) int proximity;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKCalendarItemAlertInfo.
h:      int _proximity;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKCalendarItemAlertInfo.
h:@property(readonly, assign, nonatomic) int proximity;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKCalendarItemAlertInfo.
h:+(id)alertInfoWithTitle:(id)title location:(id)location date:(id)date endDate:
(id)date4 timeZone:(id)zone allDay:(BOOL)day tentative:(BOOL)tentative entityID:
(id)anId action:(id)action proximity:(int)proximity externalID:(id)anId11 acknow
ledged:(BOOL)acknowledged;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKCalendarItemAlertInfo.
h:-(id)initWithTitle:(id)title location:(id)location date:(id)date endDate:(id)d
ate4 timeZone:(id)zone allDay:(BOOL)day tentative:(BOOL)tentative entityID:(id)a
nId action:(id)action proximity:(int)proximity externalID:(id)anId11 acknowledge
d:(BOOL)acknowledged;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EKPersistentAlarm.h:@pro
perty(assign, nonatomic) int proximity;
Binary file /Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/EventKit/EventKit mat
ches
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/iCalendar/ICSAlarm.h:@property(re
tain) NSString *x_apple_proximity;
Binary file /Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/MobilePhone/MobilePho
ne matches
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/MobilePhone/VoicemailPlaybackCont
roller.h:-(void)proximityStateChanged:(id)changed;
Binary file /Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/Reminders/Reminders m
atches
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/Reminders/XXUnknownSuperclass.h:-
(id)locationTitleWithProximity:(int*)proximity andGeocoded:(BOOL*)geocoded;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/SAObjects/SAFmfLocationSnippet.h:
@property(copy, nonatomic) NSString *proximity;
/Users/snakeninny/Code/iOSPrivateHeaders/6.0.1/SAObjects/SAFmfSearch.h:@property
(copy, nonatomic) NSString *proximity;
```

图 5-17 grep 搜索

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没有文档可供参考，如果运气够好，谷歌可能会帮上你的忙，但也说明你想做的东西很可能别人已经做过了；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 C/C++ 函数要简单得多，有 MobileSubstrate 和 Cycript 两种方法可供选择。

1. MobileSubstrate

在测试函数功能时，主要利用 MobileSubstrate 来 hook 住一个函数，从而判断这个函数的调用时机。例如，我们怀疑 SBScreenShotter 中的 saveScreenshot: 在截屏时得到调用，于是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”，用 Theos 编译打包成 deb，安装在 iOS 中，respring 一次；如果感觉有些生疏，不要着急，是正常的，等你把上面的步骤走完。等锁屏界面完全出现后，同时按下 home 键和 lock 键截屏，然后打开 Terminal，ssh 进 iOS，然后输入下面的指令，看看是否得到了图 5-18 所示的结果。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Jul 22 23:52:37 Touch-1 SpringBoard[861]: iOSRE: saveScreenshot: is called
Touch-1:~ root#
```

图 5-18 查看 syslog

可以看到，syslog 中出现了我们的自定义信息，说明在截屏时，saveScreenshot: 得到调用。此时，读者一定会跟我一样好奇：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？

在 iOS 的世界中，好奇不会害死猫，不好奇却会害了我们。那么，就用 Cycript 来进一步满足我们的好奇心吧！

2. Cycript

在知道 Cycript 之前，笔者测试函数功能的工具是 Theos。针对上面的例子，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，这种方法还算简单，但是当笔者用 Cycript 达到相同目的时，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，所以这里利用 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，整个编译过程完全透明，事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。”

先 ssh 到 iOS，然后输入如图 5-19 所示的三行代码，按 Enter 键。你的屏幕是不是也白光一闪，“咔嚓”一声，截屏一张？好了，现在确认这个函数可以完成截屏操作。为了进一步满足我们的好奇心和求知欲，在刚才的 Cycript 提示符下按“↑”键，重复上一次输入的命令，然后把 YES 改成 NO，再按 Enter 键，看看效果吧。

```
Touch-1:~ root# cycript -p SpringBoard
cy# var shotter = [SBScreenShotter sharedInstance]
@"<SBScreenShotter: 0x1eb6cc00>"
cy# [shotter saveScreenshot:YES]
cy#
```

图 5-19 用 Cycript 测试函数

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，但这里还是拿不准在调用时到底是传 YES 还是 NO，只能靠猜。浏览 class-dump 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！”MobileSubstrate 和 Theos 异口同声地说道。

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。刚才完成了截屏的操作，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，我们很快就能定义到可疑的 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，参数类型貌似就是 UIColor 吧？编写下面的代码，我们来满足一下自己的好奇心。

```
%hook SBScreenFlash

-(void)flashColor:(id)color;
{
    %orig;
    NSLog(@"iOSRE: flashColor: %s, %@", object_getClassName(color), color);
// [color description] 可以直接写成 color
}

%end
```

作为练习，由你把上面的代码变成一个可用的 tweak。respring 完成后，截屏、ssh、然后输入图 5-20 所示的代码，是否得到了以下结果？

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Jul 23 10:13:41 Touch-1 SpringBoard[1290]: iOSRE: flashColor: UICachedDeviceWhit
eColor, UIDeviceWhiteColorSpace 1 1
Touch-1:~ root#
```

图 5-20 查看 syslog

可以看出，color 是一个 UICachedDeviceWhiteColor 对象，它的描述是“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 头文件中利用 5.2.3 节介绍的方法找到 UICachedDeviceWhiteColor.h，如图 5-21 所示。

```
 1 /**
 2  * This header is generated by class-dump-z 0.2a.
 3  * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 4  *
 5  * Source: /System/Library/Frameworks/UIKit.framework/UIKit
 6  */
 7
 8 #import "UIKit-Structs.h"
 9 #import "UIDeviceWhiteColor.h"
10
11
12 __attribute__((visibility("hidden")))
13 @interface UICachedDeviceWhiteColor : UIDeviceWhiteColor {
14 }
15 -(unsigned)retainCount;
16 -(id)retain;
17 -(id)copyWithZone:(NSZone*)zone;
18 -(void)dealloc;
19 -(oneway void)release;
20 -(id)copy;
21 -(id)autorelease;
22 -(void)_forceDealloc;
23 -(BOOL)retainWeakReference;
24 -(BOOL)allowsWeakReference;
25 -(void)finalize;
26 @end
```

图 5-21 class-dump 出的头文件

因为 UICachedDeviceWhiteColor 继承自 UIDeviceWhiteColor，于是继续找到 UIDevice WhiteColor.h，如图 5-22 所示。

```
 8 #import "UIKit-Structs.h"
 9 #import "UIColor.h"
10
11
12 __attribute__((visibility("hidden")))
13 @interface UIDeviceWhiteColor : UIColor {
14         float whiteComponent;
15         float alphaComponent;
16         CGColorRef cachedColor;
17 }
18 -(id)description;
19 -(unsigned)hash;
20 -(BOOL)isEqual:(id)equal;
21 -(id)copyWithZone:(NSZone*)zone;
22 -(void)dealloc;
23 -(CGColorRef)_createCGColorWithAlpha:(float)alpha;
24 -(float)alphaComponent;
25 -(void)setStroke;
26 -(id)colorSpaceName;
27 -(id)colorWithAlphaComponent:(float)alphaComponent;
28 -(id)initWithWhite:(float)white alpha:(float)alpha;
29 -(void)setFill;
30 -(BOOL)getWhite:(float*)white alpha:(float*)alpha;
31 -(id)initWithCGColor:(CGColorRef)cgcolor;
32 -(void)set;
33 -(CGColorRef)CGColor;
34 @end
```

图 5-22　class-dump 出的头文件

进一步得知 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，大同小异。

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，建议读者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。在 Cycript 提示符下，输入如图 5-23 所示的命令。

```
Touch-1:~ root# cycript -p SpringBoard
cy# var flash = [SBScreenFlash sharedInstance]
@"<SBScreenFlash: 0x1f548ff0>"
cy# var color = [UIColor magentaColor]
@"UIDeviceRGBColorSpace 1 0 1 1"
cy# [flash flashColor:color]
```

图 5-23　用 Cycript 测试函数

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，我们并没有看到新截屏的生成，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：我们可以钩住这里的 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，由你独自完成。

5.2.6 class-dump 的局限性

通过分析 class-dump 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效果。对于这个函数而言，如果单看函数名而不作测试，很难猜测参数的作用，所以在 5.2.5 节里，绕到了 SBScreenFlash 类的 flashColor: 函数，转而解析了这个函数的参数。从 flashColor: 的效果来看，它应该发生在 saveScreenshot: 的内部，而我们仅根据 class-dump 的头文件，结合 MobileSubstrate，最多也只能判断出 saveScreenshot: 和 flashColor: 的调用先后关系，至于前者是否包含后者，则不得而知；虽然对 flashColor: 进行 hook 操作，传入自定义的 UIColor 对象可以成功改变闪光颜色，但是我们还想知道，原函数的 UICachedDeviceWhiteColor 是谁传进去的，即 flashColor: 的调用者是谁？按理说，完成了一个新 tweak，应该庆祝，但在高兴之余还应清楚地意识到，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持我们看清这些未知的东西，就好像我们正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 GDB。像所有超级巨星一样，它们高贵且难以接近。所以，请读者系牢安全带，准备加速喽！

5.3 实例演示

在加速之前，笔者觉得应该针对刚才讲过的理论作一次全面的实战演练，为读者提供一些缓冲，以期能够更平稳地过渡到第 6 章。同时这也是一个真实的例子，它按照 5.2 节所示的套路，完整地讲述了笔者的插件“Speaker SBSettings Toggle”（如图 5-24 所示）的开发过程。

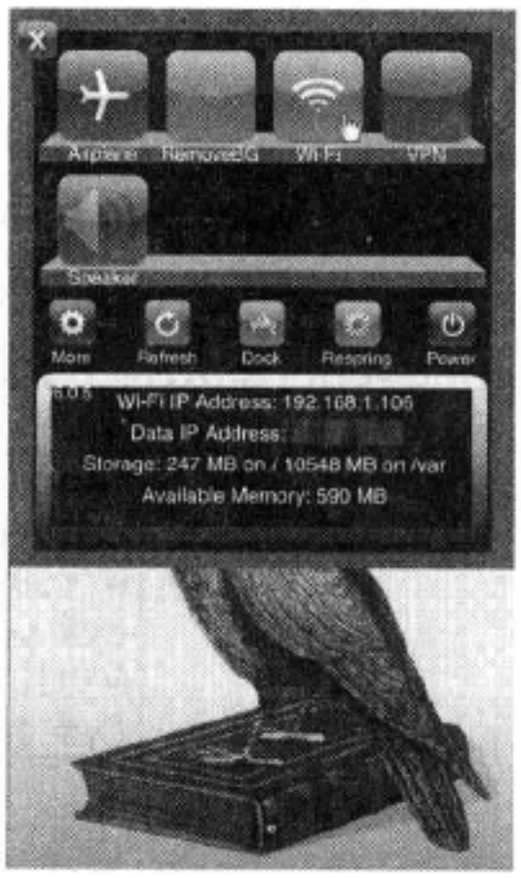


图 5-24 Speaker SBSettings Toggle

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，接听来电时可以默认打开免提，而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，设置起来非常烦琐。SBSettings 上各种各样的开关就是为了解决这类问题而存在的，所以，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。

5.3.2 分析文件

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平常使用时看到的关键词就更有可能产生对应关系。

2. 发现 Accessibility

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 关键词引起了我的注意，因为不结合语境的话，Accessibility 是不会直译成“辅助功能”的。于是我 ssh 到 iOS 中，以 Accessibility 为关键词进行了一次 grep 操作，如图 5-25 所示。

```
[redacted]-FunMaker-5:/Applications root# grep -r Accessibility /Applications/Prefe
rences.app
Binary file /Applications/Preferences.app/English.lproj/General-Simulator.string
s matches
Binary file /Applications/Preferences.app/English.lproj/General~iphone.strings m
atches
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 mat
ches
[redacted]-FunMaker-5:/Applications root#
```

图 5-25 grep 关键词 Accessibility

得到的结果不少，但最吸引我的是图 5-28 中这几个 strings 后缀的文件，如果不出所料，它们是程序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 打开 /Applications/Preferences.app/English.lproj/General~iphone.strings，如图 5-26 所示。

虽然大部分是乱码，但我们还是能够迅速定位到仅有的两处匹配——Accessibility 和 ACCESSIBILITY。对比两处关键词出现的上下文，发现 Accessibility 区分大小写，且周围出现了若干“%@”，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，回过头去看图 5-25，刚才没有深究的 /Applications/Preferences.app/General.plist 显得格外醒目。在 5.2.2 节中，特意提到了 Preference specifier plist 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，如图 5-27 所示。

4. 发现 AccessibilitySetting.bundle

哈，果不其然，这个文件就是一个标准的 Preference specifier plist，在这个文件里搜索 accessibility，刚输入 acce，Xcode 就自动帮我们展开了 item 28，如图 5-28 所示。



图 5-26 用 Xcode 查看 General~iphone.strings

Key	Type	Value
▼ Root	Dictionary	(2 items)
title	String	General
▼ items	Array	(32 items)
▶ Item 0	Dictionary	(1 item)
▶ Item 1	Dictionary	(3 items)
▶ Item 2	Dictionary	(5 items)
▶ Item 3	Dictionary	(4 items)
▶ Item 4	Dictionary	(2 items)
▶ Item 5	Dictionary	(4 items)
▶ Item 6	Dictionary	(2 items)
▶ Item 7	Dictionary	(3 items)
▶ Item 8	Dictionary	(3 items)
▶ Item 9	Dictionary	(3 items)
▶ Item 10	Dictionary	(3 items)
▶ Item 11	Dictionary	(1 item)
▶ Item 12	Dictionary	(3 items)
▶ Item 13	Dictionary	(4 items)
▶ Item 14	Dictionary	(2 items)
▶ Item 15	Dictionary	(8 items)
▶ Item 16	Dictionary	(7 items)
▶ Item 17	Dictionary	(6 items)
▶ Item 18	Dictionary	(7 items)
▶ Item 19	Dictionary	(5 items)
▶ Item 20	Dictionary	(4 items)
▶ Item 21	Dictionary	(4 items)
▶ Item 22	Dictionary	(4 items)
▶ Item 23	Dictionary	(7 items)
▶ Item 24	Dictionary	(1 item)
▶ Item 25	Dictionary	(3 items)
▶ Item 26	Dictionary	(1 item)
▶ Item 27	Dictionary	(3 items)
▶ Item 28	Dictionary	(5 items)
▶ Item 29	Dictionary	(1 item)

图 5-27 用 Xcode 查看 General.plist

可以看到，大写的 ACCESSIBILITY 映入眼帘。对比 Preferences specifier plist 格式，将目标锁定在 AccessibilitySetting 这个 bundle 中；由 AccessibilitySetting 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，AccessibilitySetting.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……

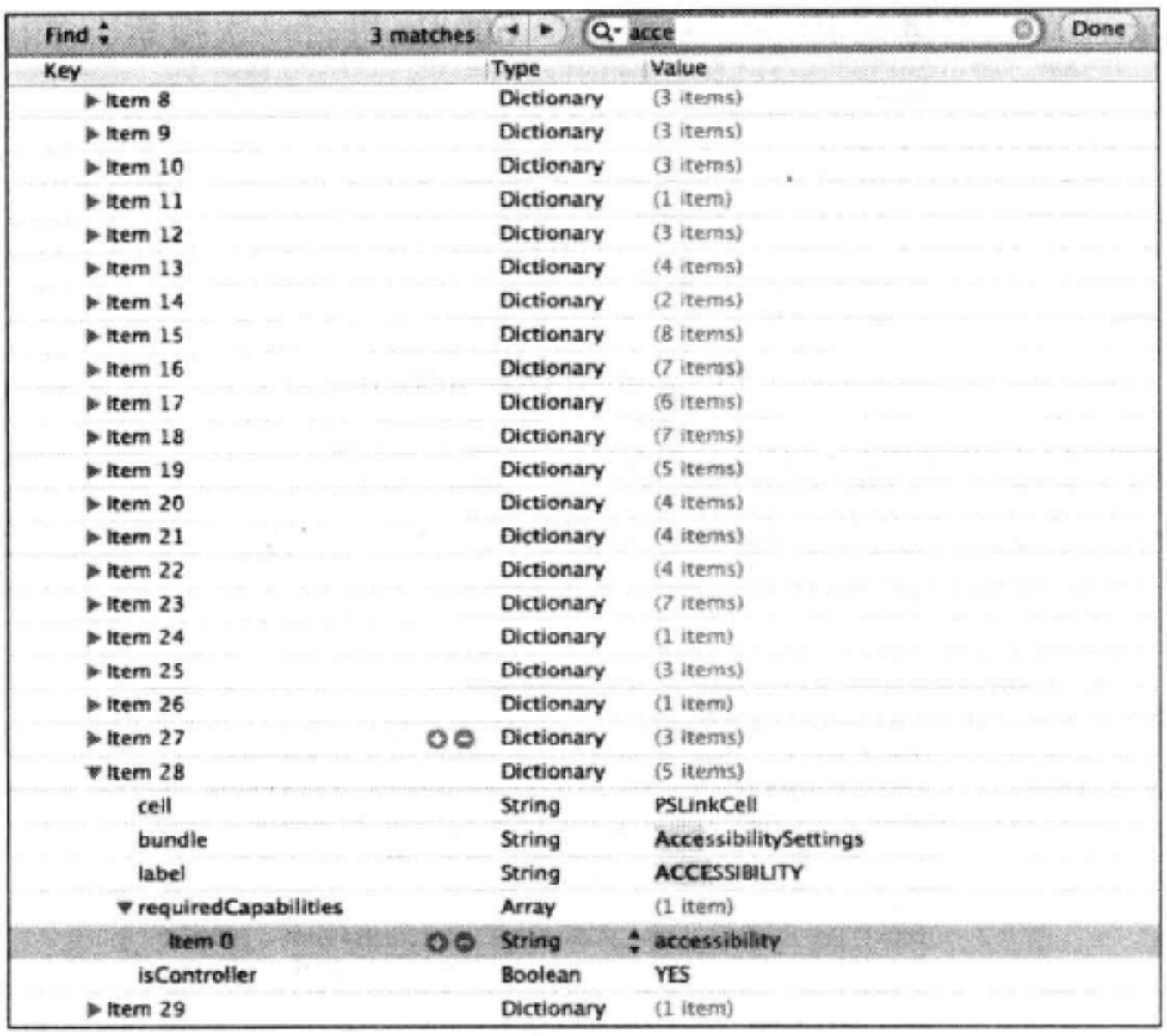


图 5-28　搜索 General.plist

在 iFunBox 中打开 /System/Library/PreferenceBundles/AccessibilitySetting.bundle，里面的文件名印证了我的猜测，如图 5-29 所示。

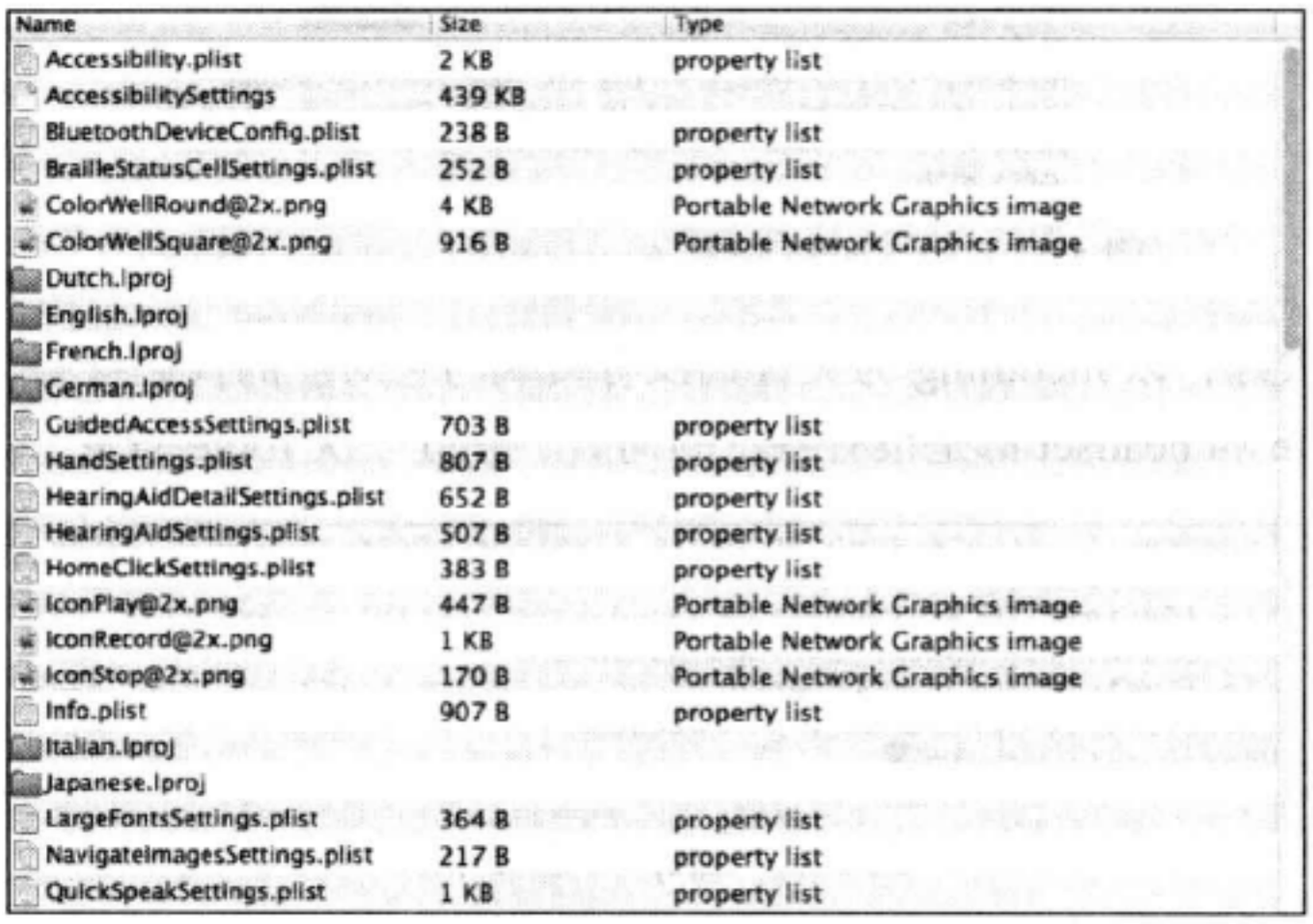


图 5-29　AccessibilitySetting.bundle 的内容

这里的 GuidedAccess、HearingAid 和 HomeClick 等字眼和我们在 Settings → General → Accessibility 中看到的内容吻合。

5. 查找 Incoming

借助强大的 grep，以关键词 incoming 对这个 bundle 进行一次搜索，如图 5-30 所示。

```
-FunMaker-5:~ root# grep -r incoming /System/Library/PreferenceBundles/Ac
cessibilitySettings.bundle
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Engli
sh.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/en_GB
.lproj/Accessibility~iphone.strings matches
-FunMaker-5:~ root#
```

图 5-30 grep 关键词 incoming

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，搜索 incoming，如图 5-31 所示。



图 5-31 用 Xcode 查看 Accessibility~iphone.strings

这次运气不太好，以 incoming 为关键词没有搜到与它对应的代码符号，但它的上下文给了我们提示：route 和 headset 都是很有特点的词语。grep 一次 route 看看，如图 5-32 所示。

```
-FunMaker-5:~ root# grep -r route /System/Library/PreferenceBundles/Acces
sibilitySettings.bundle
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Engli
sh.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/en_GB
.lproj/Accessibility~iphone.strings matches
-FunMaker-5:~ root#
```

图 5-32 grep 关键词 route

还是刚才的文件，搜“route”，如图 5-33 所示。



图 5-33　在 Accessibility~iphone.strings 里搜索 route

这次运气更差，只有一个匹配。不要气馁，逆向工程就是在不断地追踪和错误中一步一步接近目标。

6. 查找 headset

继续 grep 关键词 headset，如图 5-34 所示。

```
[redacted]-FunMaker-5:~ root# grep -r headset /System/Library/PreferenceBundles/Acc
essibilitySettings.bundle
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Dutch
.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Engli
sh.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/en_GB
.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/id.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/ms.lp
roj/Accessibility~iphone.strings matches
```

图 5-34　grep 关键词 headset

文件不变，搜索 headset，如图 5-35 所示。

搜到 6 个匹配的，且有纯大写字符串，直觉告诉我这轮搜索到此为止了。观察这几个 headset，可以看到上方的 RightStereoBalanceIdentifierWHEADSETVCANCEL、DEFAULT_HEADSET 大致与下方的 Left-Right Stereo BalanceQRWHeadsetVCancelQL、Default to headset 分别形成了对应关系，而更关键的是，DEFAULT_HEADSET_FOOTER 的字眼已经明确告诉了我们这个符号的作用，即 UITableView 中某个部分的脚注。对比 Accessibility 界面（如图 5-36 所示）中 Incoming Calls 下的脚注和图中最下两个高亮词之间的文字，哈哈，你是不是也发现了关键的线索？



图 5-35 在 Accessibility~iphone.strings 里搜索 headset

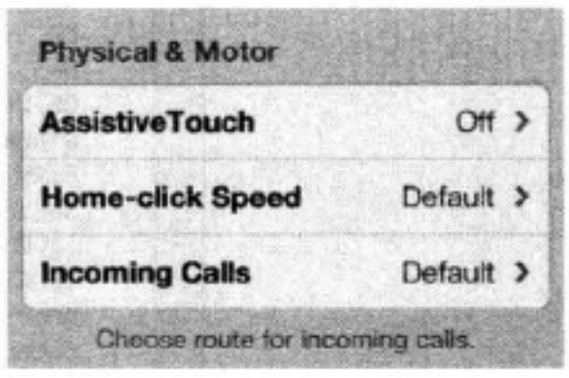


图 5-36 Accessibility 界面

7. 查找 DEFAULT_HEADSET_FOOTER

越来越接近谜底了。grep 关键词 DEFAULT_HEADSET_FOOTER，得到如图 5-37 所示结果。

第一条记录便是一个 plist 文件。无须多言，直接打开它搜索“DEFAULT_HEADSET_FOOTER”，如图 5-38 所示。

```
[redacted]-FunMaker-5:~ root# grep -r DEFAULT_HEADSET_FOOTER /System/Library/Prefer
enceBundles/AccessibilitySettings.bundle
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Acces
sibility.plist matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Dutch
.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Engli
sh.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Frenc
h.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Germa
n.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Itali
an.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Japan
ese.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/Spani
sh.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/ar.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/ca.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/cs.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/da.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/el.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/en_GB
.lproj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/fi.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/he.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/hr.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/hu.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/id.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/ko.lp
roj/Accessibility~iphone.strings matches
Binary file /System/Library/PreferenceBundles/AccessibilitySettings.bundle/ms.lp
```

图 5-37 grep 关键词 DEFAULT_HEADSET_FOOTER

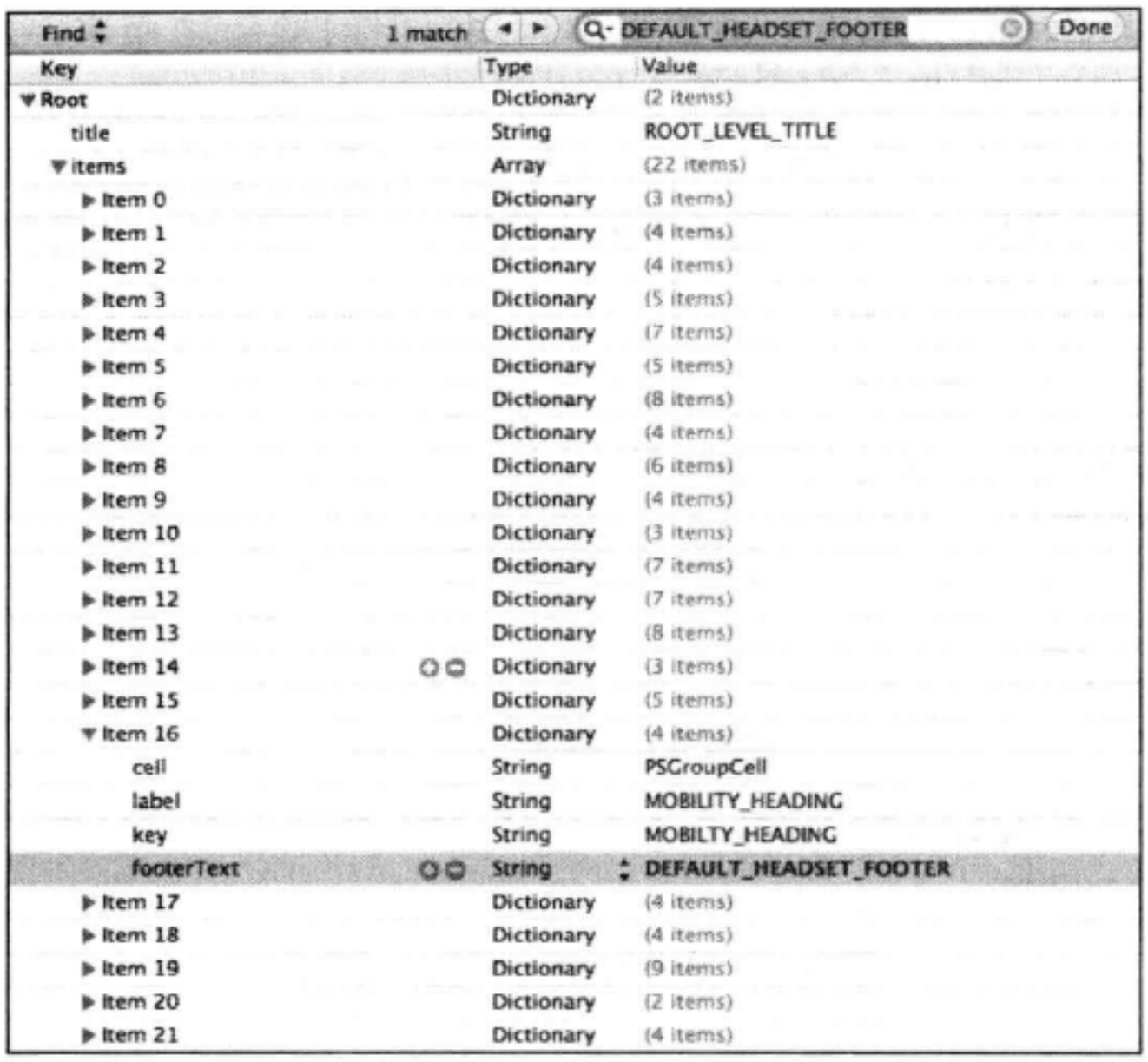


图 5-38　在 Accessibility.plist 里搜索关键词 DEFAULT_HEADSET_FOOTER

瞬间定位到 Item 16，那么就是这个文件，没跑了。

8. 定位 headset

一共就 22 项，手动遍历也好，直接搜“ headset”也好，都能轻松定位到 Item 19，将其展开，一目了然，如图 5-39 所示。

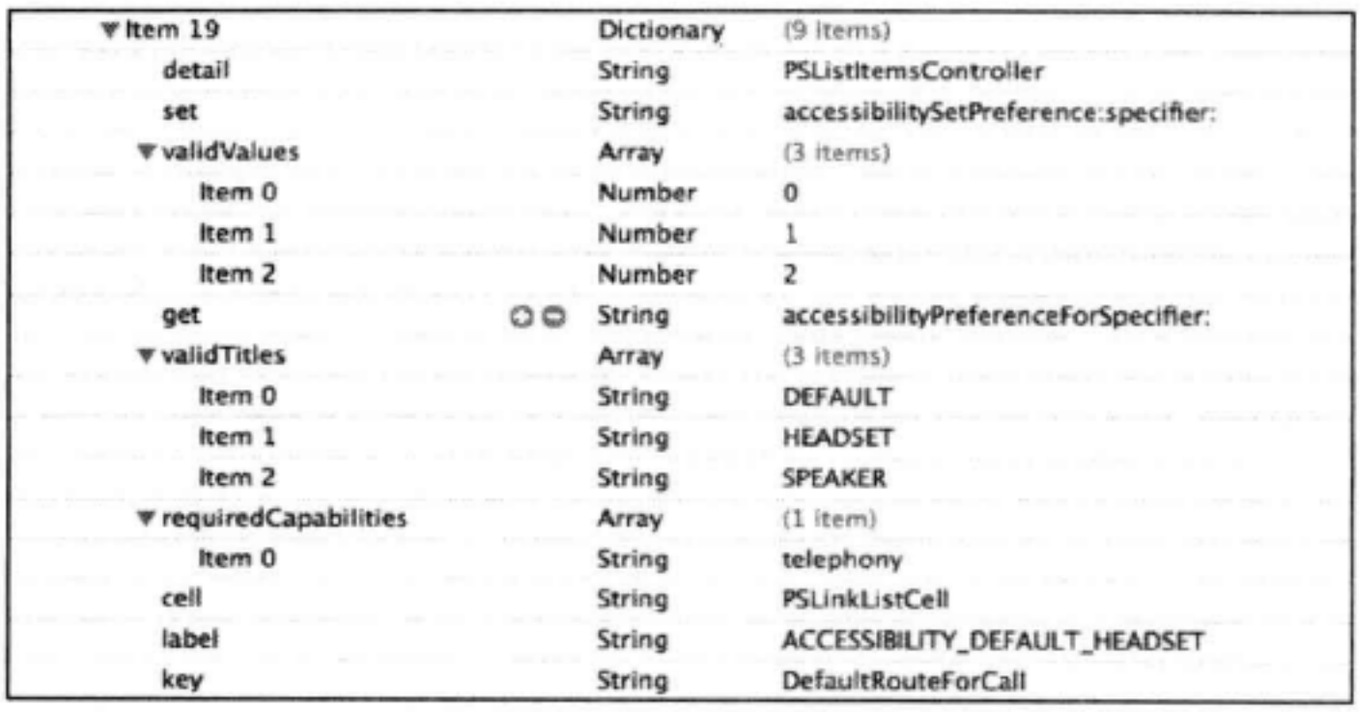
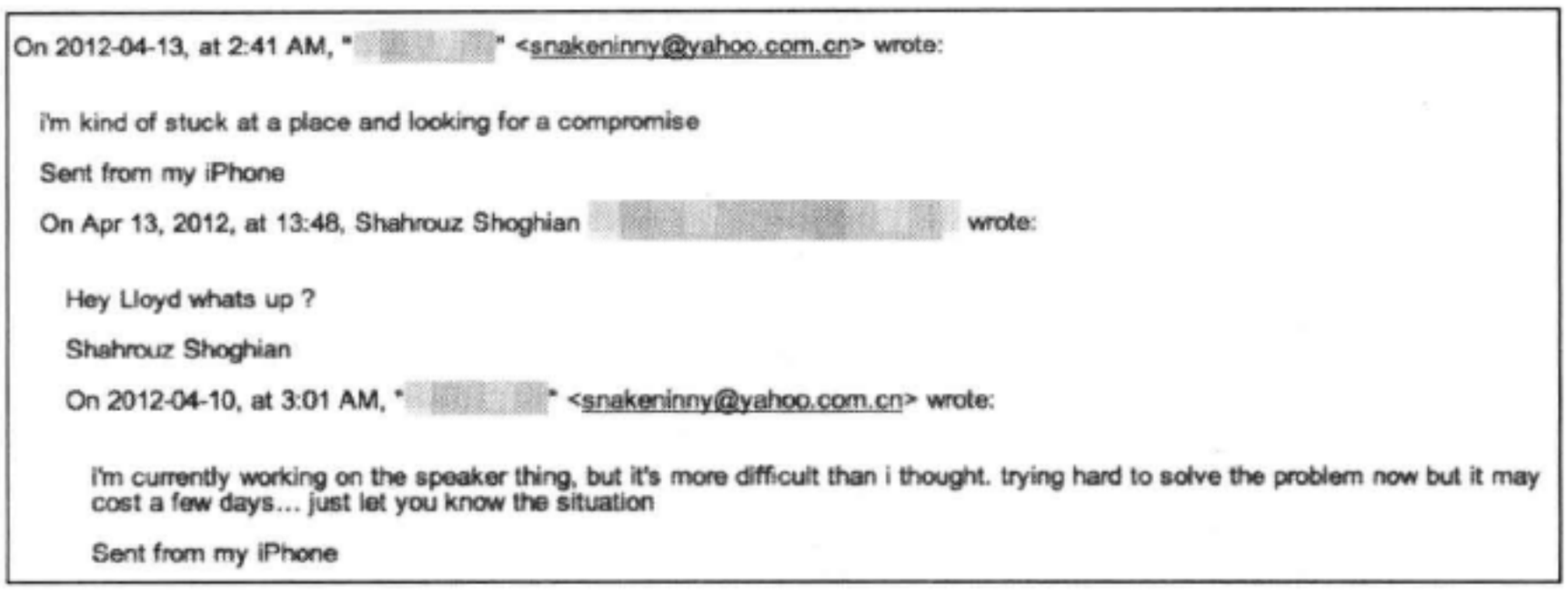


图 5-39　在 Accessibility.plist 里定位目标

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，不知道怎么将这个 bundle 加载进内存，所以没法调用这个函数；笔者当时还不会用 IDA 和 GDB，在 class-dump 的函数里找了又找，也没有发现任何线索，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 5-40 所示。


On 2012-04-13, at 2:41 AM, " " <snakeninny@yahoo.com.cn> wrote:

i'm kind of stuck at a place and looking for a compromise

Sent from my iPhone

On Apr 13, 2012, at 13:48, Shahrouz Shoghian wrote:

Hey Lloyd whats up ?

Shahrouz Shoghian

On 2012-04-10, at 3:01 AM, " " <snakeninny@yahoo.com.cn> wrote:

i'm currently working on the speaker thing, but it's more difficult than i thought. trying hard to solve the problem now but it may cost a few days... just let you know the situation

Sent from my iPhone

图 5-40 笔者给 Shoghian 发的邮件

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型，那么，accessibilitySetPreference:specifier: 的作用会不会是改动这个配置，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，成功了，如图 5-41 所示。

```
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: CFNotifica
tionCenterPostNotification center=<CFNotificationCenter 0x1e529b50 [0x3b75b0f0]>
 name=com.apple.accessibility.defaultrouteforcall userInfo=(null) deliverImmedia
tely=1
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: notify_pos
t com.apple.accessibility.defaultrouteforcall
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: <CFNotific
ationCenter 0x1e52b750 [0x3b75b0f0]> postNotificationName:UIViewAnimationDidComm
itNotification object:UIViewAnimationState userInfo:{
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: <CFNotific
ationCenter 0x1e52b750 [0x3b75b0f0]> postNotificationName:UIViewAnimationDidComm
itNotification object:UIViewAnimationState userInfo:{
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: <CFNotific
ationCenter 0x1e52b750 [0x3b75b0f0]> postNotificationName:UIViewAnimationDidStop
Notification object:<UIViewAnimationState: 0x1f8c2780> userInfo:{
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: <CFNotific
ationCenter 0x1e52b750 [0x3b75b0f0]> postNotification: NSConcreteNotification 0x
1f956f00 {name = UITableViewSelectionDidChangeNotification; object = <UITableVie
w: 0x1f0bfc00; frame = (0 0; 320 504); clipsToBounds = YES; autoresize = W+H; ge
stureRecognizers = <NSArray: 0x1f877900>; layer = <CALayer: 0x1f8c5aa0>; content
Offset: {0, 0}>}
Jul 23 19:28:08 [redacted]-FunMaker-5 Preferences[2536]: LibNotifyWatch: <CFNotifi
ationCenter 0x1e52b750 [0x3b75b0f0]> postNotificationName:_UIApplicationSystemG
stureStateChangedNotification object:(null) userInfo:(null)
```

图 5-41 用 LibNotifyWatch 捕获系统通知

注意看最上面的两条记录，通知的名字均是“com.apple.accessibility.defaultrouteforcall”，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/Applications中全是App，/var/mobile/Media中全是媒体文件，/var/mobile/Documents是空的，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，其内容如图5-42所示。

Key	Type	Value
▼ Root	Dictionary	(4 items)
VoiceOverTouchEnabled	Boolean	NO
DefaultRouteForCallPreference	Number	0
AssistiveTouchEnabled	Boolean	NO
AssistiveTouchHardwareEnabled	Boolean	NO

图5-42 用Xcode查看com.apple.Accessibility.plist

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，与图5-39的配置吻合。

5.3.4 测试函数

在经过漫长的推理之后，笔者总算得出一个可能的解决方案，且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:@"DefaultRouteForCal
lPreference"];
    [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 5-43 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 10 000 的下载量（如图 5-44 所示），对此笔者已经很满意了。更重要的是，这个 tweak 的创作过程历经坎坷，看似简单的功能却是笔者花费九牛二虎之力才得以实现的，无疑给了当时刚刚上路，有些轻飘飘的笔者当头一棒！

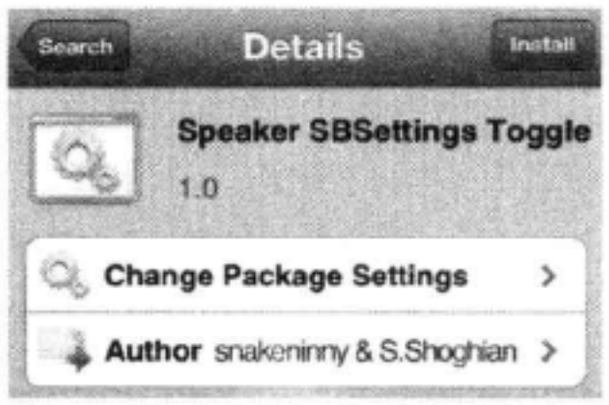


图 5-43 Speaker SBSettings Toggle 的发布页面

Downloads for **snakeninny**

App	Total Count	Installer	Cydia
SMSNinja (v1.2)	22754	0	22754
LowPowerBanner (v0.0.1-42)	35484	0	35484
LowPowerBanner (v1.1)	45466	0	45466
LowPowerBanner (v1.2)	32408	0	32408
LowPowerBanner (v1.3)	13858	0	13858
LowPowerBanner (v1.3.1)	21813	0	21813
LowPowerBanner (v1.3.2)	15447	0	15447
LowPowerBanner (v1.3.2)	1973	0	1973
LowPowerBanner (v1.3.3-1)	179676	0	179676
SMSNinja (v1.3)	4609	0	4609
SMSNinja (v1.3.1)	102315	0	102315
Speaker SBSettings Toggle (v0.0.1-3)	9508	0	9508

图 5-44 Speaker SBSettings Toggle 的下载量

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，熟练掌握相关技术是进一步研究深层技术的基本条件，希望读者多思考、勤练习、早日提炼出更适合自己的方法，进一步达到更高的水平。

第 6 章

ARM 汇编相关的 iOS 逆向理论基础

前面的章节中介绍了 iOS 逆向工程的基础知识，包括一些常见工具的组合使用，在掌握那些知识之后，基本上也就可以玩一玩了，拿下普通的 App 已经没有多少困难了。但是，相信各位读者都是爱学习的，想提高，需要尝试一些有挑战性的内容。那么，从这一章开始，iOS 逆向工程就将进入“极昼”，将零距离接触编程世界里让人头大的知识。请先深呼吸一分钟，“我是否真的适合深入到 iOS 逆向工程这个方向去?”在学完本章之后，相信读者会得到答案的。

下面即将面对 iOS 逆向工程中的第一个难点：阅读 ARM 汇编语言。经过前几章的学习，相信读者已经知道，Objective-C 代码在经过编译后形成机器码，它们由 iDevice 的 CPU 直接执行。别说编写，阅读机器码都已经是一个非常恼人的工作；好在 Objective-C 和机器码之间有汇编语言这座桥，它的可读性虽然远不如 Objective-C，但比机器码要强多了——如果你能够解决它，那么恭喜你，你有着成为逆向工程师的天分；如果你没能力解决，那么 App Store 开发才是你更好的归宿……

6.1 ARM 汇编基础

对于很多 iOS 程序员来说，ARM 汇编是一门全新的语言；如果是计算机专业的读者，应该已经对汇编语言有了初步的印象，只是对于很多人来说，大学期间的汇编语言课简直跟天书一样深奥，它在我们心里埋下了恐惧的种子，仿佛一提到汇编语言，它就会像紧箍咒一样勒紧我们的头，让我们疼痛不已。汇编语言真的有这么难？是，因为汇编的语法晦涩难懂；但另一方面，毕竟它只是一门语言，跟英语一样，熟能生巧，关键还在于我们自己。

我们一般的工作中与汇编打交道的机会并不多，如果不刻意练习，陡然面对时必然掌握不了，所以会觉得它很难。不过归根到底还是投入的时间和精力是否足够的问题——好了，iOS 逆向工程给你学习 ARM 汇编提供了一个绝佳的条件——当我们在逆向一个功能时，可能需要分析大量 ARM 汇编代码，并把它们翻译成高级语言，试图重新实现这个功能；虽然

暂时还不需要写汇编代码，但大量的阅读必然能加深我们对这门语言的理解。如果想在 iOS 逆向工程这条路上走下去，ARM 汇编是必须掌握的语言，也是读者一定能够掌握的语言；跟英语类似，ARM 汇编的基本概念相当于 26 个字母和音标；指令相当于单词，它们的变种相当于单词的各种形态；调用规则相当于语法，以定义句子之间的联系。接下来，让我们一步步地深入。

6.1.1　基本概念

如果要完整地介绍 ARM 汇编，ARM 公司的用户手册已经做得足够好了。笔者对 ARM 汇编也只是略知一二，肯定没有用户手册那么全面，但对于 iOS 逆向工程初学者来说，这些知识足以应对，适度就好。

1. 寄存器、内存和栈

在高级语言，如 Objective-C 和 C/C++ 里，操作对象是变量；在 ARM 汇编里，操作对象是寄存器（register）、内存和栈（stack）。其中，寄存器可以看成是 CPU 自带的变量，它们的数量一般是很有限的；当我们需要更多变量时，就可以把它们放在一个个内存段中；不过，数量上去了，质量也下来了，对内存的操作比对寄存器的操作要慢得多。不管是寄存器还是内存，它们存放的都是全局变量，牵一发而动全身；如果你要定义本地变量，就要用到栈。

栈其实也是一片内存区域，但它具有栈的特点：先进后出。ARM 的栈是满递减（Full Descending）的，向下增长，也就是开口朝下，越靠近栈底，内存地址越小，如图 6-1 所示。

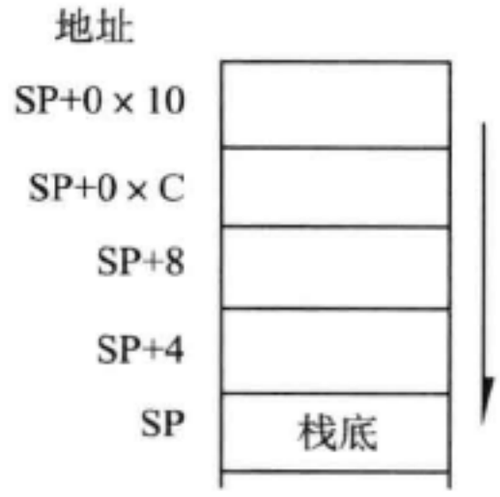


图 6-1　ARM 的栈

一个名为 Stack Pointer（简称 SP）的寄存器保存栈的栈底地址，称为栈地址；我们可以把一个变量给入（push）栈以保存它的值，也可以让它出（pop）栈，恢复变量的原始值。在实际操作中，栈地址会不断变化；但是在执行一块代码的前后，栈地址应该是不变的，不然程序就要出问题了。为什么？举例说明如下：

```
void foo(void)
{
    bar();
    // 其他操作;
}
```

在上面 4 行代码中，函数 foo() 用到了 A、B、C、D 四个寄存器；foo() 内部调用了 bar()，bar() 用到了 A、B、C 三个寄存器。因为 2 个不同的函数用到了 3 个相同的寄存器，所以 bar() 在开始执行前需要将 3 个寄存器中原来的值入栈以保存其原始值，在结束执行前将它们出栈以恢复其原始值，保证 foo() 能够正常执行。用伪汇编代码表示如下：

```
// foo() 函数
foo:
```

```
// 将A、B、C、D入栈，保存它们的原始值
入栈    {A, B, C, D}
// 使用A ~ D
移动    A, #1           // A = 1
移动    B, #2           // B = 2
移动    C, #3           // 你猜猜这行是什么意思?
调用    bar
移动    D, global_var0
// global_var1 = A + B + C + D
相加    A, B            // A = A + B，注意此处A的值
相加    A, C            // A = A + C，还要注意此处A的值
相加    A, D            // 你再猜猜这行是什么意思?
移动    global_var1, A
// 将A、B、C、D出栈，恢复它们的原始值
出栈    {A-D}
返回

// bar()函数
bar:
    // 将A、B、C入栈，保存它们的原始值A == 1，B == 2，C == 3
    入栈    {A-C}
    // 使用A ~ C
    移动    A, #2           // 还需要注释吗?
    移动    B, #5
    移动    C, A
    相加    C, B            // C = 7
    // global_var0 = A + B + C (== 2 * C)
    相加    C, C
    移动    global_var0, C      // A = 2, B = 5, C = 14
    // 现在你知道入栈和出栈的重要意义了吗?
    出栈    {A-C}
    返回
```

简单解释一下这段伪代码：foo() 先将 A、B、C 分别设置为 1、2、3，然后调用 bar()，bar() 改变了 A、B、C 的值，并将全局变量 global_var0 的值设置为 ABC 三者之和。如果把此时的 A、B、C 直接用于 foo()，计算出的另一个全局变量 global_var1 的值就是错的，因此在 bar() 执行前先要让 A、B、C 入栈，保存它们的值，执行完成后再出栈，使得 foo() 能够得到正确的 global_var1。注意一点，出于同样的目的，foo() 在执行前后也对 A、B、C、D 执行了入栈和出栈操作，所以 foo() 的调用者也能够正常工作。

2. 分支跳转与条件判断

处理器中名为 Program Counter（简称 PC）的寄存器用于存放下一条指令的地址。一般情况下，计算机一条接一条地顺序执行指令，处理器执行完一条指令后将 PC 加 1，让它指向下一条指令，如图 6-2 所示。

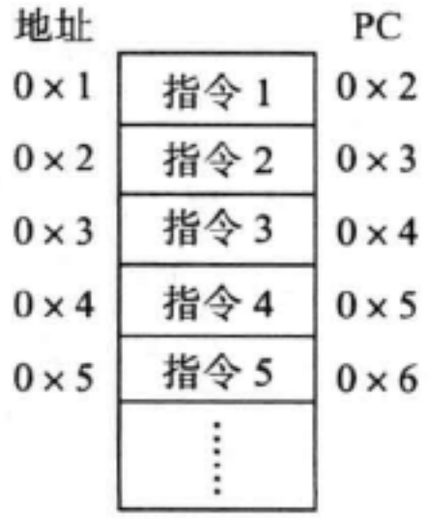


图 6-2 顺序执行指令

处理器顺序执行指令 1 ～指令 5，稀松平常、沉闷无聊。但是如果把 PC 的值变一变，指令的执行顺序就完全不同了，如图 6-3 所示。

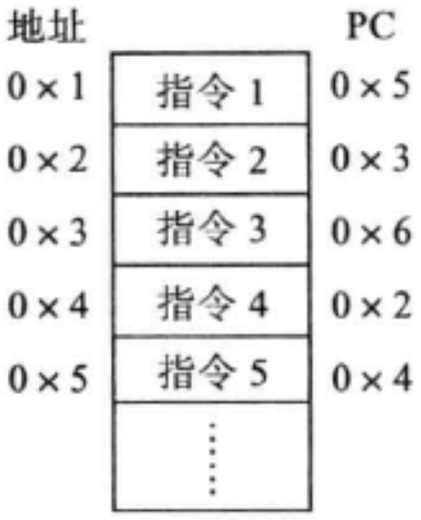


图 6-3　乱序执行指令

指令的执行顺序被打乱，变成指令 1、指令 5、指令 4、指令 2、指令 3、指令 6，光怪陆离，百花齐放。这种“乱序”的学名叫“分支”(branch)，或“跳转”(jump)，它使得循环和 subroutine 成为可能，例如：

```
// endless() 函数
endless:
    操作     操作数 1, 操作数 2
    分支     endless
    返回     // 死循环，执行不到这里啦!
```

在实际情况中，满足一定条件才得以触发的分支是最实用的，这种分支称为条件分支。if else 和 while 都是基于条件分支实现的。在 ARM 汇编中，分支的条件一般有 4 种：

- 操作结果为 0（或不为 0）。
- 操作结果为负数。
- 操作结果有进位。
- 运算溢出（比如两个正数相加得到的数超过了寄存器位数）。

这些条件的判断准则（flag）存放在程序状态寄存器（Program Status Register，简称 PSR）中，数据处理相关指令会改变这些 flag，分支指令再根据这些 flag 决定是否跳转。下面的伪代码展示了一个 for 循环：

```
for:
    相加     A, #1
    比较     A, #16
    不为 0 则跳转到 for
```

此循环将 A 和 #16 作比较，如果两者不相等，则将 A 加 1，继续比较；如果两者相等，则不再循环，继续往下执行。

6.1.2　ARM/THUMB 指令解读

ARM 处理器用到的指令集分为 ARM 和 THUMB 两种，ARM 指令长度均为 32 bit[⊖]，THUMB 指令长度均为 16 bit。所有指令可大致分为 3 类，分别是数据操作指令、内存操作指令和分支指令。

1. 数据操作指令

数据操作指令有两条规则：

（1）所有操作数均为 32 bit。

（2）所有结果均为 32 bit，且只能存放在寄存器中。

总的来说，数据操作指令的基本格式是：

⊖ ARMv8 为 64 bit，但本书完成时 iPhone 5s 尚未越狱，故不予考虑。

```
op{cond}{s} Rd, Rn, Op2
```

其中，“cond”和“s”是两个可选后缀；“cond”的作用是指定指令“op”在什么条件下执行，共有 17 种条件：

```
EQ          结果为 0 (EQual to 0)
NE          结果不为 0 (Not Equal to 0)
CS          有进位或借位 (Carry Set)
HS          同 CS (unsigned Higher or Same)
CC          没有进位或借位 (Carry Clear)
LO          同 CC (unsigned LOwer)
MI          结果小于 0 (MInus)
PL          结果大于等于 0 (PLus)
VS          溢出 (oVerflow Set)
VC          无溢出 (oVerflow Clear)
HI          无符号比较大于 (unsigned HIgher)
LS          无符号比较小于等于 (unsigned Lower or Same)
GE          有符号比较大于等于 (signed Greater than or Equal)
LT          有符号比较小于 (signed Less Than)
GT          有符号比较大于 (signed Greater Than)
LE          无符号比较小于等于 (signed Less than or Equal)
AL          无条件 (ALways, 默认)
```

cond 的用法很简单，例如：

```
比较 R0, R1
移动 GE R2, R0
移动 LT R2, R1
```

比较 R0 和 R1 的值，如果 R0 大于等于 R1，则 R2 = R0；否则 R2 = R1。

s 的作用是指定指令 op 是否设置 flag，共有 4 种 flag：

```
N (Negative)
如果结果小于 0 则置 1，否则置 0；

Z (Zero)
如果结果是 0 则置 1，否则置 0；

C (Carry)
对于加操作 (包括 CMN) 来说，如果产生进位则置 1，否则置 0；对于减操作 (包括 CMP) 来说，Carry 相当于 Not-Borrow，如果产生借位则置 0，否则置 1；对于有移位操作的非加 / 减操作来说，C 置移出值的最后一位；对于其他的非加 / 减操作来说，C 的值一般不变；

V (oVerflow)
如果操作导致溢出，则置 1，否则置 0。
```

需要注意一点，C flag 表示无符号数运算结果是否溢出；V flag 表示有符号数运算结果是否溢出。

数据操作指令可以大致分为以下 4 类：

❑ 算术操作

```
ADD R0, R1, R2              ; R0 = R1 + R2
```

```
ADC R0, R1, R2              ; R0 = R1 + R2 + C(arry)
SUB R0, R1, R2              ; R0 = R1 - R2
SBC R0, R1, R2              ; R0 = R1 - R2 - !C
RSB R0, R1, R2              ; R0 = R2 - R1
RSC R0, R1, R2              ; R0 = R2 - R1 - !C
```

算术操作中，ADD 和 SUB 为基础操作，其他均为两者的变种。RSB 是 Reverse SuB 的缩写，仅仅是把 SUB 的两个操作数调换了位置而已；以 C（即 Carry）结尾的变种代表有进位和借位的加减法，当产生进位或没有借位时，将 Carry flag 置 1。

❑ 逻辑操作

```
AND R0, R1, R2              ; R0 = R1 & R2
ORR R0, R1, R2              ; R0 = R1 | R2
EOR R0, R1, R2              ; R0 = R1 ^ R2
BIC R0, R1, R2              ; R0 = R1 & ~R2
MOV R0, R2                  ; R0 = R2
MVN R0, R2                  ; R0 = ~R2
```

逻辑操作指令没什么可多说的，它们的作用都已经用 C 操作符表示出来了，大家应该很熟悉；但是 C 操作符里的移位操作并没有对应的逻辑操作指令，因为 ARM 采用了桶式移位，共有 4 种指令：

```
LSL         逻辑左移，见图 6-4
LSR         逻辑右移，见图 6-5
```

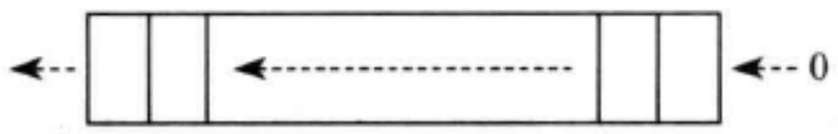


图 6-4　逻辑左移

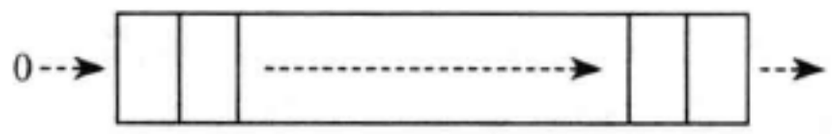


图 6-5　逻辑右移

```
ASR         算术右移，见图 6-6
ROR         循环右移，见图 6-7
```


图 6-6　算术右移

图 6-7　循环右移

❑ 比较操作

```
CMP R1, R2          ; 执行 R1 - R2 并依结果设置 flag
CMN R1, R2          ; 执行 R1 + R2 并依结果设置 flag
TST R1, R2          ; 执行 R1 & R2 并依结果设置 flag
TEQ R1, R2          ; 执行 R1 ^ R2 并依结果设置 flag
```

比较操作其实就是改变 flag 的算术操作或逻辑操作，只是操作结果不保留在寄存器里而已。

❑ 乘法操作

```
MUL R4, R3, R2              ; R4 = R3 * R2
MLA R4, R3, R2, R1          ; R4 = R3 * R2 + R1
```

乘法操作的操作数必须来自寄存器。

2. 内存操作指令

内存操作指令的基本格式如下：

```
op{cond}{type} Rd, [Rn, Op2]
```

其中 Rn 是基址寄存器，用于存放基地址；cond 的作用与数据操作指令相同；type 指定指令 op 操作的数据类型，共有 4 种：

```
B (unsigned Byte)
无符号 byte (执行时扩展到 32bit，以 0 填充);

SB (Signed Byte)
有符号 byte (仅用于 LDR 指令；执行时扩展到 32bit，以符号位填充);

H (unsigned Halfword)
无符号 halfword (执行时扩展到 32bit，以 0 填充);

SH (Signed Halfword)
有符号 halfword (仅用于 LDR 指令；执行时扩展到 32bit，以符号位填充)。
```

如果不指定 type，则默认数据类型是 word。

ARM 内存操作基础指令只有两个：LDR（LoaD Register）将数据从内存中读出来，存到寄存器中；STR（STore Register）将数据从寄存器中读出来，存到内存中。两个指令的使用情况如下：

❑ LDR

```
LDR Rt, [Rn {, #offset}]          ; Rt = *(Rn {+ offset}), {}代表可选
LDR Rt, [Rn, #offset]!            ; Rt = *(Rn + offset); Rn = Rn + offset
LDR Rt, [Rn], #offset             ; Rt = *Rn; Rn = Rn + offset
```

❑ STR

```
STR Rt, [Rn {, #offset}]          ; *(Rn {+ offset}) = Rt
STR Rt, [Rn, #offset]!            ; *(Rn {+ offset}) = Rt; Rn = Rn + offset
STR Rt, [Rn], #offset             ; *Rn = Rt; Rn = Rn + offset
```

此外，LDR 和 STR 的变种 LDRD 和 STRD 还可以操作双字（Doubleword），即一次性操作 2 个寄存器，其基本格式是：

```
op{cond} Rt, Rt2, [Rn {, #offset}]
```

其用法与原型类似，如下：

❑ STRD

```
STRD R4, R5, [R9,#offset]     ;  *(R9 + offset) = R4; *(R9 + offset + 4) = R5
```

❑ LDRD

```
LDRD R4, R5, [R9,#offset]     ;  R4 = *(R9 + offset); R5 = *(R9 + offset + 4)
```

除了 LDR 和 STR 外，还可以通过 LDM（LoaD Multiple）和 STM（STore Multiple）进行块传输，一次性操作多个寄存器。块传输指令的基本格式是：

```
op{cond}{mode} Rd{!}, reglist
```

其中 Rd 是基址寄存器，可选的“!”指定 Rd 变化后的值是否写回 Rd；reglist 是一系列寄存器，用大括号括起来，它们之间可以用“,”分隔，也可以用“-”表示一个范围，比如，{R4 – R6, R8} 表示寄存器 R4、R5、R6、R8；这些寄存器的顺序是按照自身的编号由小到大排列的，与大括号内的排列顺序无关。

需要特别注意的是，LDM 和 STM 的操作方向与 LDR 和 STR 完全相反：LDM 是把从 Rd 开始，地址连续的内存数据存入 reglist 中，STM 是把 reglist 中的值存入从 Rd 开始，地址连续的内存中。此处特别容易混淆，大家一定要注意！

cond 的作用与数据操作指令相同。mode 指定 Rd 值的 4 种变化规律如下：

```
IA (Increment After)
每次传输后增加 Rd 的值;

IB (Increment Before)
每次传输前增加 Rd 的值;

DA (Decrement After)
每次传输后减少 Rd 的值;

DB (Decrement Before)
每次传输前减少 Rd 的值。
```

这是什么意思呢？下面以 LDM 为代表，举一个简单的例子，相信大家一看就明白了。在图 6-8 中，R0 指向的值是 5。执行以下命令后，R4、R5、R6 的值分别变成：

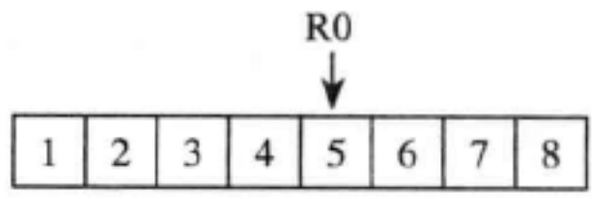


图 6-8　块传输指令模拟环境

```
foo():
    LDMIA R0, {R4 - R6}                    ; R4 = 5, R5 = 6, R6 = 7
    LDMIB R0, {R4 - R6}                    ; R4 = 6, R5 = 7, R6 = 8
    LDMDA R0, {R4 - R6}                    ; R4 = 5, R5 = 4, R6 = 3
    LDMDB R0, {R4 - R6}                    ; R4 = 4, R5 = 3, R6 = 2
```

STM 指令的作用方式与此类似，不再赘述。再次提醒你，LDM 和 STM 的操作方向与 LDR 和 STR 完全相反，切记切记！

3. 分支指令

分支指令可以分为无条件分支和条件分支两种。

❑ 无条件分支

```
B Label              ; PC = Label
BL Label             ; LR = PC - 4; PC = Label
BX Rd                ; PC = Rd 并切换指令集
```

无条件分支很简单，举一个小例子就能了解。

```
foo():
    B Label                    ; 跳转到 Label 处往下执行
    ...                        ; 得不到执行
Label:
    ...
```

❑ 条件分支

条件分支的 cond 是依照 6.1.2 节提到的 4 种 flag 来判断的，它们的对应关系如下：

```
cond        flag
EQ          Z = 1
NE          Z = 0
CS          C = 1
HS          C = 1
CC          C = 0
LO          C = 0
MI          N = 1
PL          N = 0
VS          V = 1
VC          V = 0
HI          C = 1 & Z = 0
LS          C = 0 | Z = 1
GE          N = V
LT          N != V
GT          Z = 0 & N = V
LE          Z = 1 | N != V
```

在条件分支指令前会有一条数据操作指令来设置 flag，分支指令根据 flag 的值来决定代码走向，例如：

```
Label:
    LDR R0, [R1], #4
    CMP R0, 0                  ; 如果 R0 == 0, Z = 1; 否则 Z = 0
    BNE Label                  ; Z == 0 则跳转
```

4. THUMB 指令

THUMB 指令集是 ARM 指令集的一个子集，每条 THUMB 指令均为 16 bit；因此 THUMB 指令比 ARM 指令更节省空间，且在 16 位数据总线上的传输效率更高。除了 b 之外，所有 THUMB 指令均无法条件执行；桶式移位无法结合其他指令执行；大多数 THUMB

指令只能使用 R0 ～ R7 这 8 个寄存器。相对于 ARM 指令，THUMB 指令的特点如下。

❑ 指令数量减少

既然 THUMB 只是一个子集，指令数量必然会减少。例如，乘法指令中只有 MUL 保留了下来，其他的都被精简了。

❑ 无条件执行

除分支指令外，其他指令无条件执行。

❑ 所有指令默认附带 s

即所有 THUMB 指令都会设置 flag。

❑ 桶式移位无法结合其他指令执行

移位指令只能单独执行，无法与其他指令结合执行。即可以：

```
LSL R0 #2
```

而不可：

```
ADD R0, R1, LSL #2
```

❑ 寄存器使用受限

除非显式声明，否则 THUMB 指令只能使用 R0 ～ R7 寄存器；但也有例外：ADD、MOV 和 CMP 指令可以将 R8 ～ R15 作为操作数使用；LDR 和 STR 可以使用 PC 或 SP 寄存器；PUSH 可以使用 LR，POP 可以使用 PC；BX 可以使用所有寄存器。

❑ 立即数和第二操作数使用受限

大多数 THUMB 数据操作指令的形式是“ op Rd, Rm”，只有移位指令、ADD、SUB、MOV 和 CMP 是例外。

❑ 不支持数据写回

除了 LDMIA 和 STMIA 外，其他 THUMB 指令均不支持数据写回，即“!”不可用。

我们在 iOS 逆向工程初级阶段经常会碰到以上指令，如果你对前两节的内容还是感到一知半解，没关系，自己动手分析两个程序实际操作后就好了。这一节的内容只是一个引子，在实际操作中如果对指令作用不清楚，ARM 的官方文档 http://infocenter.arm.com 永远是你最好的参考资料。

6.1.3　ARM 调用规则

了解了常用的 ARM 指令后，相信大家已经能够基本读懂一个函数的汇编代码了。当一个函数调用另一个函数时，常常需要传递参数和返回值；如何传递这些数据，称为 ARM 汇编的调用规则。

1. 前言与后记

在 6.1.1 节提到在执行一段代码时，其前后栈地址应该是不变的，这个操作是通过被执行代码块的前言（prologs）和后记（epilogs）完成的。前言所做的工作主要有：

- ❑ 将 LR 入栈；
- ❑ 将 R7 入栈；
- ❑ R7 = SP；
- ❑ 将需要保留的寄存器原始值入栈；
- ❑ 为本地变量开辟空间。

后记所做的主要工作跟前言正好相反：

- ❑ 释放本地变量占用的空间；
- ❑ 将需要保留的寄存器原始值出栈；
- ❑ 将 R7 出栈；
- ❑ 将 LR 出栈，PC = LR。

前言和后记中的这些工作并不是必需的，如果这段代码根本就没有用到栈，就不需要“保留寄存器原始值”这一步了。在阅读 ARM 汇编代码的过程中前言与后记的影响不大，大家稍作了解即可。

2. 传递参数与返回值

如果想详细了解参数传递规则，可以通读 http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf；一般情况下，记住最重要的一个金句就行：函数的前 4 个参数存放在 R0 ～ R3 中，其他参数存放在栈中；返回值放在 R0 中。

这句话的意思很好理解，为了加深印象，我们看一个例子：

```
int TestFunction(void)
{
    NSLog(@"%d, %d, %d, %d, %d", 1, 2, 3, 4, 5);
    return 6;
}
```

上面函数对应的汇编代码如图 6-9 所示。

BLX _NSLog 执行 NSLog() 函数，它的 6 个参数分别存放在 R0、R1、R2、R3、[SP, #0xC + var_C] 和 [SP, #0xC + var_8] 中，返回值存放在 R0 里。还需要解释吗？

金句的含义非常重要，一定要牢记。

函数的前 4 个参数存放在 R0 ～ R3 中，其他参数存放在栈中；返回值放在 R0 中。

请大家一定要谨记！

3. 特殊用途的寄存器

ARM 处理器中的部分寄存器有特殊用途，如下所示：

R0 ～ R3	传递参数与返回值
R7	帧指针，指向母函数与被调用子函数在栈中的交界
R9	在 iOS 3.0 以前被系统保留
R12	内部过程调用寄存器，dynamic linker 会用到它
R13	SP 寄存器
R14	LR 寄存器，保存返回地址
R15	PC 寄存器

```
; Segment type: Pure code
AREA __text, CODE, READWRITE
; ORG 0xEDC
CODE16


; Attributes: bp-based frame

; TestFunction(void)
EXPORT __Z12TestFunctionv
__Z12TestFunctionv

var_C= -0xC
var_8= -8

PUSH            {R4,R7,LR}
ADD             R7, SP, #4
SUB             SP, SP, #8
MOVW            R0, #(:lower16:(cfstr_DDDDD - 0xEF4)) ; "%d, %d, %d, %d, %d"
MOVS            R1, #5
MOVT.W          R0, #(:upper16:(cfstr_DDDDD - 0xEF4)) ; "%d, %d, %d, %d, %d"
MOVS            R4, #4
STR             R4, [SP,#0xC+var_C]
ADD             R0, PC  ; "%d, %d, %d, %d, %d"
STR             R1, [SP,#0xC+var_8]
MOVS            R1, #1
MOVS            R2, #2
MOVS            R3, #3
BLX             _NSLog
MOVS            R0, #6
ADD             SP, SP, #8
POP             {R4,R7,PC}
; End of function TestFunction(void)
```

图 6-9　TestFunction 的汇编代码

因为现在还没有开始自己写汇编代码，所以对上述知识有简单的了解就足够了。

本节只是把 ARM 汇编级别 iOS 逆向工程最基本的知识过了一遍，难免有遗漏，但说白了，只要记住刚才的“金句”，配合 ARM 官方网站中的资料，相信大家肯定能分析程序了。接下来，就来实践下，看看如何把刚刚学到的知识运用到 iOS 逆向工程中。

6.2　在 IDA 中分析目标文件

还记得 3.5.3 节展示的 IDA 使用示例吗？当时只是初步介绍了 IDA 的用法，虽然那时对 ARM 汇编语言还一无所知但也能勉强明白系统都干了些什么，不过这不是长久之计；在 5.3 节中，完整地介绍了 Speaker SBSettings Toggle 的开发经历，整个过程连 IDA 都没有用上，更别说读汇编代码了。虽然最后侥幸达到了目标，但心里还是有些犯怵的。可见，在离开 IDA 和 ARM 汇编的情况下凭运气也能解决一些小问题，但运气这个东西是虚无的；在学习了 IDA 的基本用法和 ARM 汇编的基础知识后，终于可以把重心放在分析程序上了，并且减少了对飘忽不定的运气的依赖。当然，运气也是 iOS 逆向工程中非常重要的一环。

学习了 ARM 汇编基础之后，几乎所有的 App 都变成“开源”的了，不知亲爱的读者是否意识到，只要你耐着性子一步步分析，所有 App 的实现细节都可以搞得一清二楚。当然，在实际情况中，我们往往只需要分析关键函数的用法，就可以设计出功能类似的软件。下面，推翻之前的 Speaker SBSettings Toggle，利用我们刚刚掌握的技术，分析系统是怎么实现这个功能的，并把它提取出来，用于升级 Speaker SBSettings Toggle。

6.2.1 分析函数实现细节

根据第 5 章的分析，我们已经知道“在选择‘Incoming Calls’中的某一行时，accessibility-SetPreference:specifier: 函数会得到调用”。在 IDA 中定义到这个函数，双击之，如图 6-10 所示。

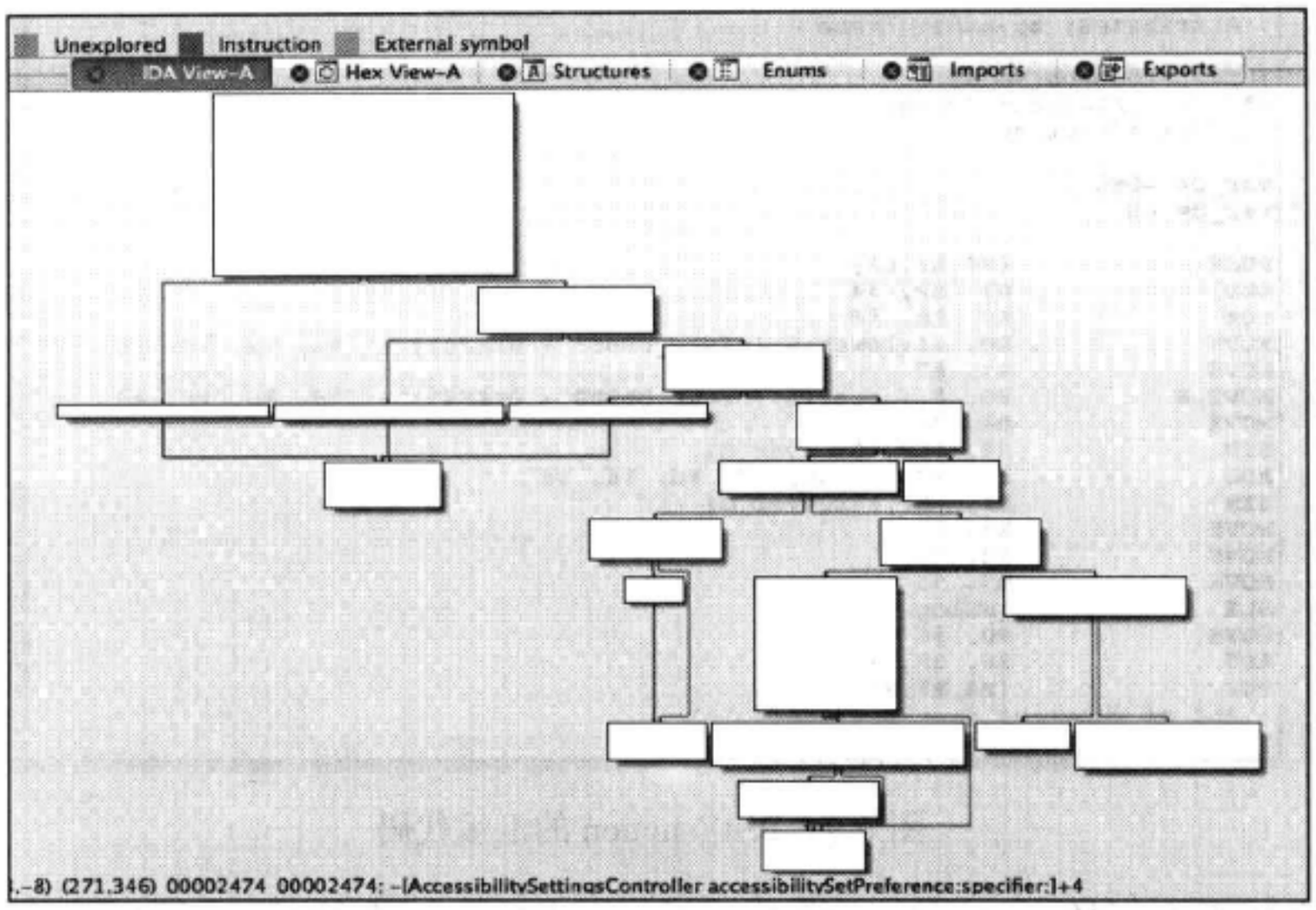


图 6-10 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 的整体逻辑

可以看出，其函数实现还算逻辑清晰、棱角分明。很多时候，最长的那条路往往才是通向成功的捷径，所以我们就从起始部分开始一点点分析吧，如图 6-11 所示。

```
; AccessibilitySettingsController - (void)accessibilitySetPreference:(id) specifier:(id)
; Attributes: bp-based frame

; void __cdecl -[AccessibilitySettingsController accessibilitySetPreference:specifier:](struct AccessibilitySettingsController *self, SEL, id, id)
__AccessibilitySettingsController_accessibilitySetPreference_specifier__

var_14= -0x14

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
PUSH.W          {R8,R10}
SUB.W           R4, SP, #0x10
BIC.W           R4, R4, #0xF
MOV             SP, R4
VST1.64         {D8-D9}, [R4@128]
MOV             R8, R0
MOV             R0, #(selRef_properties - 0x2696) ; selRef_properties
MOV             R10, R2
ADD             R0, PC ; selRef_properties
LDR             R1, [R0] ; "properties"
MOV             R0, R3
BLX             _objc_msgSend
MOV             R2, #(_PSKeyNameKey_ptr - 0x24AC) ; _PSKeyNameKey_ptr
MOVW            R1, #(:lower16:(selRef_objectForKey_ - 0x24B4))
ADD             R2, PC ; _PSKeyNameKey_ptr
MOVT.W          R1, #(:upper16:(selRef_objectForKey_ - 0x24B4))
LDR             R2, [R2] ; _PSKeyNameKey
ADD             R1, PC ; selRef_objectForKey_
LDR             R1, [R1] ; "objectForKey:"
LDR             R2, [R2]
BLX             _objc_msgSend
MOV             R4, R0
MOV             R0, #(selRef_boolValue - 0x24C8) ; selRef_boolValue
ADD             R0, PC ; selRef_boolValue
LDR             R1, [R0] ; "boolValue"
MOV             R0, R10
BLX             _objc_msgSend
MOV             R6, R0
MOV             R0, #(_kAXSInvertColorsEnabledPreference_ptr - 0x24E0) ; _kAXSInvertColorsEnabledPreference_ptr
MOVW            R1, #(:lower16:(selRef_isEqualToString_ - 0x24E8))
ADD             R0, PC ; _kAXSInvertColorsEnabledPreference_ptr
MOVT.W          R1, #(:upper16:(selRef_isEqualToString_ - 0x24E8))
LDR             R0, [R0] ; _kAXSInvertColorsEnabledPreference
ADD             R1, PC ; selRef_isEqualToString_
LDR             R5, [R1] ; "isEqualToString:"
MOV             R1, R5
LDR             R2, [R0]
MOV             R0, R4
BLX             _objc_msgSend
TST.W           R0, #0xFF
BEQ             loc_2520
```

图 6-11 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 的起始部分

第一个调用的函数是 objc_msgSend。Apple 官方文档的注释如图 6-12 所示。

Discussion
When it encounters a method call, the compiler generates a call to one of the functions `objc_msgSend`, `objc_msgSend_stret`, `objc_msgSendSuper`, or `objc_msgSendSuper_stret`. Messages sent to an object's superclass (using the `super` keyword) are sent using `objc_msgSendSuper`; other messages are sent using `objc_msgSend`. Methods that have data structures as return values are sent using `objc_msgSendSuper_stret` and `objc_msgSend_stret`.

图 6-12　objc_msgSend 的官方释义

依据第 5 章中对象、方法和实现的关系来进一步探索，[receiver message] 在执行时会被转换为 objc_msgSend(receiver, @selector(message))；当方法有参数时，则由 [receiver message:arg1 foo:arg2 bar:arg3] 转换为 objc_msgSend(receiver, @selector(message), arg1, arg2, arg3)，依次类推。

所以依照 ARM 调用规则，objc_msgSend 调用时的参数应该是 objc_msgSend(R0, R1, R2, R3, stack...) 形式的，即 [R0 R1:R2 foo:R3 bar:stack ...]。把这个标准套用在第一个 objc_msgSend 上，想要知道它的 Objective-C 方法原型，需要从下往上推导。

首先，R0 来自“MOV R0, R3”，即 R3；R3 又是哪里来的？在 6.1.1 节提到，不管是寄存器还是内存，它们存放的都是全局变量，牵一发而动全身，那么 R3 的值就是 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 的调用者传过来的；前面刚刚讨论过，[AccessibilitySettingsController accessibilitySetPreference:arg1 specifier:arg2] 会转换成 objc_msgSend(AccessibilitySettingsController, @selector(accessibilitySetPreference:specifier:), arg1, arg2)，那么 R3 就是 arg2，即 specifier: 的参数；R1 是 properties，所以第一个 objc_msgSend 在转换前应该是 [specifier properties]。

其次，objc_msgSend 的“方法”是 R1，即“objectForKey:”，那它的对象 R0 在哪里？还是来自“MOV R0, R3”吗？非也非也。还记得金句的后半句吗？“返回值放在 R0 中”——如果你读过 Apple 官方编写的“Cocoa 编码规范”（https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingMethods.html）就会知道，[specifier properties] 是有返回值的。事实上，我们在 Preferences.app 的头文件中搜索关键词“properties”，只有两个结果，如图 6-13 所示。

那么这里的 specifier 显然是一个 PSSpecifier 类的对象（如果你认真阅读了 5.3.4 节 的 Preference specifier plist 部 分，就应该知道 specifier 是哪个类的对象了）。打开 PSSpecifier.h，可以看到 [PSSpecifier properties] 确实是有返回值的，如图 6-14 所示。

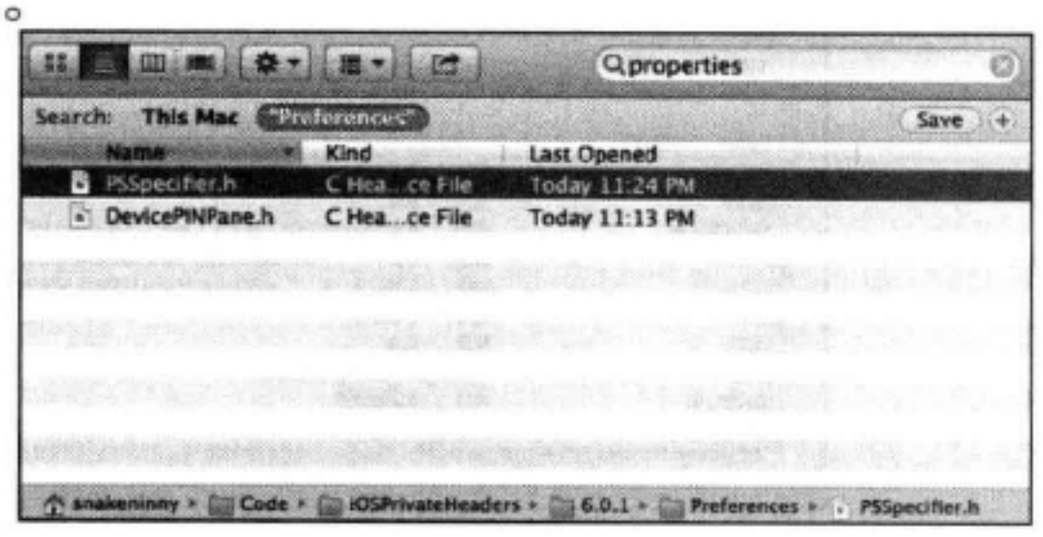


图 6-13　在 Preferences.app 的头文件里搜索 properties

所以第二个 objc_msgSend 在转换前应该是 [[specifier properties] objectForKey:]，至于它的参数，当然就存放在 R2 里，即 @"PSKeyNameKey"，所以完整的方法是 [[specifier properties] objectForKey:@ "PSKeyNameKey"]，并把返回值通过“MOV R4, R0”存入 R4。

```
67 - (void)loadValuesAndTitlesFromDataSource;
68 - (id)properties;
69 - (void)setProperties:(id)arg1;
70 - (void)removePropertyForKey:(id)arg1;
71 - (void)setProperty:(id)arg1 forKey:(id)arg2;
72 - (id)propertyForKey:(id)arg1;
```

图 6-14 PSSpecifier.h 的部分内容

继续看第三个 objc_msgSend，它的第一个参数 R0 来自于“MOV R0, R10”，即 R10；R10 来自“MOV R10, R2”，即 R2；R2 来自于 [AccessibilitySettingsController accessibility SetPreference:arg1 specifier:arg2] 中的 arg1。R1 是 boolValue，因此它的原型是 [preference boolValue]，返回值通过“MOV R6, R0”存入 R6。

第四个 objc_msgSend，R0 来自于“MOV R0, R4”，即 R4；R4 来自于“MOV R4, R0”，即第二个 objc_msgSend 的返回值。R1 来自于 R5，“isEqualToString”，R2 来自于 [R0]，即 @"kAXSInvertColorsEnabledPreference"，所以它的原型是 [[[specifier properties] objectForKey:@ "PSKeyNameKey"] isEqualToString:@ " kAXSInvertColorsEnabledPreference"]。

如果感觉还有点模棱两可，没关系，把第 1 ～第 4 个 objc_msgSend 的推导过程再看一遍，对照着 IDA，很快就能明白。

接下来是一个分支，“BEQ loc_2520”（是什么意思还记得吧？）“Branch if EQual to 0”，判断标准自然是“TST.W R0, #0xFF”。因为 TST 实际是 & 操作，所以只有当 R0 是 0 时，结果才是 0，从而导致 Branch；Objective-C 的 BOOL 类型里，0 为 NO，1 为 YES，所以 [[[specifier properties] objectForKey:@ "PSKeyNameKey"] isEqualToString:@ "kAXSInvertColorsEnabled-Preference"] 的结果为 NO 时才会执行跳转到 loc_2520。那么这个函数的执行结果是 NO 还是 YES 呢？观察函数的参数 @ "kAXSInvertColorsEnabledPreference"，我们猜测它跟 Accessibility 面板里的“Invert Colors”开关是对应的，如图 6-15 所示。



图 6-15 Accessibility 界面

显然我们的目标并不是这个开关，所以执行结果是 NO，跳转到 loc_2520，如图 6-16 所示。

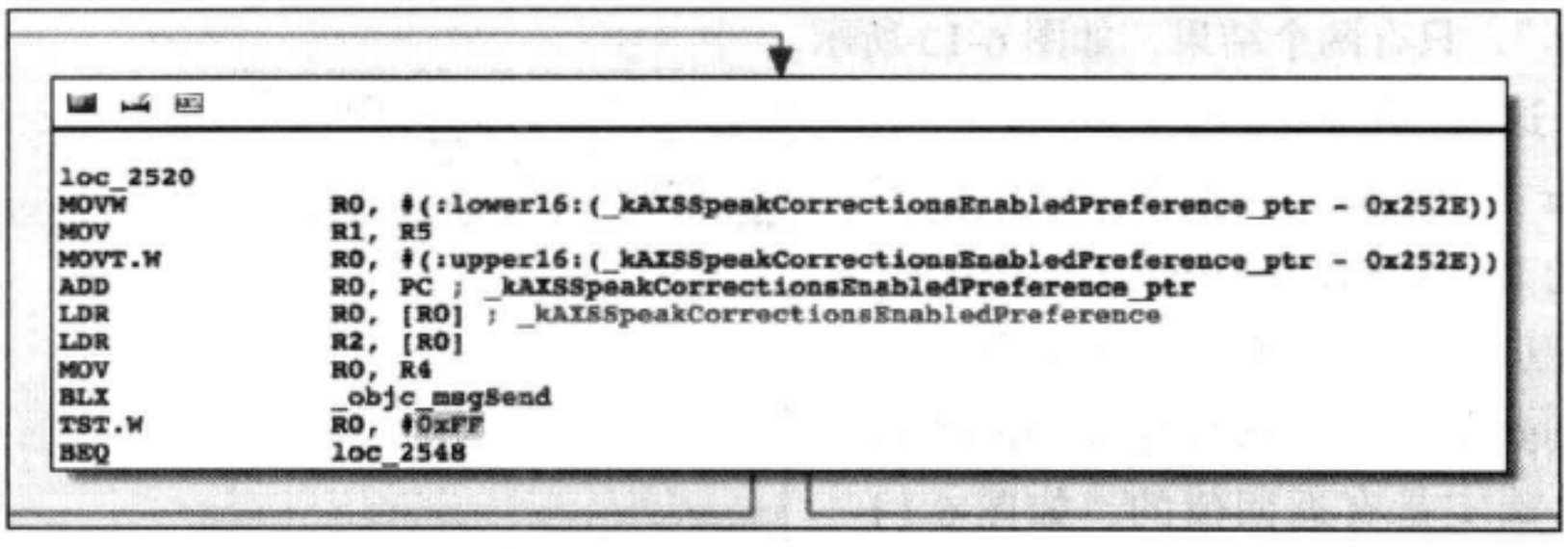


图 6-16 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 片段

又是一个“kAXSXXX”？还是[[[specifier properties] objectForKey:@ " PSKeyNameKey"] isEqualToString:@ " kAXSSpeakCorrectionsEnabledPreference"]吗？看看R0的值，来自R4；看看R1的值，来自R5；跳回到图6-11，观察R4和R5的值——还真是[[[specifier properties] objectForKey:@ " PSKeyNameKey"] isEqualToString:@ " kAXSSpeakCorrectionsEnabledPreference"]，而且它对应的貌似是图6-15中的“Speak Auto-text”开关（注意这个开关下面两行小字中的“speak auto-corrections”字眼）。不是它，则跳转到loc_2548，如图6-17所示。

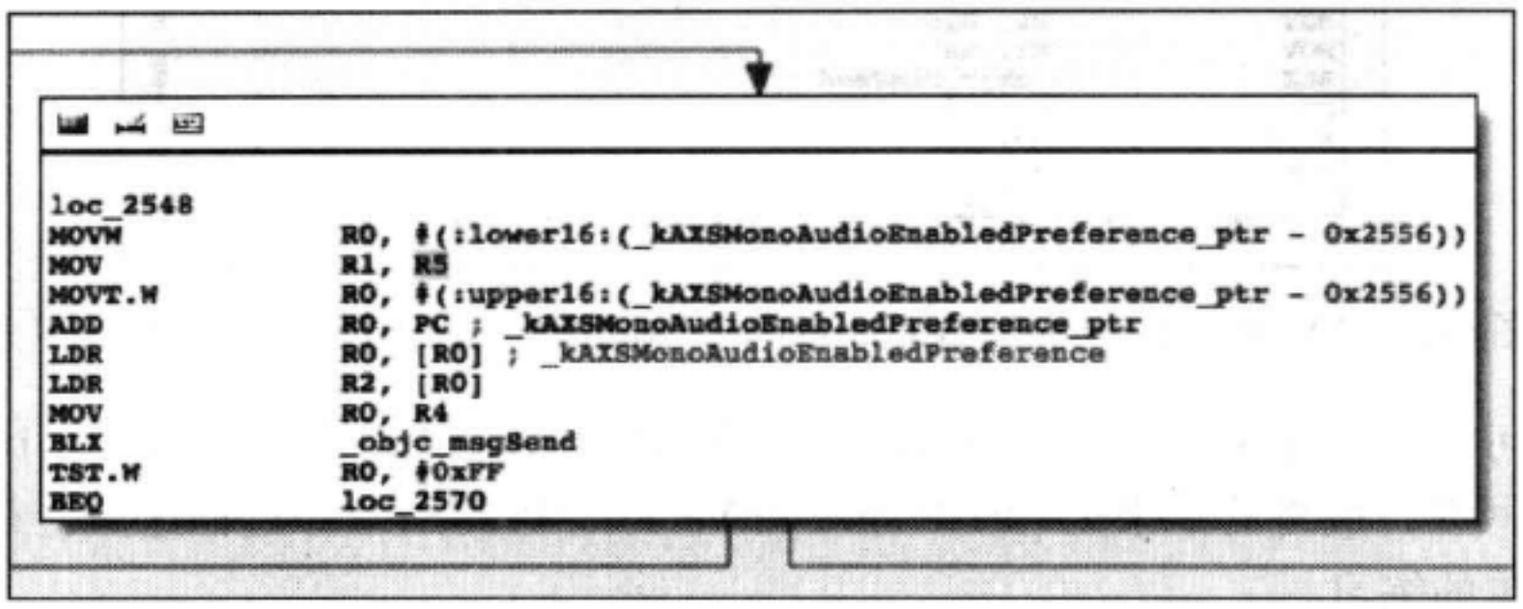


图6-17　[AccessibilitySettingsController accessibilitySetPreference:specifier:]片段

继续前面的推理，现在跳转到loc_2570，如图6-18所示。推理过程留给你，相信你没问题。

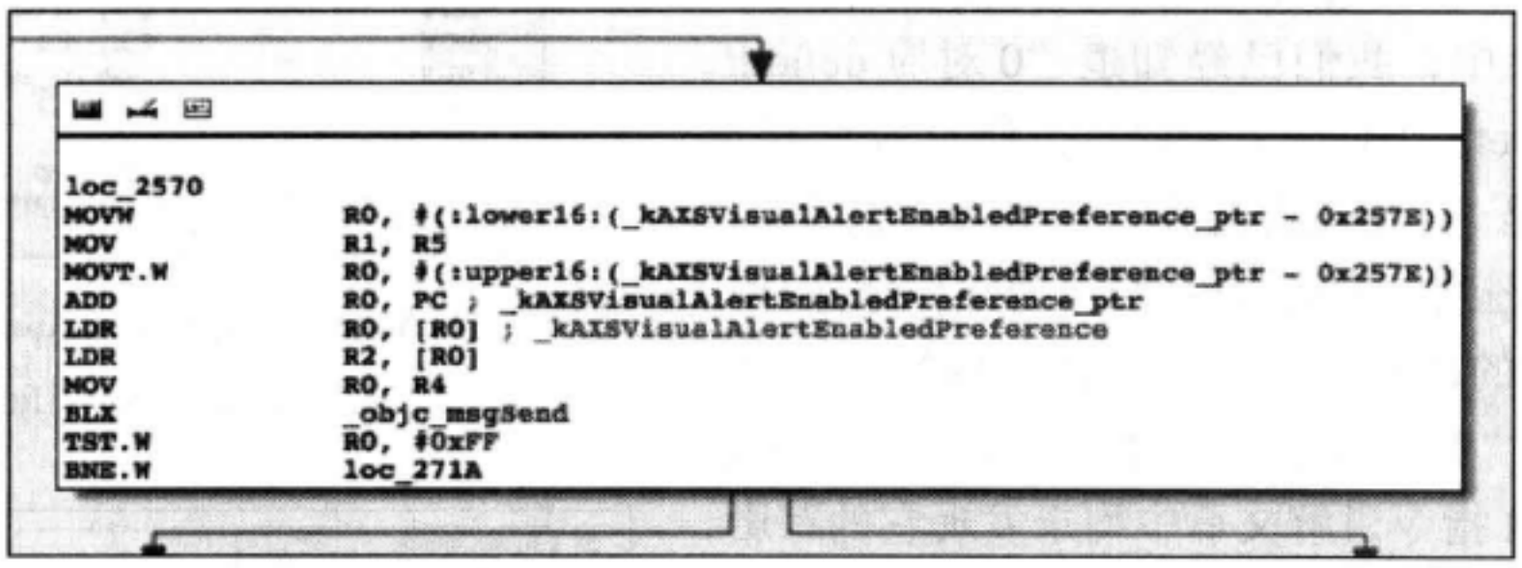


图6-18　[AccessibilitySettingsController accessibilitySetPreference:specifier:]片段

接下来跳转到loc_271A，对吧？

错！这里的分支指令是BNE，如图6-19所示。推理过程还是你来，这里不介绍了。

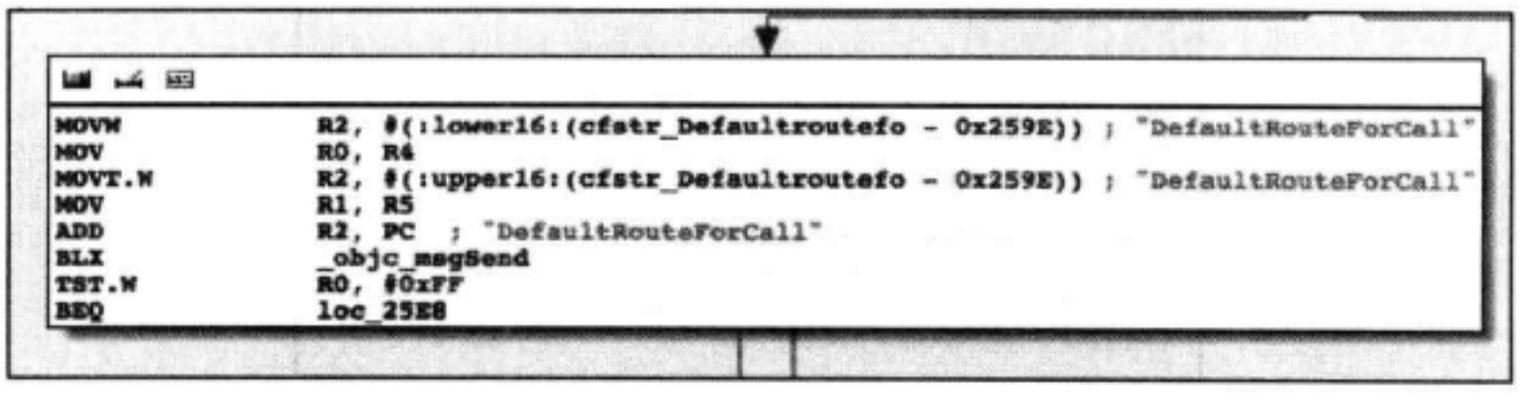


图6-19　[AccessibilitySettingsController accessibilitySetPreference:specifier:]片段

到这里，有点意思了。这时，[[[specifier properties] objectForKey:@ "PSKeyNameKey"] isEqualToString:@ "DefaultRouteForCall"] 的返回值应该是 YES，即 1，所以“TST R0, #0xFF”的结果不是 0，走左边的路，到达图 6-20 处。

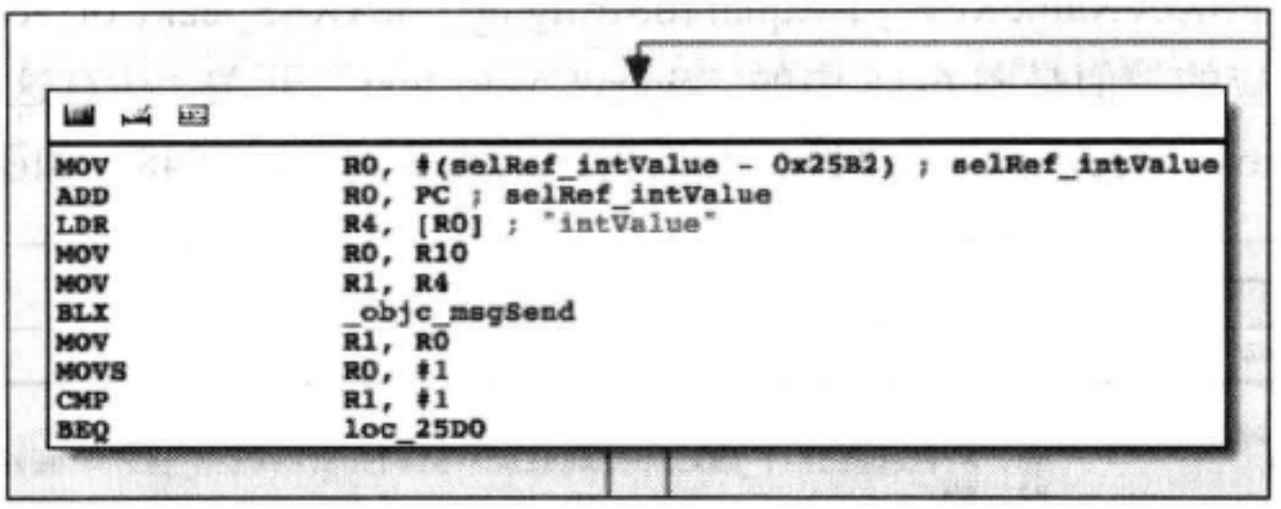


图 6-20 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 片段

贯彻金句，这里的 objc_msgSend 原型应该是 [R10 R4]。R4 是 intValue，R10 是什么？顺着原路推导回去，最终的结果是 [preference intValue]。中间的推导过程这里全都省略了，留给你作为简单的练习。

紧接着又是个“BEQ loc_25D0”，它的判断标准是“CMP R1, #1”。因为“CMP”实际是“-”，所以当 R1 为 1 时，结果才为 0，跳转到 loc_25D0；否则走左边的线。因为 R1 的值是 R0 传过来的，即 [preference intValue] 的结果；而在 5.3.3 节中，我们已经知道“0 对应 default，1 对应 headset，2 对应 speaker”，所以我们想要的 R1 并不是 1，走左边的线，到达图 6-21 处。

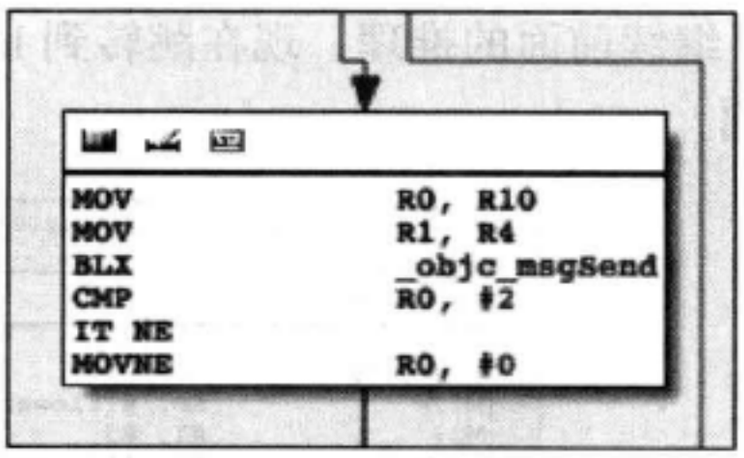


图 6-21 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 片段

一直推理到“CMP R0, #2”为止，这些都是前面讲解过的内容，把 [preference intValue] 跟 2 比较。但是“IT”是什么？事实上，笔者阅读过的好些 ARM 指令讲解文章中均未发现它的踪影，在 ARM 官网搜索，竟然也找不到任何结果，如图 6-22 所示。

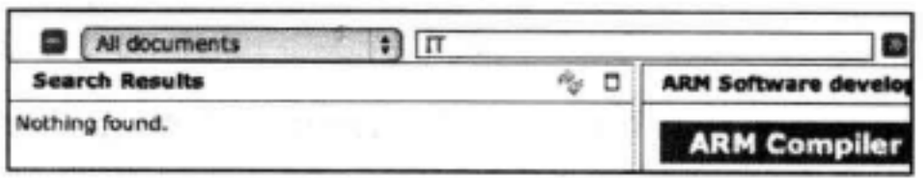


图 6-22 在 ARM 官网搜索关键词

最后还是谷歌帮上了忙，如图 6-23 所示。



图 6-23 用谷歌搜索关键词

感兴趣的读者可以自己看看官网对这条指令的详细解释，简单介绍如下。

IT 是“If-Then”的缩写，它围起一个块，这个块里最多有 4 条指令，可以条件执行。它的语法是 IT{x{y{z}}} {cond}，即：

```
IT <cond>                    ; 围起 1 条指令
IT<x> <cond>                 ; 围起 2 条指令
IT<x><y> <cond>              ; 围起 3 条指令
IT<x><y><z> <cond>           ; 围起 4 条指令
```

其中 <x><y><z> 分别对应着第 2、3、4 条指令的执行条件，取值可以是 T（Then）或 E（Else），前者代表满足 cond 则执行，后者代表不满足 cond 则执行。所以：

```
CMP R0, #2
IT NE
MOVNE R0, #0
```

意思是如果 R0 != 2，则满足“NE”，从而执行“MOVNE R0, #0”，即 R0 = 0；如果 R0 == 2，则不执行“MOVNE R0, #0”，R0 不变。还是有点绕，不太懂？好的，再举一个例子：

```
CMP R0, #2
ITE NE
MOVNE R0, #0
MOV R0, #1
```

这段代码的意思是，如果 R0 != 2，则满足 NE，从而执行“MOVNE R0, #0”，即 R0 = 0；而“MOV R0, #1”的执行条件是“E NE”，即不满足 NE 才执行，所以当 R0 != 2 时得不到执行，R0 == 2 时才能得到执行，即 R0 = 1。翻译成 C 语言就是：

```
if (R0 != 2)
    R0 = 0;
else R0 = 1;
```

回到图 6-21，因为 2 对应 speaker，所以我们预想的 R0 值为 2，不执行“MOVNE R0, #0”，前进至图 6-24。

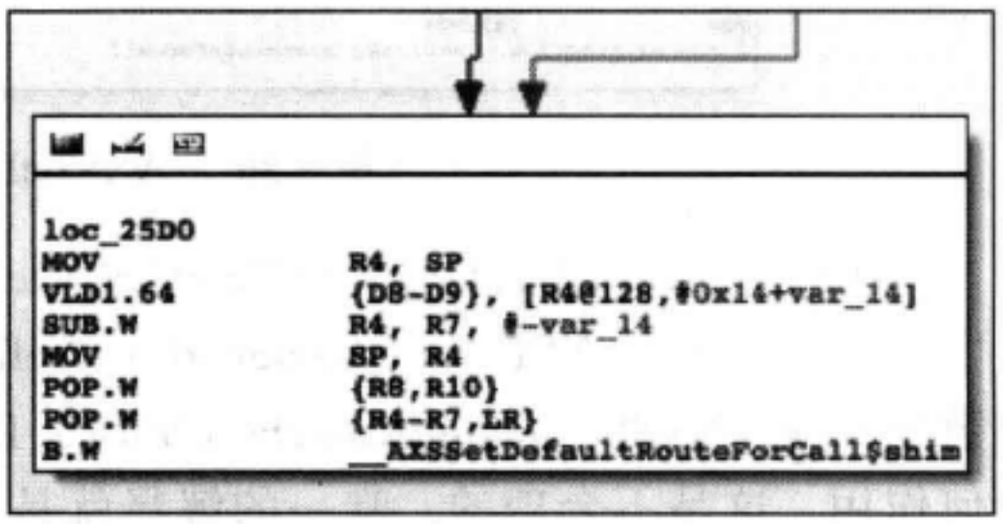


图 6-24　[AccessibilitySettingsController accessibilitySetPreference:specifier:] 片段

现在来到了 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 的尾声，一个不认识的函数“_AXSSetDefaultRouteForCall”出现在我们面前。它从哪里来？参数有几个？要到哪里去（有返回值吗）？面对这一系列扑朔迷离的问题，我们将何去何从？是时候再次运用我们的“金句”了：函数的前 4 个参数存放在 R0 ～ R3 中，其他参数存放在栈中；返回值放在 R0 中。

套用到函数“_AXSSetDefaultRouteForCall”中，就是 _AXSSetDefaultRouteForCall(R0,

R1, R2, R3, stack...)。参照上面的推理，R0 是一个 int，我们希望它是 2；R1 是通过“MOV R1, R4”得来的，即 R4，而 R4 是 intValue，即一个 @selector；R3 是 specifier。一般的非 runtime 函数是不会拿 @selector 作为参数的，而 specifier 作为参数在这个函数里似乎没有意义，于情于理，都觉得 _AXSSetDefaultRouteForCall() 应该只有一个参数，即 _AXSSetDefaultRouteForCall(int route)。要怎么验证我们的猜测呢？当然是去 _AXSSetDefaultRouteForCall 的实现里一探究竟啦！

双击图 6-24 里的“B.W __AXSSetDefaultRouteForCall$shim”，跟踪它的“实现”，发现它是一个从外部导入的函数，如图 6-25 所示。

```
UNDEF:00069828                          ; DATA XREF: __AXSQuickSpeakSetPreferredLocaleForLanguageio ...
UNDEF:0006982C          IMPORT __imp___AXSQuickSpeakSetRate
UNDEF:0006982C                          ; CODE XREF: __AXSQuickSpeakSetRateij
UNDEF:0006982C                          ; DATA XREF: __AXSQuickSpeakSetRateio ...
UNDEF:00069830          IMPORT __imp___AXSSetDefaultRouteForCall
UNDEF:00069830                          ; CODE XREF: __AXSSetDefaultRouteForCallij
UNDEF:00069830                          ; DATA XREF: __AXSSetDefaultRouteForCallio ...
UNDEF:00069834          IMPORT __imp___AXSSetLeftRightAudioBalance
UNDEF:00069834                          ; CODE XREF: __AXSSetLeftRightAudioBalanceij
UNDEF:00069834                          ; DATA XREF: __AXSSetLeftRightAudioBalanceio ...
```

图 6-25 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 片段

往上滚动，发现它是从“/usr/lib/libAccessibility.dylib”导入的。在 IDA 中打开这个 dylib，定位到“_AXSSetDefaultRouteForCall”，如图 6-26 所示。

```
EXPORT __AXSSetDefaultRouteForCall
__AXSSetDefaultRouteForCall

var_C= -0xC

PUSH            {R7,LR}
MOV             R7, SP
SUB             SP, SP, #4
MOV             R2, #(_kAXSDefaultRouteChangedNotification - 0x3467D102) ; _kAXSDefaultRouteChangedNotification
MOV             R1, #(_kAXSDefaultRouteForCallPreference - 0x3467D106) ; _kAXSDefaultRouteForCallPreference
ADD             R2, PC ; _kAXSDefaultRouteChangedNotification
STR             R0, [SP,#0xC+var_C]
ADD             R1, PC ; _kAXSDefaultRouteForCallPreference
LDR             R3, [R2]
MOV             R2, SP
LDR             R0, [R1]
MOVS            R1, #9
BL              __setNumberPreference
ADD             SP, SP, #4
POP             {R7,PC}
; End of function __AXSSetDefaultRouteForCall
```

图 6-26 _AXSSetDefaultRouteForCall 片段

注意，“MOV R2, #(_kAXSDefaultRouteChangedNotification - 0x3467D102)”更改了 R2；“MOV R1, #(_kAXSDefaultRouteForCallPreference - 0x3467D106)”更改了 R1，说明如果函数定义是 _AXSSetDefaultRouteForCall(R0, R1, R2)，那传进来的参数 R1 和 R2 就没起到任何作用，这是不合理的，唯一的解释就是 _AXSSetDefaultRouteForCall 只有一个参数，即 R0。再看“STR R0, [SP,#0xC+var_C]”，在没有给 R0 赋值的情况下就把 R0 中的值取出来存进 [SP,#0xC+var_C]，说明 R0 在 _AXSSetDefaultRouteForCall 得到调用前就已经初始化了——这不就是函数参数的特征吗？所以，我们可以确定 _AXSSetDefaultRouteForCall 只有一个参数。

还有一个问题，_AXSSetDefaultRouteForCall 有返回值吗？根据这个函数的作用和名称来看，如果它有返回值，也应该是告知函数是否执行成功，即返回值应为 BOOL _AXSSetDefaultRouteForCall(int route) 或者 int _AXSSetDefaultRouteForCall(int route)。在

图 6-26 中，R0 最后的值是由“LDR R0, [R1]”得到的，即“kAXSDefaultRouteForCallPreference”，不像返回值；R0 的其他变化都发生在 _setNumberPreference() 里，如图 6-27 所示。

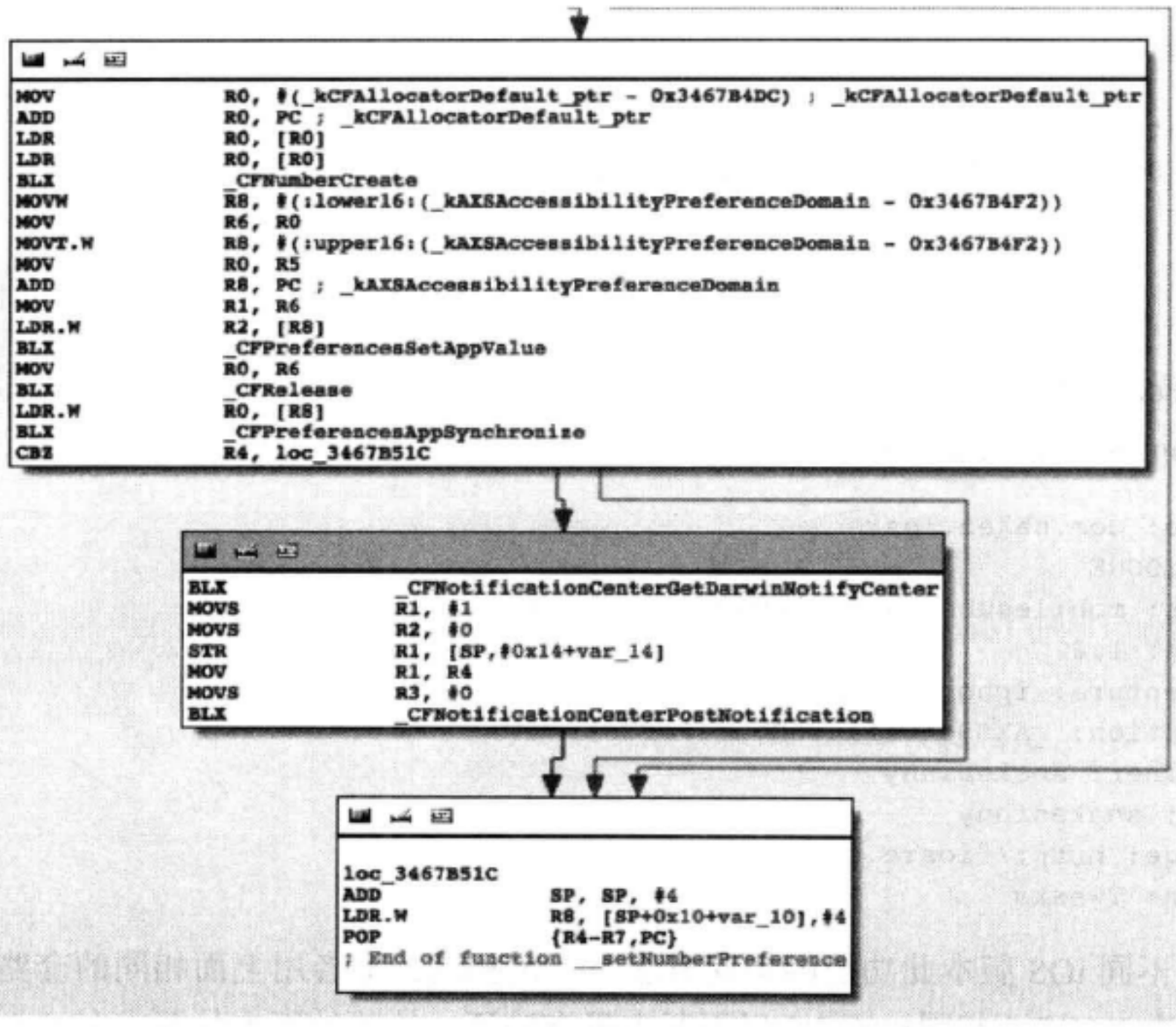


图 6-27　_AXSSetDefaultRouteForCall 片段

R0 被“LDR.W R0, [R8]”赋值后，变成了“kAXSAccessibilityPreferenceDomain”，也不像返回值；紧接着是一系列通用的 CF 操作，即使修改了 R0，也应该跟 Accessibility 毫无关系；因此，基本可以下定论了：_AXSSetDefaultRouteForCall 没有返回值。

6.2.2　验证分析结果

综上所述，修改来电默认接听方式实际只需调用一个 void _AXSSetDefaultRouteForCall(int route) 即可，其中“0 对应 default，1 对应 headset，2 对应 speaker”。这么简单？耳听为虚，眼见为实，试试便知（因为 Cycript 对 C 函数的支持度不高，所以此处用 Theos 测试）。

1. 编写 Tweak.xm

编写的 Tweak.xm 如下：

```
static int route = 0;

extern "C" void _AXSSetDefaultRouteForCall(int route); // 私有函数，我们不知道该导入
哪个 .h，所以就直接把它定义成 extern，好通过编译
```

```
%hook SpringBoard

- (void)menuButtonDown:(id)down
{
    %orig;

    route = route == 0 ? 2 : 0;
    _AXSSetDefaultRouteForCall(route);
}

%end
```

2. 编辑 control

编辑 control 的情况如下：

```
Package: com.naken.iosre
Name: iOSRE
Depends: mobilesubstrate, firmware (>= 5.0)
Version: 1.0
Architecture: iphoneos-arm
Description: _AXSSetDefaultRouteForCall 测试
Maintainer: snakeninny
Author: snakeninny
Homepage: http://iosre.com
Section: Tweaks
```

注意，不同 iOS 版本此功能的实现方法不一定相同，笔者用上面相同的套路验证了 iOS 5 的实现方法与 iOS 6 相同；iOS 5 以下版本没有验证，故此处版本号要求 iOS 5 及以上，特此说明。

3. 编辑 iOSRE.plist

设置 Filter 为 com.apple.springboard。

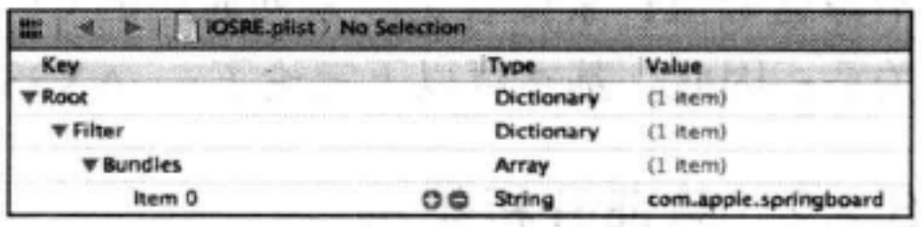


图 6-28 iOSRE.plist

4. 编辑 Makefile

来看看编辑 Makefile 的情况：

```
THEOS_DEVICE_IP = 192.168.1.102 # 换成你的 IP
ARCHS = armv7
TARGET = iphone:6.1:4.3

include theos/makefiles/common.mk

TWEAK_NAME = iOSRE
iOSRE_FILES = Tweak.xm
iOSRE_LDFLAGS = -lAccessibility # 还记得是什么意思吧?

include $(THEOS_MAKE_PATH)/tweak.mk
```

```
after-install::
    install.exec "killall -9 SpringBoard"
```

好了，现在大家自己编译安装测试一下吧！

现在已经知道怎么设置来电默认接听方式了，不过，在根据 SBSettings toggle 的编写格式，彻底升级 Speaker SBSettings Toggle 之前，还有一个亟待解决的问题：怎么得到来电默认接听方式？有一种方式是从头至尾彻底再寻找一次线索，另一种方式是根据 setter 和 getter 的命名法则来分析，如果 setter 的名称是 _AXSSetDefaultRouteForCall，那么 getter 的名称就是 _AXSDefaultRouteForCall。

现在布置一难一易两个练习，读者可以根据自己对前面知识的掌握情况，选一个做：

（1）从头开始分析，找到来电默认接听方式的查询函数；

（2）从 _AXSDefaultRouteForCall 开始，分析它的参数和返回值。

“Speaker SBSettings Toggle”升级后的最终 Toggle.mm 文件如下：

```
extern "C" void _AXSSetDefaultRouteForCall(int route);
extern "C" int _AXSDefaultRouteForCall(void);

// 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()
{
    // :)
    return _AXSDefaultRouteForCall() == 0 ? NO : YES;
}

// 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)
{
    // :)
    _AXSSetDefaultRouteForCall(enabled ? 2 : 0);
}

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

```
extern "C" float getDelayTime()
{
    return 0.6f;
}
```

以上就是用 IDA 分析目标文件的大致方法，学会了这个套路，万变不离其宗。

6.3 在 GDB 中分析目标文件

对于本书的部分读者来说，6.2 节会为你开启 iOS 逆向工程的另一扇门。IDA 简直是太强大了，配合 ARM 指令集文档，似乎已经达成了“IDA 在手，天下我有”的境界；你是不是已经迫不及待，想要废寝忘食地开始实践刚学到的新知识了呢?

先别急。依据笔者的个人经验，在度过了一个短暂的新鲜期后，就会意识到，阅读大量的 ARM 汇编代码是一件有些枯燥的事；当你寻找的功能横跨若干函数、程序的逻辑比较复杂、变量和参数不能望“名”生义时，你会希望精简 IDA 给出的信息，只保留目标功能的实现逻辑就好。此时，用 GDB 跟踪代码执行流程，可以帮你从乱麻中理出一些头绪来，让你跟没必要的工作量说拜拜!

道高一尺，魔高一丈。有时候我们想要分析的目标是加过密的，直接拖到 IDA 里显示出的是一堆乱码，6.2 节的技术变成了战斗力只有 5 的渣渣。例如从 App Store 下载的 App，必须用 AppCrackr 之类的软件脱壳后才能用 IDA 进行分析，而 AppCrackr 脱壳的核心操作是由 GDB 完成的。

而且有时候 IDA 并不能把我们需要的信息全部显示出来，例如 MSHookFunction() 和 MSHookMessageEx() 的参数，真是一个细节难倒英雄汉，这时 GDB 也许就是我们的大救星。

总之，IDA 不是万能的，在 IDA 束手无策时，GDB 往往能帮上我们的忙。因为 GDB 的用法我们已经在第 4 章里介绍过了，所以下面我们以几个短例示范一下 GDB 的常用场合，不求面面俱到，只是期望提供灵感，让读者能够活学活用。

6.3.1 跟踪代码执行流程

1. 判断分支走向

还记得图 6-10 中 [AccessibilitySettingsController accessibilitySetPreference:specifier:] 的实现流程吗? 这个函数内部含有多个分支指令，每条分支跳转到哪个方向，是我们“望文生义”，推测出来的，虽然最后达成了目标，但这种方法并不是 100% 保险的——只有亲眼看到程序是按这个流程执行的，才能放心，是时候让 GDB 出场了。

用 GDB 跟踪代码执行流程很简单，在每条分支指令上下断点，待程序停下来后单步执行，然后观察其后续指令的地址，这样就能知道分支走到了哪条路上，具体的操作如下。

第一条分支指令是“ BEQ loc_2520”，它的地址是 0x24f6。在它上面下一个断点，该操作的全部步骤如图 6-29 ～图 6-37 所示，希望你对它们还有印象。

```
MOV             R0, #(_kAXSInvertColorsEnabledPreference_ptr - 0x24E0) ; _kAXSInvertColorsEnabledPreference_ptr
MOVW            R1, #(:lower16:(selRef_isEqualToString_ - 0x24E8))
ADD             R0, PC ; _kAXSInvertColorsEnabledPreference_ptr
MOVT.W          R1, #(:upper16:(selRef_isEqualToString_ - 0x24E8))
LDR             R0, [R0] ; _kAXSInvertColorsEnabledPreference
ADD             R1, PC ; selRef_isEqualToString_
LDR             R5, [R1] ; "isEqualToString:"
MOV             R1, R5
LDR             R2, [R0]
MOV             R0, R4
BLX             _objc_msgSend
TST.W           R0, #0xFF
BEQ             loc_2520
```

图 6-29　定位分支指令

```
__text:000024F2  TST.W   R0, #0xFF
__text:000024F6  BEQ     loc_2520
__text:000024F8  MOV     R0, #(selRef__handleInvertColorsEnabled_ - 0x2504)
__text:00002500  ADD     R0, PC ; selRef__handleInvertColorsEnabled_
```

图 6-30　查看分支指令的地址

```
gdb$ attach Preferences
Attaching to process 10234.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ..........................................
..............................................................................
..............................................................................
.............. done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries +.. done
0x3af90e30 in mach_msg_trap ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ info sh
```

图 6-31　在 GDB 中 attach 进 Preferences

```
227 ManagedConfigurationUI                B 0x39f6d000        dyld Y Y /System/Li
brary/PreferenceBundles/ManagedConfigurationUI.bundle/ManagedConfigurationUI at
0x39f6d000 (offset 0x299c000)
                                                            (objfile is) [memory ob
ject "/System/Library/PreferenceBundles/ManagedConfigurationUI.bundle/ManagedCon
figurationUI" at 0x39f6d000]
228 AccessibilitySettings                 B 0x5058000         dyld Y Y /System/Li
brary/PreferenceBundles/AccessibilitySettings.bundle/AccessibilitySettings at 0x
5058000 (offset 0x5058000)
229 ScreenReaderOutput                    F 0x3369e000        dyld Y Y /System/Li
brary/PrivateFrameworks/ScreenReaderOutput.framework/ScreenReaderOutput at 0x336
9e000 (offset 0x299c000)
```

图 6-32　查看 AccessibilitySettings 的 ASLR 偏移

```
gdb$ b *(0x5058000 + 0x000024F6)
Breakpoint 1 at 0x505a4f6
gdb$ c

Breakpoint 1, 0x0505a4f6 in -[AccessibilitySettingsController accessibilitySetPr
eference:specifier:] ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0505a520 in -[AccessibilitySettingsController accessibilitySetPreference:speci
fier:] ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0505a524 in -[AccessibilitySettingsController accessibilitySetPreference:speci
fier:] ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 6-33　下断点并单步执行

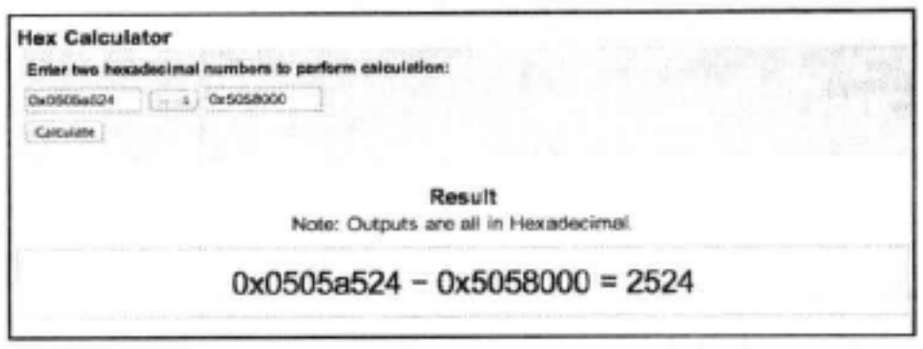


图 6-34 计算单步执行后的地址

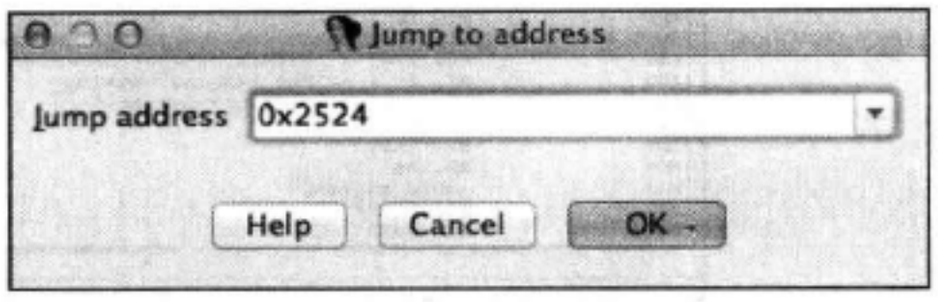


图 6-35 跳转到计算出的地址

```
loc_2520
MOVW            R0, #(:lower16:(_kAXSSpeakCorrectionsEnabledPreference_ptr - 0x252E))
MOV             R1, R5
MOVT.W          R0, #(:upper16:(_kAXSSpeakCorrectionsEnabledPreference_ptr - 0x252E))
ADD             R0, PC ; _kAXSSpeakCorrectionsEnabledPreference_ptr
LDR             R0, [R0] ; _kAXSSpeakCorrectionsEnabledPreference
LDR             R2, [R0]
MOV             R0, R4
BLX             _objc_msgSend
TST.W           R0, #0xFF
BEQ             loc_2548
```

图 6-36 跳转到新地址处的指令

```
__text:00002520 loc_2520                                ; CODE XREF: -[AccessibilitySettingsController
__text:00002520                 MOVW            R0, #(:lower16:(_kAXSSpeakCorrectionsEnabledPreference
__text:00002524                 MOV             R1, R5
__text:00002526                 MOVT.W          R0, #(:upper16:(_kAXSSpeakCorrectionsEnabledPreference
__text:0000252A                 ADD             R0, PC ; _kAXSSpeakCorrectionsEnabledPreference_ptr
```

图 6-37 验证地址

所有的分支指令都可以用上面的步骤来判断走向，从全部岔路中挑出系统实际走的那一条路，然后专门分析这一条路，就能大大提高效率。怎么样，很实用吧？

2. 寻找函数调用者

在判断分支走向时，是顺着程序的执行逻辑，从上往下跟踪的；但很多时候，需要从下往上，来倒推出函数的调用者，如下例。

当 iOS 处于桌面时，按下锁屏键，[SBAwayController dimScreen:] 会得到调用，如果想找出它的调用者，要怎么办呢？在 IDA 里搜索 dimScreen:，得到的结果很多，如图 6-38 所示。

Function	Instruction
-[SpringBoard _proximityChanged:]	LDR R1, [R0] ; "dimScreen:"
-[SBAwayController frontLocked:withAnimation:autom...	LDR R1, [R0] ; "dimScreen:"
-[SBAwayController attemptUnlockFromSource:]	LDR R1, [R0] ; "undimScreen:"
-[SBAwayController relockForButtonPress:afterCall:]	LDR R1, [R0] ; "dimScreen:"
-[SBAwayController dimScreen:]	; SBAwayController - (void)dimScreen:(char)
-[SBAwayController dimScreen:]	; End of function -[SBAwayController dimScreen:]
-[SBAwayController undimScreen]	LDR R1, [R1] ; "undimScreen:"
-[SBAwayController undimScreen:]	; SBAwayController - (void)undimScreen:(char)
-[SBAwayController undimScreen:]	; End of function -[SBAwayController undimScreen:]
-[SBAwayController activate]	LDR R1, [R0] ; "dimScreen:"
-[SBAwayController activate]	LDR R1, [R0] ; "dimScreen:"
-[SBAwayController activateAlertItem:]	LDR R1, [R1] ; "activateAlertItem:didUndimScreen:"
-[SBUserAgent dimScreen:]	; SBUserAgent - (void)dimScreen:(char)
-[SBUserAgent dimScreen:]	; End of function -[SBUserAgent dimScreen:]
-[SBAwayBulletinListController _alertForBulletin:]	LDR R1, [R1] ; "dimScreen:"

图 6-38 SpringBoard 内调用 [SBAwayController dimScreen:] 的地方

其中，[SBAwayController frontLocked:withAnimation:automatically:disableLockSound:]、[SBAwayController relockForButtonPress:afterCall:]、[SBAwayController activate] 和 [SBAwayBulletinListController _alertForBulletin:] 都调用了 [SBAwayController dimScreen:]，但单从函数名来看，不太能确定每个函数的作用，它们都有可能；前两个函数含有关键词 lock，更加让人怀疑。难道要逐个分析再排除？有了 GDB 当然不用，真相只有一个！

用 GDB 寻找函数调用者比跟踪流程还要简单，只需要在被调用函数的结尾处下一个断点，待程序停下来后单步执行，就会进入其调用者的地址范围，进而找出它的调用者。具体的操作如下。

最后一条指令是“POP {R4-R7,PC}”，它的地址是 0x457C6。在它上面下一个断点，该操作的全部步骤如图 6-39 ～图 6-48 所示。

```
loc_457C0
ADD             SP, SP, #0xC
POP.W           {R8,R10}
POP             {R4-R7,PC}
; End of function -[SBAwayController dimScreen:]
```

图 6-39　定位函数结尾处的指令

```
__text:000457C0 loc_457C0                                   ; CODE XREF
__text:000457C0                 ADD             SP, SP, #0xC
__text:000457C2                 POP.W           {R8,R10}
__text:000457C6                 POP             {R4-R7,PC}
__text:000457C6 ; End of function -[SBAwayController dimScreen:]
```

图 6-40　查看指令地址

```
gdb$ attach SpringBoard
Attaching to process 10390.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ..........................................
................................................................................
................................................................................
```

图 6-41　在 GDB 中 attach 进 SpringBoard

```
gdb$ info sh
The DYLD shared library state has not yet been initialized.
                                                Requested State Current Stat
e
Num Basename                                Type Address         Reason I I Source

  I I                                          I I                   I I I I

  1 SpringBoard                                - 0x2c000             exec Y Y /System/Li
brary/CoreServices/SpringBoard.app/SpringBoard at 0x2c000 (offset 0x2b000)
```

图 6-42　查看 SpringBoard 的 ASLR 偏移

```
gdb$ b *(0x2b000 + 0x457C6)
Breakpoint 1 at 0x707c6
gdb$ c

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x440e0000
0x440e0000 in ?? ()
```

图 6-43　下断点并触发断点

GDB 的 bug 把 SpringBoard 搞崩了，算我们倒霉（依据笔者的经验，断点最好不要下在

POP 指令和 B 指令上，容易触发 bug）。不过没关系，把断点稍微前移一些就好，如图 6-44 所示（因 SpringBoard 重启，其 ASLR 偏移发生变化，此处已重新获取）。

```
gdb$ b *(0xca000 + 0x457C0)
Breakpoint 1 at 0x10f7c0
gdb$ c

Breakpoint 1, 0x0010f7c0 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0010f7c0 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a2769c in dyld_stub_object_getClass ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0010fe64 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0010fe68 in _mh_execute_header ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 6-44 重新下断点并单步执行

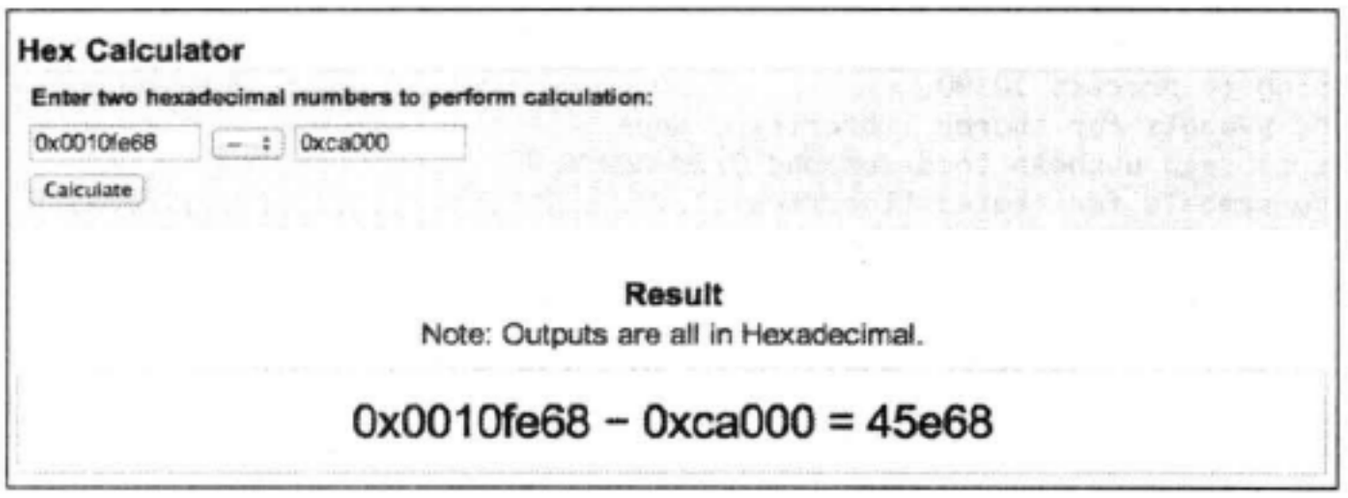


图 6-45 计算单步执行后的地址

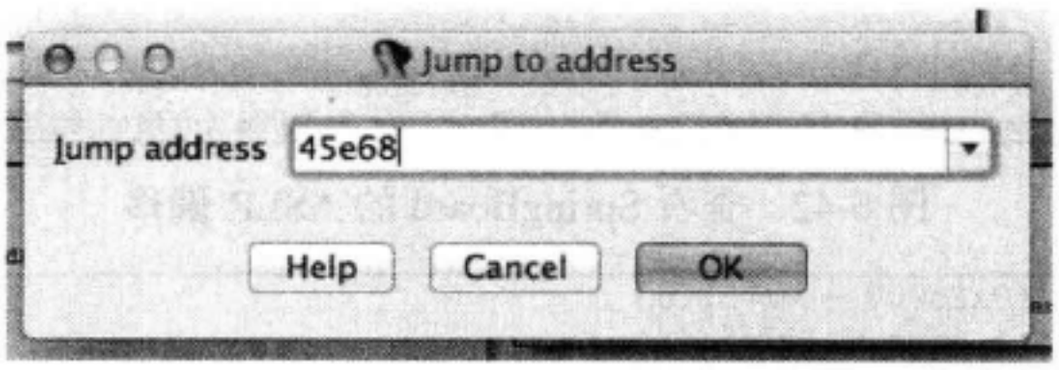


图 6-46 跳转到计算出的地址

这样就找到了当前环境下 dimScreen: 的调用者，既不是 frontLocked:withAnimation:automatically:disableLockSound:，也不是 relockForButtonPress:afterCall:，而是 activate。值得一提的是，当图 6-45 得到的地址跳转失败时，则一般说明这个函数的调用者来自外部文件，常用办法就是“grep”一下全系统，找到可疑的二进制文件后再继续分析。

```
MOVW            R0, #(:lower16:(selRef_dimScreen_ - 0x45E60))
MOVS            R2, #0
MOVT.W          R0, #(:upper16:(selRef_dimScreen_ - 0x45E60))
MOV.W           R8, #0
ADD             R0, PC ; selRef_dimScreen_
LDR             R1, [R0] ; "dimScreen:"
MOV             R0, R4
BLX             _objc_msgSend
LDR             R0, [R4,R5]
BIC.W           R0, R0, #0x80
STR             R0, [R4,R5]
B               loc_45FFC
```

图 6-47　跳转到新地址处的指令

```
MOVW            R0, #(:lower16:(se
MOVS            R2, #0
MOVT.W          R0, #(:upper16:(se
MOV.W           R8, #0
ADD             R0, PC ; selRef_di
LDR             R1, [R0] ; "dimScr
MOV             R0, R4
BLX             _objc_msgSend
LDR             R0, [R4,R5]
BIC.W           R0, R0, #0x80
STR             R0, [R4,R5]
B               loc_45FFC
```

```
From:   -[SBAwayController activate]+54

; SBAwayController - (void)activate
; Attributes: bp-based frame

; void __cdecl -[SBAwayController activate](struct
__SBAwayController_activate_

var_1C= -0x1C
var_18= -0x18

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
PUSH.W          {R8,R10}
SUB             SP, SP, #8
MOV             R4, R0
```

图 6-48　验证地址

6.3.2　动态更改代码

为什么要动态更改代码？可能的原因有很多，其中之一是因为有些时候，你想要调试的代码需要满足一定的条件才能触发执行，而这种条件不借助外界力量很难重现，所以可以动态更改代码，把程序引向你的目标代码，从而调试它们。这句话听起来很拗口，有读者可能会问，真有这种情况发生吗？下面就举一个真实的例子，介绍笔者在 iOS 逆向工程中是怎么用到 GDB 动态更改代码功能的。

通话录音功能是很多 iOS 开发者最想要实现的功能之一，笔者也不例外；但其难度如何，相信研究过的人们都明白。Audio Recorder 是世面上第一款通话录音软件，借鉴它的技术实现比我们从头开始分析电话录音这个功能要容易得多；Audio Recorder 的核心部分是一个 tweak，我们最感兴趣的便是它 hook 了哪些函数，就从它入手吧。

首先，定位 hook 函数——基于 MobileSubstrate 的 tweak 都会调用 MSHookFunction 和 MSHookMessageEx 来进行 hook，而 Audio Recorder 的作者 Elias Limneos 又是一位用惯了 MobileSubstrate 的开发者，因此把 callrecorder.dylib 拖进 IDA，在 Function window 中找到 MSHookFunction，并显示它的 xref，如图 6-49 所示。

5 处 MSHookFunction 都发生在 sub_3fbc 中，那么到 sub_3fbc 里看看——里面又是判

断设备型号又是判断进程名称，很明显，这里应该就是定义所有 hook 的地方；进一步查看 MSHookMessageEx 的 xref，也都是来自 sub_3fbc，如图 6-50 所示，这就验证了我们的推断。

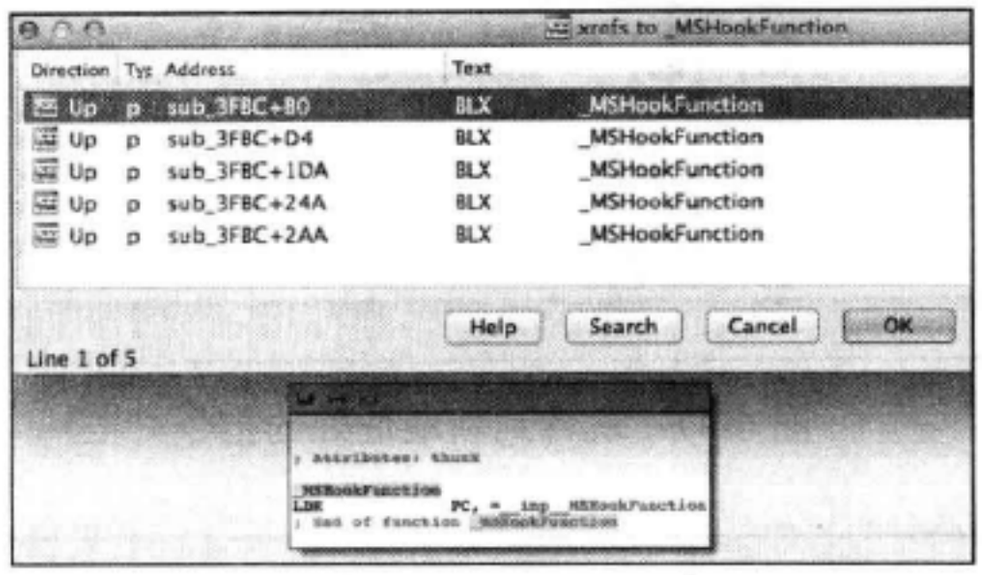


图 6-49 Audio Recorder 的所有 MSHookFunction

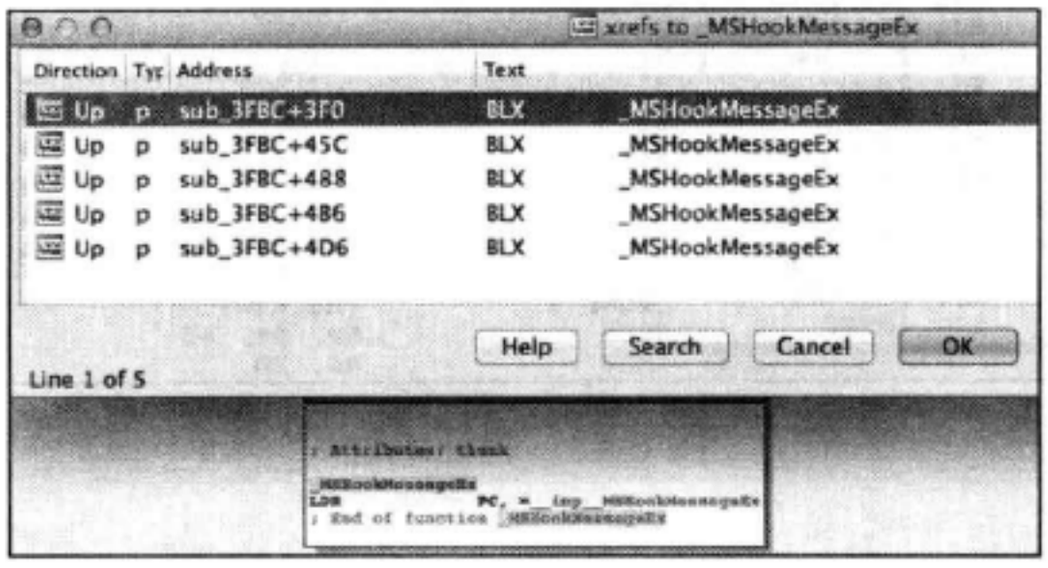


图 6-50 Audio Recorder 的所有 MSHookMessageEx

逐一浏览这 5 个 MSHookFunction，发现最后两个 MSHookFunction 的参数来自于 dlsym()，而 IDA 没有显示 dlsym() 的参数，如图 6-51 所示。出大事了！这就意味着我们不知道 MSHookFunction 到底 hook 了哪个函数；这么重要的信息如果不搞清楚，是没法往下分析的。

嘿嘿，还好我们有两手准备，静态分析没辙，咱们就整动态的，上 GDB。dlsym() 的第二个参数是一个字符串，刚好可以用 GDB 打印出来，那么在 dlsym 上下个断点，然后“print”一下不就解决问题了么？真是天助我也！

但是别急，首先得弄清楚怎么触发这个断点——sub_3fbc 是怎样得到调用的，以及怎样才能让它执行到 dlsym 处。我们看看 sub_3fbc 的调用者是谁，打开它的 xref 窗口，如图 6-52 所示。是 _mod_init_func，即这个 tweak 的构造函数。也就是说，当这个 dylib 被 dlopen 时，sub_3fbc 得到执行。

搞清了 sub_3fbc 的调用时机，接着分析 sub_3fbc 要经过哪些分支才能到达 dlsym。因为现在是知道了终点要推理路线，所以倒着来，从最后一个 MSHookFunction 开始，如图 6-53 所示。

这段路没有分岔，是直着过来的，只不过loc_423e会循环执行，可能是动态生成dlsym()的参数？不管它，接着往上走，这样就到了图6-54处。

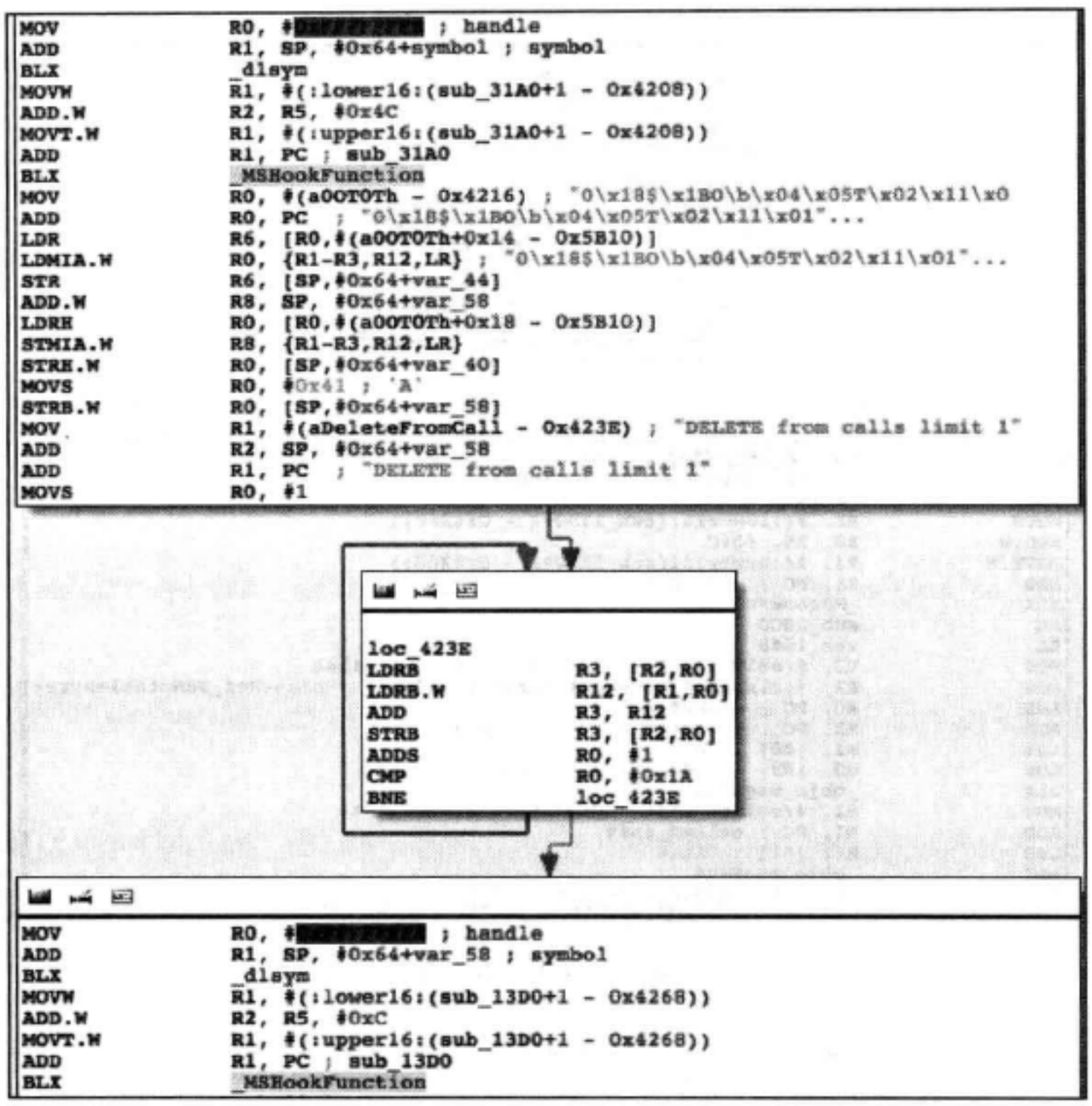


图6-51　dlsym()参数未知

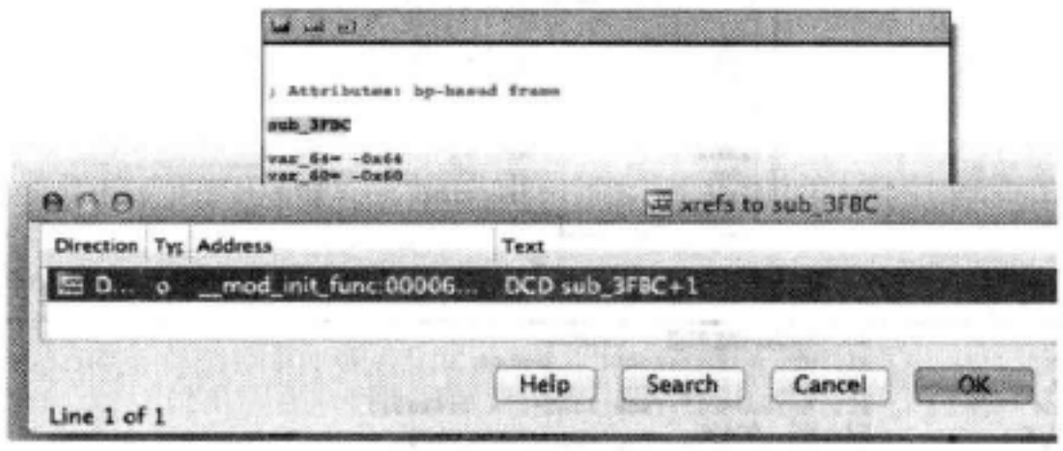


图6-52　sub_3fbc的xref

看上去跟上面仍然类似，继续往上，到了图6-55处。

这里出现了分岔，loc_4006执行完后走左边的路才能最终到达dlsym()。因为刚才在IDA部分已经详细解读了用IDA分析目标文件的方法，这里就不再手把手教了，只简单阐述程序逻辑，作为你的参考答案。当[[[NSProcessInfo processInfo] processName] isEqual:@" mediaserverd"]返回YES时，dlsym()才会得到执行；也就是说，MobileSubstrate在轮询进程

```
STMIA.W         R8, {R1-R3,R12,LR}
STRH.W          R0, [SP,#0x64+var_40]
MOVS            R0, #0x41 ; 'A'
STRB.W          R0, [SP,#0x64+var_58]
MOV             R1, #(aDeleteFromCall - 0x423E) ; "DELETE from calls limit 1"
ADD             R2, SP, #0x64+var_58
ADD             R1, PC  ; "DELETE from calls limit 1"
MOVS            R0, #1
```

```
loc_423E
LDRB            R3, [R2,R0]
LDRB.W          R12, [R1,R0]
ADD             R3, R12
STRB            R3, [R2,R0]
ADDS            R0, #1
CMP             R0, #0x1A
BNE             loc_423E
```

```
MOV             R0, #[redacted] ; handle
ADD             R1, SP, #0x64+var_58 ; symbol
BLX             _dlsym
MOVW            R1, #(:lower16:(sub_13D0+1 - 0x4268))
ADD.W           R2, R5, #0xC
MOVT.W          R1, #(:upper16:(sub_13D0+1 - 0x4268))
ADD             R1, PC ; sub_13D0
BLX             _MSHookFunction
BL              sub_29C0
BL              sub_1648
MOV             R0, #(selRef_alloc - 0x4286) ; selRef_alloc
MOV             R2, #(classRef_NSMutableArray - 0x4288) ; classRef_NSMutableArray
ADD             R0, PC ; selRef_alloc
ADD             R2, PC ; classRef_NSMutableArray
LDR             R1, [R0] ; "alloc"
LDR             R0, [R2] ; _OBJC_CLASS_$_NSMutableArray
BLX             _objc_msgSend
MOV             R1, #(selRef_init - 0x429A) ; selRef_init
ADD             R1, PC ; selRef_init
LDR             R1, [R1] ; "init"
BLX             _objc_msgSend
STR             R0, [R5,#(dword_6E20 - 0x6DD0)]
```

图 6-53 sub_3fbc 片段

```
STRB.W          R0, [SP,#0x64+symbol]
MOV             R1, #(aNet_limneos_10 - 0x41DE) ; "net.limneos.callrecorder.updateRec"
ADD             R2, SP, #0x64+symbol
ADD             R1, PC  ; "net.limneos.callrecorder.updateRec"
MOVS            R0, #1
```

```
loc_41DE
LDRB            R3, [R2,R0]
LDRB.W          R12, [R1,R0]
ADD             R3, R12
STRB            R3, [R2,R0]
ADDS            R0, #1
CMP             R0, #0x23 ; '#'
BNE             loc_41DE
```

```
MOV             R0, #[redacted] ; handle
ADD             R1, SP, #0x64+symbol ; symbol
BLX             _dlsym
MOVW            R1, #(:lower16:(sub_31A0+1 - 0x4208))
ADD.W           R2, R5, #0x4C
MOVT.W          R1, #(:upper16:(sub_31A0+1 - 0x4208))
ADD             R1, PC ; sub_31A0
BLX             _MSHookFunction
MOV             R0, #(a0OTOTh - 0x4216) ; "0\x18$\x1B0\b\x04\x05T\x02\x11\x0
ADD             R0, PC  ; "0\x18$\x1B0\b\x04\x05T\x02\x11\x01"...
LDR             R6, [R0,#(a0OTOTh+0x14 - 0x5B10)]
LDMIA.W         R0, {R1-R3,R12,LR} ; "0\x18$\x1B0\b\x04\x05T\x02\x11\x01"...
STR             R6, [SP,#0x64+var_44]
ADD.W           R8, SP, #0x64+var_58
LDRH            R0, [R0,#(a0OTOTh+0x18 - 0x5B10)]
STMIA.W         R8, {R1-R3,R12,LR}
STRH.W          R0, [SP,#0x64+var_40]
MOVS            R0, #0x41 ; 'A'
STRB.W          R0, [SP,#0x64+var_58]
MOV             R1, #(aDeleteFromCall - 0x423E) ; "DELETE from calls limit 1"
ADD             R2, SP, #0x64+var_58
ADD             R1, PC  ; "DELETE from calls limit 1"
MOVS            R0, #1
```

图 6-54 sub_3fbc 片段

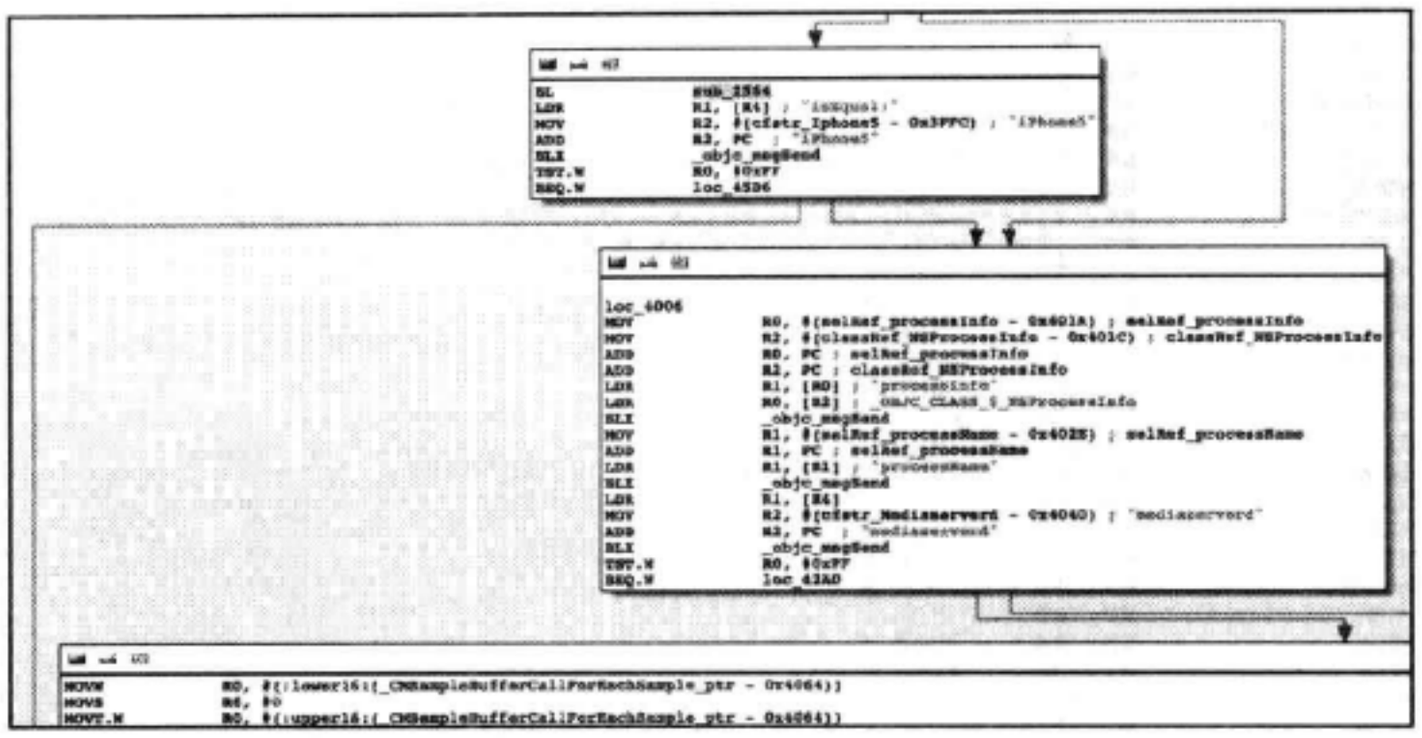


图 6-55　sub_3fbc 片段

时，只有碰到了 mediaserverd，才会把左边这条路的代码注入其中运行起来。继续往上走，看图 6-56。

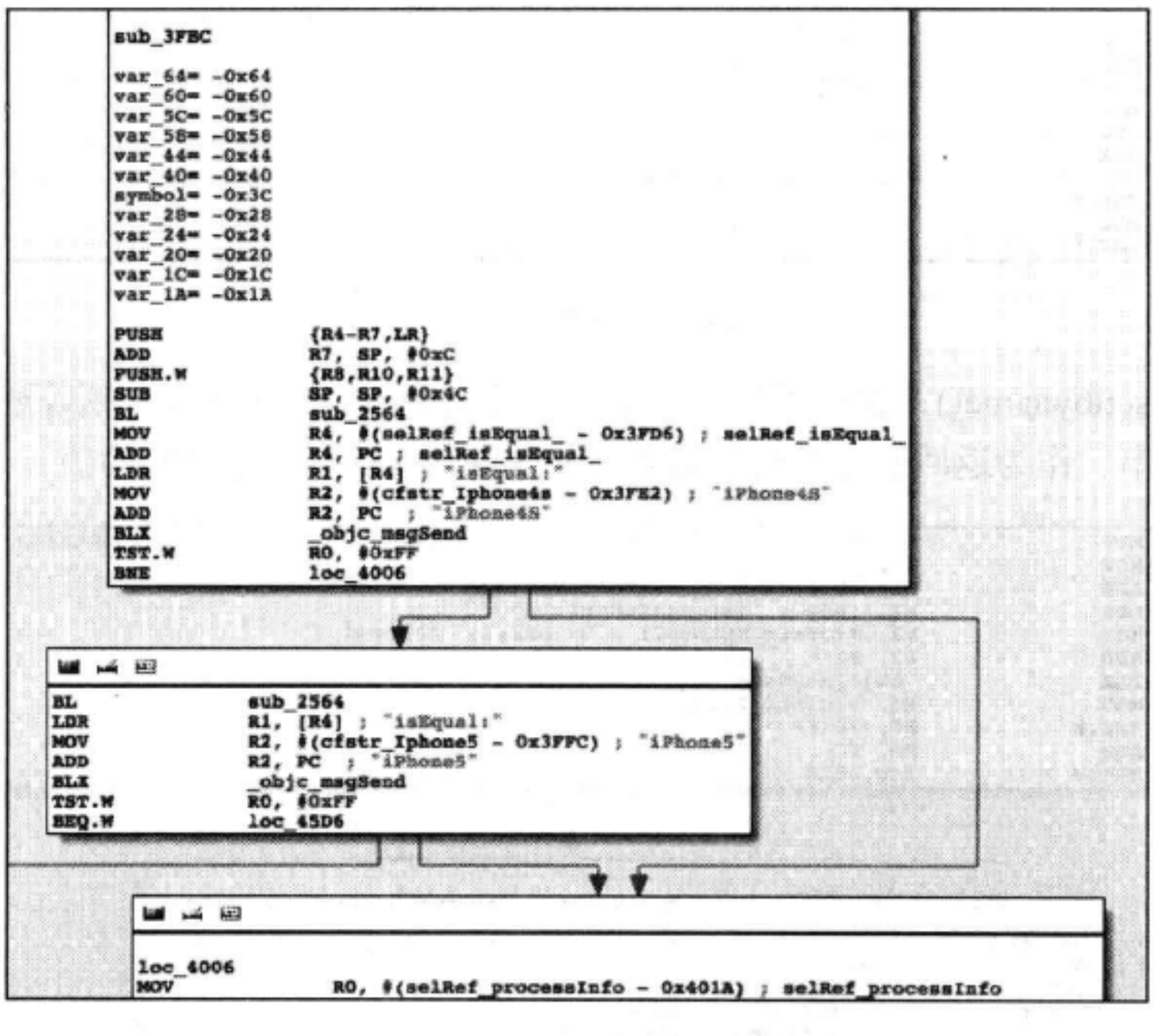


图 6-56　sub_3fbc 片段

当 [R0 isEqual:@ "iPhone4S"] 或 [R0 isEqual:@ "iPhone5"] 时，才会执行 loc_4006。那么 R0 里是什么？在两次 objc_msgSend 前都调用了 sub_2564，我们看看 sub_2564 干了什么，有没有返回值，如图 6-57 所示。

```
PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
PUSH.W          {R8,R10}
SUB             SP, SP, #0x20 ; size_t
MOVW            R4, #(:lower16:(aHw_machine - 0x257E)) ; "hw.machine"
MOVS            R5, #0
MOVT.W          R4, #(:upper16:(aHw_machine - 0x257E)) ; "hw.machine"
STR             R5, [SP,#0x34+var_34] ; size_t
ADD             R4, PC  ; "hw.machine"
ADD             R6, SP, #0x34+var_28
MOVS            R1, #0  ; void *
MOVS            R3, #0  ; void *
MOV             R0, R4  ; char *
MOV             R2, R6  ; size_t *
BLX             _sysctlbyname
LDR             R0, [SP,#0x34+var_28] ; size_t
BLX             _malloc
STR             R5, [SP,#0x34+var_34] ; size_t
MOV             R2, R6  ; size_t *
MOVS            R3, #0  ; void *
MOV             R8, R0
MOV             R0, R4  ; char *
MOV             R1, R8  ; void *
BLX             _sysctlbyname
MOVW            R0, #(:lower16:(selRef_stringWithCString_encoding_ - 0x25B6))
MOVS            R3, #4
MOVT.W          R0, #(:upper16:(selRef_stringWithCString_encoding_ - 0x25B6))
MOV             R2, #(classRef_NSString - 0x25B8) ; classRef_NSString
ADD             R0, PC ; selRef_stringWithCString_encoding_
ADD             R2, PC ; classRef_NSString
LDR             R1, [R0] ; "stringWithCString:encoding:"
LDR             R0, [R2] ; _OBJC_CLASS_$_NSString
MOV             R2, R8
BLX             _objc_msgSend
MOV             R4, R0
MOV             R0, R8  ; void *
BLX             _free
MOV             R5, #(selRef_isEqualToString_ - 0x25D6) ; selRef_isEqualToString_
MOV             R0, R4
ADD             R5, PC ; selRef_isEqualToString_
LDR             R1, [R5] ; "isEqualToString:"
MOV             R2, #(cfstr_Iphone11 - 0x25E2) ; "iPhone1,1"
ADD             R2, PC  ; "iPhone1,1"
BLX             _objc_msgSend
MOV             R6, #(cfstr_Iphone1g - 0x25F4) ; "iPhone 1G"
TST.W           R0, #0xFF
ADD             R6, PC  ; "iPhone 1G"
BNE.W           loc_27CA
```

图 6-57 sub_2564 片段

它调用 sysctlbyname()，获取了 hw.machine，即 iDevice 设备型号。然后把系统代码转化为自定义字符串，作为返回值传递给调用者，如图 6-58 所示（注意 R0 的值）。

```
MOV             R5, #(selRef_isEqualToString_ - 0x25D6) ; selRef_isEqualToString_
MOV             R0, R4
ADD             R5, PC ; selRef_isEqualToString_
LDR             R1, [R5] ; "isEqualToString:"
MOV             R2, #(cfstr_Iphone11 - 0x25E2) ; "iPhone1,1"
ADD             R2, PC  ; "iPhone1,1"
BLX             _objc_msgSend
MOV             R6, #(cfstr_Iphone1g - 0x25F4) ; "iPhone 1G"
TST.W           R0, #0xFF
ADD             R6, PC  ; "iPhone 1G"
BNE.W           loc_27CA
```

```
To:  sub_2564:loc_27CA

loc_27CA                R0, R6
MOV                     SP, SP, #0x20
ADD                     {R8,R10}
POP.W                   {R4-R7,PC}
POP
; End of function sub_2564
```

图 6-58 sub_2564

也就说，当 dlopen 这个 dylib 时，只有同时满足 iDevice 是 iPhone 4S 或 iPhone 5，且当前进程为 mediaserverd 这两个条件时，dlsym() 才得到调用。对于第一个条件，花钱就可以解决。但对于第二个条件，你要怎么破？当 MobileSubstrate 在开机 dlopen 这个 dylib 时，你

连 GDB 都来不及打开，这个时候根本没法下断点；当然，也可以自己写个程序来 dlopen 它，先用 GDB 在自己的程序里下断点，然后单步执行到 dylib 的构造函数里去；可是，这时 dylib 检测到当前进程不是 mediaserverd，不会执行到 dlsym()。傻眼了吧？不更改代码能行吗？还是上 GDB 吧。

先编写一个 tool（命令行工具）来 dlopen 目标 dylib，代码如下：

```
#include </usr/include/dlfcn.h>
int main(int argc, char **argv, char **envp)
{
    void *handle = dlopen("/Library/MobileSubstrate/DynamicLibraries/
callrecorder.dylib", RTLD_LAZY);
    dlclose(handle);
    return 0;
}
```

安装之后，就可以开始调试了，整体的思路是：在“objc_msgSend”上下断点，然后调用刚才写好的 tool，待 tweak 的构造函数开始执行之后（这一步是为了获取 dylib 的 ASLR 偏移），在 loc_4006 的分支判断条件“TST.W R0, #0xFF”下断点，然后“set $r0 = 1”，将程序引导至 dlsym() 这条路上；接着分别在两处 dlsym() 下断点，然后“p (char *)r1”打印出第二个参数，即可知道 hook 了哪个函数。全过程如图 6-59 ～图 6-69 所示。

```
[redacted]-FunMaker-5:~ root# gdb DlopenDylib
warning: unrecognized host cpusubtype 11, defaulting to host==armv7.
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4) (
Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin"...Reading symbols for shared libra
ries .. done

gdb$ b objc_msgSend
Function "objc_msgSend" not defined.
gdb$
```

图 6-59　用 GDB 启动 DlopenDylib

在第 4 章介绍 GDB 时，笔者曾说过因为调试对象在编译时的差异，GDB 不一定总能从调试对象中获取足够的调试信息，所以 b function 的方式并不总是能成功，这里就碰到了。没关系，可通过一个简单的方法绕过这个问题——先运行一遍程序，让 GDB 熟悉一下函数，然后再 b function，如图 6-60 所示。

之后继续运行程序，停在断点上，如图 6-61 所示。

这个断点是在 callrecorder.dylib 内部吗？注意，第一次运行时，GDB 一共有三次“Reading symbols for shared libraries”，而现在只出现了两次，说明有一个 dylib 还没有得到加载。可以通过“info sh”来验证，如图 6-62 所示。

```
gdb$ b objc_msgSend
Function "objc_msgSend" not defined.
gdb$ r
Reading symbols for shared libraries + done
Reading symbols for shared libraries +....+.....................................
. done
Reading symbols for shared libraries ..........................................
.......................................... done

Program exited normally.
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ b objc_msgSend
Cannot access memory at address 0x388c75c0
Breakpoint 1 at 0x388c75c0

gdb$
```

图 6-60 在 objc_msgSend 上下断点

```
gdb$ r
Reading symbols for shared libraries + done
Reading symbols for shared libraries + done

Breakpoint 1, 0x388c75c0 in objc_msgSend ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
```

图 6-61 断点被触发

```
gdb$ info sh
The DYLD shared library state has been initialized from the executable's shared
 library information.  All symbols should be present, but the addresses of some
 symbols may move when the program is executed, as DYLD may relocate library lo
ad addresses if necessary.
                                        Requested State Current State
Num Basename                 Type Address         Reason I I Source
  | |                           | |                   | | | |
  1 dyld                       - 0x2fecf000        dyld Y Y /usr/lib/dyld at 0x
2fecf000 (offset 0xcf000) with prefix "__dyld_"
  2 callrecorder.dylib         - -                 c-init Y Y /Library/MobileSubs
trate/DynamicLibraries/callrecorder.dylib at 0x10f000
```

图 6-62 验证 callrecorder.dylib 的地址

可见 callrecorder.dylib 还没有地址，即未加载进内存。用 c 命令来继续，直到第三次出现“Reading symbols for shared libraries”，如图 6-63 所示。

再 info sh 一次看看，如图 6-64 所示。

好了，callrecorder.dylib 已经得到加载了，且停在了 objc_msgSend 上，此时还未执行到判断进程名的代码部分。loc_4006 的分支判断条件“ TST.W R0, #0xFF”的地址是 0x4042，如图 6-65 所示。

所以，执行 b *(0x4042 + 0x5f000)，然后“c”停在断点上，如图 6-66 所示。

好了，动态更改代码的时机到了。set $r0 = 1 后输入 ni，看看代码执行到了哪里，如图 6-67 所示。

0x6304e – 0x5f000 = 0x404e，对照 IDA，可以看到，代码已经进入了 dlsym() 的那条路，

如图 6-68 所示。

```
Breakpoint 1, 0x388c75c0 in objc_msgSend ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$

Breakpoint 1, 0x388c75c0 in objc_msgSend ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$

Breakpoint 1, 0x388c75c0 in objc_msgSend ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$

Reading symbols for shared libraries ++ done

Breakpoint 1, 0x388c75c0 in objc_msgSend ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
```

图 6-63　出现最后一次 Reading symbols for shared libraries

```
 45 callrecorder.dylib          - 0x5f000          dyld Y Y /Library/MobileSubs
trate/DynamicLibraries/callrecorder.dylib at 0x5f000 (offset 0x5f000)
 46 liblockdown.dylib           - 0x36ac9000       dyld Y Y /usr/lib/liblockdow
n.dylib at 0x36ac9000 (offset 0xfd8000)
                                                  (objfile is) [memory object "/us
r/lib/liblockdown.dylib" at 0x36ac9000]
```

图 6-64　查看 callrecorder.dylib 的 ASLR 偏移

```
__text:00004006 loc_4006                                ; CODE XREF: sub_3FBC+2Cij
__text:00004006                 MOV     R0, #(selRef_processInfo - 0x401A) ; selRef_processInfo
__text:0000400E                 MOV     R2, #(classRef_NSProcessInfo - 0x401C) ; classRef_NSProcessInfo
__text:00004016                 ADD     R0, PC ; selRef_processInfo
__text:00004018                 ADD     R2, PC ; classRef_NSProcessInfo
__text:0000401A                 LDR     R1, [R0] ; "processInfo"
__text:0000401C                 LDR     R0, [R2] ; _OBJC_CLASS_$_NSProcessInfo
__text:0000401E                 BLX     _objc_msgSend
__text:00004022                 MOV     R1, #(selRef_processName - 0x402E) ; selRef_processName
__text:0000402A                 ADD     R1, PC ; selRef_processName
__text:0000402C                 LDR     R1, [R1] ; "processName"
__text:0000402E                 BLX     _objc_msgSend
__text:00004032                 LDR     R1, [R4]
__text:00004034                 MOV     R2, #(cfstr_Mediaserverd - 0x4040) ; "mediaserverd"
__text:0000403C                 ADD     R2, PC  ; "mediaserverd"
__text:0000403E                 BLX     _objc_msgSend
__text:00004042                 TST.W   R0, #0xFF
__text:00004046                 BEQ.W   loc_42A0
```

图 6-65　定位判断指令

```
gdb$ d
gdb$ b *(0x4042 + 0x5f000)
Breakpoint 2 at 0x63042
gdb$ c

Breakpoint 2, 0x00063042 in r_AudioUnitAddPropertyListener ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
```

图 6-66　下断点并触发

```
gdb$ set $r0 = 1
gdb$ ni
0x00063046 in r_AudioUnitAddPropertyListener ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
0x0006304a in r_AudioUnitAddPropertyListener ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
0x0006304e in r_AudioUnitAddPropertyListener ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
```

图 6-67 动态更改代码并单步执行

```
MOVW            R0, #(:lower16:(_CMSampleBufferCallForEachSample_ptr - 0x4064))
MOVS            R6, #0
MOVT.W          R0, #(:upper16:(_CMSampleBufferCallForEachSample_ptr - 0x4064))
MOV             R1, #(__Z33r_CMSampleBufferCallForEachSampleP20opaqueCMSampleBuffer
MOVW            R2, #(:lower16:(_o_CMSampleBufferCallForEachSample - 0x406C))
0000404E 0000404E: sub 3FBC+92
```

图 6-68 验证更改是否成功

剩下的事情就简单了，如图 6-69 所示。

```
gdb$ d
gdb$ b dlsym
Breakpoint 3 at 0x39131996
gdb$ c

Breakpoint 3, 0x39131996 in dlsym ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ p (char *)$r1
$1 = 0x2fdc053c "AudioConverterConvertComplexBuffer"
gdb$ c

Breakpoint 3, 0x39131996 in dlsym ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ p (char *)$r1
$2 = 0x2fdc0520 "AudioConverterGetProperty"
gdb$ c

Program exited normally.
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
```

图 6-69 在 dlsym 上下断点并打印参数

dlsym() 的断点被触发两次后，程序退出；GDB 将两个函数名打印出来，它们分别是 AudioConverterConvertComplexBuffer 和 AudioConverterGetProperty。任务完成!

6.4　小结

IDA 和 GDB 两大神器的作用当然不止本章所介绍的这些，它们的有效范围很广，小到分析 App，大到动手越狱，是两款万能的工具。不过，相信在 iOS 逆向工程的初级阶段，读者应用它们最多的场景应该跟本章的内容差不多，当然，熟练掌握它们以后，对 iOS 的理解一定会上升到一个新的层次；届时，读者就能举一反三，根据自己的需求摸索它们的新用法了。

本章的内容虽然有些艰深，不过认真学完本章的内容之后，读者就能根据自己的情况判断出来是知难而退，还是迎难而上了。无论如何，这是一个有意思的方向，能走多远则完全看个人的功底和兴趣了。

第四部分

实 战 篇

前面三个部分重点介绍了逆向工程的基础概念、工具应用和对应的理论，并穿插讲解了一些方法、实例，但都是针对某个工具或某个理论为主来应用的。在对前面的概念有所了解之后，读者也都已经清楚，必须将这些工具和技术有机地组合起来，才能更好地展开逆向工程。

相信前面那些零零散散的小实例无法让读者有一种酣畅淋漓的感觉，于是就有了这个实战篇，本篇共分为 4 章，逆向的目标分别是：

- 去除 OPlayer Lite 的广告
- WhatsApp 消息拦截
- Instagram 图片保存
- iOS 电话操作

接下来，就请进入本书最精彩的部分，通过一个个精彩实例去实际感受 iOS 逆向工程的强大火力。

第 7 章

实战 1：去除 OPlayer Lite 的广告

7.1 OPlayer Lite 简介

OPlayer 是一款由 olimsoft 出品的操作简单但功能强大的多媒体播放器，支持多种音频视频格式且播放画面流畅，是您出差、打针、排队、坐车、开会、无聊时看电影打发时间的利器。OPlayer 的界面如图 7-1 所示。

OPlayer Lite 是 OPlayer 的免费版本，总体功能变化不大，只是多了一个广告横幅，还明确告诉你，购买 Full 版的目的就是去掉这个大广告，如图 7-2 所示。



图 7-1 OPlayer 运行界面



图 7-2 播放时显示广告横幅

在欣赏电影时屏幕总有一块被挡着，终归让人不爽，而本章的任务就是自己动手多角度、全方位地去除这个广告横幅。

7.2　分析源代码结构，并定位广告横幅的代码

7.2.1　class-dump 获取 .h 文件

当前的 OPlayer Lite 版本为 2.0.11，可从 App Store 免费下载。用 AppCrackr 来帮 OPlayer Lite 脱壳的过程很简单，不再赘述。打开 OPlayer Lite.app，找到 OPlayer Lite 的可执行文件，如图 7-3 所示。

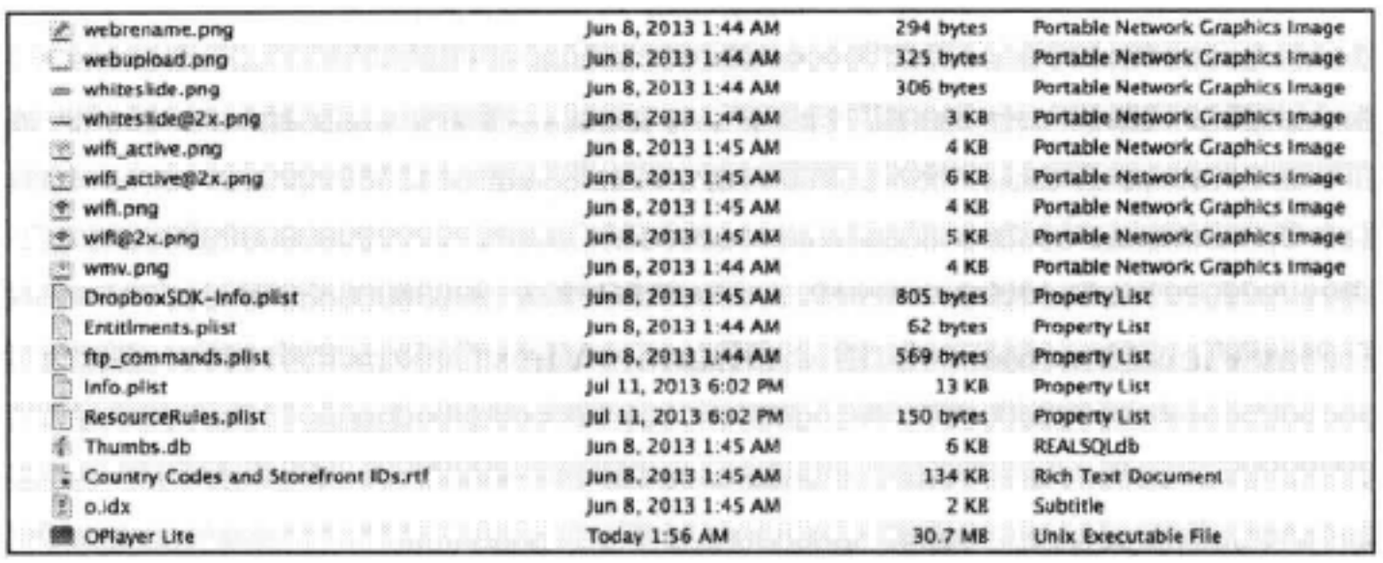


图 7-3　OPlayer Lite 目录文件

class-dump 获取 .h 文件：

```
class-dump-z -H -I OPlayer\ Lite -o /Path/To/OPlayer/Headers/
```

OPlayer Lite 的可执行文件有 30.7MB，共 class-dump 出 978 个 .h 文件，结构比较庞大，如图 7-4 所示。

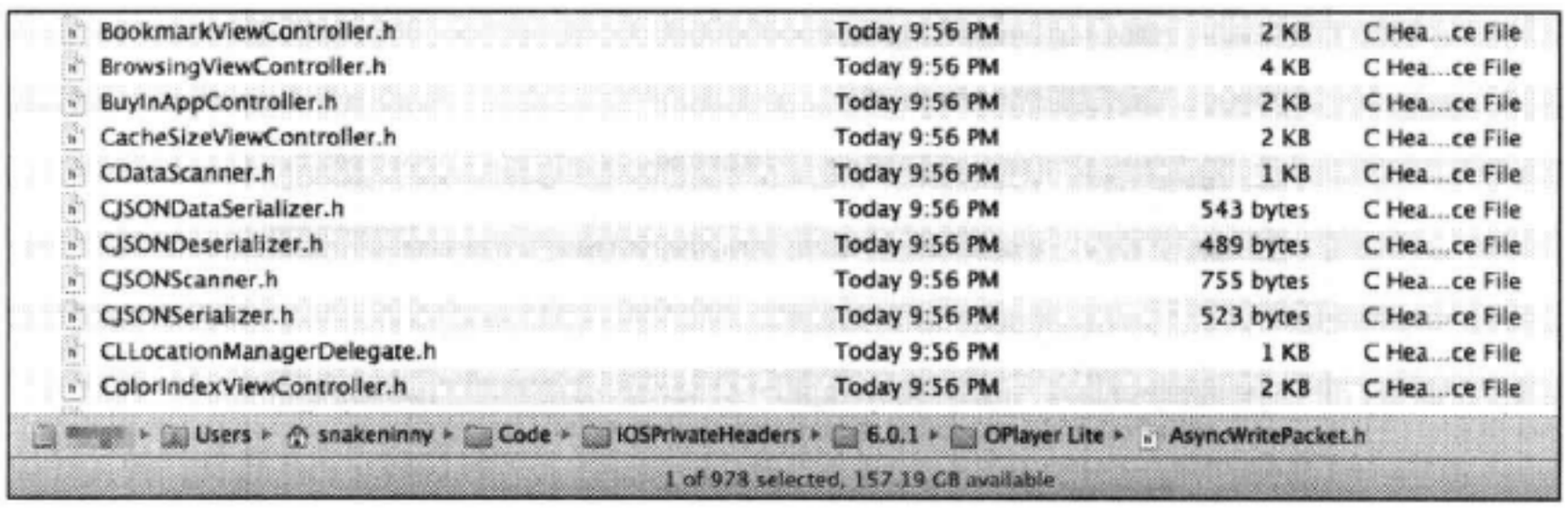


图 7-4　OPlayer Lite 的 .h 文件

7.2.2　猜测广告横幅的出现位置

既然 OPlayer 和 OPlayer Lite 的差异就在于有没有广告横幅，那么按常理来说，olimsoft 应该会把广告横幅做成一个单独的 UIView，而不是把它和播放界面整合在一起，因为这样会大大增加代码的复杂度，维护起来非常麻烦。既然是一个单独的 UIView，又是 iOS 中的广告，那么 iOS 开发者自然会想到 Apple 的移动广告平台 iAd；在 iAd.framework 中有一个重要的类，即 ADBannerView，正是 UIView 的一个子类。那么，尝试在 978 个 .h 文件中按

内容搜索 ADBannerView，看看能不能有所收获，如图 7-5 所示。

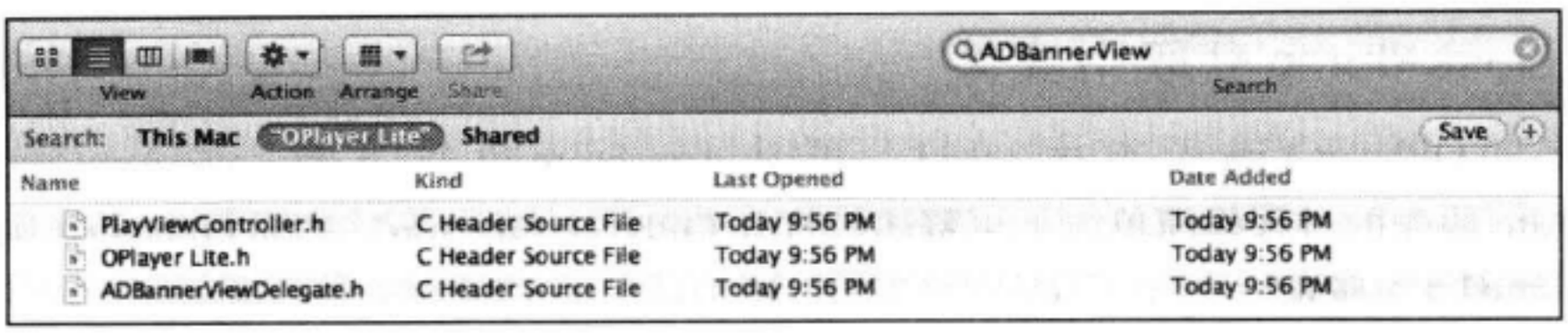


图 7-5 搜索 ADBannerView

总共就3条结果，特征也太明显了：OPlayer Lite.h 的内容是所有 .h 的文件名。ADBannerViewDelegate.h 是 ADBannerView 的标准协议；而对于 PlayViewController.h，看名字都能猜出就是播放界面的管理器。由这个搜索结果，已经可以猜个八九不离十：广告横幅的代码就隐藏在 PlayViewController.h 和 ADBannerViewDelegate.h 里，整个文件定位过程轻松愉快。

7.2.3 分析 PlayViewController.h

打开 PlayViewController.h，观察其内容，如图 7-6 所示。

```
 3  * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 4  *
 5  * Source: (null)
 6  */
 7
 8 #import "TencentSessionDelegate.h"
 9 #import "GADBannerViewDelegate.h"
10 #import "UIAlertViewDelegate.h"
11 #import "OPlayer Lite-Structs.h"
12 #import <XXUnknownSuperclass.h> // Unknown library
13 #import "WBEngineDelegate.h"
14 #import "WBLogInAlertViewDelegate.h"
15 #import "WBSendViewDelegate.h"
16 #import "ADBannerViewDelegate.h"
17
18 @class NSMutableArray, FloatingView, UIImageView, PlayerView, InfoViewControll
   er, ADBannerView, PlaySeekView, UINavigationController, MPVolumeView, AVPlayer
   , RoundedRectView, AVPlayerItem, UILabel, AVAssetReader, UIImage, GADBannerVie
   w, SubtitleLabel, NSDictionary, NSURL, WBEngine, NSTimer, AVPlayerDemoPlayback
   View, NSString, UIActivityIndicatorView, UIButton, UIView;
19
20 __attribute__((visibility("hidden")))
21 @interface PlayViewController : XXUnknownSuperclass <ADBannerViewDelegate, GAD
   BannerViewDelegate, UIAlertViewDelegate, TencentSessionDelegate, WBEngineDeleg
   ate, WBLogInAlertViewDelegate, WBSendViewDelegate> {
22 @private
23         NSString* url;
24         PlayerView* playerView;
25         UIButton* backButton;
26         UIButton* forwardButton;
27         UIButton* playPauseButton;
28         UIButton* prevButton;
29         UIButton* nextButton;
                                                        29,4-11       0%
```

图 7-6 PlayViewController.h 内容一览

只是粗略地扫一眼，已经觉得信息量太大了：@class 里的 ADBannerView、GADBannerView、SubtitleLabel；@interface 里 的 ADBannerViewDelegate、GADBannerViewDelegate；@private 里 的 backButton、forwardButton、playPauseButton、prevButton 和 nextButton 无 一 不表明这个 PlayViewController 就是播放界面的管理器，除非这些明显的字眼都是 olimsoft

在迷惑我们，不然在这个.h文件中一定能找到广告横幅的身影。浏览这个文件中带有ADBannerView关键词的字眼，很容易发现，第149行的iAdView很关键，如图7-7所示。

```
138         int currentConvertIndex;
139         BOOL hasCallConvertFull;
140         NSTimer* refreshConvertInfoTimer;
141         int playerIndex;
142         BOOL continueFromPreview;
143         int last_sync_time;
144         int convert_current_time;
145         BOOL nextConverReady;
146         BOOL isAbsSeeking;
147         float theRate;
148 }
149 @property(retain, nonatomic) ADBannerView* iAdView;
150 @property(retain, nonatomic) UIView* localAdView;
151 @property(retain, nonatomic) UILabel* statusInfoLabel;
152 @property(retain, nonatomic) UILabel* convertInfoLabel;
153 @property(retain, nonatomic) AVPlayerItem* mNextPlayerItem;
154 @property(retain, nonatomic) AVPlayer* mNextPlayer;
155 @property(retain, nonatomic) AVPlayerDemoPlaybackView* mNextPlaybackView;
156 @property(retain, nonatomic) AVPlayerDemoPlaybackView* mPlaybackView;
157 @property(retain) AVPlayerItem* mPlayerItem;
158 @property(retain, getter=player, setter=setPlayer:) AVPlayer* mPlayer;
159 @property(retain, nonatomic) WBEngine* weiBoEngine;
160 @property(retain, nonatomic) UIImage* sharedImage;
161 @property(retain, nonatomic) FloatingView* audioDelayFloatView;
162 @property(retain, nonatomic) FloatingView* speedControlFloatView;
163 @property(retain, nonatomic) FloatingView* leftSubDelayFloatView;
164 @property(retain, nonatomic) FloatingView* rightFloatView;
165 @property(retain, nonatomic) FloatingView* leftFloatView;
166 @property(retain, nonatomic) UIImageView* imageTopView;
167 @property(retain, nonatomic) RoundedRectView* portraitTapbarView;
168 @property(retain, nonatomic) RoundedRectView* portraitProgressView;
169 @property(retain, nonatomic) FloatingView* gesturePuaseView;
170 @property(retain, nonatomic) UILabel* speedScrubbingTipLabel;
```

图7-7　关键字iAdView

iAdView正是一个ADBannerView类型的指针，很难让人不去怀疑这个指针的动机。它是不是就是指向广告横幅的指针？做一个简单的实验就可以验证了——UIView有一个hidden属性，你猜猜我们要干嘛？

```
#import <iAd/ADBannerView.h>

@interface PlayViewController : UIViewController
@property(retain, nonatomic) ADBannerView* iAdView;
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;
    NSLog(@"iOSRE: viewDidLoad was called");
    self.iAdView.hidden = YES;
}
%end
```

这段代码的功能非常简单：在PlayViewController的viewDidLoad函数调用时输出指定的调试信息，并将iAdView隐藏起来。按照OPlayer的说明导入一个视频文件，新建一个tweak工程，在MakeFile文件里添加iAd.framework，然后将Filter设置为com.olimsoft.oplayer.lite，编译并安装；ssh登录到iOS设备上，播放这个视频，然后在Terminal中输入

“grep iOSRE /var/log/syslog”，如图 7-8 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Jul 31 15:18:13 Touch-1 OPlayer Lite[1813]: iOSRE: viewDidLoad was called
Touch-1:~ root#
```

图 7-8 syslog 输出日志，证明拦截函数已被执行

图 7-8 说明 hook 起作用了，但是广告横幅并没有消失，如图 7-9 所示。



图 7-9 广告横幅依然在

广告横幅没有消失，可能有多种原因：viewDidLoad 调用时 iAdView 还未初始化，或者 iAdView 根本就不代表广告横幅。下面先验证第一种猜测。

```
#import <iAd/ADBannerView.h>

@interface PlayViewController : UIViewController
@property(retain, nonatomic) ADBannerView* iAdView;
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;
    NSLog(@"iOSRE: %@", self.iAdView);
}
%end
```

重复刚才的步骤，然后在 Terminal 中输入 grep iOSRE /var/log/syslog，如图 7-10 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Jul 31 15:47:19 Touch-1 OPlayer Lite[1892]: iOSRE: (null)
```

图 7-10 syslog 输出日志，获取对象为 null

显然，iAdView 在 viewDidLoad 时仍未初始化，在此时操作 iAdView 为时过早；在 iOS 中，发生在一个对象上的关键事件，如程序退出前台、界面完成加载等，往往定义在 protocol 中。回想刚才在 .h 头文件中搜索 ADBannerView 的结果，我们还没有研究过 ADBannerViewDelegate 这个协议。打开 ADBannerViewDelegate.h，只有寥寥可数的几行代码，如图 7-11。

```
 7
 8 #import "NSObject.h"
 9
10
11 @protocol ADBannerViewDelegate <NSObject>
12 @optional
13 -(void)bannerViewActionDidFinish:(id)bannerViewAction;
14 -(BOOL)bannerViewActionShouldBegin:(id)bannerViewAction willLeaveApplication:(BOOL)application;
15 -(void)bannerView:(id)view didFailToReceiveAdWithError:(id)error;
16 -(void)bannerViewDidLoadAd:(id)bannerView;
17 -(void)bannerViewWillLoadAd:(id)bannerView;
18 @end
```

图 7-11 ADBannerViewDelegate.h 一览

大致浏览完这些函数，先武断地望文生义一下：-(void)bannerViewDidLoadAd:(id) bannerView 应该会在广告横幅加载成功之后得到调用。在开发文档中查找这个函数的说明，验证了我们的猜测，如图 7-12 所示。

bannerViewDidLoadAd:

Called when a new banner advertisement is loaded.

- (void)bannerViewDidLoadAd:(ADBannerView *)*banner*

Parameters
banner
The banner view that loaded a new advertisement.

Discussion
Your application implements this method to be notified when a new advertisement is ready for display.

Availability
Available in iOS 4.0 and later.

Declared In
ADBannerView.h

图 7-12　文档中 bannerViewDidLoadAd 的定义

那么，如果 bannerViewDidLoadAd: 得到调用，iAdView 就一定完成初始化了，没问题吧？用代码再次验证：

```
%hook ADBannerView
-(void)bannerViewDidLoadAd:(id)bannerView
{
    %orig;
    NSLog(@"iOSRE: %@", bannerView);
}
%end
```

再次重复刚才的步骤（下文不再赘述），观察 Terminal 的输出，如图 7-13 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Touch-1:~ root#
```

图 7-13　syslog 中没有拦截到 bannerViewDidLoadAd 输出信息

bannerViewDidLoadAd: 没有得到调用？难道 iAdView 没有设置 delegate？没什么好说的，继续用代码验证：

```
%hook ADBannerView
-(void)setDelegate:(ADBannerViewDelegate *)delegate
{
    %orig;

    NSLog(@"iOSRE: %@", delegate);
}
%end
```

iAdBannerView 只要设置了 delegate，setDelegate: 就会得到调用。还是在 Terminal 中看结果，如图 7-14 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Touch-1:~ root#
```

图 7-14　syslog 中没有拦截到 setDelegate 输出信息

仍然没有任何输出，说明 iAdBannerView 连 delegate 都没有设置。这时就要回过头来审视一下，iAdBannerView 真是广告横幅吗？如果 iAdBannerView 是广告横幅，那之前在 PlayViewController 里发现的 GADBannerView 是干嘛的？以 GADBannerView 为关键词按内容搜索所有 .h 文件，得到 9 个结果，如图 7-15 所示。

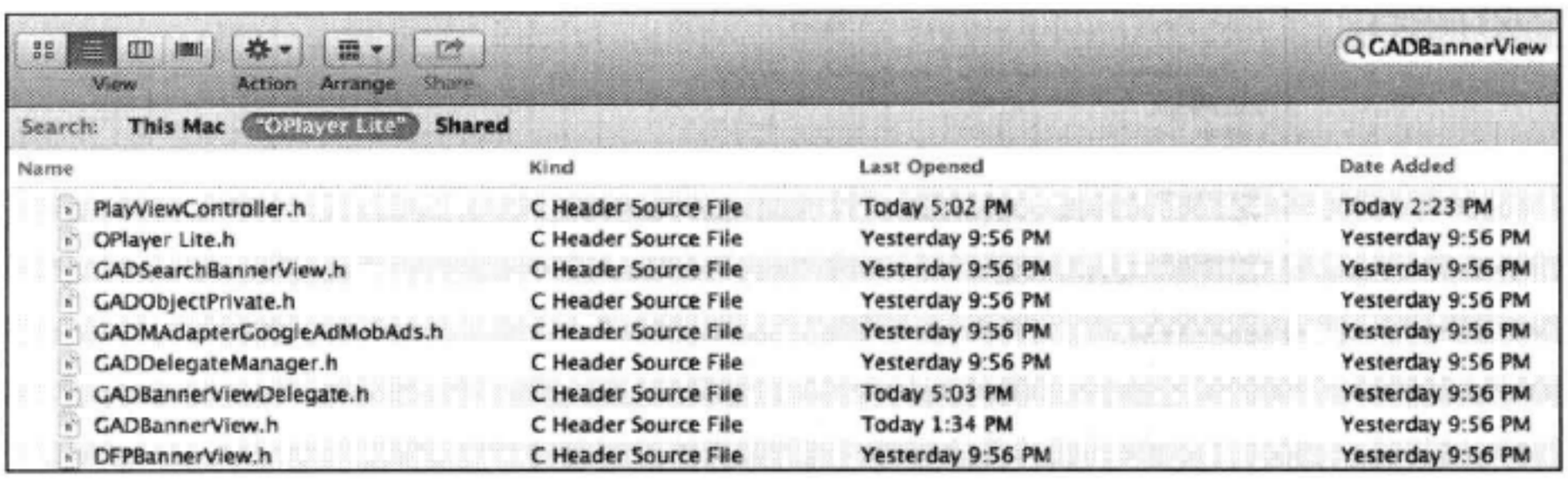


图 7-15 搜索 GADBannerView

注意第 5 行，GADMAdapterGoogleAdMobAds.h，gAd 就是 googleAd？很有可能！搜索一下 OPlayer Android，看看搜索结果吧，如图 7-16 所示。



图 7-16 在谷歌中查找 OPlayer Android

OPlayer 有 Android 版本，采用谷歌的移动广告平台顺理成章。olimsoft 会不会为了省事，在 iOS 和 Android 两个版本都采用谷歌广告？或者是 iAd 收益不如 gAd？个中原因并不重要，不用再进行猜测。GADBannerView 相关的头文件有 9 个之多，仅此一点就足以说明这个类的重要性。那么，试着把分析 iAdView 的狠招放在 gAdView 上再来一次，看它到底结果如何？

```
%hook PlayViewController
- (void)viewDidLoad
{
    %orig;

    id gAdView = MSHookIvar<id>(self, "gAdView");
    NSLog(@"iOSRE: %@", gAdView);
}
%end
```

注意，因为 gAdView 只是 PlayViewController 的一个全局变量，而不是属性，所以要通过 MSHookIvar 的方式获取它。Terminal 的输出结果，如图 7-17 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Jul 31 17:27:31 Touch-1 OPlayer Lite[2108]: iOSRE: <GADBannerView: 0x10dc43f0; f
rame = (80 0; 320 50); clipsToBounds = YES; layer = <CALayer: 0x10dc43c0>>
Touch-1:~ root#
```

图 7-17　syslog 中输出了 gAdView 对象

终于还是被发现了！注意看 gAdView 的 frame 中纵坐标的值，是 0，意味着 gAdView 应该紧贴屏幕上沿，与广告横幅的特征吻合，继续代码伺候！

```
@interface GADBannerView : UIView
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;

    GADBannerView *gAdView = MSHookIvar<GADBannerView *>(self, "gAdView");
    gAdView.hidden = YES;
}
%end
```

广告横幅消失，整个世界一下清静了，如图 7-18 所示。

折腾了一大圈才定位到广告横幅，最后用两行代码就搞定了它，你是不是也有些意犹未尽？其实上面的方法只是“隐藏”了这个广告横幅，并没有真正“去除”它，而且 OPlayer 仍然会下载广告，耗费一定的流量。接下来，就要让 OPlayer 不再联网下载广告，去掉这个横幅以节省流量。

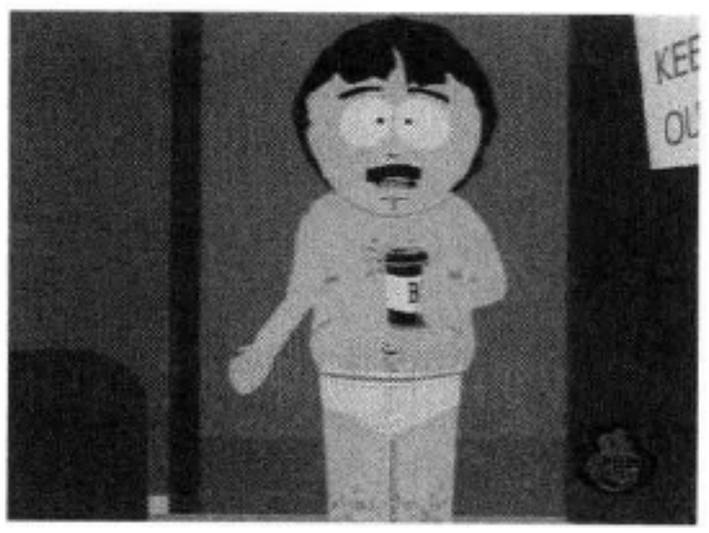


图 7-18　广告横幅被隐藏了

7.3　去除广告横幅，节省流量

7.3.1　找到“准备”广告的位置

在继续阅读前，请先把 OPlayer Lite 的可执行文件拖进 IDA 分析。不知道大家有没有注意到这个细节：在广告横幅的左下方有个带圆圈的感叹号，点击后会展开，显示“Google 提供的广告”，如图 7-19 所示，这似乎印证了我们的猜测，即此版本的 OPlayer Lite 采用了 gAd，而未采用 iAd。



图 7-19　展开的“Google 提供的广告”

既然 gAdView 也是来自 PlayViewController 类，那么就在这个类的头文件里找找看有没

有关于 gAdView 的其他线索。重新审视 PlayViewController.h，又有新发现，如图 7-20 所示。

```
65          UIView* localAdView;
66          ADBannerView* iAdView;
67          GADBannerView* gAdView;
```

图 7-20 PlayViewController.h 中的 AdView

由图 7-20 可知，PlayViewController 应该至少可以显示三种广告类型，分别是本地广告、iAd 和 gAd。至于每种广告的显示时机，本地广告可能在断网时显示，而 iAd 和 gAd 的选择条件暂且不得而知，只是在笔者的测试环境中，OPlayer Lite 显示了 gAd。所以，本节开头“未采用 iAd”的说法并不准确，不同地区、不同网络显示的广告可能不同，在某些情况下也可能显示 iAd，本书不对其作严格测试。下面验证一下 localAdView 的显示时机，先将 iOS 断网，然后打开一个视频，观察广告横幅，如图 7-21 所示。

断网时，广告横幅不显示任何内容，而是以纯灰色的背景覆盖在画面上，非常难看；上一节编写的“隐藏”横幅代码不起作用，说明这个横幅并不是 gAdView，那么它是 localAdView 吗？试一试就知道了。

```
@interface GADBannerView : UIView
@end

@interface PlayViewController
@property(retain, nonatomic) UIView* localAdView;
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;

    GADBannerView *gAdView = MSHookIvar<GADBannerView *>(self, "gAdView");
    gAdView.hidden = YES;

    UIView* localAdView = self.localAdView;
    localAdView.hidden = YES;
}
%end
```

我们猜得没错，灰色横幅消失了（如图 7-22 所示），它就是 localAdView。

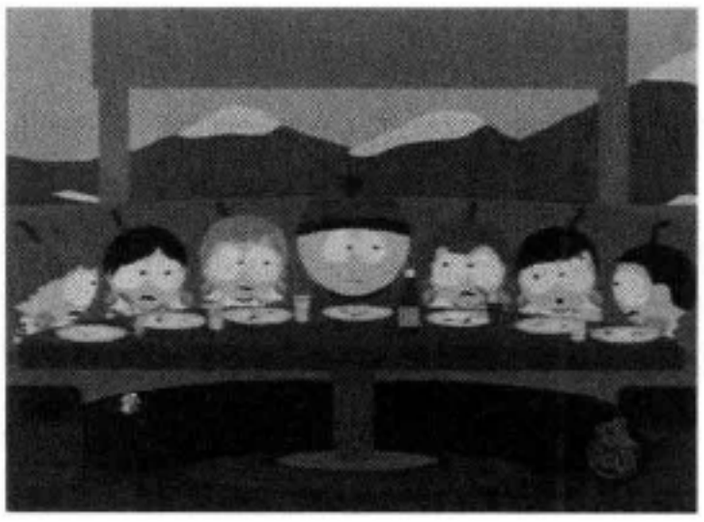


图 7-21 断网后显示无字的广告横幅

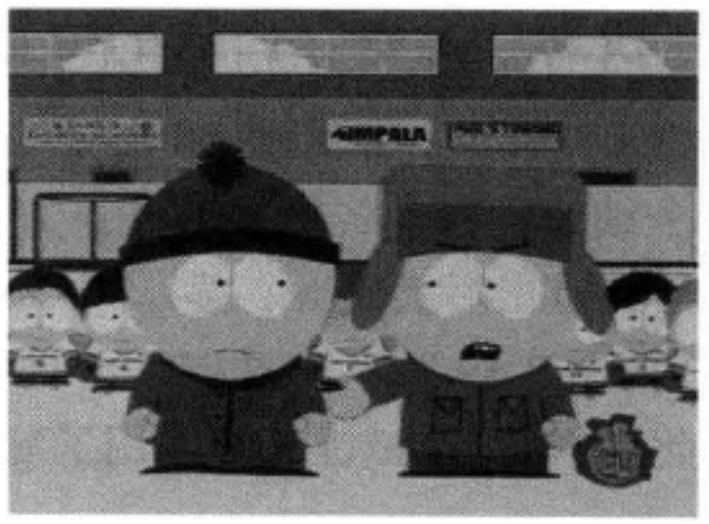


图 7-22 将 localAdView 隐藏后，广告横幅消失了

为了一绝后患，考虑对代码进行如下改进：

```
#import <iAd/ADBannerView.h>

@interface GADBannerView : UIView
@end

@interface PlayViewController
@property(retain, nonatomic) UIView* localAdView;
@property(retain, nonatomic) ADBannerView* iAdView;
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;

    GADBannerView *gAdView = MSHookIvar<GADBannerView *>(self, "gAdView");
    gAdView.hidden = YES;

    UIView* localAdView = self.localAdView;
    localAdView.hidden = YES;

    ADBannerView* iAdView = self.iAdView;
    iAdView.hidden = YES;
}
%end
```

三种广告横幅都消失了，下一步就是集中力量办大事——阻止广告下载。在发现三种广告类型后，不知大家有没有注意一个问题：一个视频刚刚打开时，OPlayer Lite 显示的是灰色的 localAdView，短暂的几秒后才显示有内容的 gAdView，两者的间隔应该就是下载广告的过程；因此，OPlayer Lite 在播放视频时会先初始化 localAdView，并开始联网下载广告，待广告下载完成后用 gAdView 替换或覆盖 localAdView；这样的过程在 iOS 中通常是通过回调函数来完成的，当“广告下载完成”这个事件出现时回调函数得到调用，而 iOS 的回调函数一般定义在协议中。我想，现在是时候把关注点转向 GADBannerViewDelegate.h 了。

```
11 @protocol GADBannerViewDelegate <NSObject>
12 @optional
13 -(void)adViewWillLeaveApplication:(id)adView;
14 -(void)adViewDidDismissScreen:(id)adView;
15 -(void)adViewWillDismissScreen:(id)adView;
16 -(void)adViewWillPresentScreen:(id)adView;
17 -(void)adView:(id)view didFailToReceiveAdWithError:(id)error;
18 -(void)adViewDidReceiveAd:(id)adView;
19 @end
```

图 7-23　GADBannerViewDelegate.h 内容展示

GADBannerViewDelegate.h 的内容也很简洁，只有 6 个函数，如图 7-23 所示。

其中，adViewDidReceiveAd: 应该就是我们要寻找的回调函数了，它出现在 3 个 .h 文件里，如图 7-24 所示。

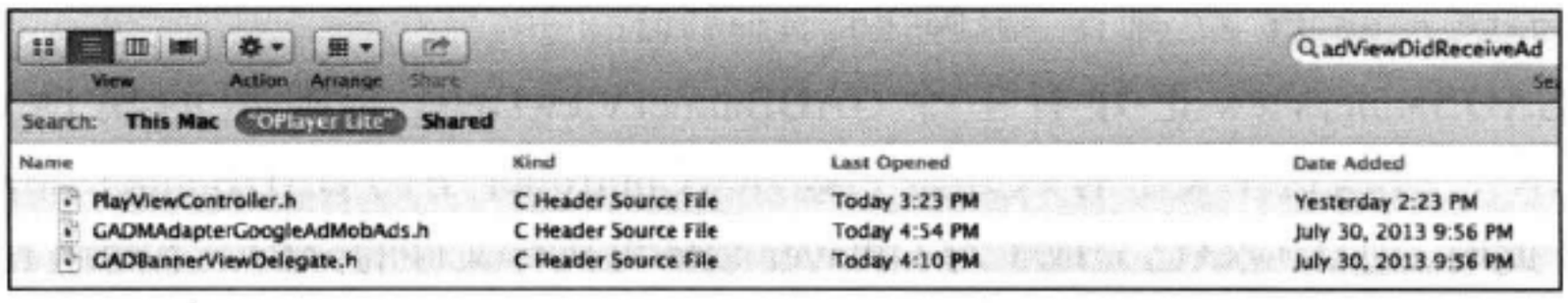


图 7-24　搜索 adViewDidReceiveAd

PlayViewController.h 和 GADMAdapterGoogleAdMobAds.h 都有可能，到底是哪个呢？没关系，用代码验证一下就知道了：

```
%hook PlayViewController
-(void)adViewDidReceiveAd:(id)adView
{
    %orig;

    NSLog(@"iOSRE1: adViewDidReceiveAd:%@ was called", adView);
}
%end

%hook GADMAdapterGoogleAdMobAds
-(void)adViewDidReceiveAd:(id)adView
{
    %orig;

    NSLog(@"iOSRE2: adViewDidReceiveAd:%@ was called", adView);
}
%end
```

Terminal 的输出如图 7-25 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Aug  1 16:59:58 Touch-1 OPlayer Lite[3172]: iOSRE1: adViewDidReceiveAd:<GADBann
erView: 0xbfde030; frame = (80 0; 320 50); clipsToBounds = YES; layer = <CALaye
r: 0xbfde390>> was called
Touch-1:~ root#
```

图 7-25 syslog 拦截的 adView 对象信息

接着，检查所有按内容搜索 GADBannerViewDelegate 得到的结果，如图 7-26 所示。

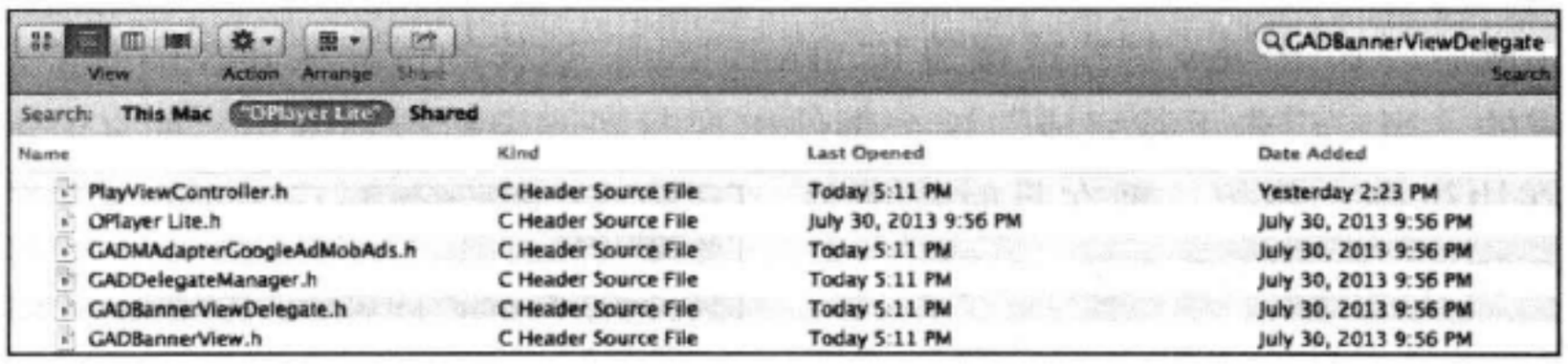


图 7-26 搜索 GADBannerViewDelegate

实现 GADBannerViewDelegate 的类只有 PlayViewController 和 GADMAdapterGoogleAdMobAds，而 GADMAdapterGoogleAdMobAds 经上面的测试已经被淘汰掉了，所以一定是 PlayViewController.h 里的某个对象 x 作出了如下操作：

```
x.delegate = self; // 即 [x setDelegate:self]
```

因为 GADBannerView 正好有一个 GADBannerViewDelegate 类型的属性（如图 7-27 第 20 行所示），所以这个 x 十有八九是 PlayViewController 中的 GADBannerView 对象，即 gAdView；此时，IDA 应该已完成了对 OPlayer Lite 可执行文件的分析，下面用 IDA 和 GDB 完成后续工作。

```
14 __attribute__((visibility("hidden")))
15 @interface GADBannerView : XXUnknownSuperclass {
16 }
17 @property(readonly, assign, nonatomic) GADWebView* webView;
18 @property(readonly, assign, nonatomic) UIView* mediatedAdView;
19 @property(readonly, assign, nonatomic) BOOL hasAutoRefreshed;
20 @property(assign, nonatomic) NSObject<GADBannerViewDelegate>* delegate;
21 @property(assign, nonatomic) GADAdSize adSize;
```

图 7-27　GADBannerView.h 内容一览

首先验证 [gAdView setDelegate:self] 是否得到调用，然后找出其调用者；此调用者很可能就是发起网络连接的“真凶”。在 Functions window 中找到 [GADBannerView setDelegate:] 函数，双击跳转到其实现代码，如图 7-28 所示。

```
__text:00135860
__text:00135860 ; =============== S U B R O U T I N E =======================================
__text:00135860
__text:00135860 ; GADBannerView - (void)setDelegate:(id)
__text:00135860 ; Attributes: bp-based frame
__text:00135860
__text:00135860 ; void __cdecl -[GADBannerView setDelegate:](struct GADBannerView *self, SEL, id)
__text:00135860 __GADBannerView_setDelegate__           ; DATA XREF: __objc_const:014FFA88↓o
__text:00135860                 STMFD           SP!, {R4,R5,R7,LR}
__text:00135864                 MOV             R1, #(:lower16:(selRef_private - 0x13587C))
__text:00135868                 ADD             R7, SP, #8
__text:0013586C                 MOVT            R1, #(:upper16:(selRef_private - 0x13587C))
__text:00135870                 MOV             R5, R2
__text:00135874                 LDR             R1, [PC,R1] ; selRef_private ; "private"
__text:00135878                 MOV             R4, R0
__text:0013587C                 BL              _objc_msgSend.island
__text:00135880                 MOV             R1, #(selRef_delegateManager - 0x135890) ; selRef_delegateManager
__text:00135888                 LDR             R1, [PC,R1] ; selRef_delegateManager ; "delegateManager"
__text:0013588C                 BL              _objc_msgSend.island
__text:00135890                 MOV             R1, #(:lower16:(selRef_setBannerDelegate_ - 0x1358A4))
__text:00135894                 MOV             R2, R5
__text:00135898                 MOVT            R1, #(:upper16:(selRef_setBannerDelegate_ - 0x1358A4))
__text:0013589C                 LDR             R1, [PC,R1] ; selRef_setBannerDelegate_ ; "setBannerDelegate:"
__text:001358A0                 BL              _objc_msgSend.island
__text:001358A4                 MOV             R1, #(:lower16:(selRef_private - 0x1358B8))
__text:001358A8                 MOV             R0, R4
__text:001358AC                 MOVT            R1, #(:upper16:(selRef_private - 0x1358B8))
__text:001358B0                 LDR             R1, [PC,R1] ; selRef_private ; "private"
__text:001358B4                 BL              _objc_msgSend.island
__text:001358B8                 MOV             R1, #(selRef_delegateManager - 0x1358C8) ; selRef_delegateManager
__text:001358C0                 LDR             R1, [PC,R1] ; selRef_delegateManager ; "delegateManager"
__text:001358C4                 BL              _objc_msgSend.island
__text:001358C8                 MOV             R1, #(:lower16:(selRef_setAdView_ - 0x1358DC))
__text:001358CC                 MOV             R2, R4
__text:001358D0                 MOVT            R1, #(:upper16:(selRef_setAdView_ - 0x1358DC))
__text:001358D4                 LDR             R1, [PC,R1] ; selRef_setAdView_ ; "setAdView:"
__text:001358D8                 BL              _objc_msgSend.island
__text:001358DC                 LDMFD           SP!, {R4,R5,R7,PC}
__text:001358DC ; End of function -[GADBannerView setDelegate:]
```

图 7-28　在 IDA 中查看 [GADBannerView setDelegate:]

记下最后一行“ LDMFD SP!, {R4,R5,R7,PC}”的地址 0x1358dc，等会要用 GDB 在此处下断点。打开 GDB，attach 进 OPlayer Lite，如图 7-29 所示。

```
Touch-1:~ root# gdb
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4)
(Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
gdb$ attach "OPlayer Lite"
Attaching to process 3282.
Reading symbols for shared libraries . done
Reading symbols for shared libraries ..........................................
..............................................................................
..............................................................................
.. done
0x3bff5eb4 in mach_msg_trap ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.

gdb$
```

图 7-29　GDB 加载 OPlayer Lite

利用“info sh”命令找到 OPlayer Lite 的 ASLR 偏移，如图 7-30 所示。

```
gdb$ info sh
The DYLD shared library state has not yet been initialized.
                                        Requested State Current State
Num Basename                Type Address         Reason | | Source
  | |                         | |                   | | | |
  1 OPlayer Lite              - 0x1000            exec C C /private/var/mobile
/Applications/85C4D646-B723-42EF-80B7-5337043CF203/OPlayer Lite.app/OPlayer Lit
e (offset 0x0)
                                                (objfile is) [memory object '/pr
ivate/var/mobile/Applications/85C4D646-B723-42EF-80B7-5337043CF203/OPlayer Lite
.app/OPlayer Lite' at 0x1000]
  2 dyld                    - 0x2fe93000        dyld Y Y /usr/lib/dyld at 0x
2fe93000 (offset 0x93000) with prefix "__dyld_"
  3 SubstrateBootstrap.dylib - 0x2261000        dyld Y Y /Library/Frameworks
/CydiaSubstrate.framework/Libraries/SubstrateBootstrap.dylib at 0x2261000 (offs
et 0x2261000)
  4 libbz2.1.0.dylib        - 0x3b4ff000        dyld Y Y /usr/lib/libbz2.1.0
.dylib at 0x3b4ff000 (offset 0x2d62000)
                                               (objfile is) [memory object '/us
r/lib/libbz2.1.0.dylib' at 0x3b4ff000]
  5 libSystem.B.dylib       - 0x3b3d4000        dyld Y Y /usr/lib/libSystem.
B.dylib at 0x3b3d4000 (offset 0x2d62000)
                                               (objfile is) [memory object '/us
r/lib/libSystem.B.dylib' at 0x3b3d4000]
```

图 7-30 GDB 查找 ASLR 偏移

ASLR 的偏移是 0，那么断点的位置就是 0x1358dc（0x1358dc + 0x0）。断点设置好之后，打开一个视频，触发断点，然后用 ni 来单步执行代码，如图 7-31 所示。

```
gdb$ b *0x1358dc
Breakpoint 1 at 0x1358dc
gdb$ c

Breakpoint 1, 0x001358dc in dyld_stub_pthread_key_create ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ ni
0x00289b88 in dyld_stub_pthread_key_create ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
0x00289b90 in dyld_stub_pthread_key_create ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
0x00289b94 in dyld_stub_pthread_key_create ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$
0x00289b94 in dyld_stub_pthread_key_create ()
Error while running hook_stop:
No symbol table is loaded.  Use the "file" command.
gdb$ c
```

图 7-31 GDB 使用 ni 单步执行

[gAdView setDelegate:self] 得到调用，且 0x289b88 位于调用者内部。在 IDA 里跳转到 0x289b88，如图 7-32 所示。

```
__text:00289B40 MOV     R0, #(selRef_setAdUnitID_ - 0x289B4C) ; selRef_setAdUnitID_
__text:00289B48 ADD     R0, PC ; selRef_setAdUnitID_
__text:00289B4A LDR     R1, [R0] ; "setAdUnitID:"
__text:00289B4C LDR.W   R0, [R4,R8]
__text:00289B50 MOV     R2, #(cfstr_A14c59222b5e63 - 0x289B5C) ; "a14c59222b5e633"
__text:00289B58 ADD     R2, PC  ; "a14c59222b5e633"
__text:00289B5A BLX.W   _objc_msgSend.island
__text:00289B5E MOVW    R0, #(:lower16:(selRef_setRootViewController_ - 0x289B6C))
__text:00289B62 MOV     R2, R4
__text:00289B64 MOVT.W  R0, #(:upper16:(selRef_setRootViewController_ - 0x289B6C))
__text:00289B68 ADD     R0, PC ; selRef_setRootViewController_
__text:00289B6A LDR     R1, [R0] ; "setRootViewController:"
__text:00289B6C LDR.W   R0, [R4,R8]
__text:00289B70 BLX.W   _objc_msgSend.island
__text:00289B74 MOVW    R0, #(:lower16:(selRef_setDelegate_ - 0x289B82))
__text:00289B78 MOV     R2, R4
__text:00289B7A MOVT.W  R0, #(:upper16:(selRef_setDelegate_ - 0x289B82))
__text:00289B7E ADD     R0, PC ; selRef_setDelegate_
__text:00289B80 LDR     R1, [R0] ; "setDelegate:"
__text:00289B82 LDR.W   R0, [R4,R8]
__text:00289B86 BLX.W   _objc_msgSend.island
__text:00289B8A MOVW    R0, #(:lower16:(selRef_loadRequest_ - 0x289B98))
__text:00289B8E MOVS    R2, #0
```

图 7-32 IDA 定位到 _objc_msgSend.island

由 IDA 显示可知，调用者是 [PlayViewController planAds]。该函数名就已经很有内涵了——“准备”广告，难道这就是初始化广告的地方？我们来一探究竟！

7.3.2 追溯到网络连接的起点

planAds 的函数逻辑比较清晰，大致如下：判断当前版本是否支持 iAd，如果支持，则按 iAd 标准流程来初始化一个 ADBannerView 对象；如果不支持，则按屏幕尺寸和设备分辨率来初始化一个 GADBannerView 对象，然后调用 loadRequest: 函数。对于 iAd 来说，开发者只需要调用 iAd.framework 指定显示 iAd 的位置即可，其内容由 Apple 提供，开发者无须操心，毕竟 iOS 是 Apple 开发的系统，这种程度的代码整合是合理的；但是 gAd 是由谷歌提供的广告，iOS 无现成的 framework 来实现此功能，因此 gAd 的内容并不能自动生成。这么看来，planAds 好像只规划了 gAds 的显示位置，而没有规划内容。等一下，iOS 里常用的 HTTP 请求类基本都是以 Request 结尾的，如 ASIHTTPRequest 和 NSURLRequest，如图 7-33 所示。那么 loadRequest: 函数是干嘛的？

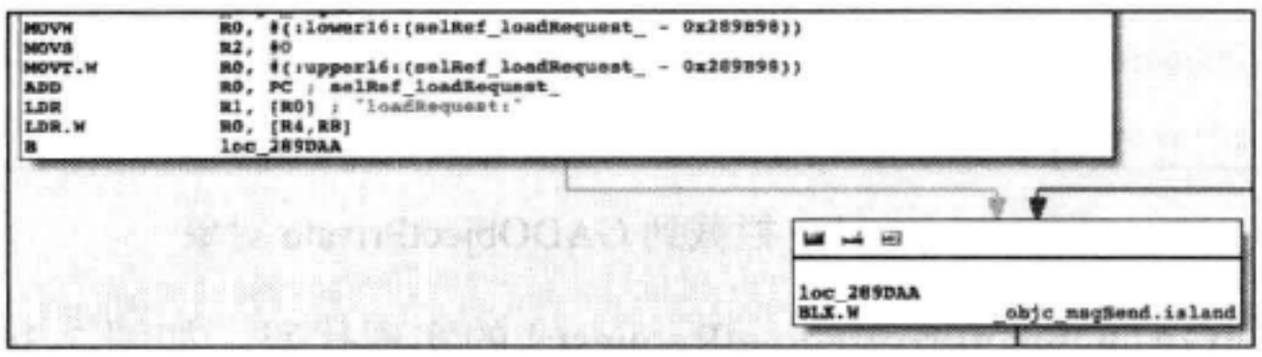


图 7-33　IDA 定位 loadRequest

由图 7-33 可知，loadRequest: 的对象是 [R4,R8]，倒推可知，[R4,R8] 是由 0x289A48 处的“ STR.W R0, [R4,R8]”决定的，而 R0 是一个已初始化的 GADBannerView 对象。在 Functions window 中找到 [GADBannerView loadRequest:] 函数，双击跳转到其实现代码，如图 7-34 所示。

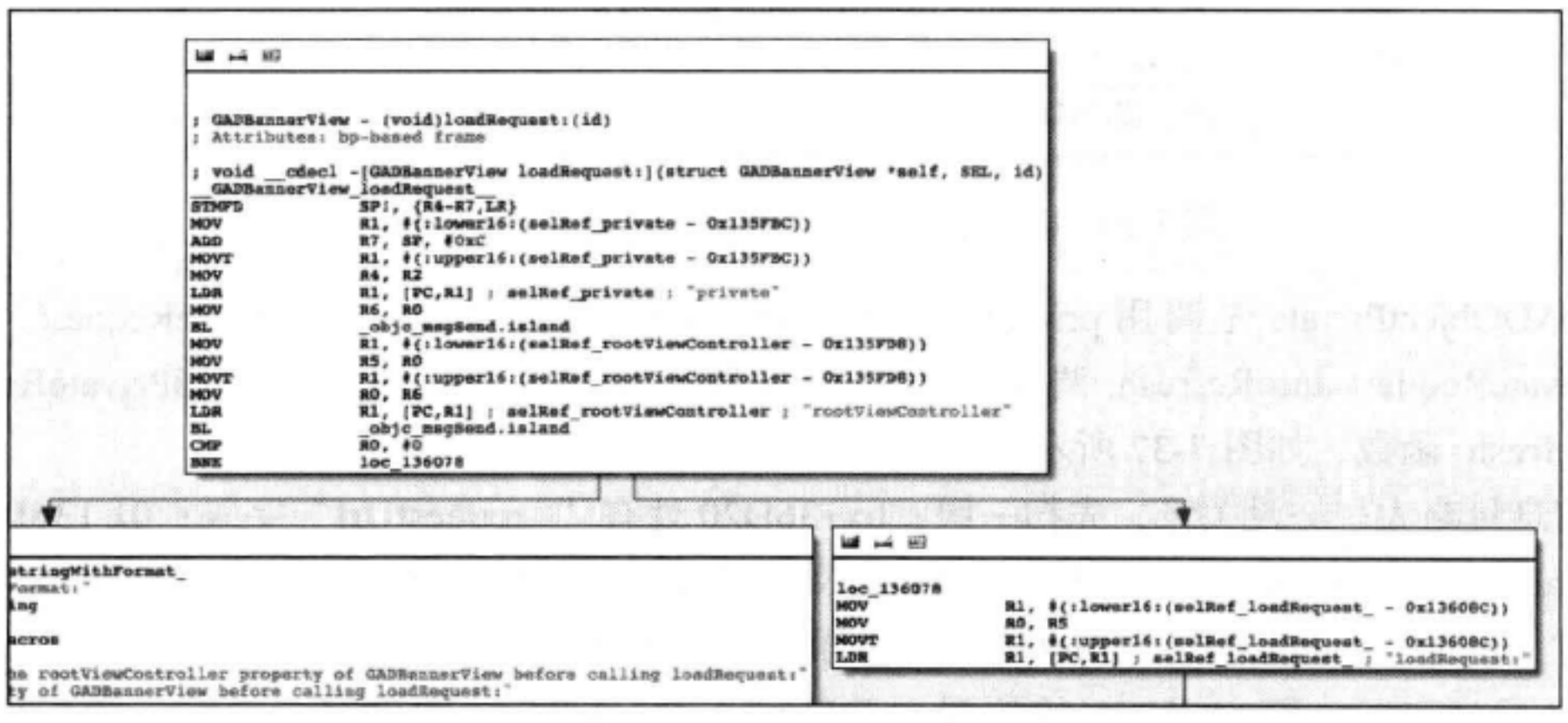


图 7-34　IDA 定位 [GADBannerView loadRequest:]

函数的逻辑也很简单，如果 GADBannerView 对象没有 rootViewController，则报错；如果有，则先调用 GADBannerView 对象的 private 函数，然后对返回值调用 loadRequest: 函数。编写以下代码检测 private 函数返回值的类型：

```
%hook GADBannerView
-(id)private
{
    id result = %orig;
    NSLog(@"iOSRE: %@", result);

    return result;
}
%end
```

在 Terminal 中查看调试信息，如图 7-35 所示。

```
Touch-1:~ root# grep iOSRE /var/log/syslog
Aug  1 23:06:05 Touch-1 OPlayer Lite[3425]: iOSRE: (null)
Aug  1 23:06:06 Touch-1 OPlayer Lite[3425]: iOSRE: <GADObjectPrivate: 0xbf40c70
>
Aug  1 23:06:06 Touch-1 OPlayer Lite[3425]: iOSRE: <GADObjectPrivate: 0xbf40c70
>
Touch-1:~ root#
```

图 7-35 syslog 拦截到 GADObjectPrivate 对象

在 IDA 里展开 [GADObjectPrivate loadRequest:] 的实现代码，如图 7-36 所示。

```
; GADObjectPrivate - (void)loadRequest:(id)
; Attributes: bp-based frame

; void __cdecl -[GADObjectPrivate loadRequest:](struct GADObjectPrivate *self, SEL, id)
__GADObjectPrivate_loadRequest__
STMFD           SP!, {R4,R7,LR}
MOV             R1, #(:lower16:(selRef_privateRequestWithRequest_ - 0x138CD4))
ADD             R7, SP, #4
MOVT            R1, #(:upper16:(selRef_privateRequestWithRequest_ - 0x138CD4))
MOV             R4, R0
LDR             R1, [PC,R1] ; selRef_privateRequestWithRequest_ ; "privateRequestWithRequest:"
BL              _objc_msgSend.island
MOV             R1, #(:lower16:(selRef_loadPrivateRequest_autoRefresh_ - 0x138CEC))
MOV             R2, R0
MOVT            R1, #(:upper16:(selRef_loadPrivateRequest_autoRefresh_ - 0x138CEC))
MOV             R0, R4
LDR             R1, [PC,R1] ; selRef_loadPrivateRequest_autoRefresh_ ; "loadPrivateRequest:autoRefresh:"
MOV             R3, #0
BL              _objc_msgSend.island
LDMFD           SP!, {R4,R7,PC}
; End of function -[GADObjectPrivate loadRequest:]
```

图 7-36 IDA 定位 [GADObjectPrivate loadRequest:]

GADObjectPrivate 先调用 privateRequestWithRequest: 函数获取一个 privateRequest，作为 loadPrivateRequest:autoRefresh: 调用的第一个参数。继续在 IDA 里展开 loadPrivateRequest:autoRefresh: 函数，如图 7-37 所示。

信息量略大……没关系，先扫一眼：0x136D20 处有“_requestUrl”字眼；0x 136DB4 处有“_activationOverlayUrl”字眼；0x136E20 处有“loadAdHTMLFromURL:”字眼；0x136E70 处有“executeJavaScriptFileFromURL:”字眼……看到这里，planAds 的作用渐渐明晰：它初始化了 iAdView 和 gAdView 的位置，并联网下载 gAdView 的内容，等等。我们已经可以断定 planAds 就是初始化广告的操作，不需要再继续追踪了。干掉这个函数，就干掉了

iAdView 和 gAdView，更重要的是，在去除广告横幅的同时节省了流量，任务完成！

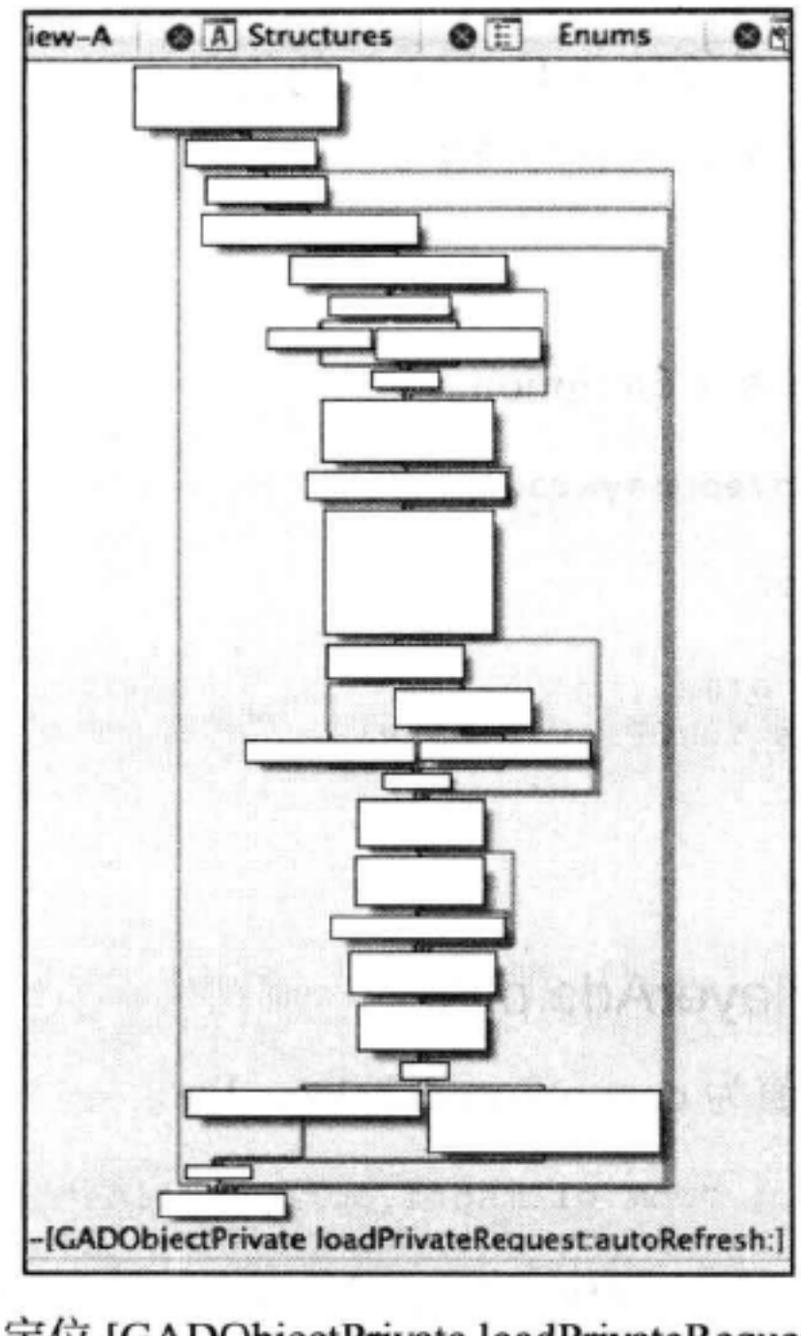


图 7-37　IDA 定位 [GADObjectPrivate loadPrivateRequest:autoRefresh:]

7.4　编写调试代码

7.4.1　编写 Tweak_OPlayerLite.xm

在上面的分析过程中，我们的代码雏形其实已经完成了，只需稍微整理即可：

```
@interface PlayViewController
@property(retain, nonatomic) UIView* localAdView;
@end

%hook PlayViewController
- (void)viewDidLoad
{
    %orig;

    self.localAdView.hidden = YES;
}

- (void)planAds
{
    // Bye bye Ads!
    return;
```

```
}
%end
```

不用再说明了吧？虽然只是简单的不到 20 行的代码，却凝结了我们对 OPlayer Lite 广告横幅的大量分析，真可以称得上是浓缩的精华。

7.4.2 编辑 control

编辑 control，填写 Name 和 Description。

```
Package: com.naken.removeoplayerads
Name: RemoveOPlayerAds
Depends: mobilesubstrate
Version: 1.0
Architecture: iphoneos-arm
Description: Remove Ads in OPlayer Lite!
Maintainer: snakeninny
Author: snakeninny
Section: Tweaks
```

7.4.3 编辑 RemoveOPlayerAds.plist

指定 Filter 中的 Bundles 值为 com.olimsoft.oplayer.lite。

```
{ Filter = { Bundles = ( "com.olimsoft.oplayer.lite" ); }; }
```

填入的 Bundle Indentifier 可从 OPlayer Lite 目录中的 Info.plist 里查到，如图 7-38 所示。

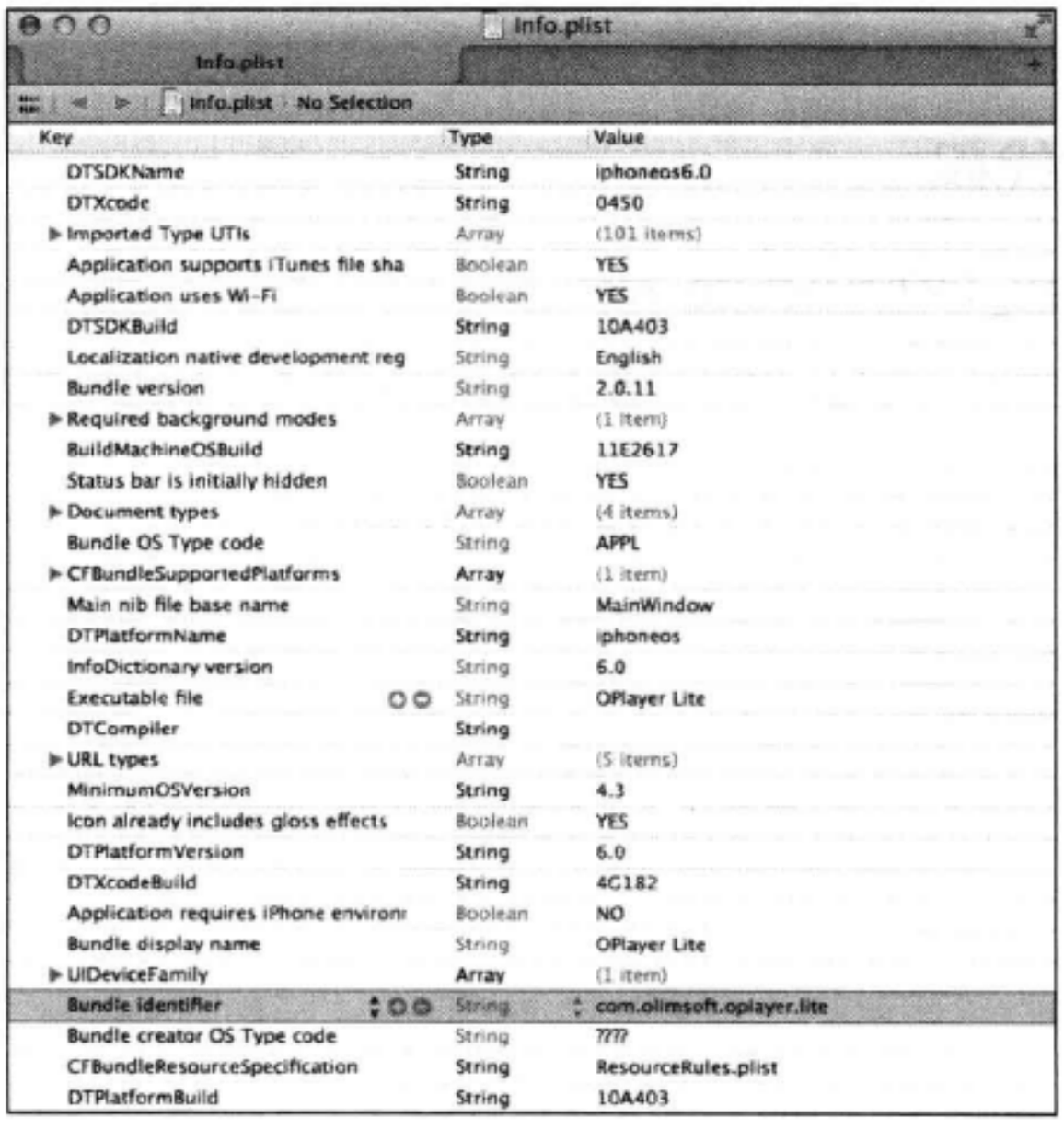


图 7-38 OPlayer Lite 的 Info.plist

7.4.4　编辑 Makefile

编辑 Makefile，指定 RemoveOPlayerAds_FILES 等参数。

```
export ARCHS = armv7
export TARGET = iphone:latest:4.3

include theos/makefiles/common.mk

TWEAK_NAME = RemoveOPlayerAds
RemoveOPlayerAds_FILES = Tweak_OPlayerLite.xm

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 OPlayer\ Lite"
```

7.4.5　编译 + 打包

利用 Theos 编译、打包非常方便，如图 7-39 所示。

```
[redacted]-MacBook-Pro:removeoplayerads snakeninny$ make package
Making all for tweak RemoveOPlayerAds...
 Preprocessing Tweak_OPlayerLite.xm...
 Compiling Tweak_OPlayerLite.xm...
 Linking tweak RemoveOPlayerAds...
 Stripping RemoveOPlayerAds...
 Signing RemoveOPlayerAds...
Making stage for tweak RemoveOPlayerAds...
dpkg-deb: building package `com.naken.removeoplayerads' in `./com.naken.removeop
layerads_1.0-2_iphoneos-arm.deb'.
[redacted]-MacBook-Pro:removeoplayerads snakeninny$
```

图 7-39　Theos 编译 + 打包

7.4.6　安装

安装 deb 的方式有很多种：使用 ssh 命令登录到 iOS 设备然后手动运行“dpkg -i com.naken.removeoplayerads_1.0-2_iphoneos-arm.deb”、iFile 安装，或者“make package install”等都很方便，读者可根据个人习惯自行选择。

7.5　调试去广告插件

1. 断网时的效果

前面的去广告插件过程应该涵盖了 Google Ads、iAd 和本地广告，对各种网络状况均适用，是否真是这样，我们来测试一下。首先断开网络，观察插件效果，如图 7-40 所示。

2. 连接 Wi-Fi 时的效果

图 7-41 所示为连接 Wi-Fi 时的效果。

3. 连接 3G 时的效果

图 7-42 是连接 3G 时的效果。

4. 连接 EDGE 时的效果

图 7-43 是连接 EDGE 时的效果。

5. 连接 GPRS 时的效果

图 7-44 是连接 GPRS 时的效果。

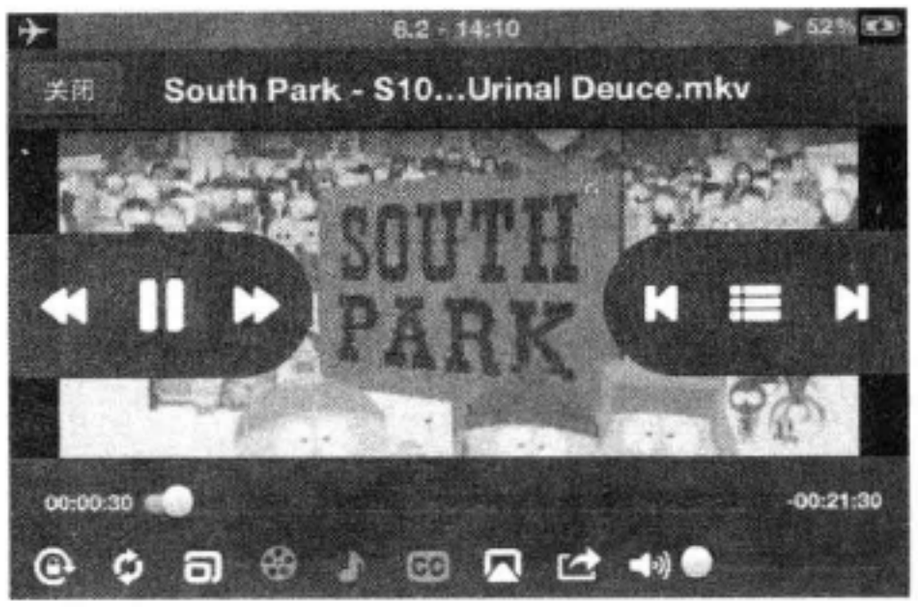


图 7-40 断网时的效果

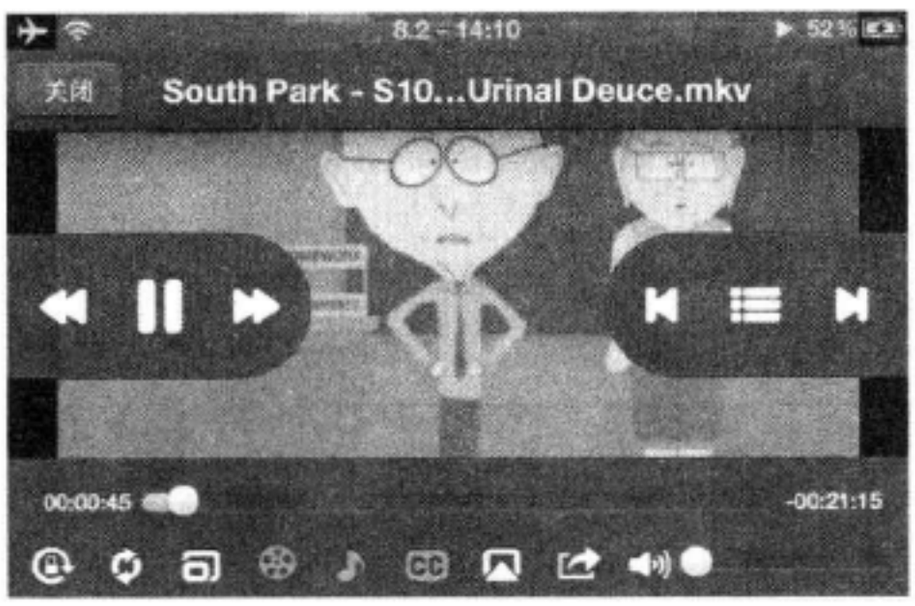


图 7-41 连接 Wi-Fi 时的效果



图 7-42 连接 3G 时的效果

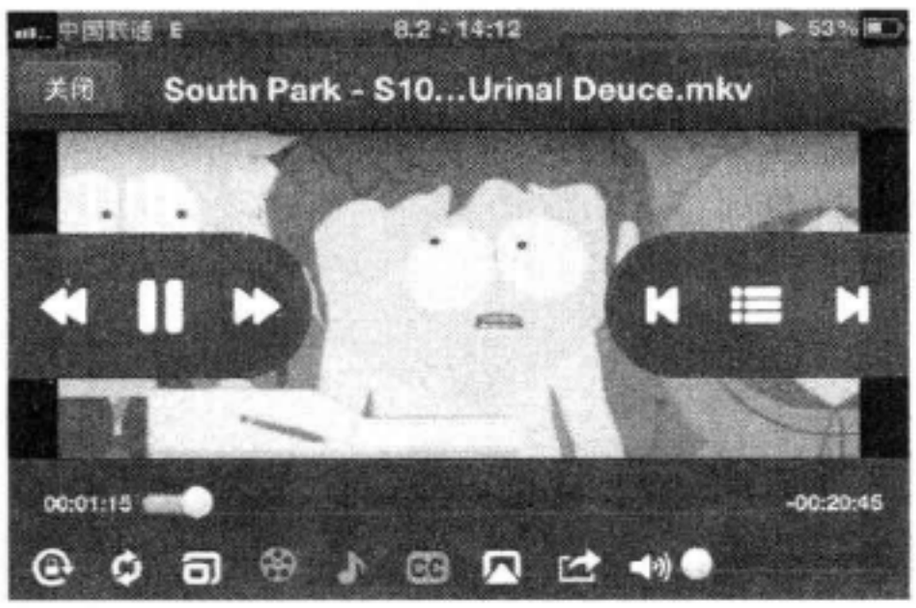


图 7-43 连接 EDGE 时的效果

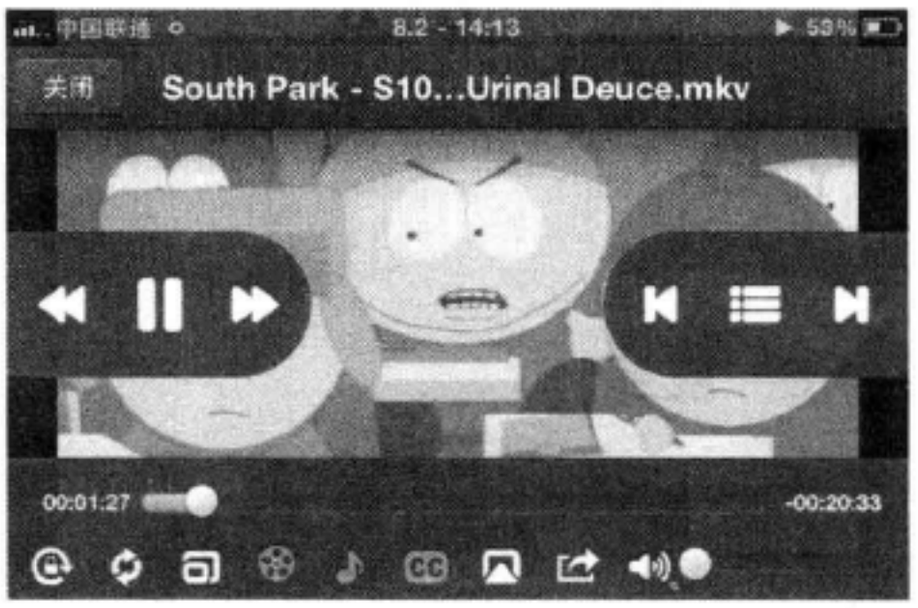


图 7-44 连接 GPRS 时的效果

可见，在各种网络状况下广告横幅都消失了，插件工作正常。

7.6 代码结果整理

虽然最后生成的 deb 解包后小到只有 16KB，但个中的分析过程却并不简单。本章基本沿用了第 5 章的分析套路，辅以 IDA 和 GDB，一环扣一环地找到了广告横幅产生的根源。

为了帮助读者举一反三，本节将前文的分析过程从实例中提炼出来，总结为如下步骤：

（1）由 iAd.framework 中最常见的 ADBannerView 开始分析和测试，广告横幅不受影响，失败。

（2）转向之前忽略的 GADBannerView，刚把架子摆上，广告横幅就消失了，取得初步成效。

（3）进一步发现三种广告类型：iAd、gAd 和 localAd，必须三管齐下，才能“隐藏”广告横幅。

（4）结合 iOS 下载功能的特点，从回调函数和协议入手，定位 GADBannerView Delegate。

（5）从 setDelegate: 开始，左手 IDA 右手 GDB，陆续追踪 [PlayViewController planAds]、[GADBannerView loadRequest:]、[GADObjectPrivate loadRequest:]、[GADObjectPrivate loadPrivateRequest: autoRefresh:] 直到 [GADObjectPrivate loadAdHTMLFromURL: usingActivationOverlayURL:]，找到联网线索。

（6）由一系列线索推测，planAds 就是广告横幅的初始化起点，屏蔽这个函数就可以跟广告横幅彻底说再见了。

6 个步骤，5 个文件（如图 7-45 所示），一个对策，你值得拥有！

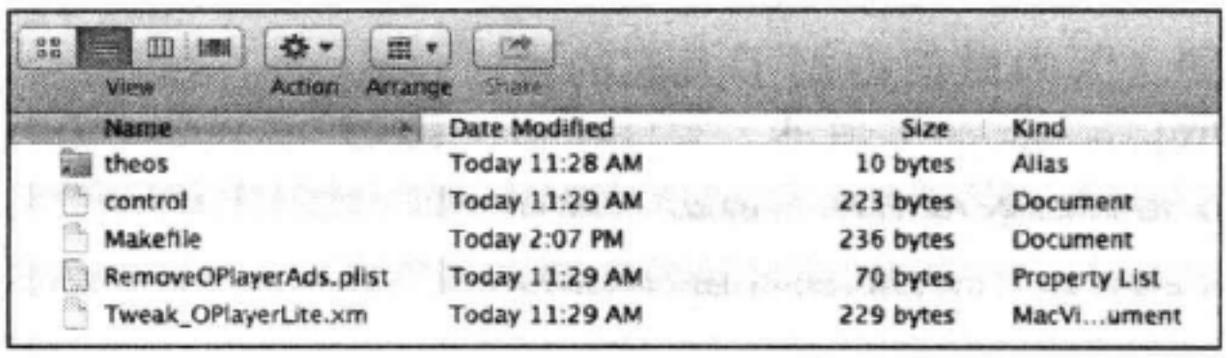


图 7-45　tweak 目录，仅有 5 个文件

7.7　小结

本章为 OPlayer Lite 制作了去除广告插件，让我们在观看视频时不再受到广告困扰。虽然最后的代码只有不到 20 行，但整个分析过程还真不简单，iOS 逆向的三大件，class-dump、IDA 和 GDB 都出马了，才算从源头上制服了广告横幅。值得一提的是，OPlayer Lite 2.0.10 版的广告横幅采用的还是 iAd，因此不敢保证本章所编写的插件一定能有效去除 OPlayer Lite 2.0.12 版的广告横幅，但整体分析思路是不变的，那么我们就以不变应万变吧！

对不熟悉逆向工程的读者来说，可能会觉得这一章的运气成分太大；没错，逆向工程不比 App 开发，没有官方文档作为参考，最有效的方式就是不断尝试，不断犯错，再不断改错。在计算机行业里，犯错的代价比其他行业小得多，它顶多是浪费了一些电、消耗了你的一些精力和时间，并不会造成其他实质性的损失（除非是已经公开发布的软件），但是这些花费换来的是你对逆向工程更深入的理解，产出远大于投入，所以不要害怕犯错，大胆去做，成功就在前面等着你。

第 8 章

实战 2：WhatsApp 消息拦截

8.1 WhatsApp 简介

WhatsApp，国外非常流行的聊天软件，也是最早进入移动领域的聊天软件，之后才有 Kiki、米聊、微信等。因为微信占据了大家的视线，所以在中国用 WhatsApp 的人很少，不过作为一款经典 App，它是很有研究价值的。据称，其月活跃用户已经超过 2.5 亿人，每天的消息超过 10 亿条。WhatsApp 的界面如图 8-1 所示。

从 WhatsApp 的 UI 上看，它实在是一个非常简单、甚至有点简陋的 App，和微信比起来真是简单得不能再简单了。不过简单归简单，老外他们就是喜欢。

本章的目标是拦截收发的文本聊天信息，不是去读取数据库记录，而是从消息收发的层面去做更多的工作。



图 8-1 WhatsApp 的运行界面

8.2 分析源代码结构，并定位监听点

8.2.1 class-dump 获取 .h 文件

前面已经提过如何获取破解版的 IPA 文件，不管是通过 AppCrackr 破解还是从 91 等网站下载破解版都可以，在此不再赘述。

打开 WhatsApp 的目录，马上就能看到 WhatsApp 的运行文件，如图 8-2 所示。

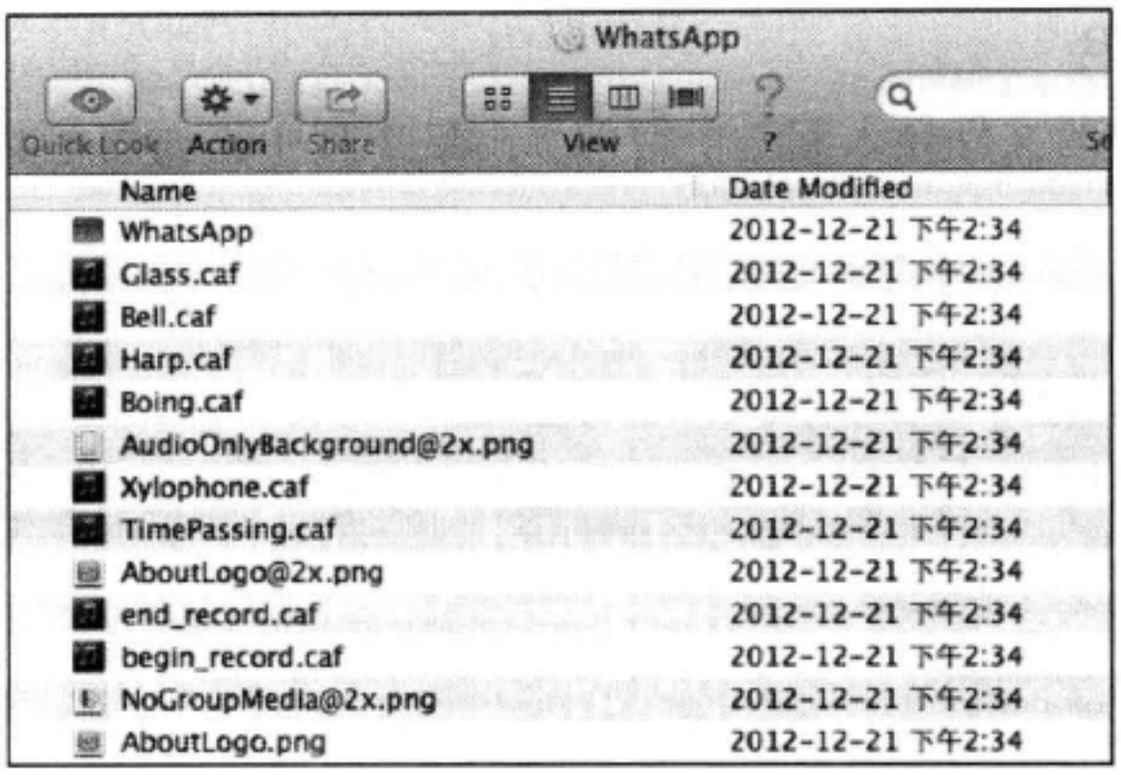


图 8-2　WhatsApp 目录，运行文件是 WhatsApp

导出之后该通过 class-dump 来获取 WhatsApp 的 .h 文件了。

```
class-dump -H WhatsApp -o WhatsApp.h
```

很顺利就拿到了 WhatsApp 的 .h 文件，看一下总共 317 个文件，如图 8-3 所示。这里面应该还包括一些第三方的代码，可以想象 WhatsApp 的整体结构有多轻巧。



图 8-3　WhatsApp 的 .h 目录，共 317 个文件

8.2.2 导入 Xcode

把 WhatsApp.h 中的文件导入到一个空的 Xcode 工程中，如图 8-4 所示，然后再来做代码的分析。

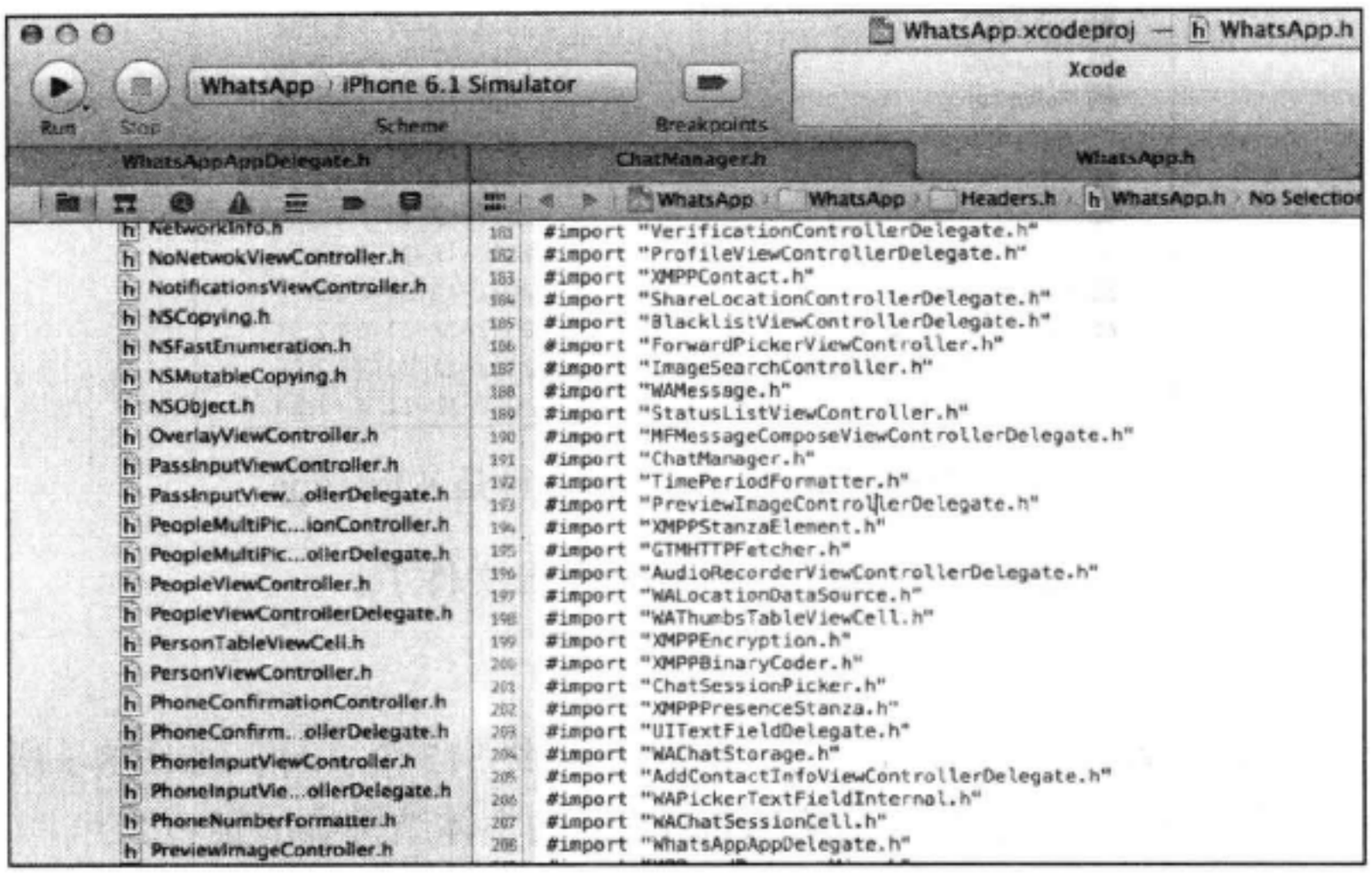


图 8-4 将 WhatsApp 的 .h 文件导入 Xcode

8.2.3 找到 WhatsAppAppDelegate

搜索 didFinishLaunchingWithOptions，如图 8-5 所示。

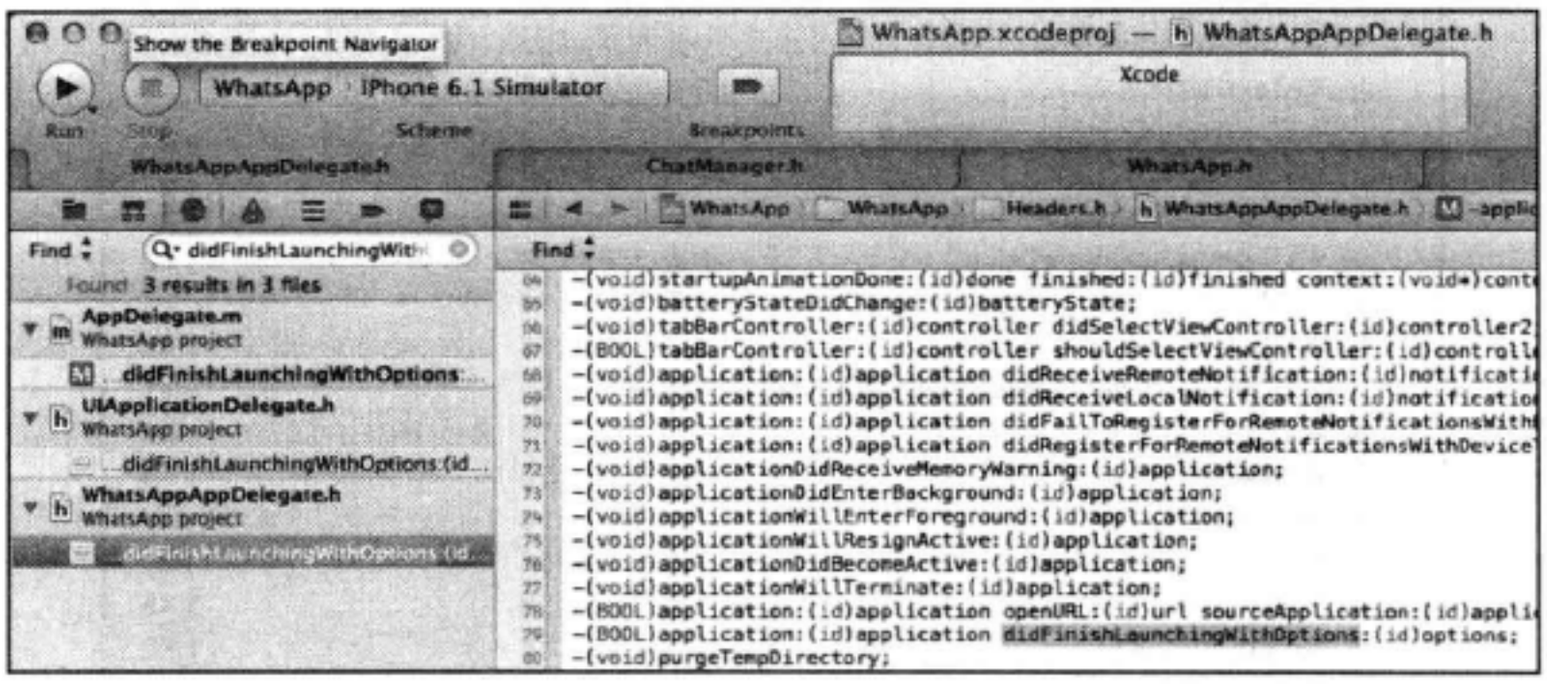


图 8-5 搜索 didFinishLaunchingWithOptions

可以看到有 3 个文件入选，但是前 2 个：AppDelegate 和 UIApplicationDelegate 明显不是我们要找的文件，再看第 3 个 WhatsAppAppDelegate。

各位读者都是经验丰富的 iOS 开发者，一眼就能看出这个文件代表什么：是的，找到 WhatsApp 的入口了！再仔细的查看 WhatsAppAppDelegate，很快就能看出眉目了。

```
-(void)xmppConnectionDidConnect:(id)xmppConnection;
-(void)chatStorageDidRemoveExistingStore:(id)chatStorage;
-(void)chatStorageDidInitializePSC:(id)chatStorage;
-(void)chatStorageDidDeleteChatSessions:(id)chatStorage;
-(void)chatStorageDidUpdateChatSessionList:(id)chatStorage;
-(void)chatStorage:(id)storage didUpdateChatSession:(id)session;
-(void)chatStorage:(id)storage didReceiveMessage:(id)message;
-(void)chatStorage:(id)storage didFailWithError:(id)error;
```

对 WhatsApp 稍微有些了解的读者都知道，WhatsApp 的聊天协议是 xmpp，那么搜索一下 xmpp 这个关键字，竟然发现紧挨着的就是 chatStorage 这样的函数名，简直是意外之喜。

8.2.4　找到 ChatManager

从 WhatsAppAppDelegate 的函数定义可以看到一组针对 chatStorage 的函数，其中有两个函数：didReceiveMessage 和 didFailWithError。

相信各位读者对函数命名规则都是非常熟悉的，再加上对 WhatsApp 开发者水平的信任，已经可以断定这两个函数名是定义在某个 delegate 中，并且它们极有可能就是被定义来接收消息和处理错误的。搜索一下 didReceiveMessage 试试，如图 8-6 所示。

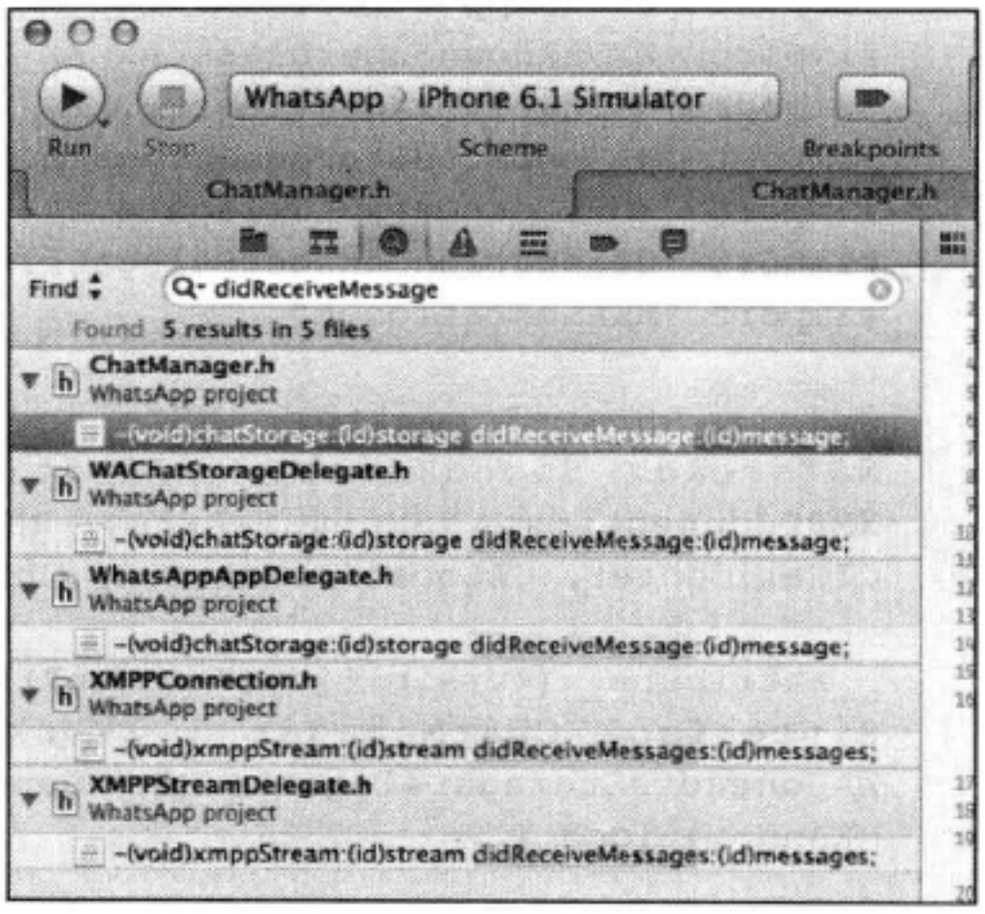


图 8-6　含有 didReceiveMessage 的代码文件

didReceiveMessage 果然是 WAChatStorageDelegate 的接口。

```
@protocol WAChatStorageDelegate
-(BOOL)isRecipientExists:(id)exists;
-(void)getPresenseForChatSession:(id)chatSession;
-(id)imageForContactID:(int)contactID jid:(id)jid ofSize:(int)size;
-(id)contactForJID:(id)jid;
-(void)disableProfilePics;
-(BOOL)profilePicsEnabled;
-(id)connection;
-(void*)sharedAddressBook;
-(void)chatStorageDidRemoveExistingStore:(id)chatStorage;
-(void)chatStorageDidInitializePSC:(id)chatStorage;
-(void)chatStorage:(id)storage didUpdateParticipantWithJID:(id)jid;
-(void)updateChatSessionWithContactData:(id)contactData;
-(void)requestLocationUpdate;
-(void)chatStorage:(id)storage didDeleteMessagesInSession:(id)session;
-(void)chatStorage:(id)storage willDeleteChatSession:(id)session;
-(void)chatStorage:(id)storage didUpdateChatSession:(id)session;
-(void)chatStorage:(id)storage didDeleteChatSessionAtIndex:(id)index;
-(void)chatStorageDidDeleteChatSessions:(id)chatStorage;
```

```
-(void)chatStorageDidUpdateChatSessionList:(id)chatStorage;
-(void)chatStorageDidUpdateBlacklist:(id)chatStorage;
-(id)UIDelegateForMessage:(id)message;
-(void)mediaDownloadFailedForMessage:(id)message withError:(id)error;
-(void)mediaUploadFailedForMessage:(id)message;
-(void)chatStorage:(id)storage didUpdateMessage:(id)message;
-(void)chatStorage:(id)storage didReceiveMessage:(id)message;
-(void)chatStorage:(id)storage didAddMessages:(id)messages;
-(void)chatStorage:(id)storage didFailWithError:(id)error;
@end
```

然而 delegate 本身只是个接口规范，干不了什么实际的事，那么再来看看 ChatManager：

```
#import "WhatsApp-Structs.h"
#import <XXUnknownSuperclass.h> // Unknown library
#import "WAChatStorageDelegate.h"
#import "WAContactsStorageDelegate.h"
#import "XMPPConnectionDelegate.h"
#import "CLLocationManagerDelegate.h"
#import "MKAnnotation.h"

@class NSDateFormatter, NSCalendar, WAContactsStorage, NSDate, NSString,
NSThread, SoundEffect, WAChatStorage, NSArray, NSMutableDictionary,
XMPPConnection, NSMutableArray, NSMutableSet, UIFont, UIColor, UIAlertView,
CTCallCenter, CLLocationManager, CLLocation, NSNumberFormatter;

__attribute__((visibility("hidden")))
@interface ChatManager : XXUnknownSuperclass <WAChatStorageDelegate,
WAContactsStorageDelegate, XMPPConnectionDelegate, CLLocationManagerDelegate,
MKAnnotation> {
...

-(void)chatStorage:(id)storage didUpdateMessage:(id)message;
-(void)chatStorage:(id)storage didReceiveMessage:(id)message;
-(void)chatStorage:(id)storage didAddMessages:(id)messages;
-(void)chatStorage:(id)storage didFailWithError:(id)error;
```

限于篇幅，就不把完整的 ChatManager 贴出来了，各位读者可以自行试验。

在 ChatManager.h 长长的接口函数列表中，除了有 chatStorage 系列之外，还有 xmppConnection、contactsStorage 系列，从函数名上也大致能区分出来各函数的用途：xmppConnection 用于 xmpp 连接，contactsStorage 用于联系人管理，chatStorage 用于聊天会话。核心功能如此集中的一个类，即使最终不是我们要定位的点，也能了解它的作用是很大的！

接下来，就该用代码来验证一下这个判断了。

8.3 编写调试代码

8.3.1 分析函数列表

因为这次的目标是拦截聊天记录，所以关注点应该是 chatStorage 系列函数。

```
-(void)chatStorage:(id)storage didUpdateMessage:(id)message;
-(void)chatStorage:(id)storage didReceiveMessage:(id)message;
-(void)chatStorage:(id)storage didAddMessages:(id)messages;
-(void)chatStorage:(id)storage didFailWithError:(id)error;
```

其中，didFailWithError 可以暂时忽略，那么就只需要关注 didUpdateMessage、didReceive Message 和 didAddMessages 这三个函数即可。

8.3.2　编写 Tweak_WhatsApp.xm

既然初步定位到了函数，那么接下来就要开始考虑在函数中加监听代码了，如下所示：

```
#import "DB/Trace.h"
#import "DB/MonitorDB+Public.h"

%hook WhatsAppAppDelegate

- (BOOL)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2 {
   NSInteger level = [MonitorDB getLogLevel];
   [MonitorDB setLevel:level];
   [Trace trace:[NSString stringWithFormat:@ "---------- Start WhatsApp
   ---------- %@, %d", self, level] level:4];

   %orig;
   return YES;
}

%end

%hook ChatManager

-(void)chatStorage:(id)storage didUpdateMessage:(id)message
{
   [Trace trace:[NSString stringWithFormat:@ "didUpdateMessage %@ %@ %@: %@",
   self, storage, message, [NSThread callStackSymbols]]];
   %orig;
}

-(void)chatStorage:(id)storage didReceiveMessage:(id)message
{
   [Trace trace:[NSString stringWithFormat:@ "didReceiveMessage %@ %@ %@: %@",
   self, storage, message, [NSThread callStackSymbols]]];
   %orig;
}

-(void)chatStorage:(id)storage didAddMessages:(id)messages
{
   [Trace trace:[NSString stringWithFormat:@ "didAddMessages %@ %@ %@: %@",
   self, storage, messages, [NSThread callStackSymbols]
```

```
]];
    %orig;
}

%end
```

在上述代码中，我们除了对 3 个被监听函数各加了一行代码用于输出信息之外，没有再做其他的事情。

而这一行代码输出的内容就是我们所要监听的信息，它包括了函数名、入口参数，以及当前线程的调用栈信息（callStack）。

Trace 输出信息的目标可以是日志，也可以指定到某个特定文件，在本例中，Trace 输出到了一个类似 /var/mobile/Applications/1A96A27A-9CEF-4B2D-ACF4-ACB7EB311071/Documents 目录下的 monitor.db 文件中，其中的 1A96A27A-9CEF-4B2D-ACF4-ACB7EB311071 是 WhatsApp 对应的目录，在不同的设备上其数值不同，配套的代码将在后面整理出来。

8.3.3 编辑 control

编辑 control，填写 Name 和 Description。

```
Package: com.hangcom.monitor
Name: monitor
Depends: mobilesubstrate
Version: 0.0.1
Architecture: iphoneos-arm
Description: WhatsApp 消息监听插件
Maintainer: sam
Author: sam
Section: Tweaks
```

主要目的是修改描述信息。

8.3.4 编辑 monitor.plist

指 定 Filter 中 的 Bundles 值 为 net.whatsapp.WhatsApp。

```
{Filter = { Bundles = ("net.whatsapp.WhatsApp"); }; }
```

指定插件监听的对象是 WhatsApp。这个 Bundle Indentifier 来源于 WhatsApp 目录中的 Info.plist，如图 8-7 所示。

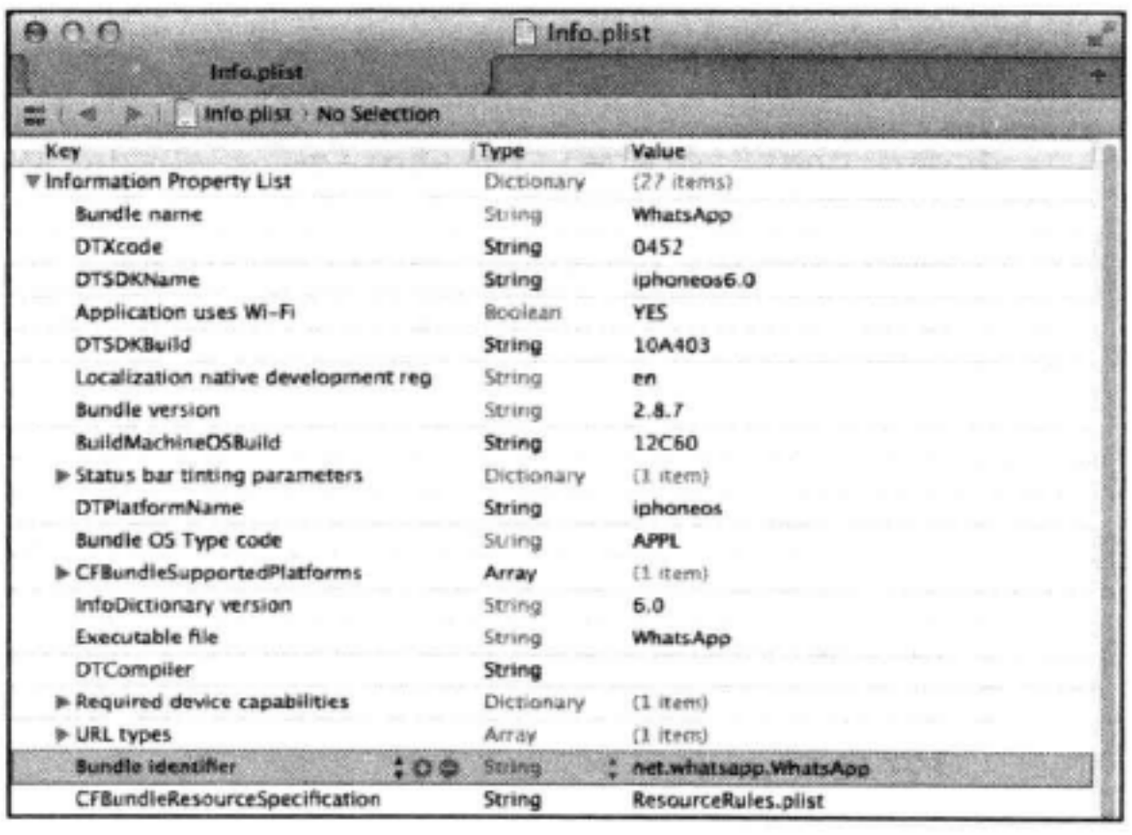


图 8-7 WhatsApp 的 Info.plist

8.3.5 编辑 Makefile

编辑 Makefile，指定 monitor_FILES 等参数。

```
include theos/makefiles/common.mk

TWEAK_NAME = monitor
monitor_FILES = Tweak.xm Tweak_WhatsApp.xm RevealUtil.m DB/NameInfo.m
     DB/Trace.m DB/MonitorDB.m DB/DBM.m
     DB/MonitorDB+Trace.m DB/MonitorDB+Setting.m
monitor_FRAMEWORKS = UIKit CFNetwork
monitor_PRIVATE_FRAMEWORKS = AppSupport
monitor_LDFLAGS = -lsqlite3.0

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
     install.exec "killall -9 WhatsApp"
```

从 monitor_FILES 引用的文件来看，除了 Tweak_WhatsApp.xm 之外，在 DB 目录下还有一组文件，这些文件都是一些附属文件，可以暂不关注。

monitor_FRAMEWORKS 和 monitor_PRIVATE_FRAMEWORKS 用于指定需要依赖的 framework。

monitor_LDFLAGS 是新增的参数，用于导入 SQLite。

8.3.6　编译 + 打包

各配置文件更新完之后就可以进行编译、打包了，如图 8-8 所示。

```
sammatoMacBook-Pro:monitor sam$ vi Makefile
sammatoMacBook-Pro:monitor sam$ make
Making all for tweak monitor...
 Linking tweak monitor...
 Stripping monitor...
 Signing monitor...
sammatoMacBook-Pro:monitor sam$ make package
Making all for tweak monitor...
make[2]: Nothing to be done for `internal-library-compile'.
Making stage for tweak monitor...
dpkg-deb: building package `com.hangcom.monitor' in `./com.hangc
om.monitor_0.0.1-5_iphoneos-arm.deb'.
```

图 8-8　编译 + 打包，生成 deb

顺利完成之后会生成一个新的 deb 文件。

8.3.7　安装

第 3 章已经介绍了使用 iFile 安装的方法，在此不再重复。

安装完成之后，就要开始调试运行一下，看看添加的代码是否生效了。

8.4　调试运行监听插件

8.4.1　运行 WhatsApp，SSH 连接

首先打开 iOS 设备上的 SSH 服务，然后启动 WhatsApp。

SSH 登上设备并进入 WhatsApp/Documents 目录，打开 monitor.db 实时查看输出的日志信息，能看到一条信息：Start WhatsApp，这说明这个 WhatsApp 插件已经正常工作了，如图 8-9 所示。

```
/usr/bin/sqlite3 monitor.db
SQLite version 3.7.13
Enter ".help" for instructions
sqlite> select * from trace;
1|2013-06-26 16:11:22|Trace setLevel 2|4|
2|2013-06-26 16:11:22|---------- Start WhatsApp ---------- <WhatsAppAppDelegate: 0x1
ddb43b0>, 2|4|
```

图 8-9 使用 sqlite 实时监控 monitor.db 采集到的日志信息

8.4.2 发送 / 接收消息，监控日志

接下来我们做一次消息收发的测试，发送的消息是 send message 001，接收的消息是 receive message 001，如图 8-10 所示。

然后再来看输出的信息，如图 8-11 所示。

从发送时间和日志中对应消息的正文来看，发送消息 send message 001 触发了三条日志，即 didAddMessages、didUpdateMessage、didUpdateMessage；接收消息 receive message 001 触发了后两条日志 didAddMessages、disReceiveMessage。

相信基于各位读者开发应用程序的经验，已经大致可以猜到其运行逻辑了，如下：

发送消息：didAddMessages 通知有消息进入队列→ didUpdate Message 更新当前状态为发送 ing → didUpdateMessage。

接收消息：didAddMessages 通知有消息进入队列→ disReceive Message 确认消息接收状态。

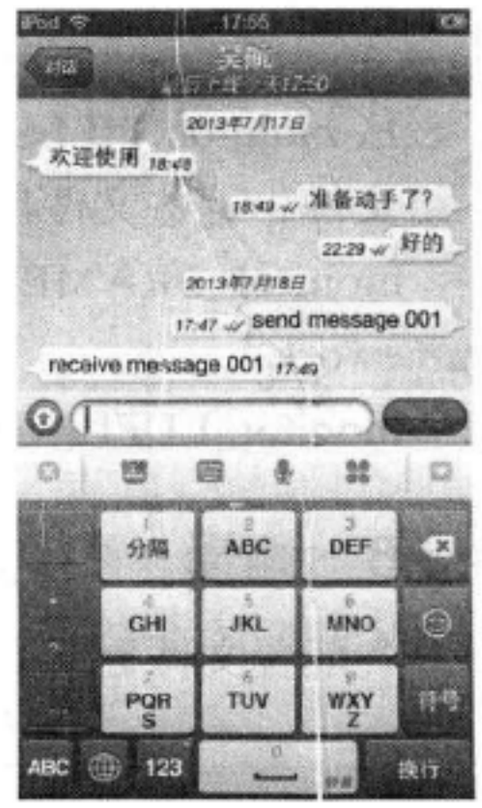


图 8-10 发送、接收消息

time	info
2013-07-18 17:47:06	Trace setLevel 2
2013-07-18 17:47:06	---------- Start WhatsApp ---------- <WhatsAppAppDelegate: 0x1cf56e10>, 2
2013-07-18 17:47:55	didAddMessages <ChatManager: 0x1c6748[70> <WAChatStorage: 0x1c6749f0> (
2013-07-18 17:47:55	didUpdateMessage <ChatManager: 0x1c67 4870> <WAChatStorage: 0x1c6749f0> <...
2013-07-18 17:48:49	didUpdateMessage <ChatManager: 0x1c67 4870> <WAChatStorage: 0x1c6749f0> <...
2013-07-18 17:49:06	didAddMessages <ChatManager: 0x1c6748 70> <WAChatStorage: 0x1c6749f0> (
2013-07-18 17:49:06	didReceiveMessage <ChatManager: 0x1c67 4870> <WAChatStorage: 0x 1c6749f0> ...

图 8-11 monitor.db 中的日志信息，此时是在 Mac 桌面上打开的

然后再进一步看日志的详情，如图 8-12 所示。

```
3|2013-07-18 17:47:55|didAddMessages <ChatManager: 0x1c674870> <WAChatStorage: 0x1c67
49f0> (
    "<WAMessage: 0x1db7b490> (entity: WAMessage; id: 0x1da72f80 <x-coredata://FF8C8D5
4-AFBB-4ED4-808D-E67E45A31AC4/WAMessage/p7> ; data: {\n    chatSession = \"0x1c67b9f0
 <x-coredata://FF8C8D54-AFBB-4ED4-808D-E67E45A31AC4/WAChatSession/p4>\";\n    fromJID
 = nil;\n    groupEventType = 0;\n    groupMember = nil;\n    isFromMe = 1;\n    isSe
nding = 1;\n    lastSession = \"0x1c67b9f0 <x-coredata://FF8C8D54-AFBB-4ED4-808D-E67E
45A31AC4/WAChatSession/p4>\";\n    mediaItem = nil;\n    messageDate = \"2013-07-18 0
9:47:55 +0000\";\n    messageStatus = 0;\n    messageType = 0;\n    participantJID =
nil;\n    pushName = nil;\n    sort = 4;\n    stanzaID = \"1374140829-12\";\n    text
 = \"send message 001\";\n    toJID = \"8613910530574@s.whatsapp.net\";\n    words =
    (\n        \"0x1dace660 <x-coredata://FF8C8D54-AFBB-4ED4-808D-E67E45A31AC4/WAMess
ageWord/p16>\",\n        \"0x1dace6d0 <x-coredata://FF8C8D54-AFBB-4ED4-808D-E67E45A31
AC4/WAMessageWord/p17>\",\n        \"0x1dace5a0 <x-coredata://FF8C8D54-AFBB-4ED4-808D
-E67E45A31AC4/WAMessageWord/p15>\"\n    );\n})"
```

图 8-12 didAddMessages 的详情，此时是通过 SSH 在 iOS 设备上看到的

didAddMessages 信息中，只是看到各 Message 对象的值，没有 send message 001 那样明确的消息内容，暂且把它忽略。

再往下看 didUpdateMessage 信息，显示 text=" send message 001 "，如图 8-13 所示。

```
2013-07-18 17:47:55|didUpdateMessage <ChatManager: 0x1c67
9f0> <WAMessage: 0x1db7b490> (entity: WAMessage; id: 0x1c
-AFBB-4ED4-808D-E67E45A31AC4/WAMessage/p7> ; data: {
    chatSession = "0x1c67b9f0 <x-coredata://FF8C8D54-AFB
tSession/p4>";
    fromJID = nil;
    groupEventType = 0;
    groupMember = nil;
    isFromMe = 1;
    isSending = 0;
    lastSession = "0x1c67b9f0 <x-coredata://FF8C8D54-AFB
tSession/p4>";
    mediaItem = nil;
    messageDate = "2013-07-18 09:47:55 +0000";
    messageStatus = 1;
    messageType = 0;
    participantJID = nil;
    pushName = nil;
    sort = 4;
    stanzaID = "1374140829-12";
    text = "send message 001";
```

图 8-13　didUpdateMessage 收到的信息

再接再厉，didReceiveMessage 也能看到 text = " receive message 001 "，如图 8-14 所示。

```
2013-07-18 17:49:06|didReceiveMessage <ChatManager: 0x1c
49f0> <WAMessage: 0x1db74800> (entity: WAMessage; id: 0x
age/t3614D02B-DD1C-4213-8176-13A2D0F6D2446> ; data: {
    chatSession = "0x1c67b9f0 <x-coredata://FF8C8D54-AFB
tSession/p4>";
    fromJID = "8613910530574@s.whatsapp.net";
    groupEventType = 0;
    groupMember = nil;
    isFromMe = 0;
    isSending = 0;
    lastSession = "0x1c67b9f0 <x-coredata://FF8C8D54-AFB
tSession/p4>";
    mediaItem = nil;
    messageDate = "2013-07-18 09:49:07 +0000";
    messageStatus = 2;
    messageType = 0;
    participantJID = nil;
    pushName = Sam;
    sort = 5;
    stanzaID = "1374140930-17";
    text = "receive message 001";
```

图 8-14　didReceiveMessage 收到的信息

传说有着数亿用户的 WhatsApp 就是这样被逆向找到关键点的，其非常核心的收发消息被拦截了。感觉难吗？似乎太简单了。

8.4.3　保存数据

8.4.2 已经能看到拦截出来的消息内容了，胜利就在眼前，不过要完成一个完整的插件，还需要把功能补全才行。

从上面看到的日志记录中，以及功能的完整程度上考虑，还有 3 个目标要达到：

- ❑ 发送消息时有两条 didUpdateMessage，显然这是同一条信息的不同状态所致。
- ❑ 消息内容是一个 WAMessage 对象，需要从中提取出正文内容。
- ❑ 将消息保存到数据库中。

首先，回到 Xcode 去查看一下 WAMessage 这个类文件：

```
@interface WAMessage : XXUnknownSuperclass {
@private
        BOOL _offlineStorage;
}
@property(assign, nonatomic) BOOL offlineStorage;
@property(retain, nonatomic) WAGroupMember* groupMember;
@property(retain, nonatomic) WAMediaItem* mediaItem;
@property(retain, nonatomic) WAChatSession* chatSession;
@property(retain, nonatomic) NSString* pushName;
@property(retain, nonatomic) NSString* participantJID;
@property(retain, nonatomic) NSNumber* groupEventType;
@property(retain, nonatomic) NSSet* words;
@property(retain, nonatomic) NSNumber* sort;
@property(retain, nonatomic) NSNumber* isSending;
@property(retain, nonatomic) NSNumber* isFromMe;
@property(retain, nonatomic) NSNumber* messageStatus;
@property(retain, nonatomic) NSNumber* messageType;
@property(retain, nonatomic) NSDate* messageDate;
@property(retain, nonatomic) NSString* text;
@property(retain, nonatomic) NSString* stanzaID;
@property(retain, nonatomic) NSString* toJID;
@property(retain, nonatomic) NSString* fromJID;
-(void)awakeFromInsert;
-(void)awakeFromFetch;
@end
```

根据这个定义再来看前面拦截的 didUpdateMessage 和 didReceiveMessage 函数信息，可以看到在发送文本消息时，第 1 次 didUpdateMessage 时的关键字段如下：

```
isFromMe = 1;
messageStatus = 1;
messageType = 0;
text ="send message 001";
```

第 2 次 didUpdateMessage 时的关键字段如下：

```
isFromMe = 1;
messageStatus = 2;
messageType = 0;
text = "send message 001";
```

而在接收信息时，只触发一次 didReceiveMessage，就能看到 messageStatus=2。

```
isFromMe = 0;
```

```
messageStatus = 2;
messageType = 0;
text = "receive message 001";
```

由 didReceiveMessage 的 messageStatus，可以反向地得出结论：前面两个 didUpdateMessage 消息，只需要关心 messageStatus=2 这条，这个值 2 应该就相当于已完成!

综合这个结论，就可以开始动手把需要的代码补充完整了，步骤如下：

首先，更新 Tweak_WhatsApp.xm，更新 ChatManager 部分的代码。

```
%hook ChatManager

-(void)chatStorage:(id)storage didUpdateMessage:(id)message
{
    //[Trace trace:[NSString
    stringWithFormat:@ "didUpdateMessage %@ %@ %@: %@", self, storage, message,
[NSThread callStackSymbols]]];
    %orig;
    [MonitorDB sendWhatsAppMsg:message];
}

-(void)chatStorage:(id)storage didReceiveMessage:(id)message
{
     //[Trace trace:[NSString stringWithFormat:@ "didReceiveMessage %@ %@ %@:
%@", self, storage, message, [NSThread callStackSymbols]]];
    %orig;
    [MonitorDB receiveWhatsAppMsg:message];
}

-(void)chatStorage:(id)storage didAddMessages:(id)messages
{
     //[Trace trace:[NSString stringWithFormat:@ "didAddMessages %@ %@ %@: %@",
self, storage, messages, [NSThread callStackSymbols]]];
    %orig;
}

%end
```

从上面代码中可看到：

- ❑ didUpdateMessage 函数中增加了一个 [MonitorDB sendWhatsAppMsg:message];
- ❑ didReceiveMessage 函数中增加了一个 [MonitorDB receiveWhatsAppMsg:message];
- ❑ didAddMessages 函数已经没有价值，可以注释掉或删除。

从上面的内容可以看到，增加的 [MonitorDB sendWhatsAppMsg:message] 和 [MonitorDB receiveWhatsAppMsg:message] 这一对函数，它们存在于前面提到过的 MonitorDB 中。

MonitorDB 是笔者基于监控目的慢慢积累形成的一个 DB 代码目录，由一组小的数据库功能类组成，由图 8-15 可见，除了 MonitorDB+WhatsApp 之外，其他类文件都是用于完成基础配置功能的，比如 MonitorDB+Trace 用于保存日志信息，MonitorDB+Setting 用于设置参

数（比如 Log Level 等设定参数）等。

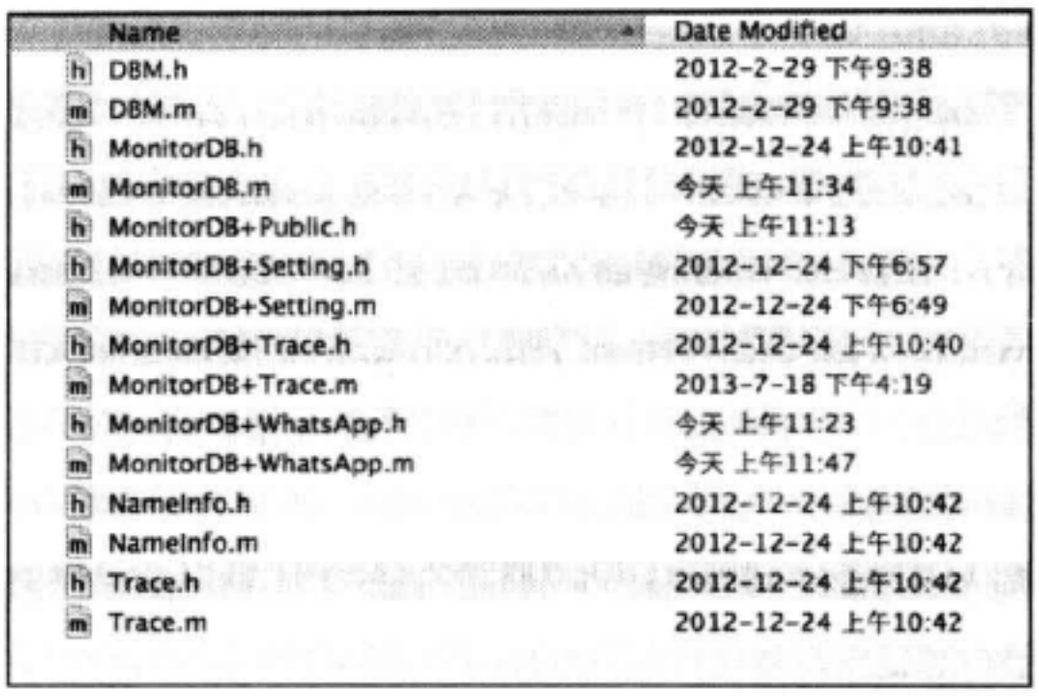


图 8-15 MonitorDB 的目录结构

现在再来看看这两个新增的函数所在的位置：MonitorDB+WhatsApp.m。

```
//
//  MonitorDB+WhatsApp.m
//
//  Created by sam on 11-11-20.
//  Copyright 2011 年 __MyCompanyName__. All rights reserved.
//

#import "MonitorDB+Trace.h"
#import "NameInfo.h"
#import "../WAMessage.h"

@implementation MonitorDB (WhatsApp)

- (BOOL)createTable_WhatsApp:(sqlite3*)db
{
    char *sql = "create table WhatsApp ( \
    ROWID integer primary key, \
    time timestamp, \
    whatsAppid text, \
    text text, \
    type integer, \
    message text, \
    reverse1 text, \
    reverse2 text) ";
    return [self createTable:db SQL:sql Table:@ "whatsapp"];
}


- (BOOL)insertWhatsAppMsg:(NSDictionary *)dict type:(NSInteger)type
{
    BOOL isOK = YES;
```

```
    static char *line = "INSERT INTO whatsapp (time, whatsAppid, text, type,
message, reverse1, reverse2) VALUES(?,?,?,?,?,?,?)";
    sqlite3_stmt *statement;
    int success = sqlite3_prepare_v2(database, line, -1, &statement, NULL);
    if (success != SQLITE_OK) {
        isOK = NO;
    } else {
        int i = 1;
        sqlite3_bind_double(statement, i++, [[dict objectForKey:@ "time"]
timeIntervalSince1970]);
        sqlite3_bind_text(statement, i++, [[dict objectForKey:@ "remoteid"]
UTF8String], -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(statement, i++, [[dict objectForKey:@ "body"]
UTF8String], -1, SQLITE_TRANSIENT);
        sqlite3_bind_int(statement, i++, type);
        sqlite3_bind_text(statement, i++, [[NSString stringWithFormat:@ "%@ ",
dict] UTF8String], -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(statement, i++, [@"1" UTF8String], -1, SQLITE_
TRANSIENT);
        sqlite3_bind_text(statement, i++, [@"" UTF8String], -1, SQLITE_
TRANSIENT);

        success = sqlite3_step(statement);
        sqlite3_finalize(statement);

        if (success != SQLITE_DONE) {
            isOK = NO;
        } else {
            //NSLog(@"insert %d\n", id);
        }
    }
    return isOK;
}

+ (NSDictionary *)generateDict:(WAMessage *)msg
{
    NSMutableDictionary *dict = [NSMutableDictionary dictionary];
    [dict setObject:msg.messageDate forKey:@ "time"];
    [dict setObject:msg.toJID forKey:@ "remoteid"];
    [dict setObject:msg.text forKey:@ "body"];
    // extra message
    [dict setObject:msg.fromJID forKey:@ "fromJID"];
    [dict setObject:msg.isFromMe forKey:@ "isFromMe"];
    [dict setObject:msg.messageType forKey:@ "messageType"];
    return dict;
}

+ (BOOL)sendWhatsAppMsg:(WAMessage *)msg
{
    BOOL tf = NO;
    if ([msg.messageStatus intValue] == 2) {
```

```
        tf = [[MonitorDB shareInstance] insertWhatsAppMsg:[MonitorDB
generateDict:msg] type:1];
    }
    return tf;
}

+ (BOOL)receiveWhatsAppMsg:(WAMessage *)msg
{
    BOOL tf = NO;
    if ([msg.messageStatus intValue] == 2) {
        tf = [[MonitorDB shareInstance] insertWhatsAppMsg:[MonitorDB
generateDict:msg] type:0];
    }
    return tf;
}

@end
```

从上面的代码中可看到：

- ❑ createTable_WhatsApp 函数创建了 whatsapp 表。此处代码使用基于 SQLite 的原生代码，因为这部分代码是一直积累下来的，读者可以用自己喜欢的方式来处理这部分数据库代码。
- ❑ insertWhatsAppMsg 函数用于添加一条消息记录。
- ❑ generateDict 将 WAMessage 对象中的参数提取出来，供 insertWhatsAppMsg 使用。
- ❑ sendWhatsAppMsg 用于插入发送的 WAMessage 消息。插入之前需要检查一下 messageStatus，因为之前已经提到，只有 messageStatus==2 才是已完成的消息。
- ❑ receiveWhatsAppMsg 用于插入接收的 WAMessage 消息。此处的 messageStatus==2 其实已不重要，不过是为了与 sendWhatsAppMsg 函数保持一致。
- ❑ #import " ../WAMessage.h "，使用的就是 WhatsApp 中的 WAMessage.h。不过，直接引入时会报编译错误，需要去掉如下这样一行代码：

```
#import <XXUnknownSuperclass.h> // Unknown library
```

并且将 WAMessage 的继承关系从 XXUnknownSuperclass 改成 NSObject：

```
@interface WAMessage : NSObject {
```

然后看看 WhatsApp 的文本消息是不是从此被留在这么一个小小的 monitor.db 中？除非 WhatsApp 更换 didUpdateMessage、didReceiveMessage 这样的接口，否则无论它怎么升级 App，这些记录该如何截取还会如何截取！掌握方法之后，即使接口变了，我们的插件只要稍做调整就同样能升级。

8.5 代码结果整理

在前面 4 节中，一步步地讲解了如何定位 WhatsApp 的关键运行点、截取并保存数据的

知识点。因为这些知识点散落在各处有些零乱，所以本节对这些关键信息做一个整体的梳理，以期能更好地帮助读者理解本章的内容。

（1）获取 WhatsApp 的 .h 文件，分析函数列表，并定位到 ChatManager。

（2）编写 tweak 代码，为 ChatManager 加入调试代码。发现关键函数是 didUpdateMessage 和 didReceiveMessage。

（3）将 didUpdateMessage 和 didReceiveMessage 中的 WAMessage 保存到 MonitorDB 中。

就这么几步，即可完成一个插件的开发，而一个经典的 App 就这样被攻陷了。除了辅助的 DB 代码，一个 Tweak_WhatsApp.xm 就完成了任务。最后生成的 tweak 目录也非常简单，如图 8-16 所示。相对于大而全的 App，这是真正的小而美！

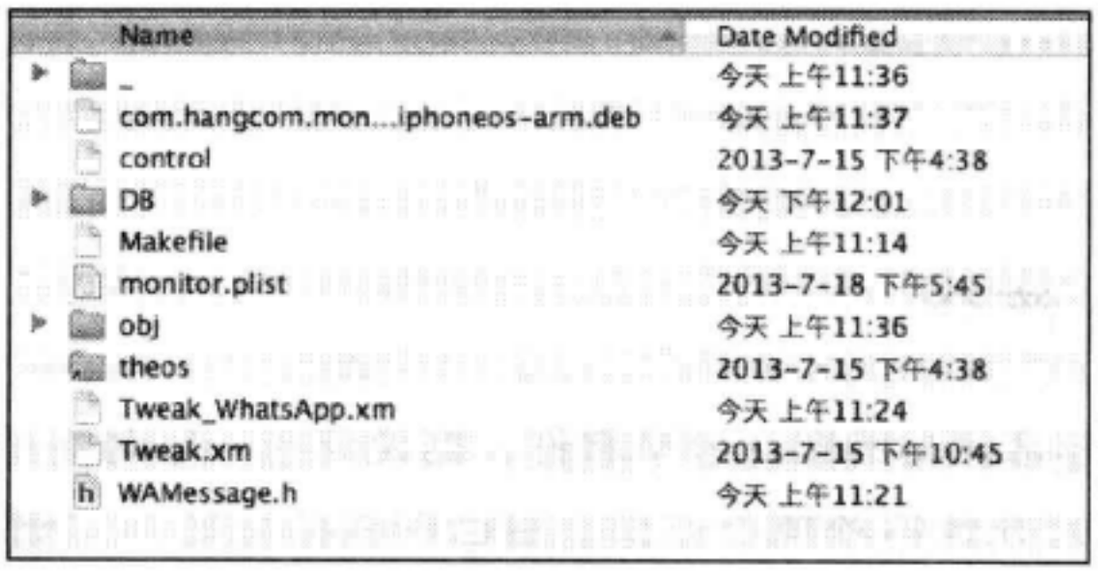


图 8-16　监控 WhatsApp 的 tweak 目录文件

8.6　小结

本章以 WhatsApp 为目标制作了一个小插件，安装之后能够对 WhatsApp 的聊天消息进行拦截记录。

作为一家拥有数亿用户的顶尖软件，WhatsApp 的制作水平、开发质量都是毋庸置疑的，然而却能在这么短的时间里被轻松攻破，恰恰证明了一个很有意思的结论：程序越经典，架构设计其实越清晰；而架构越清晰，逆向分析起来也就越容易。

这个结论相信读者们都是能理解的：我们参与到别的项目中，软件开发质量的高低直接决定我们掌握它的速度。通过对 WhatsApp 的进一步分析，我们也能对这一经典 App 有更深的了解，而不仅仅是这一点消息机制。

最后，赞美一下 WhatsApp 漂亮的代码结构。

第 9 章

实战 3：Instagram 图片保存

9.1 Instagram 简介

Instagram，一款非常流行的图片分享软件，据说仅在一年时间里就有了 3000 万的 iOS 平台用户。它允许用户在任何环境下抓拍下自己的生活记忆，选择图片的滤镜样式，一键分享至各大社交网站。2012 年被 Facebook 花 10 亿美元收购，真是令人既羡慕又嫉妒呀！

Instagram 那么值钱也确实是因为它很牛，用过的朋友恐怕都会对 Instagram 的精彩内容以及使用的流畅感觉记忆犹新。这么经典的一款 App 自然是值得研究的。图 9-1 是 Instagram 的运行界面。

照片很美，但是只能浏览，无法保存到本地（除非截屏，或者使用 Instagram 提供的某些付费 VIP 功能）。事实上，相信部分读者都知道，这些图片是放在对应的 UIImageView 上的，只要取出对应的 UIImage 对象就可以实现保存到本地的任务了。



图 9-1 Instagram 的运行界面

于是，本章的目标就有了：将 Instagram 的图片保存到本地相册。

9.2 分析源代码结构，并定位图片展示点

9.2.1 用 class-dump 获取 .h 文件

前面已经介绍过如何获取破解版的 IPA 文件，在此不再赘述。

首先，打开 Instagram 的目录，马上就能看到 Instagram 的运行文件，如图 9-2 所示。

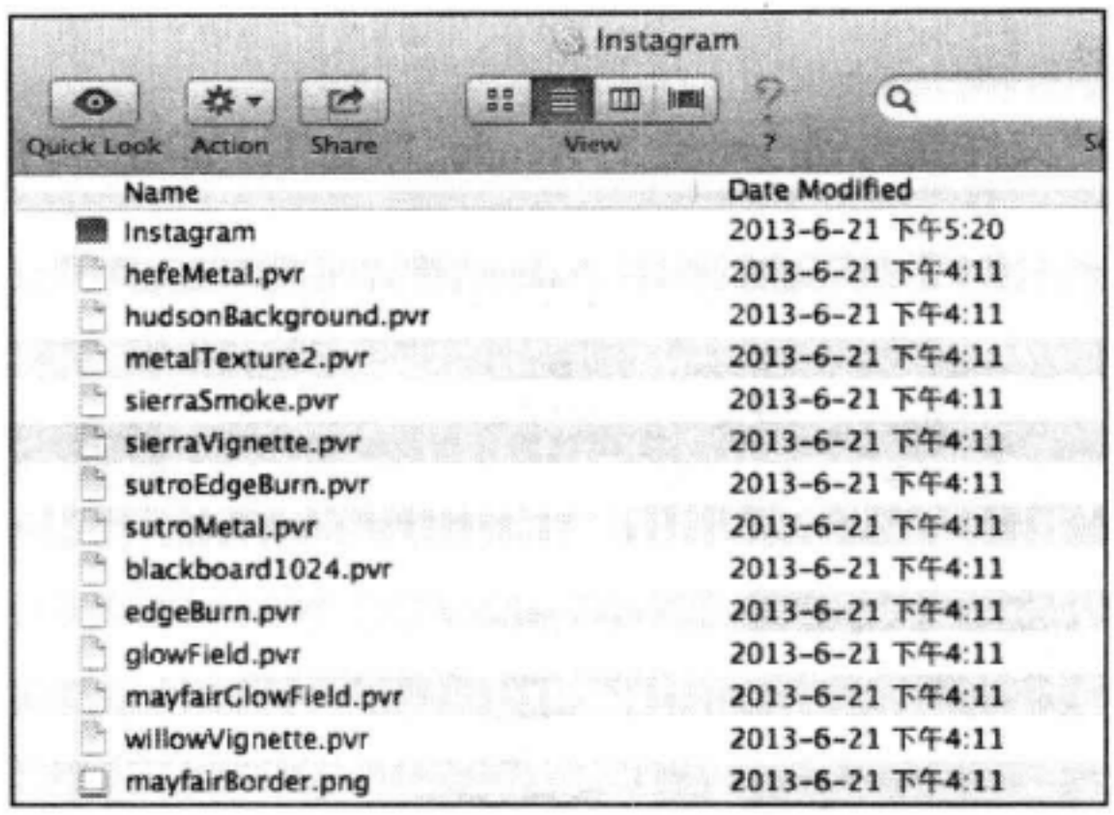


图 9-2　Instagram 目录，运行文件是 Instagram

然后用 class-dump 导出 .h 文件。

```
class-dump -H Instagram -o Instagram.h
```

很顺利就拿到了 Instagram 的 .h 文件，进去看一下，总共有 785 个文件，如图 9-3 所示。

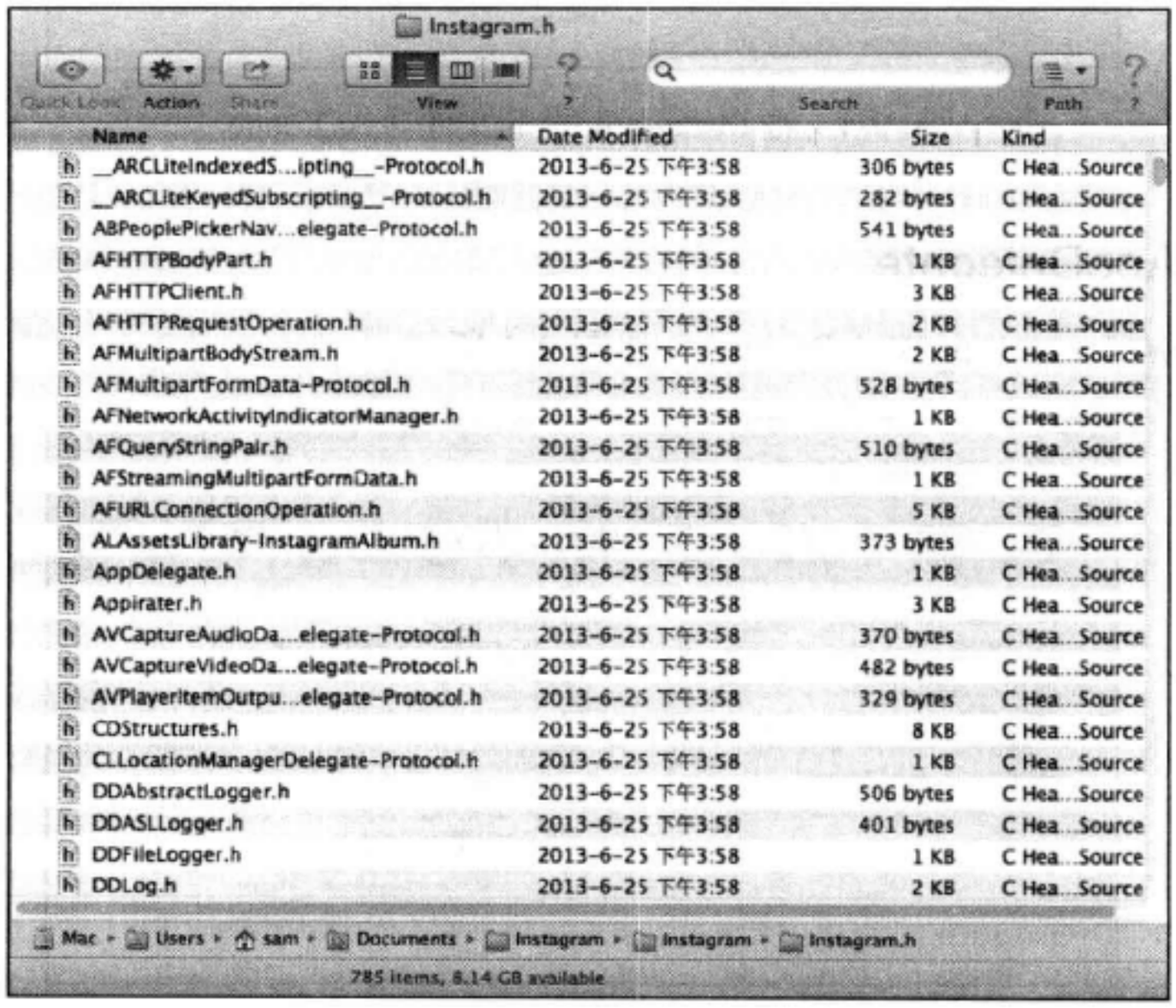


图 9-3　Instagram 的 .h 目录，共 785 个文件

“785”这个文件数目比较适中，因为毕竟 Instagram 的界面那么复杂，而且功能也很丰富。一般来说，常见的 App 所拥有的 .h 个数都在 300 ～ 1000，超过 1000 就有点大了，要是达到 2000 ～ 3000，难以想象那个 App 有多复杂。

9.2.2 导入 Xcode

把 Instagram.h 中的文件导入到一个空的 Xcode 工程中，如图 9-4 所示，然后再来分析代码。

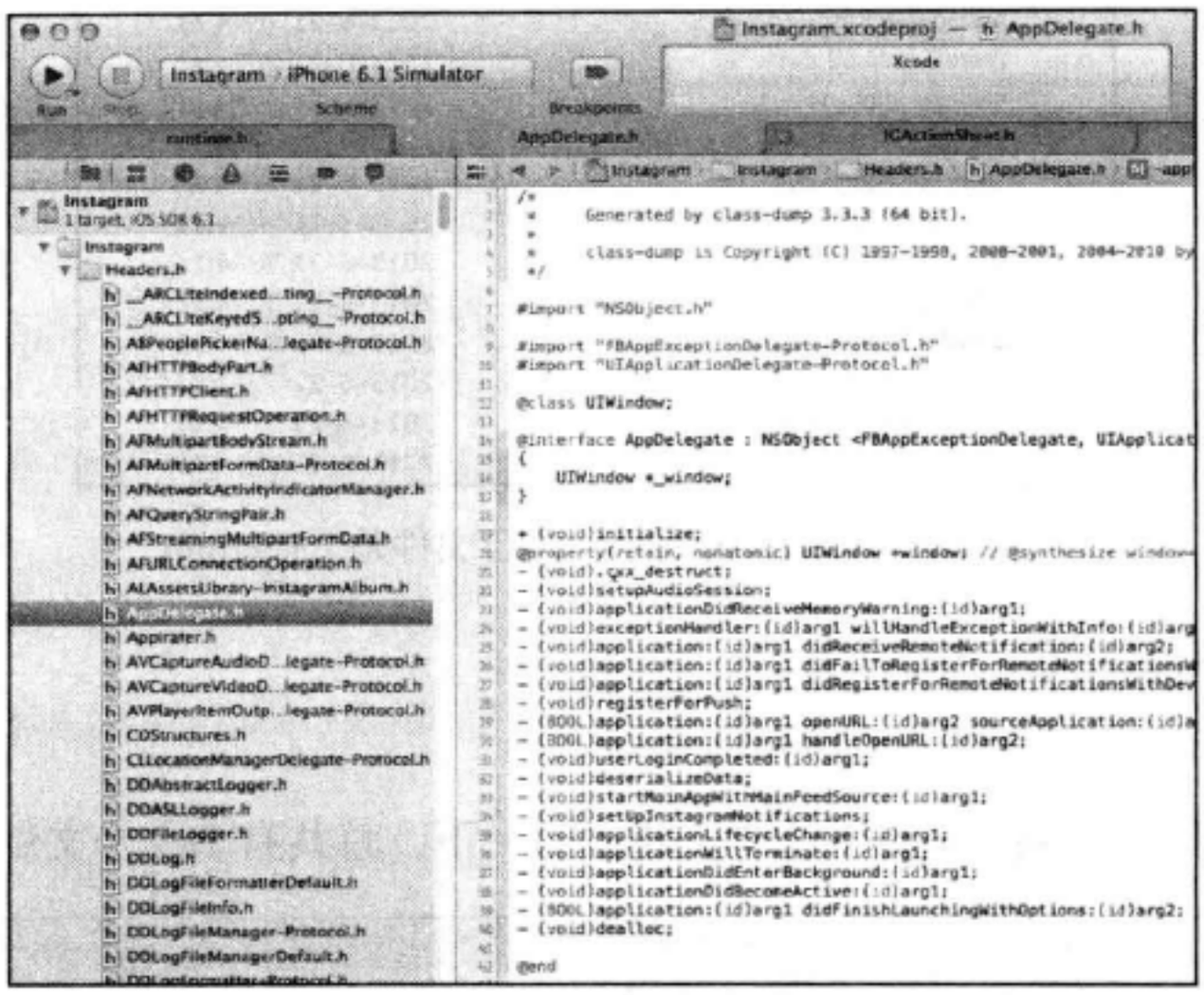


图 9-4 将 Instagram 的 .h 文件导入 Xcode

9.2.3 找到 AppDelegate

搜索 didFinishLaunchingWithOptions，如图 9-5 所示。

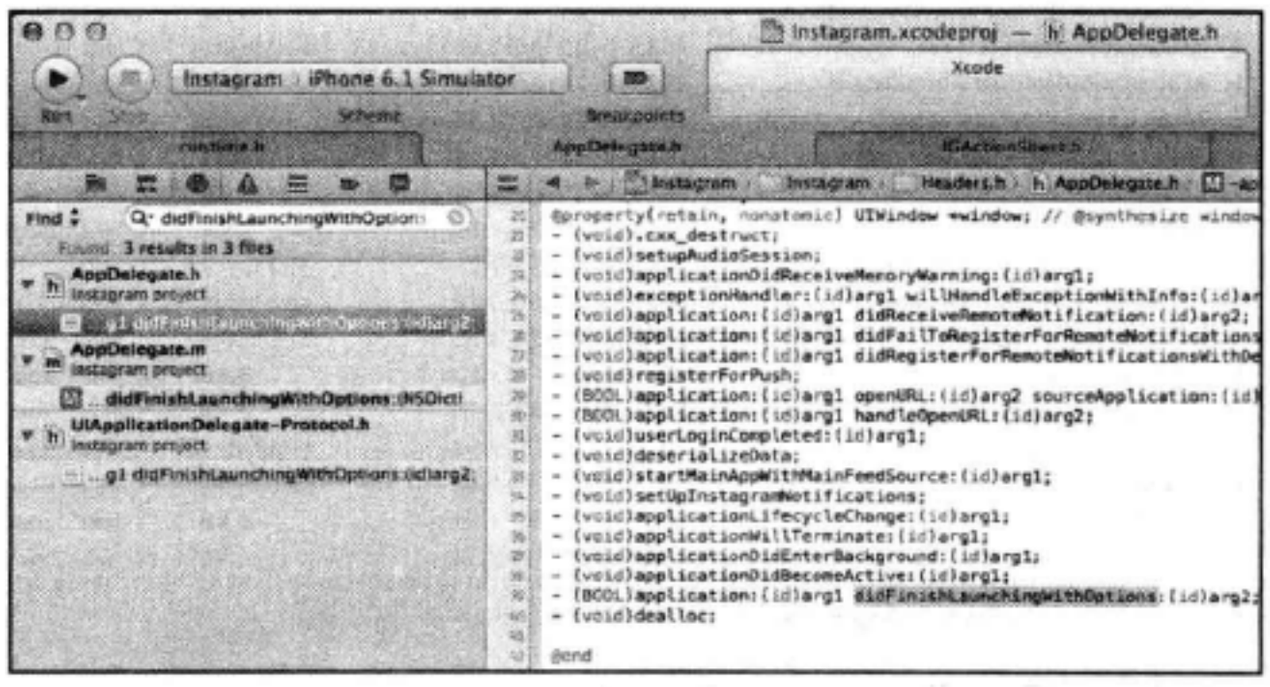


图 9-5 搜索 didFinishLaunchingWithOptions

可以看到有 3 个文件入选，但是只有第 1 个 AppDelegate 才是 Instagram 对应的入口 delegate。

不过，AppDelegate 中并没有什么一眼就能看出具有很大价值的东西，幸好，我们要的

也只是一个入口，有了这个 AppDelegate，就可以把 Reveal 注入进来了。

```
+ (void)initialize;
@property(retain, nonatomic) UIWindow *window; // @synthesize window=_window;
- (void).cxx_destruct;
- (void)setupAudioSession;
- (void)applicationDidReceiveMemoryWarning:(id)arg1;
- (void)exceptionHandler:(id)arg1 willHandleExceptionWithInfo:(id)arg2;
- (void)application:(id)arg1 didReceiveRemoteNotification:(id)arg2;
- (void)application:(id)arg1 didFailToRegisterForRemoteNotificationsWithError:(i
d)arg2;
- (void)application:(id)arg1 didRegisterForRemoteNotificationsWithDeviceToken:(i
d)arg2;
- (void)registerForPush;
- (BOOL)application:(id)arg1 openURL:(id)arg2 sourceApplication:(id)arg3
annotation:(id)arg4;
- (BOOL)application:(id)arg1 handleOpenURL:(id)arg2;
- (void)userLoginCompleted:(id)arg1;
- (void)deserializeData;
- (void)startMainAppWithMainFeedSource:(id)arg1;
- (void)setUpInstagramNotifications;
- (void)applicationLifecycleChange:(id)arg1;
- (void)applicationWillTerminate:(id)arg1;
- (void)applicationDidEnterBackground:(id)arg1;
- (void)applicationDidBecomeActive:(id)arg1;
- (BOOL)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2;
- (void)dealloc;
```

9.2.4　注入 Reveal

前面已经介绍过如何通过 Reveal 注入 App，在此不作过多的说明，只是把对应的步骤和代码贴出来。

（1）上传 libReveal.dylib

把 libReveal.dylib 上传到 Instagram 的 Documents 目录下。

（2）创建 Tweak_Instagram.xm

```
#import "RevealUtil.h"

%hook AppDelegate
- (BOOL)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2 {
    %orig;

    RevealUtil *ru = [[RevealUtil alloc] init];
    [ru startReveal];

    return YES;
}

%end
```

可以看到我们并没有做别的事情，只是在 didFinishLaunchingWithOptions 中启动了 RevealUtil。然后将这个 Tweak_Instagram.xm 加入到 monitor 工程目录中。

（3）更新 monitor.plist，指定 Bundle Identifier 为 com.burbn.instagram

```
{Filter = { Bundles = ("com.burbn.instagram"); }; }
```

（4）更新 Makefile，添加 Tweak_Instagram.xm 和 RevealUtil.m

```
include theos/makefiles/common.mk

TWEAK_NAME = monitor
monitor_FILES = Tweak_Instagram.xm RevealUtil.m DB/NameInfo.m DB/Trace.
m DB/MonitorDB.m DB/DBM.m DB/MonitorDB+Trace.m DB/MonitorDB+Setting.m DB/
MonitorDB+WhatsApp.m
monitor_FRAMEWORKS = UIKit CFNetwork
monitor_PRIVATE_FRAMEWORKS = AppSupport
monitor_LDFLAGS = -lsqlite3.0

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 Instagram"
```

（5）编译、打包、安装

这些步骤与 3.2.2 节是一致的，这里不作过多说明。

9.2.5 寻找图片页相关 class

完成 Reveal 的注入之后，启动 Instagram，看 UI 对应的元素，如图 9-6 所示。



图 9-6 在 Reveal 中看到 Instagram 的 UI 结构

图 9-6 的左侧是整个 UI 元素的构成图，右侧是 Instagram 的界面展现，如果直接点击中央的图片，可以看到选中效果以及左侧的对应元素变换。如图 9-7 中所看到的，此时选中的 UI 元素已经指向了 IGFeedItemPhotoCell 和 IGImageView，而在这两者之间，还隔了一层 IGImageProgressView，所以这 3 个类对应的 .h 文件就是我们目前需要关注的。



图 9-7　选中 IGFeedItemPhotoCell 时的效果

首先，让我们来看一看这 3 个 .h 文件。

1. IGFeedItemPhotoCell.h

IGFeedItemPhotoCell.h 的内容如下：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve
Nygard.
*/

#import "IGFeedItemMediaCell.h"

#import "IGImageProgressViewDelegate-Protocol.h"
#import "UIGestureRecognizerDelegate-Protocol.h"

@class IGFeedItem, IGImageProgressView, IGUserInTagDisplayView, UIImageView,
UITapGestureRecognizer;

@interface IGFeedItemPhotoCell : IGFeedItemMediaCell
   <IGImageProgressViewDelegate, UIGestureRecognizerDelegate>
```

```
{
    UIImageView *_inTagIndicatorView;
    IGUserInTagDisplayView *_inTagView;
    BOOL _isShowingUsertagsIndicator;
    UITapGestureRecognizer *_doubleTapRecognizer;
    UITapGestureRecognizer *_singleTapRecognizer;
    id <IGUserInTagDisplayDelegate> _tagViewDelegate;
    id <IGFeedItemPhotoCellDelegate> _imageLoadDelegate;
    BOOL _isShowingUsertags;
    IGFeedItem *_feedItem;
    IGImageProgressView *_photoImageView;
}

+ (float)height;
@property(readonly, nonatomic) IGImageProgressView *photoImageView; // @
synthesize photoImageView=_photoImageView;
@property(retain, nonatomic) IGFeedItem *feedItem; // @synthesize feedItem=_
feedItem;
@property(readonly, nonatomic) BOOL isShowingUsertags; // @synthesize
isShowingUsertags=_isShowingUsertags;
@property(nonatomic) id <IGFeedItemPhotoCellDelegate> imageLoadDelegate; // @
synthesize imageLoadDelegate=_imageLoadDelegate;
@property(nonatomic) id <IGUserInTagDisplayDelegate> tagViewDelegate; // @
synthesize tagViewDelegate=_tagViewDelegate;
- (void).cxx_destruct;
- (void)onUsertagGroupWasEdited;
- (void)progressImageView:(id)arg1 didFailLoadWithError:(id)arg2;
- (void)progressImageViewDidLoadImage:(id)arg1;
- (void)showUsertags:(BOOL)arg1 animated:(BOOL)arg2;
- (void)animationDidStop:(id)arg1 finished:(BOOL)arg2;
- (void)showUsertagsIndicator:(BOOL)arg1 animated:(BOOL)arg2;
- (BOOL)hasUsertags;
- (void)updateSingleTapRecognizer;
- (BOOL)gestureRecognizer:(id)arg1 shouldReceiveTouch:(id)arg2;
- (BOOL)gestureRecognizerShouldBegin:(id)arg1;
- (void)onDoubleTap:(id)arg1;
- (void)onTap:(id)arg1;
- (void)swiped:(id)arg1;
- (void)onTagIndicatorTapped;
- (id)initWithReuseIdentifier:(id)arg1;
- (void)dealloc;

@end
```

IGFeedItemPhotoCell 是一个标准的 UITableViewCell 文件，并且已经定义了 onTap、onDoubleTap 等手势事件。理论上，可以直接覆盖这些函数，这样就相当于改写了 Instagram 原来的动作逻辑。

根据 Reveal 中显示的从属关系来分析，在这个类中，需要重点关注的是 IGImageProgress-View *_photoImageView 对象，因此接下来就来看看 IGImageProgressView.h 文件。

2. IGImageProgressView.h

IGImageProgressView.h 的内容如下：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve
Nygard.
*/

#import "UIView.h"

#import "IGImageViewDelegate-Protocol.h"

@class IGCircularProgressView, IGFeedItem, IGImageView, IGRequest,
IGSimpleButton, NSURL;

@interface IGImageProgressView : UIView <IGImageViewDelegate>
{
    NSURL *_imageURL;
    IGRequest *_imageLoadRequest;
    IGSimpleButton *_failureView;
    IGCircularProgressView *_progressView;
    id <IGImageProgressViewDelegate> _delegate;
    IGImageView *_photoImageView;
    BOOL _delegateNeedsImageNow;
    IGFeedItem *_feedItem;
    int _loadStatus;
}

@property(readonly, nonatomic) int loadStatus; // @synthesize loadStatus=_
loadStatus;
@property(retain, nonatomic) IGFeedItem *feedItem; // @synthesize feedItem=_
feedItem;
@property(nonatomic) BOOL delegateNeedsImageNow; // @synthesize
delegateNeedsImageNow=_delegateNeedsImageNow;
@property(readonly, nonatomic) IGImageView *photoImageView; // @synthesize
photoImageView=_photoImageView;
@property(nonatomic) id <IGImageProgressViewDelegate> delegate; // @synthesize
delegate=_delegate;
@property(retain, nonatomic) IGCircularProgressView *progressView; // @
synthesize progressView=_progressView;
- (void).cxx_destruct;
- (void)imageViewFailedToLoadImage:(id)arg1 error:(id)arg2;
- (void)imageViewLoadedImage:(id)arg1;
- (void)doMediaFetch;
- (void)update;
- (void)makeFailureView;
- (id)initWithFrame:(struct CGRect)arg1;
- (void)dealloc;

@end
```

IGImageProgressView 只是一个中间层，有一些 LoadImage 动作，不过不需要关心它，因为我们需要的只是 IGImageView *_photoImageView 对象。

3. IGImageView.h

IGImageView.h 的内容如下：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve
Nygard.
*/

#import "UIImageView.h"

#import "IGMediaRequestDelegate-Protocol.h"

@class NSURL, UIImage;

@interface IGImageView : UIImageView <IGMediaRequestDelegate>
{
    id <IGImageViewDelegate> _delegate;
    BOOL _cacheDecodedImage;
    NSURL *_imageURL;
    int _imagePriority;
    UIImage *_placeholderImage;
    BOOL _delegateNeedsImageNow;
    BOOL _shouldDecodeAsyncronously;
}

@property(nonatomic) BOOL shouldDecodeAsyncronously; // @synthesize
shouldDecodeAsyncronously=_shouldDecodeAsyncronously;
@property(nonatomic) BOOL delegateNeedsImageNow; // @synthesize
delegateNeedsImageNow=_delegateNeedsImageNow;
@property(retain, nonatomic) UIImage *placeholderImage; // @synthesize
placeholderImage=_placeholderImage;
@property(nonatomic) int imagePriority; // @synthesize imagePriority=_
imagePriority;
@property(retain, nonatomic) NSURL *imageURL; // @synthesize imageURL=_
imageURL;
@property(nonatomic) BOOL cacheDecodedImage; // @synthesize cacheDecodedImage=_
cacheDecodedImage;
@property(nonatomic) __weak id <IGImageViewDelegate> delegate; // @synthesize
delegate=_delegate;
- (void).cxx_destruct;
- (void)mediaRequest:(id)arg1 didFailWithError:(id)arg2 forURL:(id)arg3;
- (void)mediaRequest:(id)arg1 didLoadMediaWithData:(id)arg2 forURL:(id)arg3;
- (void)cancelImageLoad;
- (void)stopListeningForImageLoadNotification;
- (void)notifyDelegateOfSuccess;
```

```
- (int)discaredImagePriority;
- (void)setImage:(id)arg1;
- (void)setImageWithData:(id)arg1;
- (id)initWithFrame:(struct CGRect)arg1;
- (void)dealloc;

@end
```

IGImageView 的定义非常简单，就是一个 UIImageView 的继承类，因此对我们来说，要的只是这个 IGImageView 的 image 对象。

也就是说，只要通过 IGFeedItemPhotoCell 能够取到 IGImageView 的 image，就可以达到目标。那么，接下来就可以试试用代码来实现了。

9.3　保存图片

9.3.1　尝试保存图片

首先，要确定是否可以如之前提到的那样拿到 IGImageView 中的 image 对象。为此，编辑 Tweak_Instagram.xm 并覆盖 IGFeedItemPhotoCell 的 onTap 函数：

```
%hook IGFeedItemPhotoCell
- (void)onTap:(id)arg1
{
    UIImage *image = [[[self photoImageView] photoImageView] image];
    UIImageWriteToSavedPhotosAlbum(image, nil, nil, nil);
}

%end
```

从 IGFeedItemPhotoCell 与 IGImageView 的从属关系上看，我们要取的 image 即是 [[[self photoImageView] photoImageView] image]，不过这种在 Xcode 里常见的写法会导致编译上的问题，如图 9-8 所示。

```
sammatoMacBook-Pro:monitor sam$ vi Tweak_Instagram.xm
sammatoMacBook-Pro:monitor sam$ make
Making all for tweak monitor...
 Preprocessing Tweak_Instagram.xm...
 Compiling Tweak_Instagram.xm...
cc1objplus: warnings being treated as errors
Tweak_Instagram.xm: In function 'void $_ungrouped$IGFeedItemPhotoCell$onTap$(IGFeed
ItemPhotoCell*, objc_selector*, objc_object*)':
Tweak_Instagram.xm:41: warning: no '-photoImageView' method found
Tweak_Instagram.xm:41: warning: (Messages without a matching method signature
Tweak_Instagram.xm:41: warning: will be assumed to return 'id' and accept
Tweak_Instagram.xm:41: warning: '...' as arguments.)
Tweak_Instagram.xm:41: warning: no '-photoImageView' method found
make[2]: *** [obj/Tweak_Instagram.xm.o] Error 1
make[1]: *** [internal-library-all_] Error 2
make: *** [monitor.all.tweak.variables] Error 2
sammatoMacBook-Pro:monitor sam$
```

图 9-8　编译时报错，因为无法识别 photoImageView 对象

从图 9-8 中反馈的信息可以看到，photoImageView 对象无法识别，从而导致编译失败。

玩过编译的读者都明白，这是当前编译环境无法识别出 IGFeedItemPhotoCell、IGImageProgressView、IGImageView 这些类文件而导致的错误。虽然我们可以对 IGFeedItem-PhotoCell 做 hook 操作，但是并不等于我们可以直接访问它里面的所有对象，更不用说子对象了。

当然，也可以通过强制转换等方式获取对象，不过这些简单粗暴的方式不为笔者所好。那么就另寻一条路：把 IGFeedItemPhotoCell、IGImageProgressView、IGImageView 对应的 .h 文件导入到当前编译环境中。不过，根据前面这 3 个 .h 文件的内容也可以看到，每个 .h 文件中都引用了很多其他类和对象，如果直接导入，编译错误反而会更多。在这种情况下，就需要修改 .h 文件来瘦身了。

9.3.2 对 .h 文件进行瘦身

原则上来说，.h 文件只是给出了一个函数、变量的范围定义，而我们的目的也只是为了让编译通过，对 .h 文件的修改并不能真正影响到目标 App，所以可以放心试错。

以下代码是瘦身之后的 IGFeedItemPhotoCell.h：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve Nygard.
*/

@class IGImageProgressView;
@class UITableViewCell;

@interface IGFeedItemPhotoCell : UITableViewCell
{
    IGImageProgressView *_photoImageView;
}

@property(readonly, nonatomic) IGImageProgressView *photoImageView; // @
synthesize photoImageView=_photoImageView;

@end
```

是不是非常简单？

接下来再修改 IGImageProgressView.h，这是瘦身之后的 IGImageProgressView.h：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*       class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve
Nygard.
*/
```

```
@class IGImageView;

@interface IGImageProgressView : UIView
{
    IGImageView *_photoImageView;
}

@property(readonly, nonatomic) IGImageView *photoImageView; // @synthesize
    photoImageView=_photoImageView;

@end
```

瘦身之后的 IGImageView.h 就更简单了，只是标明它是一个 UIImageView 的继承类而已：

```
/*
*     Generated by class-dump 3.3.3 (64 bit).
*
*     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2010 by Steve
    Nygard.
*/

@class UIImageView;

@interface IGImageView : UIImageView

@end
```

把这 3 个 .h 文件放在一个 Instagram 子目录中，然后在 Tweak_Instagram.xm 中加上对应的 import 语句：

```
#import "Instagram/IGFeedItemPhotoCell.h"
#import "Instagram/IGImageProgressView.h"
#import "Instagram/IGImageView.h"
```

以下是完整的 Tweak_Instagram.xm 代码：

```
#import "DB/Trace.h"
#import "DB/MonitorDB+Public.h"
#import "RevealUtil.h"
#import "Instagram/IGFeedItemPhotoCell.h"
#import "Instagram/IGImageProgressView.h"
#import "Instagram/IGImageView.h"

%hook AppDelegate

- (BOOL)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2 {
    NSInteger level = [MonitorDB getLogLevel];
    [MonitorDB setLevel:2];
    [Trace trace:[NSString stringWithFormat:@ "---------- Start Instagram -----
    ----- %@, %d", self, level] level:4];
```

```
    %orig;

    RevealUtil *ru = [[RevealUtil alloc] init];
    [ru startReveal];

    return YES;
}

%end
```

```
%hook IGFeedItemPhotoCell

- (void)onTap:(id)arg1
{
    UIImage *image = [[[self photoImageView] photoImageView] image];
    UIImageWriteToSavedPhotosAlbum(image, nil, nil, nil);
}

%end
```

9.3.3 编译测试

编译成功之后安装测试一下，在图片上点击之后再去相册里查看，是不是多了一张图？再点一下，又多一张，再点一下……

虽然从功能的角度看已经达到目标，不过这也太糙了，好歹我们也是号称追求极致体验的 iOS App 程序员。

根据笔者的理解，这种图片保存功能并不是一个常用的功能，所以应该需要使用者通过某个动作来触发，最好能有明确的提示来完成这个动作。

9.4 弹出提示菜单

按照前面提出的要求，长按图片之后弹出“Save”菜单就是一个看起来比较不错的选择，效果如图 9-9 所示。

点击“Save”之后，再触发保存的动作，这样看起来就自然多了。



图 9-9 长按图片，弹出“Save”菜单

9.4.1 弹出菜单代码

长按图片弹出菜单，以及调整后的保存图片的代码如下所示：

```
%hook IGFeedItemPhotoCell

- (id)initWithReuseIdentifier:(id)arg1
{
```

```
    id cell = %orig;
    {
        UILongPressGestureRecognizer *gr = [[UILongPressGestureRecognizer alloc]
    initWithTarget:self action:@selector(longPress:)];
        [self addGestureRecognizer:gr];
        [gr release];
    }
    return cell;
}

- (void)longPress:(UILongPressGestureRecognizer *)gestureRecognizer
{
    if ([gestureRecognizer state] == UIGestureRecognizerStateBegan) {
        [[gestureRecognizer view] becomeFirstResponder];
        CGPoint location = [gestureRecognizer locationInView:[gestureRecognizer
    view]];
        UIMenuController *menuController = [UIMenuController
    sharedMenuController];
        UIMenuItem *delMenuItem = [[UIMenuItem alloc] initWithTitle:@ "Save"
    action:
@selector(clickSave:)];
        [menuController setMenuItems:[NSArray arrayWithObject:delMenuItem]];
        [menuController setTargetRect:CGRectMake(location.x, location.y-20,
    0.0f, 0.0f) inView:[gestureRecognizer view]];
        [menuController setMenuVisible:YES animated:YES];
        [delMenuItem release];
    }
}

- (void)clickSave:(id)sender
{
    if ([NSStringFromClass([self class])
    isEqualToString:@"IGFeedItemPhotoCell"] == YES) {
        IGFeedItemPhotoCell *cell = (IGFeedItemPhotoCell *)self;
        UIImage *image = [[[cell photoImageView] photoImageView] image];
        UIImageWriteToSavedPhotosAlbum(image, nil, nil, nil);
    }
}

- (BOOL)canPerformAction:(SEL)selector withSender:(id)sender
{
    if (selector == @selector(clickSave:)) {
        return YES;
    }
    return NO;
}

- (BOOL)canBecomeFirstResponder
{
    return YES;
}

%end
```

从上述代码中可以看到，我们在 IGFeedItemPhotoCell 的 initWithReuseIdentifier 函数中添加了长按手势事件，事件会触发 longPress，再由点击“ Save”的动作来保存图片。而 canPerformAction 和 canBecomeFirstResponder 这两个函数只是用来辅助弹出菜单的。

但是，如果这么编译并安装到真机上运行，肯定是会崩溃的，连弹出菜单的影子都见不着。因为除了 initWithReuseIdentifier 是原本就存在的函数，其他的 longPress、clickSave、canPerformAction、canBecomeFirstResponder 这 4 个都是我们根据需要新增加的，而这些函数符号 (symbol) 并不存在于 App 对应的二进制文件中，一旦调用就会产生崩溃。

但是，肯定是有方法解决这个问题的，而且还不止一种。

9.4.2 使用 Theos 新增函数

Theos 是一个很方便也很强大的工具，除了我们常用的 %orig 功能之外，还可以在函数前面加上 %new 符号来定义新函数，也就是说，可以在 longPress、clickSave、canPerformAction、canBecomeFirstResponder 这 4 个函数前都加上 %new 来标明它们是新增函数，Tweak_Instagram.xm 对应的代码如下所示：

```
%hook IGFeedItemPhotoCell

- (id)initWithReuseIdentifier:(id)arg1
{
    id cell = %orig;
    {
        UILongPressGestureRecognizer *gr = [[UILongPressGestureRecognizer
    alloc] initWithTarget:self action:@selector(longPress:)];
        [self addGestureRecognizer:gr];
        [gr release];
    }
    return cell;
}

%new
- (void)longPress:(UILongPressGestureRecognizer *)gestureRecognizer
{
    if ([gestureRecognizer state] == UIGestureRecognizerStateBegan) {
        [[gestureRecognizer view] becomeFirstResponder];
        CGPoint location = [gestureRecognizer locationInView:[gestureRecognizer
view]];
        UIMenuController *menuController = [UIMenuController
    sharedMenuController];
        UIMenuItem *delMenuItem = [[UIMenuItem alloc] initWithTitle:@ "Save"
    action:
@selector(clickSave:)];
        [menuController setMenuItems:[NSArray arrayWithObject:delMenuItem]];
        [menuController setTargetRect:CGRectMake(location.x, location.y-20,
    0.0f, 0.0f) inView:[gestureRecognizer view]];
        [menuController setMenuVisible:YES animated:YES];
```

```
        [delMenuItem release];
    }
}

%new
- (void)clickSave:(id)sender
{
    if ([NSStringFromClass([self class])
    isEqualToString:@"IGFeedItemPhotoCell"] == YES) {
        IGFeedItemPhotoCell *cell = (IGFeedItemPhotoCell *)self;
        UIImage *image = [[[cell photoImageView] photoImageView] image];
        UIImageWriteToSavedPhotosAlbum(image, nil, nil, nil);
    }
}

%new
- (BOOL)canPerformAction:(SEL)selector withSender:(id)sender
{
    if (selector == @selector(clickSave:)) {
        return YES;
    }
    return NO;
}

%new
- (BOOL)canBecomeFirstResponder
{
    return YES;
}

%end
```

这次再重新编译安装，就可以看到“Save”菜单如约出现了，而且可以正常保存图片，保存图片的功能至此可告一段落。

不过，对于一部分对代码有“洁癖”或者对架构调整有“强迫症”的读者，这是否有一点杂乱的感觉呢？明明只 hook 了一个 initWithReuseIdentifier，却不得不加上一系列函数，而这些新函数本来不属于 hook 这个范畴的。也许可以把新增的代码放到一个专门的类中，而让 IGFeedItemPhotoCell 的 hook 代码保持精致。如果不用 %new 来标注新函数，还可以用什么来做呢？

9.4.3　使用 runtime 新增函数

有成熟经验的 iOS 平台开发人员会对 runtime 有或多或少的了解，稍微深入一点就知道可以给某个类添加函数，也可以覆盖函数。iOS runtime 强大到了几乎能篡改一切，而我们一直强调 Theos 如何强大、如何方便，其实 Theos 也不过是包装好的一个 runtime 工具而已。

回到传统的 Objective-C 代码中，如果想要添加一个新函数，可以使用 class_addMethod

来完成。比如上面提到的 longPress 函数，就可以通过如下代码来注入到目标类中：

```
SEL s = @selector(longPress:);
Method m = class_getInstanceMethod([self class], s);
class_addMethod([target class], s, method_getImplementation(m),
method_getTypeEncoding(m));
```

既然这些辅助性的代码可以用 class_addMethod 的方式注入到 IGFeedItemPhotoCell 中，那么留在 Tweak_Instagram.xm 中的 hook 代码就可以非常简洁，如下所示：

```
%hook IGFeedItemPhotoCell

- (id)initWithReuseIdentifier:(id)arg1
{
    id cell = %orig;
    {
        [PhotoCellLongGesture addMethod:cell];
        //
        UILongPressGestureRecognizer *gr = [[UILongPressGestureRecognizer
alloc] initWithTarget:self action:@selector(longPress:)];
        [self addGestureRecognizer:gr];
        [gr release];
    }
    return cell;
}

%end
```

从上面的代码可以看到，UILongPressGestureRecognizer 依旧是指向了 self 里的 longPress。而 longPress 以及对应的辅助代码都移到了 PhotoCellLongGesture 中，并通过 addMethod 注入到了当前对象中。

PhotoCellLongGesture 的代码如下所示：

```
#import "PhotoCellLongGesture.h"
#import "IGFeedItemPhotoCell.h"
#import "IGImageProgressView.h"
#import "IGImageView.h"
#import <objc/runtime.h>

@implementation PhotoCellLongGesture

+ (void)addMethod:(id)target
{
    Class c = [self class];
    {
        SEL s = @selector(longPress:);
        Method m = class_getInstanceMethod(c, s);
        class_addMethod([target class], s, method_getImplementation(m),
    method_getTypeEncoding(m));
    }
```

```
{
    SEL s = @selector(clickSave:);
    Method m = class_getInstanceMethod(c, s);
    class_addMethod([target class], s, method_getImplementation(m),
method_getTypeEncoding(m));
}
{
    SEL s = @selector(canPerformAction:withSender:);
    Method m = class_getInstanceMethod(c, s);
        class_addMethod([target class], s, method_getImplementation(m),
    method_getTypeEncoding(m));
    }
    {
        SEL s = @selector(canBecomeFirstResponder);
        Method m = class_getInstanceMethod(c, s);
        class_addMethod([target class], s, method_getImplementation(m),
    method_getTypeEncoding(m));
    }
}

- (void)longPress:(UILongPressGestureRecognizer *)gestureRecognizer
{
    if ([gestureRecognizer state] == UIGestureRecognizerStateBegan) {
        [[gestureRecognizer view] becomeFirstResponder];
        //
        CGPoint location = [gestureRecognizer
    locationInView:[gestureRecognizer view]];
        UIMenuController *menuController = [UIMenuController
    sharedMenuController];
        UIMenuItem *delMenuItem = [[UIMenuItem alloc] initWithTitle:@ "Save"
    action:@selector(clickSave:)];
        [menuController setMenuItems:[NSArray arrayWithObject:delMenuItem]];
        [menuController setTargetRect:CGRectMake(location.x, location.y-20,
    0.0f, 0.0f) inView:[gestureRecognizer view]];
        [menuController setMenuVisible:YES animated:YES];
        [delMenuItem release];
    }
}

- (void)clickSave:(id)sender
{
    if ([NSStringFromClass([self class])
    isEqualToString:@"IGFeedItemPhotoCell"] == YES) {
        IGFeedItemPhotoCell *cell = (IGFeedItemPhotoCell *)self;
        UIImage *image = [[[cell photoImageView] photoImageView] image];
        UIImageWriteToSavedPhotosAlbum(image, nil, nil, nil);
    }
}

- (BOOL)canPerformAction:(SEL)selector withSender:(id)sender
{
```

```
    if (selector == @selector(clickSave:)) {
        return YES;
    }
    return NO;
}

- (BOOL)canBecomeFirstResponder
{
    return YES;
}

@end
```

整个 PhotoCellLongGesture 的写法都是完全遵照标准 Objective-C 风格来写的，与 Theos 的链接代码界限分隔得非常清晰。

其实不管是哪种写法，都不复杂，愿意怎么用就全凭读者自己的爱好了。

因为新增了 PhotoCellLongGesture 文件，所以 Makefile 也需要相应地加上 PhotoCellLong Gesture.m，相关内容如下所示：

```
include theos/makefiles/common.mk

TWEAK_NAME = monitor
monitor_FILES = Tweak_Instagram.xm RevealUtil.m DB/NameInfo.m DB/Trace.m DB/MonitorDB.m
DB/DBM.m DB/MonitorDB+Trace.m DB/MonitorDB+Setting.m
DB/MonitorDB+WhatsApp.m Instagram/PhotoCellLongGesture.m
monitor_FRAMEWORKS = UIKit CFNetwork
monitor_PRIVATE_FRAMEWORKS = AppSupport
monitor_LDFLAGS = -lsqlite3.0

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 Instagram"
```

9.5 代码结果整理

本章前 4 节一步步讲解了如何通过 Reveal 定位 IGFeedItemPhotoCell 以及 IGImageView 对象，并且通过添加菜单的方式完成了保存图片的动作，甚至还就注入类函数的两种方式进行了讨论。

本节对以上这些关键信息做一个梳理，以期能帮助读者更好地理解本章的内容。

1）通过 Reveal 注入 Instagram，查看到与保存图片关联的类文件是 IGFeedItemPhoto Cell、IGImageProgressView、IGImageView。

2）编写 tweak 代码，通过覆盖 IGFeedItemPhotoCell 的 onTap 来保存图片。

3）编译失败，需要导入 IGFeedItemPhotoCell、IGImageProgressView、IGImageView 三个类的 .h 文件，并且对这 3 个文件进行瘦身，成功编译。

4）放弃 onTap 函数，改用长按图片弹出菜单的方式，不过运行会失败，因为新增函数无法注入。

5）使用 Theos 的 %new 符号来标注新增函数，测试成功。

6）重构，使用 runtime 的 class_addMethod 将新增函数注入，测试成功。

在图 9-10 中，Tweak_Instagram.xm 仅仅负责注入 IGFeedItemPhotoCell，而实际的运行代码都移到了 Instagram 子目录中。而本 tweak 插件安装之后，就给 Instagram 这个 App 新增了一个保存图片的功能。

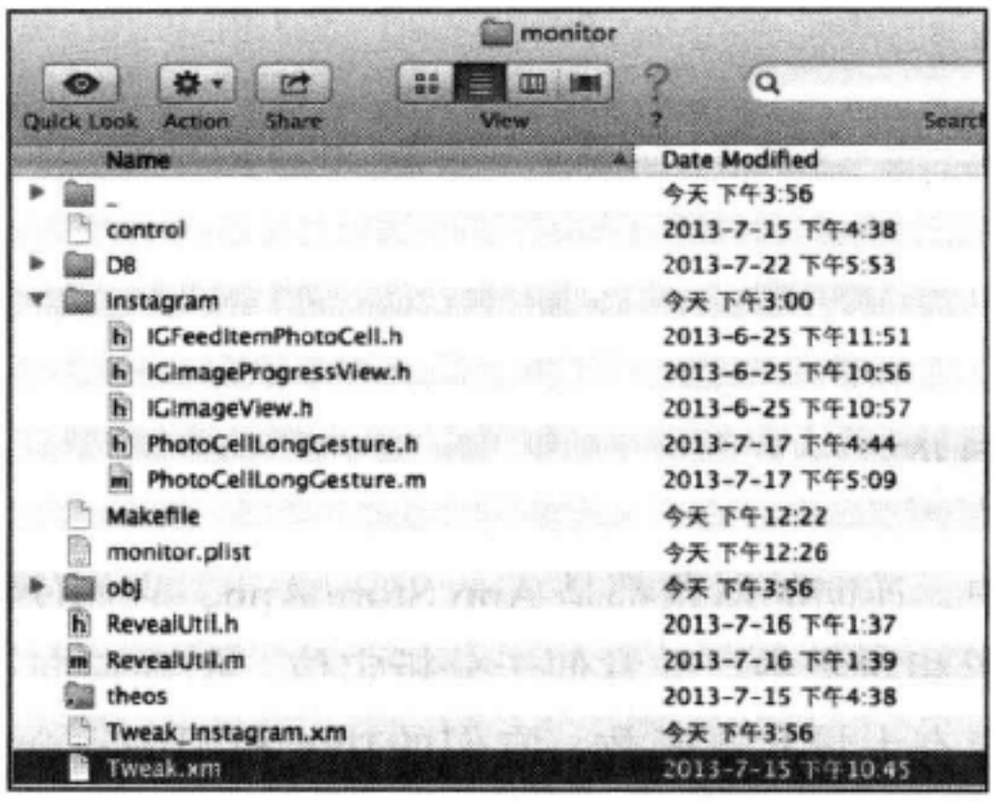


图 9-10　定制 Instagram 的 tweak 目录文件

9.6　小结

本章以 Instagram 为目标制作了一个小插件，安装之后能够在浏览那些精美图片的同时保存图片。

因为使用了 Reveal，所以定位这些 UI 层的元素非常快，也非常精确，因此也能够很快地达到保存图片的目标。

达到基础目标之后，又对功能做了一点扩展，增加长按图片弹出菜单的功能。由于需要添加一些原本不属于目标 App 中已有的新函数，文中采用了 Theos 中的 %new 以及 runtime 中的 class_addMethod 来注入，最终达成目标。

Instagram 是一个优美的 App，值得借鉴学习的地方很多，建议有兴趣的读者可花时间深入地探索，相信能得到的远不止界面上展现的这些内容。

第 10 章

实战 4：iOS 电话操作

10.1 常用电话操作

在前面的 3 个实战中，逆向的对象都是 App Store App，它们使用的均为公开函数，程序层次纵深较浅，文件位置相对集中，非常便于分析定位。但在逆向 iOS 系统功能时，情况往往没有这么简单：iOS 含有大量私有函数，它们的功能和用法完全未知；iOS 的某一项功能往往由多个部件组合而成，且这些部件一般散落在系统的各个角落，大大增加了逆向难度。

毫无疑问，越狱是每个 iOS 逆向工程爱好者的终极目标，但作为初学者的我们，要到达目的地还有很长一段路要走。相较于越狱来说，iOS 自带 App 的逆向虽然要简单得多，但能大大加深我们对 iOS 架构的了解，对我们朝着越狱努力有积极的推动作用；同时，系统 App 比 App Store App 的逆向难度要大，适合作为进阶练习。

在信息流通异常发达的今天，无数人通过手机谈论工作、联络感情，手机已经成为了人们日常生活的必备品；但是，随之出现的副作用——垃圾电话的泛滥让我们不堪其扰。相信不少读者都希望自己动手解决这个问题，而这同样也是笔者编写 SMSNinja 的初衷；所以，本章就以 iOS 的电话操作为目标来进行分析，分析的内容主要针对 5 种电话操作：拨打电话、接听电话、挂断电话、拦截电话以及删除通话记录。

10.2 分析源代码结构，并定位电话操作的代码

10.2.1 class-dump 获取 .h 文件

iOS 的电话程序位于 /Applications/MobilePhone.app，而且由于是系统程序，不需手动脱壳。打开 MobilePhone.app 即找到可执行文件，如图 10-1 所示。

对 class-dump 进行操作，获取 .h 文件：

```
class-dump-z -H -I MobilePhone -o /Path/To/MobilePhone/Headers/
```

VMGreetingRecord@2x~iphone.png	Jul 24, 2013 10:02 AM	1 KB	Portable Network Graphics Image
VMGreetingStop@2x~iphone.png	Jul 24, 2013 10:02 AM	203 bytes	Portable Network Graphics Image
VMLevelBkgnd@2x~iphone.png	Jul 24, 2013 10:02 AM	953 bytes	Portable Network Graphics Image
VMLevelFeedback@2x~iphone.png	Jul 24, 2013 10:02 AM	299 bytes	Portable Network Graphics Image
VMOutOfOrderImage@2x~iphone.png	Jul 24, 2013 10:02 AM	8 KB	Portable Network Graphics Image
VMPause@2x~iphone.png	Jul 24, 2013 10:02 AM	1 KB	Portable Network Graphics Image
VMPlay@2x~iphone.png	Jul 24, 2013 10:02 AM	2 KB	Portable Network Graphics Image
VMSetupButton@2x~iphone.png	Jul 24, 2013 10:02 AM	2 KB	Portable Network Graphics Image
VMSetupButtonPressed@2x~iphone.png	Jul 24, 2013 10:02 AM	2 KB	Portable Network Graphics Image
VMSetupButtonReflection@2x~iphone.png	Jul 24, 2013 10:02 AM	1 KB	Portable Network Graphics Image
VMSetupButtonReflectionPressed@2x~iphone.png	Jul 24, 2013 10:02 AM	1 KB	Portable Network Graphics Image
Info.plist	Jul 24, 2013 10:02 AM	3 KB	Property List
ResourceRules.plist	Jul 24, 2013 10:02 AM	150 bytes	Property List
EmergencyCallStrings.strings	Jul 24, 2013 10:02 AM	276 bytes	Strings File
MobilePhone	Jul 24, 2013 10:02 AM	667 KB	Unix Executable File
PkgInfo	Jul 24, 2013 10:02 AM	8 bytes	Unix Executable File

图 10-1　MobilePhone.app 的内容

可以看到对 MobilePhone 的 class-dump 操作只获得了 94 个 .h 文件，如图 10-2 所示。iPhone 最核心的功能之一竟然这么小，有点难以置信吧？

DialerController.h	Feb 15, 2013 10:06 PM	4 KB	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerLCDFieldDelegate.h	Feb 15, 2013 10:06 PM	352 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerLCDNameAndLabelView.h	Feb 15, 2013 10:06 PM	689 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerLCDView.h	Feb 15, 2013 10:06 PM	2 KB	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerPhonePad.h	Feb 15, 2013 10:06 PM	504 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerView.h	Feb 15, 2013 10:06 PM	1 KB	C Hea...ce File	Feb 15, 2013 10:06 PM
DialerViewDelegate.h	Feb 15, 2013 10:06 PM	338 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
EndCallExtrudeAnimation.h	Feb 15, 2013 10:06 PM	473 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
FavoritesViewController.h	Feb 15, 2013 10:06 PM	5 KB	C Hea...ce File	Feb 15, 2013 10:06 PM
InCallBottomButton.h	Feb 15, 2013 10:06 PM	400 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
InCallController.h	Feb 15, 2013 10:06 PM	15 KB	C Hea...ce File	Feb 15, 2013 10:06 PM
InCallLCDField.h	Feb 15, 2013 10:06 PM	979 bytes	C Hea...ce File	Feb 15, 2013 10:06 PM
InCallLCDView.h	Feb 15, 2013 10:06 PM	3 KB	C Hea...ce File	Feb 15, 2013 10:06 PM

Users › snakeninny › Code › IOSPrivateHeaders › 6.0.1 › MobilePhone › PhoneViewController.h

图 10-2　MobilePhone 的头文件

10.2.2　寻找拨打电话的函数

拨打电话的操作特征非常明显，在 MobilePhone 的 Keypad 界面中，绿色的 Call 按钮非常抢眼，如图 10-3 所示。



图 10-3　拨打电话界面

输入一个号码，按下这个按钮就会调用“拨打电话”的函数，这种逻辑是非常合乎情理的，我们用 Reveal 来看看这个按钮对应的代码是什么。首先要做的是定制 RevealUtil，使其适用于 MobilePhone，总共有 3 处需要更改。

❑ RevealUtil.plist

目标 App 是什么，就要把这里的 filter 改成它的 bundle identifier。为了偷懒，我们把这里的 filter 留空，如图 10-4 所示。让 MobileSubstrate 尝试对所有文件加载 RevealUtil.dylib；这样看似有些极端，其实不然——RevealUtil.dylib 起到的作用仅仅是在 App 启动时加载 libReveal.dylib，而 App 启动的点在不同 App 中一般是不一样的，即 hook 的类和函数因 App 不同而不同，RevealUtil.dylib 在绝大多数情况下只能成功加载一次，即使 MobileSubstrate 广撒网，实际

RevealUtil.plist › No Selection

Key	Type	Value
▼ Root	Dictionary	(1 item)
▶ Filter	Dictionary	(0 items)

图 10-4　RevealUtil.plist 的内容

上钩的也只有一条鱼。如果你还是不放心，那就在 filter 中填入 bundle identifier，这样是最严谨的。

❑ Tweak.xm

在本文件中，我们 hook 了 App 的启动点，一般情况下，它在不同 App 中是不同的。对于 MobilePhone，可以在 class-dump 出的 94 个 .h 文件中按内容搜索 applicationDidFinish-Launching 关键词，很容易就可以在 PhoneApplication.h 中定位到目标，如图 10-5 和图 10-6 所示。

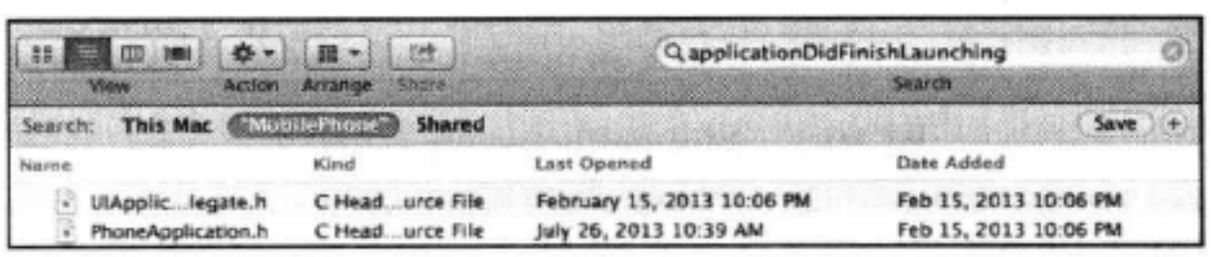


图 10-5 搜索 MobilePhone 头文件

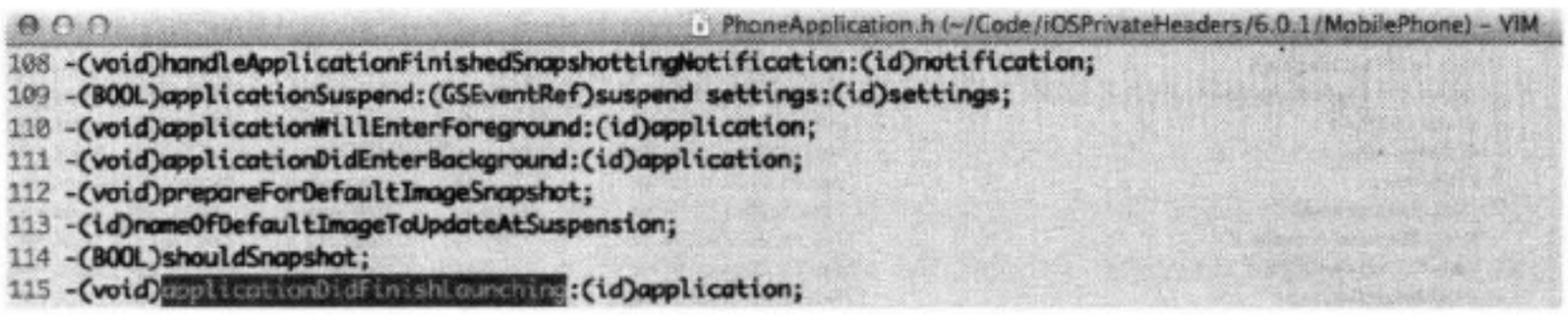


图 10-6 在 PhoneApplication.h 中定位目标

Tweak.xm 的内容随即更改为：

```
#import "RevealUtil.h"

%hook PhoneApplication
-(void)applicationDidFinishLaunching:(id)application
{

    %orig;

    RevealUtil *ru = [[RevealUtil alloc] init];
    [ru startReveal];
}
%end
```

❑ RevealUtil.mm

还记得前面是从哪个路径加载 libReveal.dylib 的吗？是 App 的 Documents 目录。对于 App Store App 来 说，NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) 函数的返回值一般是这个 App 目录下的 Documents 目录，而对于 System App 和 Cydia App 来说，它们的 App 目录下是没有 Documents 目录的，那这个函数的返回值是什么？可以通过如下试验代码来获取路径信息：

```
%hook PhoneApplication
-(void)applicationDidFinishLaunching:(id)application
{
```

```
    %orig;

    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
    NSLog(@"iOSRE: %@", paths);
}
%end
```

syslog 输出结果如图 10-7 所示。

```
64069 Dec  2 23:42:05 [redacted]-FunMaker-5 MobilePhone[631]: iOSRE: (
64070             "/var/mobile/Documents"
64071         )
```

图 10-7　syslog 片段

这就说明，不同类型 App 的 Documents 路径是不一样的，那么 Preference Bundle 的 Documents 在哪里？ WeeApp 的呢？要确定它们的 Documents 路径都得做实验，很麻烦。好在所有的 App 都是 bundle 形式的，bundle 的结构又都是相同的，为了方便使用，何不把 libReveal.dylib 放在每个 App 的 bundle 里呢？这样的话，RevealUtil.mm 的内容就可以统一成如下内容了：

```
#import "RevealUtil.h"
#import <dlfcn.h>

@implementation RevealUtil
- (void)startReveal
{
    NSString *revealLibName = @"libReveal.dylib";
    NSString *bundlePath = [[NSBundle mainBundle] bundlePath];
    NSString *dyLibPath = [NSString stringWithFormat:@"%@/%@", bundlePath,
    revealLibName];

    // 内容不变
}

- (void)stopReveal
{
    // 内容不变
}
@end
```

完成了对 RevealUtil 的定制，以后使用它只需要更改 Tweak.xm 这一个文件，减少了工作量，节省了时间，何乐而不为？

好了，安装 RevealUtil 之后把 libReveal.dylib 放到 /Applications/MobilePhone.app 下，然后打开 Reveal，定位到 Call 按钮，如图 10-8 所示。

Reveal 状态栏下方的层次链清晰地指出，Call 这个 UIButton 是呈现在 DialerView 上的。因此，再到 DialerView.h 里找到这个 UIButton，如图 10-9 所示。

Call 对应的应该是 callButton，轻松理解，没问题吧？

既然已经定位到了 Call 对应的 UIButton，要怎么样找出按下它之后调用的函数呢？

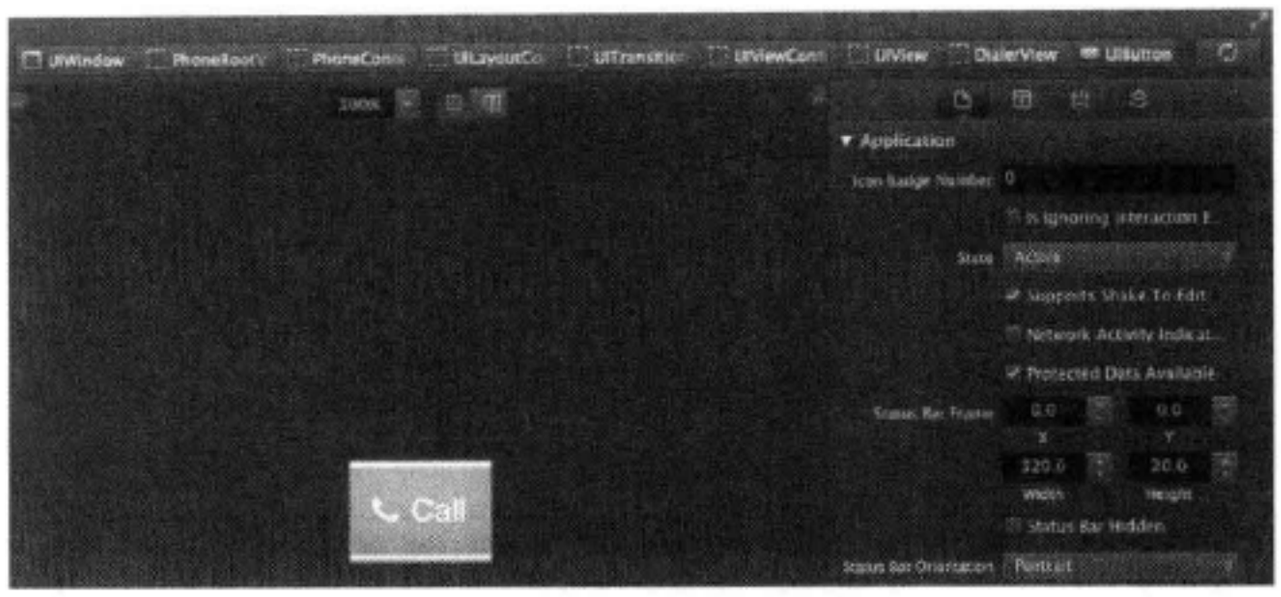


图 10-8 在 Reveal 中定位目标

```
12 @class DialerLCDView, UIButton, DialerPhonePad;
13 @protocol DialerViewDelegate;
14
15 __attribute__((visibility("hidden")))
16 @interface DialerView : XXUnknownSuperclass <DialerLCDFieldDelegate> {
17         id<DialerViewDelegate> _delegate;
18         UIButton* _deleteButton;
19         BOOL _inCallMode;
20         UIButton* _callButton;
21         UIButton* _addContactButton;
22         DialerPhonePad* _phonePadView;
23         DialerLCDView* _lcdView;
24 }
25 @property(retain, nonatomic) UIButton* deleteButton;
26 @property(retain, nonatomic) UIButton* callButton;
```

图 10-9 在 DialerView.h 里定位目标

在笔者见过的所有情况中，给 UIButton 增加响应操作的函数全是 [UIControl addTarget:action:forControlEvents:]，这里的 action 即是相应操作的方法，target 是对应的对象，所以找到了 action，也就找到了目标函数。因为 UIButton 继承自 UIControl，而 UIControl 已经提供了一个寻找 action 的方法 [UIControl actionsForTarget:forControlEvent:]，所以可以通过遍历 [UIControl allTargets]，以其中的元素为第一个参数，以 [UIControl allControlEvents] 为第二个参数，得到这个 UIButton 相应的所有 action，代码如下所示：

```
@interface DialerView : UIView
@property(retain, nonatomic) UIButton* callButton;
@end

%hook DialerView
-(void)layoutSubviews
{

    %orig;

    UIButton *callButton = [self callButton];
    for (id target in [callButton allTargets])
        NSLog(@"iOSRE: target = %@, actions = %@", target, [callButton
    actionsForTarget:target forControlEvent:[callButton allControlEvents]]);
```

```
}
%end
```

syslog 的输出如图 10-10 所示。

```
64324 Dec  2 23:42:41 [redacted]-FunMaker-5 MobilePhone[631]: iOSRE:
      target = <DialerController: 0x1d0c0850>, actions = (
64325             "_callButtonPressed:"
64326         )
```

图 10-10　syslog 片段

打印的信息直切要害，没什么可多解释的。第一阶段任务完成，结论是，在 MobilePhone 中，拨打电话的函数是 [DialerController _callButtonPressed:]。

掌握了这个信息，接下来可以做的事有：hook 这个函数，让 iPhone 打不出去电话。这种无聊的事在愚人节那天整整你的朋友还可以，在平常基本毫无意义——前面已经说过相似的例子，DialerController 是 MobilePhone 里的类，在自己的程序里无法使用，而我们更想知道的是如何自己写程序拨打电话。那么想达到这个目的，就要深入看看 [DialerController _callButtonPressed:] 的底层实现是什么了。

把 MobilePhone 拖进 IDA，定位到 _callButtonPressed:；此时由 class-dump 转移到了 IDA，这也已经说明我们的分析更进一步了，那就利用前面介绍的知识，开始分析代码吧。

首先面对的是第一个块，如图 10-11 所示。

```
; DialerController - (void)_callButtonPressed:(id)
; Attributes: bp-based frame

; void __cdecl -[DialerController _callButtonPressed:](struct DialerController *self, SEL, id)
__DialerController__callButtonPressed__

var_14= -0x14
var_10= -0x10

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
STR.W           R8, [SP,#0xC+var_10]!
SUB             SP, SP, #4
MOV             R4, R0
MOV             R0, #(_OBJC_IVAR_$_DialerController._dialerView - 0xF30C) ; DialerView *_dialerView;
MOVW            R1, #(:lower16:(selRef_lcdView - 0xF314))
ADD             R0, PC  ; DialerView *_dialerView;
MOVT.W          R1, #(:upper16:(selRef_lcdView - 0xF314))
LDR             R0, [R0] ; DialerView *_dialerView;
ADD             R1, PC ; selRef_lcdView
LDR             R1, [R1] ; "lcdView"
LDR             R0, [R4,R0]
BLX             _objc_msgSend
MOV             R1, #(selRef_text - 0xF326) ; selRef_text
ADD             R1, PC ; selRef_text
LDR             R1, [R1] ; "text"
BLX             _objc_msgSend
MOV             R5, R0
MOV             R0, #(selRef_length - 0xF338) ; selRef_length
ADD             R0, PC ; selRef_length
LDR             R1, [R0] ; "length"
MOV             R0, R5
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(_UIApp_ptr - 0xF350))
MOV             R6, R0
MOVT.W          R1, #(:upper16:(_UIApp_ptr - 0xF350))
MOVW            R0, #(:lower16:(selRef_inCall - 0xF35A))
ADD             R1, PC ; _UIApp_ptr
MOVT.W          R0, #(:upper16:(selRef_inCall - 0xF35A))
LDR.W           R8, [R1] ; _UIApp
ADD             R0, PC ; selRef_inCall
LDR             R1, [R0] ; "inCall"
LDR.W           R0, [R8]
BLX             _objc_msgSend
CMP             R6, #0
BEQ             loc_F3E6
```

图 10-11　_callButtonPressed: 汇编片段

简单地扫一眼，没有什么实质性内容。BEQ 的判断条件是 [[[_dialerView lcdView] text] length]，而 _dialerView 就是刚才已经分析过的 Keypad 界面，这个界面唯一有价值的 text 似乎就是拨打的号码，而拨出电话的前提是这个号码的长度不为空，即 [[[_dialerView lcdView] text] length] != 0。所以应该走左边那条路到图 10-12。

```
LDR.W           R8, [R1] ; _UIApp
ADD             R0, PC ; selRef_inCall
LDR             R1, [R0] ; "inCall"
LDR.W           R0, [R8]
BLX             _objc_msgSend
CMP             R6, #0
BEQ             loc_F3E6

TST.W           R0, #0xFF
BNE             loc_F380
```

图 10-12 _callButtonPressed: 汇编片段

BNE 的判断条件是 [UIApp inCall]，从字面意思上猜测，表示当前是否处于通话状态。既然是拨打电话，当然是处于通话状态了，因此 R0 的值是 1，走右边的路，到达图 10-13。

```
loc_F380                ; int _calledNumber:1;
MOV             R0, #(_OBJC_IVAR_$_DialerController._calledNumber - 0xF38C)
ADD             R0, PC  ; int _calledNumber:1;
LDR             R1, [R0] ; int _calledNumber:1;
LDR             R3, [R4,R1]
MOV             R0, #(selRef_stringByRemovingWhitespace - 0xF39E) ; selRef_stringByRemovingWhitespace
MOVW            R2, #(:lower16:(classRef_DialerPhonePad - 0xF3A8))
ADD             R0, PC ; selRef_stringByRemovingWhitespace
MOVT.W          R2, #(:upper16:(classRef_DialerPhonePad - 0xF3A8))
ORR.W           R3, R3, #1
ADD             R2, PC ; classRef_DialerPhonePad
STR             R3, [R4,R1]
LDR             R1, [R0] ; "stringByRemovingWhitespace"
MOV             R0, R5
LDR             R4, [R2] ; _OBJC_CLASS_$_DialerPhonePad
BLX             _objc_msgSend
MOV             R2, R0
MOV             R0, #(selRef_launchFieldTestIfNeeded_ - 0xF3C0) ; selRef_launchFieldTestIfNeeded_
ADD             R0, PC ; selRef_launchFieldTestIfNeeded_
LDR             R1, [R0] ; "launchFieldTestIfNeeded:"
MOV             R0, R4
BLX             _objc_msgSend
TST.W           R0, #0xFF
BNE             loc_F424
```

图 10-13 _callButtonPressed: 汇编片段

这里 BNE 的判断条件是 [DialerPhonePad launchFieldTestIfNeeded:]，这是个什么玩意？在 bing 词典里看看 fieldtest 是什么意思，如图 10-14 所示。

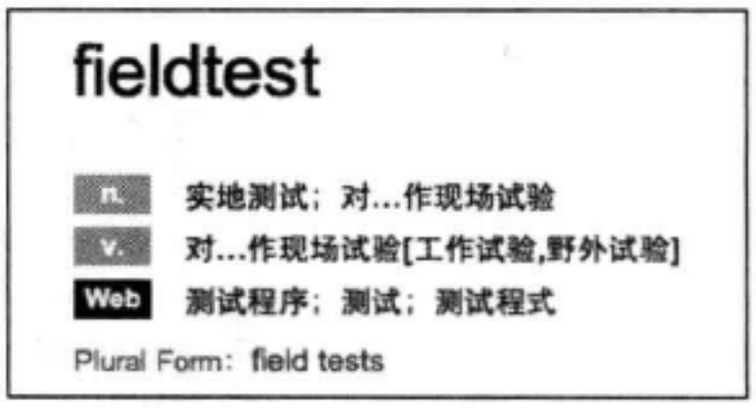


图 10-14 fieldtest 的释义

测试程序？可我们从没见过什么测试程序啊？因此猜测这里的返回值多半是 0，先按这个猜测继续推理，等出现了问题再回过头来思考，现在看图 10-15。

```
MOVW            R0, #(:lower16:(selRef_dialPhoneNumber_dialAssist_ - 0xF3DC))
MOV             R2, R5
MOVT.W          R0, #(:upper16:(selRef_dialPhoneNumber_dialAssist_ - 0xF3DC))
MOVS            R3, #0
ADD             R0, PC ; selRef_dialPhoneNumber_dialAssist_
LDR             R1, [R0] ; "dialPhoneNumber:dialAssist:"
LDR.W           R0, [R8]
BLX             _objc_msgSend
B               loc_F424
```

图 10-15 _callButtonPressed: 汇编片段

这个 [UIApp dialPhoneNumber:dialAssist:] 名字不要太直白哦！因为这个代码块执行完

后 [DialerController _callButtonPressed:] 就结束了，综合前面的分析，拨打电话的操作应该在 [UIApp dialPhoneNumber:dialAssist:] 内部，它的参数是 [[_dialerView lcdView] text] 和 0。定位到 dialPhoneNumber:dialAssist:，我们从图 10-16 开始继续分析。

```
; PhoneApplication - (char)dialPhoneNumber:(id) dialAssist:(char)

; char __cdecl -[PhoneApplication dialPhoneNumber:dialAssist:](struct PhoneApplication *self, SEL, id, char)
__PhoneApplication_dialPhoneNumber_dialAssist__

var_C= -0xC

PUSH            {R7,LR}
MOV             R7, SP
SUB             SP, SP, #4
MOV             R1, #(selRef_dialPhoneNumber_forUID_dialAssist_ - 0xF446) ; selRef_dialPhoneNumber_forUID_dialAssist_
STR             R3, [SP,#0xC+var_C]
MOV.W           R3, #0xFFFFFFFF
ADD             R1, PC ; selRef_dialPhoneNumber_forUID_dialAssist_
LDR             R1, [R1] ; "dialPhoneNumber:forUID:dialAssist:"
BLX             _objc_msgSend
ADD             SP, SP, #4
POP             {R7,PC}
; End of function -[PhoneApplication dialPhoneNumber:dialAssist:]
```

图 10-16　dialPhoneNumber:dialAssist: 汇编片段

在图 10-16 中，没什么实质性内容，仅仅调用了 [UIApp dialPhoneNumber:forUID:dialAssist:]，参数分别是 [[_dialerView lcdView] text]、0xFFFFFFFF 和 [SP]（即 R3，值是 0）。继续看图 10-17。

```
; PhoneApplication - (char)dialPhoneNumber:(id) forUID:(int) dialAssist:(char)
; Attributes: bp-based frame

; char __cdecl -[PhoneApplication dialPhoneNumber:forUID:dialAssist:](struct PhoneApplication *self, SEL, id, int, char)
__PhoneApplication_dialPhoneNumber_forUID_dialAssist__

arg_0=  8

PUSH            {R4,R5,R7,LR}
ADD             R7, SP, #8
SUB             SP, SP, #8
MOVW            R1, #(:lower16:(selRef_dialPhoneNumber_forUID_dialAssist_wasAlreadyAssisted_ - 0xF46A))
LDR             R4, [R7,#arg_0]
MOVS            R5, #0
MOVT.W          R1, #(:upper16:(selRef_dialPhoneNumber_forUID_dialAssist_wasAlreadyAssisted_ - 0xF46A))
ADD             R1, PC ; selRef_dialPhoneNumber_forUID_dialAssist_wasAlreadyAssisted_
LDR             R1, [R1] ; "dialPhoneNumber:forUID:dialAssist:wasAl"...
STRD.W          R4, R5, [SP]
BLX             _objc_msgSend
ADD             SP, SP, #8
POP             {R4,R5,R7,PC}
; End of function -[PhoneApplication dialPhoneNumber:forUID:dialAssist:]
```

图 10-17　dialPhoneNumber:forUID:dialAssist: 汇编片段

可以看到，[UIApp dialPhoneNumber:forUID:dialAssist:] 又继续调用了 [UIApp dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted:]。前两个参数没什么好说的，注意这里的第三和第四个参数，它们均来自栈，那么具体值是多少呢？在“BLX _objc_msgSend”的上面一行，“STRD.W R4, R5, [SP]”把 R4 和 R5 分别入栈，那么显然 R4 和 R5 就是后两个参数，但谁是第三个参数，谁是第四个呢？一起来分析一下。

首先，毫无疑问，R5 的值是 0，那么 R4 的值是多少？R4 来自于“LDR R4, [R7, #arg_0]”，即 R4 = *(R7 + 8)；而 R7 来自于“ADD R7, SP, #8”，即 R7 = SP + 8，因此 R4 = *(SP + 0x10)，即 R4 跟 SP 有关。注意，这里的 SP 有些容易让人混淆，在“BLX _objc_msgSend”之前，SP 的值变化了好几次[⊖]——“PUSH {R4, R5, R7, LR}”将 4 个寄存器入栈，每个寄存器占 4 字节，导致 SP2 = SP1 – 4 * 4 = SP1 – 0x10；其中 SP1 是 [UIApp dialPhoneNumber:forUID:dialAssist:] 传进来的，且存放着 [UIApp dialPhoneNumber:forUID:dialAssist:]

⊖ 方便起见，将指向不同值的 SP 分别记为 SP1、SP2 和 SP3。

的第三个参数；而“ADD R7, SP, #8”时 SP 已经是 SP2 了，所以准确地说，上面的 R4 = *(SP2 + 0x10) = [SP1]，即一路传过来的 dialAssist: 参数，值是 0。那么 R4 就是第三个参数，R5 是第四个参数。至于“SUB SP, SP, #8”，虽然也改变了 SP，相当于 SP3 = SP2 – 8，但 R4 跟 SP3 无关，所以不影响参数的值。如果你还是不太明白，对照这段文字多看几遍就好了，或者 @iOS 应用逆向工程，跟大家一起讨论。

向着图 10-18 前进。

```
; PhoneApplication - (char)dialPhoneNumber:(id) forUID:(int) dialAssist:(char) wasAlreadyAssisted:(char)
; Attributes: bp-based frame

; char __cdecl -[PhoneApplication dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted:](struct PhoneApplication *self, SEL, id, int, char, char)
__PhoneApplication_dialPhoneNumber_forUID_dialAssist_wasAlreadyAssisted__

var_30= -0x30
var_2C= -0x2C
var_28= -0x28
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10
arg_0=  8
arg_4=  0xC

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
STR.W           R8, [SP,#0xC+var_10]!
SUB             SP, SP, #0x20
MOV             R8, R0
MOV             R0, #(selRef_shouldAttemptPhoneCall - 0xF492) ; selRef_shouldAttemptPhoneCall
MOV             R6, R3
ADD             R0, PC ; selRef_shouldAttemptPhoneCall
MOV             R4, R2
LDR             R1, [R0] ; "shouldAttemptPhoneCall"
MOV             R0, R8
BLX             _objc_msgSend
MOVS            R5, #0
TST.W           R0, #0xFF
BEQ.W           loc_F616
```

图 10-18　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

在图 10-18 中看到了什么？ shouldAttemptPhoneCall？根据字面意思，we should，那就看图 10-19。

```
MOV             R0, #(selRef_inCall - 0xF4B0) ; selRef_inCall
ADD             R0, PC ; selRef_inCall
LDR             R1, [R0] ; "inCall"
MOV             R0, R8
BLX             _objc_msgSend
TST.W           R0, #0xFF
BNE             loc_F512
```

图 10-19　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

[UIApp inCall]，刚才就已经分析过了，再看图 10-20。

又是一个“LDR R0, [R7,#arg_4]”，那就按分析图 10-17 的套路再来一遍。R0 = *(R7 + 0xC)，而依图 10-18，R7 = SP2 + 0xC，因此 R0 = *(SP2 + 0x18)。因为“PUSH {R4-R7,LR}”的关系，从 [PhoneApplication dialPhoneNumber:forUID:dialAssist:] 进入 [PhoneApplication dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted:] 的时候，SP2 = SP1 – 4 * 5 = SP1 – 0x14，即 R0 = *(SP1 + 0x4)，其中 SP1 是 [PhoneApplication dialPhoneNumber:forUID:dialAssist:] 传进来的；因为 [PhoneApplication dialPhoneNumber:forUID:dialAssist:] 中“STRD.W R4, R5, [SP]”的关系，R4 和 R5 先后入栈，这条指令执行完成后 SP 的值才变成 SP1，所以此时的 [SP1] 是 R5，[SP1 + 4] 是 R4，即 R0 = R4，是 dialAssist: 的参数。它的值是 0，继续看图 10-21。

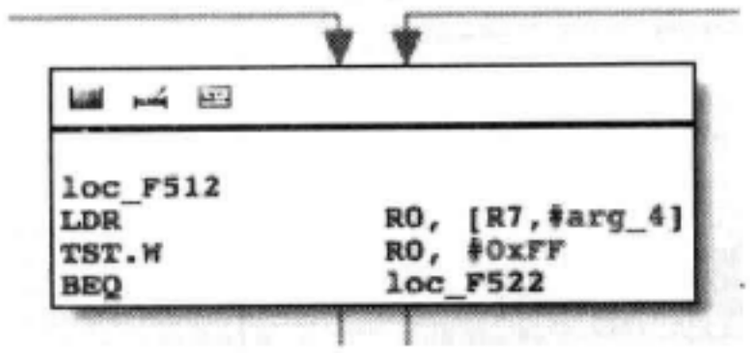


图 10-20　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

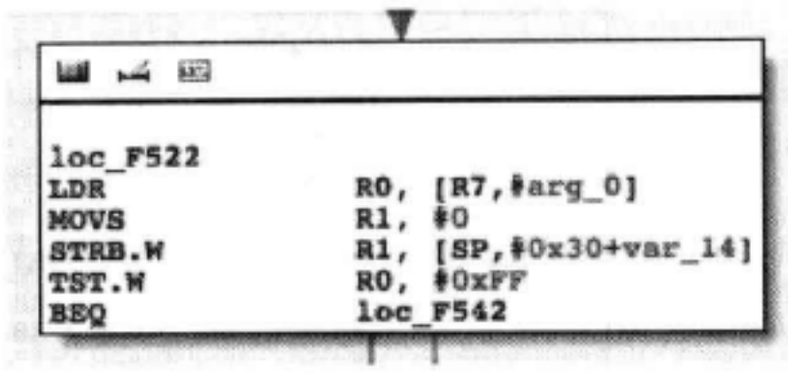


图 10-21　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

还是“LDR R0, [R7,#arg_4]”。R0 = *(R7 + 8)，R7 = SP2 + 0xC，因此 R0 = *(SP2 + 0x14)。SP2 = SP1 – 0x14，即 R0 = *(SP1) = R5，即为 wasAlreadyAssisted: 的参数。它的值还是 0，继续看图 10-22。

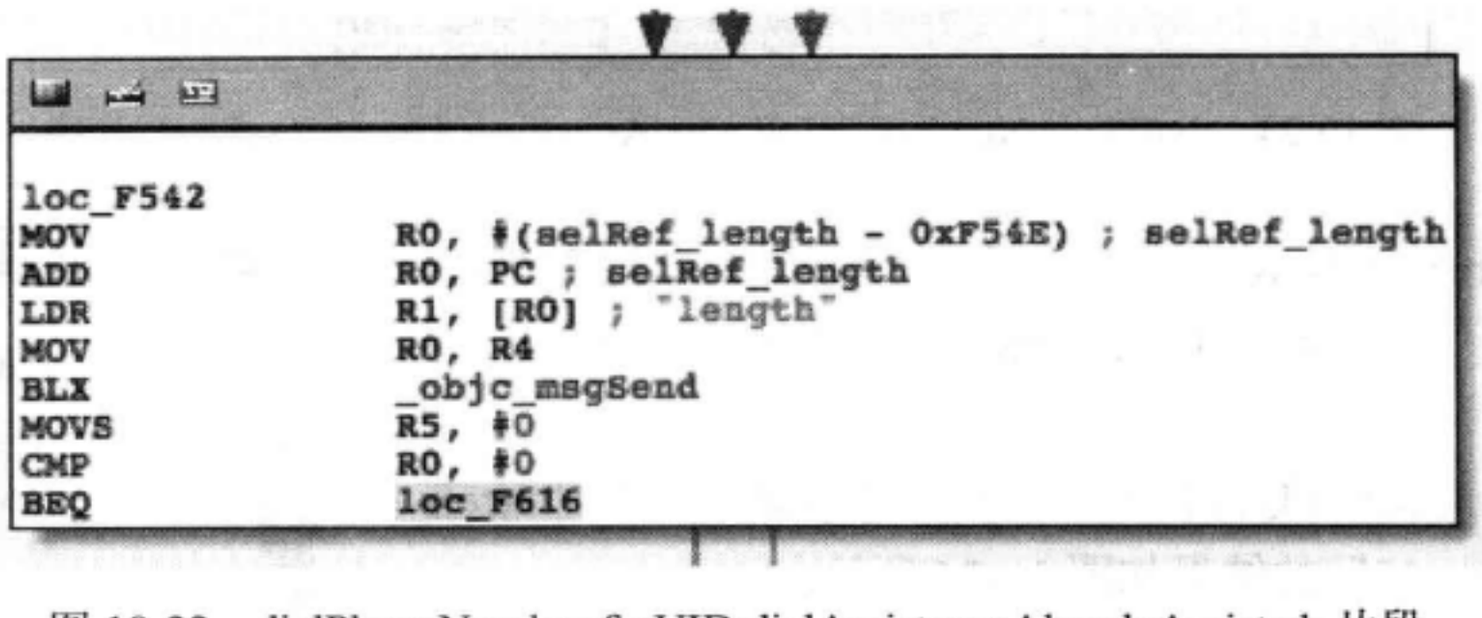


图 10-22　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

图 10-22 中的这一段前面已经分析过了，它用于判断电话号码的长度是不是 0。前面已经输入了一串电话号码，所以看图 10-23。

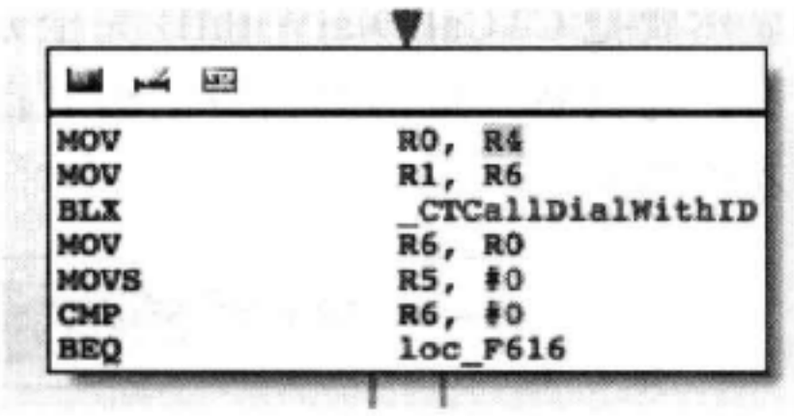


图 10-23　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

这个函数的作用很明显就是“拨号”，其前缀是 CT，即 CoreTelephony 的缩写，而 CoreTelephony 正是电话操作的核心类，看到这里，你是不是觉得我们快成功了呢？下面先分析一下它的参数：R0 是 dialPhoneNumber: 的参数，即电话号码；R1 是 forUID: 的参数，即 UID，其值是 0xFFFFFFFF，也就是 -1；R2 呢？按照刚才分析的内容来看，R2 也是 dialPhoneNumber: 的参数，跟 R0 重复了，所以可以肯定 CTCallDialWithID 只有两个参数。另外，这个函数应该还有返回值：如果它没有返回值，“CMP R6, 0”其实就是“CMP R4, 0”；而只有 R4 不为 0 时，程序才能从图 10-22 执行到图 10-23，根本就没有“CMP R4, 0”的必要，逻辑上有问题，因此推测它有返回值，且返回值类型是 int 或 BOOL。那么，CTCallDialWithID 的原型应该是 int CTCallDialWithID(NSString *phoneNumber, int UID)。为了验证我们的推测，双击 CTCallDialWithID 跳转到它的“实现”，如图 10-24 所示。

```
Imports from /System/Library/Frameworks/CoreTelephony.framework/CoreTelephony

                IMPORT __imp__CTCallAddressBlocked
                                        ; CODE XREF: _CTCallAddressBlocked↓j
                                        ; DATA XREF: _CTCallAddressBlocked↓o ...
                IMPORT __imp__CTCallCopyAddress
                                        ; CODE XREF: _CTCallCopyAddress↓j
                                        ; DATA XREF: _CTCallCopyAddress↓o ...
                IMPORT __imp__CTCallCopyCountryCode
                                        ; CODE XREF: _CTCallCopyCountryCode↓j
                                        ; DATA XREF: _CTCallCopyCountryCode↓o ...
                IMPORT __imp__CTCallCopyName ; CODE XREF: _CTCallCopyName↓j
                                        ; DATA XREF: _CTCallCopyName↓o ...
                IMPORT __imp__CTCallCopyNetworkCode
                                        ; CODE XREF: _CTCallCopyNetworkCode↓j
                                        ; DATA XREF: _CTCallCopyNetworkCode↓o ...
                IMPORT __imp__CTCallDeleteAllCallsBeforeDate
                                        ; CODE XREF: _CTCallDeleteAllCallsBeforeDate↓j
                                        ; DATA XREF: _CTCallDeleteAllCallsBeforeDate↓o ...
                IMPORT __imp__CTCallDeleteFromCallHistory
                                        ; CODE XREF: _CTCallDeleteFromCallHistory↓j
                                        ; DATA XREF: _CTCallDeleteFromCallHistory↓o ...
                IMPORT __imp__CTCallDial ; CODE XREF: _CTCallDial↓j
                                        ; DATA XREF: _CTCallDial↓o ...
                IMPORT __imp__CTCallDialEmergency
                                        ; CODE XREF: _CTCallDialEmergency↓j
                                        ; DATA XREF: _CTCallDialEmergency↓o ...
                IMPORT __imp__CTCallDialVoicemail
                                        ; CODE XREF: _CTCallDialVoicemail↓j
                                        ; DATA XREF: _CTCallDialVoicemail↓o ...
                IMPORT __imp__CTCallDialWithID
                                        ; CODE XREF: _CTCallDialWithID↓j
                                        ; DATA XREF: _CTCallDialWithID↓o ...
```

图 10-24 dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

它是 CoreTelephony 中的一个函数。在 IDA 中打开 CoreTelephony，定位到 CTCallDialWithID，如图 10-25 所示。

```
EXPORT _CTCallDialWithID
_CTCallDialWithID

var_C= -0xC

PUSH            {R7,LR}
MOV             R7, SP
SUB             SP, SP, #4
MOV             R2, R1
MOVS            R1, #0
STR             R1, [SP,#0xC+var_C]
MOVS            R1, #0
MOVS            R3, #0
BL              __redacted__10
ADD             SP, SP, #4
POP             {R7,PC}
; End of function _CTCallDialWithID
```

图 10-25 CTCallDialWithID 片段

很明显，核心功能都在 redacted__10 内完成，而“Xrefs from redacted__10”的结果如图 10-26 所示。

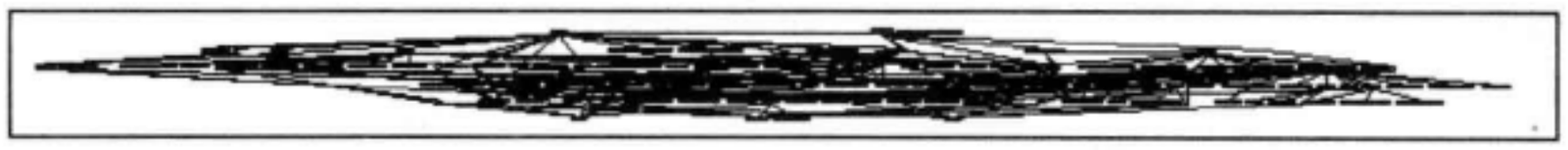


图 10-26 Xrefs from redacted__10

图 10-26 仿佛就是我们此时的真实写照——满脸黑线。显然要把 CTCallDialWithID 完全分析清楚还需要再写一本书，与其费劲地继续钻研汇编语言，不如实际验证一下刚才的推测。

```
extern "C" int CTCallDialWithID(NSString *phoneNumber, int UID);

%hook SpringBoard

- (void)menuButtonDown:(id)down
{
    %orig;

    NSLog(@"iOSRE: %d", CTCallDialWithID(@"10010", -1));
}
%end
```

我们已经多次使用这段代码进行测试，想必大家都已“眼”熟能详了；只有一点需要提

醒，就是记得在 MakeFile 中导入 CoreTelephony 这个 framework。安装之后，按下 home 键试一试效果吧。NSLog 输出内容见图 10-27。

```
Aug 23 13:17:25 [redacted]-FunMaker-5 SpringBoard[58651]: iOSRE: 521492432
Aug 23 13:18:19 [redacted]-FunMaker-5 SpringBoard[58651]: iOSRE: 522681440
Aug 23 13:29:38 [redacted]-FunMaker-5 SpringBoard[58651]: iOSRE: 522730992
```

图 10-27　测试 3 次，每次输出均不同，但都远大于 0

虽然已经找到了目标，但我们做事要善始善终，不要虎头蛇尾，接着把 [PhoneApplicationdialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted:] 的流程分析完。因为 CTCallDialWithID 的返回值不是 0，如图 10-28 所示。

```
MOVS            R1, #0
MOVS            R3, #0
BLX             _dispatch_time
MOVW            R3, #(:lower16:(__NSConcreteStackBlock_ptr - 0xF590))
ADD.W           LR, SP, #0x30+var_28
MOVT.W          R3, #(:upper16:(__NSConcreteStackBlock_ptr - 0xF590))
MOVW            R2, #(:lower16:(__dispatch_main_q_ptr - 0xF59C))
ADD             R3, PC ; __NSConcreteStackBlock_ptr
MOVT.W          R2, #(:upper16:(__dispatch_main_q_ptr - 0xF59C))
LDR             R3, [R3] ; __NSConcreteStackBlock
MOVW            R9, #(:lower16:(unk_92580 - 0xF5AC))
ADD             R2, PC ; __dispatch_main_q_ptr
MOVT.W          R9, #(:upper16:(unk_92580 - 0xF5AC))
MOVW            R12, #(:lower16:(sub_25E00+1 - 0xF5B4))
LDR             R2, [R2] ; __dispatch_main_q
MOVT.W          R12, #(:upper16:(sub_25E00+1 - 0xF5B4))
ADD             R9, PC ; unk_92580
STR             R3, [SP,#0x30+var_2C]
MOV.W           R3, #0x42000000
ADD             R12, PC ; sub_25E00
STMIA.W         LR, {R3,R5,R12}
ADD             R3, SP, #0x30+var_2C
STR.W           R9, [SP,#0x30+var_1C]
STR             R4, [SP,#0x30+var_18]
BLX             _dispatch_after
MOVW            R1, #(:lower16:(selRef_phoneCallForCall_ - 0xF5D8))
MOV             R2, R6
MOVT.W          R1, #(:upper16:(selRef_phoneCallForCall_ - 0xF5D8))
MOV             R0, #(classRef_PhoneCall - 0xF5DA) ; classRef_PhoneCall
ADD             R1, PC ; selRef_phoneCallForCall_
ADD             R0, PC ; classRef_PhoneCall
LDR             R1, [R1] ; "phoneCallForCall:"
LDR             R0, [R0] ; _OBJC_CLASS_$_PhoneCall
BLX             _objc_msgSend
MOV             R5, R0
MOV             R0, #(selRef_setWasDialAssisted_ - 0xF5F2) ; selRef_setWasDialAssisted_
LDRSB.W         R2, [SP,#0x30+var_14]
ADD             R0, PC ; selRef_setWasDialAssisted_
LDR             R1, [R0] ; "setWasDialAssisted:"
MOV             R0, R5
BLX             _objc_msgSend
MOVW            R0, #(:lower16:(selRef__performDialOfNumber_withCall_ - 0xF608))
MOV             R2, R4
MOVT.W          R0, #(:upper16:(selRef__performDialOfNumber_withCall_ - 0xF608))
MOV             R3, R5
ADD             R0, PC ; selRef__performDialOfNumber_withCall_
LDR             R1, [R0] ; "_performDialOfNumber:withCall:"
MOV             R0, R8
BLX             _objc_msgSend
MOV             R0, R6
BLX             _CFRelease
MOVS            R5, #1
```

图 10-28　dialPhoneNumber:forUID:dialAssist:wasAlreadyAssisted: 片段

又是一个“_performDialOfNumber:withCall:”，怎么会又拨打一次呢？注意 withCall: 的参数，它来自 phoneCallForCall: 的返回值，而 phoneCallForCall: 的参数来自于 R6，即 CTCallDialWithID 的返回值；在 class-dump 出的 .h 文件里找到 phoneCallForCall: 的定义，它的参数类型是 CTCallRef，如图 10-29 所示。

那么现在可以确定，CTCallDialWithID 的原型应该是 CTCallRef CTCallDialWithID (NSString *phoneNumber, int UID) 了。所以图 10-28 的意思大概是：sub_25E00 把通话记录标为已读，然后由“_performDialOfNumber:withCall:”把 CTCallDialWithID 生成的“Call”(呼叫）给“perform”(表现）在了 MobilePhone 的界面上。为了验证这个猜测，笔者使用 Cycript 做了一个小小的试验，如图 10-30 所示。

```
11 @class UIImage, NSString;
12
13 __attribute__((visibility("hidden")))
14 @interface PhoneCall : XXUnknownSuperclass {
15         CTCallRef _call;
16         NSString* _displayName;
17         NSString* _overrideName;
18         int _displayNameBreakPoint;
19         int _abUid;
20         NSString* _abLabel;
21         UIImage* _image;
22         struct {
23                 unsigned imageType : 2;
24                 unsigned abUidIsValid : 1;
25                 unsigned dialAssisted : 1;
26                 unsigned dialedFromEmergencyUI : 1;
27                 unsigned isEmergencyCall : 1;
28                 unsigned joiningConference : 1;
29                 unsigned leavingConference : 1;
30         } _phoneCallFlags;
31 }
32 +(id)phoneCallForCall:(CTCallRef)call;
```

图 10-29 phoneCallForCall: 的定义

```
[redacted]-FunMaker-5:~ root# cycript -p MobilePhone
cy# [UIApp _performDialOfNumber:@"10010" withCall:nil]
cy#
```

图 10-30 Cycript 试验

然后我的 iPhone 5 界面变成了图 10-31 所示的样子。

从 [DialerController _callButtonPressed:] 开始，到 CTCallDialWithID 结束，我们的第一个任务已经完成了。但你在之前的测试中应该已经发现了，调用 CTCallDialWithID 就会显示通话界面，无需“_performDialOfNumber:withCall:”——按照我们的推导，不是“_performDialOfNumber:withCall:”才应是显示通话界面的函数吗？这是怎么回事？

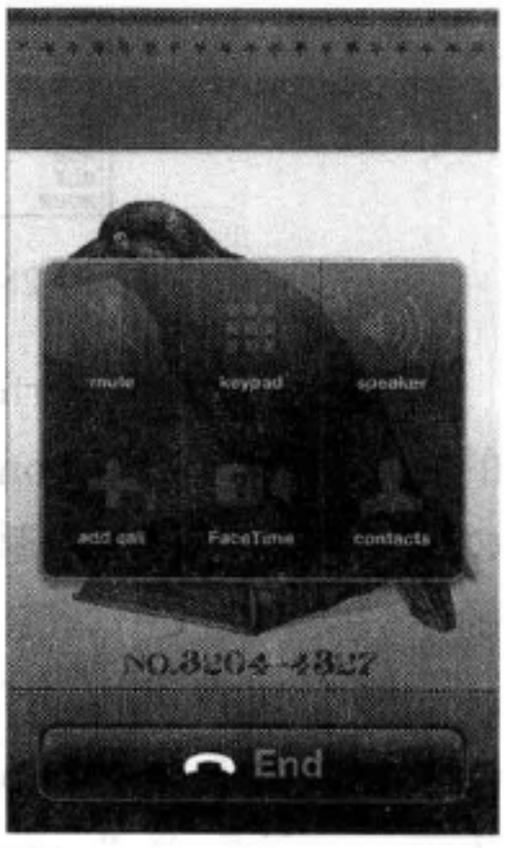


图 10-31 没有通话的通话界面

所谓“尽信书不如无书”，这个问题的唯一解释就是，显示通话界面的函数在另一个地方被调用了——这个“显示通话界面的函数”要么就是“_performDialOfNumber:withCall:”本身，要么就在“_performDialOfNumber:withCall:”内部。寻找这个函数的真身作为练习留给读者了，给个小提示：[PhoneRootViewController transitionToView:animated:completionBlock:] 才是显示通话界面的函数，它在 [PhoneApplication _displayedCallsChanged:] 和 [PhoneApplication applicationOpenURL:] 等函数里都得到了调用，如果有兴趣，可以综合运用前面的知识追根溯源，尝试彻底屏蔽这个界面，体验一把没法挂电话的刺激感！

10.2.3　寻找接听电话的函数

跟上一节的套路一样，下面先看看来电界面的特征，如图10-32所示。

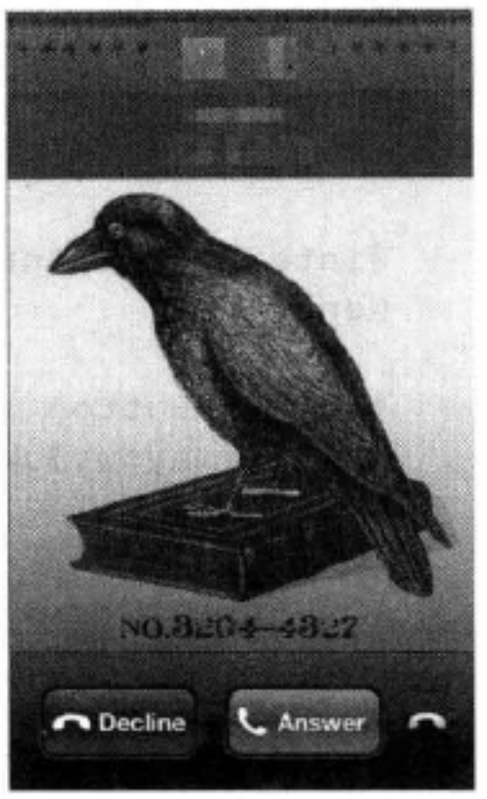


图 10-32　来电界面

我们的第一选择仍然是用 Reveal 查看 Answer 按钮对应的代码，但在实际操作中就会发现，当我们接到一个来电时，显示无法捕获到来电界面；笔者也不清楚这到底是自己的问题还是 Reveal 的 bug，那就只好用笨办法了——祭出神器“grep”。先把 /System/ 全给 grep 一遍，如图 10-33 所示。

因为 Answer 这个关键词太常见了，所以得到的结果很多。但是我们的目标显然跟电话有关，这样的话，结果里符合条件的着实不多，其中最吸引眼球的一条结果如图 10-34 所示。

根据前面的分析，我想大部分读者都会跟我一样，怀疑接听电话的操作就藏在 CoreTelephony 里。继续往下看，图 10-35 中的中间部分也很可疑。

```
[redacted]-MacBook-Pro:~ snakeninny$ grep -r "Answer" /Users/snakeninny/Code/iOSSys
temBinaries/6.0.1_iPhone5/System
```

图 10-33　grep 关键词 Answer

```
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/Frameworks/CoreTelephony.framework/CommCenter matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/Frameworks/CoreTelephony.framework/CoreTelephony matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/Frameworks/CoreTelephony.framework/Support/CommCenter matches
```

图 10-34　grep 出了 CoreTelephony

```
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/SyncedDefaults.framework/Support/syncdefaultsd matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/en_GB.lproj/General.strings matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/English.lproj/General.strings matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/TelephonyUI matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TextInput.framework/TextInput matches
```

图 10-35　grep 出了 TelephonyUI

关键词UI已经非常明显地告诉我们，“TelephonyUI”应该是“电话界面”。“Answer”关键词既出现在了 strings 里，又出现在了二进制文件中，那么直接在 TelephonyUI 的所有 .h 文件里搜 answer 看看效果，如图 10-36 所示。

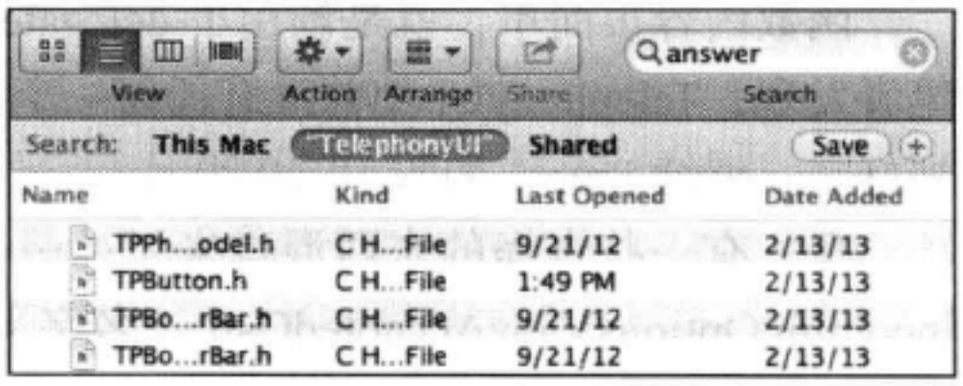


图 10-36　在 Finder 中搜索“answer”

没什么可说的了，这个 TPButton 八九不离

十就是 Answer 按钮的类，下面再看看它的内容，如图 10-37 所示。

还能再明显一点吗？让我们用事实说话，确认这个 TPButton 就是 Answer 按钮，其分析套路与上一节一模一样。编写下面的代码，得出这个 TPButton 的响应函数：

```
@interface TPButton : UIButton
@end

%hook TPButton
-(void)layoutSubviews
{

    %orig;

     for (id target in [self allTargets])
     NSLog(@"iOSRE: target = %@, actions = %@", target, [self
     actionsForTarget:target forControlEvent:[self allControlEvents]]);
}
%end
```

```
11 @interface TPButton : UIButton
12 {
13     UIImageView *_plusSeparator;
14     int _color;
15     NSString *_plusSeparatedTitle;
16     BOOL _forceReLayout;
17     BOOL _currentlyLayingOut;
18     BOOL _useStandardSizing;
19 }
20
21 + (id)acceptVideoButtonImage;
22 + (id)declineVideoButtonImage;
23 + (id)declineButtonImage;
24 + (id)answerButtonImage;
```

图 10-37 TPButton.h 的内容

记得添加一条“com.apple.TelephonyUI”的 filter。给本机打个电话，接下来的操作，你懂的，直接看输出，如图 10-38 所示。

```
65048 Dec  2 23:48:17 [redacted]-FunMaker-5 SpringBoard[696]: iOSRE:
      target = <TPIncomingCallOptionsView: 0x20a123a0; frame = (0
      472; 320 96); layer = <CALayer: 0x20a12450>>, actions = (
65049             "_decline"
65050         )
65051 Dec  2 23:48:17 [redacted]-FunMaker-5 SpringBoard[696]: iOSRE:
      target = <TPIncomingCallOptionsView: 0x20a123a0; frame = (0
      472; 320 96); layer = <CALayer: 0x20a12450>>, actions = (
65052             "_accept"
65053         )
65054 Dec  2 23:48:18 [redacted]-FunMaker-5 SpringBoard[696]: iOSRE:
      target = <TPIncomingCallOptionsView: 0x20a123a0; frame = (0
      0; 320 226); autoresize = W+TM; layer = <CALayer: 0x20a12450
      >>, actions = (
65055             "_buttonClicked:"
65056         )
```

图 10-38 syslog 片段

又是 accept 又是 decline，看来我们不光找到了接听电话的函数，还顺手把下一节挂断电话的函数也给搞定了。先去 IDA 里看看 [TPIncomingCallOptionsView _accept] 吧，如图 10-39 所示。

函数内容很简单，就是调用了 delegate 里的 incomingOptionsViewAcceptedCall: 函数，但笔者发现，TelephonyUI 中并没有任何一个类实现了 incomingOptionsViewAcceptedCall: 函数，那好吧，继续 grep，如图 10-40 所示。

嗯？有一些奇怪的东西混进来了……“IncomingCall.servicebundle”是什么？它实现了 incomingOptionsViewAcceptedCall:，名字又这么“暧昧”，我们就把它拖到 IDA 里看看好了，如图 10-41 所示。

```
; TPIncomingCallOptionsView - (void)_accept
; Attributes: bp-based frame

; void __cdecl -[TPIncomingCallOptionsView _accept](struct TPIncomingCallOptionsView *self, SEL)
__TPIncomingCallOptionsView__accept_
PUSH            {R4,R7,LR}
MOV             R4, R0
MOV             R0, #(selRef_incomingOptionsDelegate - 0x33056362) ; selRef_incomingOptionsDelegate
ADD             R7, SP, #4
ADD             R0, PC ; selRef_incomingOptionsDelegate
LDR             R1, [R0] ; "incomingOptionsDelegate"
MOV             R0, R4
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_incomingOptionsViewAcceptedCall_ - 0x33056376))
MOV             R2, R4
MOVT.W          R1, #(:upper16:(selRef_incomingOptionsViewAcceptedCall_ - 0x33056376))
ADD             R1, PC ; selRef_incomingOptionsViewAcceptedCall_
LDR             R1, [R1] ; "incomingOptionsViewAcceptedCall:"
POP.W           {R4,R7,LR}
B.W             j__objc_msgSend
; End of function -[TPIncomingCallOptionsView _accept]
```

图 10-39　_accept 片段

```
-MacBook-Pro:~ snakeninny$ grep -r "incomingOptionsViewAcceptedCall" /Use
rs/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/Caches/com.apple.dyld/dyld_shared_cache_armv7s matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/TelephonyUI matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/TelephonyUI.id0 matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/SpringBoardPlugins/IncomingCall.servicebundle/IncomingCall matches
-MacBook-Pro:~ snakeninny$
```

图 10-40　grep 关键词“incomingOptionsViewAcceptedCall”

```
; MPIncomingCallController - (void)incomingOptionsViewAcceptedCall:(id)

; void __cdecl -[MPIncomingCallController incomingOptionsViewAcceptedCall:](struct MPIncomingCallController *self, SEL, id)
__MPIncomingCallController_incomingOptionsViewAcceptedCall__
MOV             R1, #(selRef_handleOptionsSheetAnswer - 0xE6EC) ; selRef_handleOptionsSheetAnswer
ADD             R1, PC ; selRef_handleOptionsSheetAnswer
LDR             R1, [R1] ; "handleOptionsSheetAnswer"
B.W             j__objc_msgSend
; End of function -[MPIncomingCallController incomingOptionsViewAcceptedCall:]
```

图 10-41　incomingOptionsViewAcceptedCall: 片段

转战 handleOptionsSheetAnswer，如图 10-42 所示。

为什么会出现“handle failure”呢？仔细看看这里的提示信息，“Subclass of MPIncoming CallController should implement -handleOptionsSheetAnswer”，也就是说这个函数的实现应该在 MPIncomingCallController 的子类里。在功能窗口继续搜索“handleOptionsSheetAnswer”，很容易就会发现还有 [MPIncomingPhoneCallController handleOptionsSheetAnswer] 和 [MPIncoming FaceTimeCallController handleOptionsSheetAnswer] 两个函数，其中 MPIncomingPhoneCallController 是 MPIncomingCallController 关于电话的子类，MPIncomingFaceTimeCallController 是关于 FaceTime 的子类，我们研究的自然是前者，如图 10-43 所示。

它调用了 [self answer:]，继续看图 10-44。

```
; void __cdecl -[MPIncomingCallController handleOptionsSheetAnswer](struct MPIncomingCallController *self, SEL)
__MPIncomingCallController_handleOptionsSheetAnswer_

var_10= -0x10

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
SUB             SP, SP, #0xC
MOV             R4, R1
MOV             R1, #(selRef_currentHandler - 0x10A36) ; selRef_currentHandler
MOV             R5, R0
MOVW            R0, #(:lower16:(classRef_NSAssertionHandler - 0x10A3E))
ADD             R1, PC ; selRef_currentHandler
MOVT.W          R0, #(:upper16:(classRef_NSAssertionHandler - 0x10A3E))
LDR             R1, [R1] ; "currentHandler"
ADD             R0, PC ; classRef_NSAssertionHandler
LDR             R0, [R0] ; _OBJC_CLASS_$_NSAssertionHandler
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_stringWithUTF8String_ - 0x10A5C))
MOV             R6, R0
MOVT.W          R1, #(:upper16:(selRef_stringWithUTF8String_ - 0x10A5C))
MOV             R0, #(classRef_NSString - 0x10A5E) ; classRef_NSString
MOVW            R2, #(:lower16:(aSourcecacheM_0 - 0x10A68)) ; "/SourceCache/MobilePhone/MobilePhone-12"...
ADD             R1, PC ; selRef_stringWithUTF8String_
ADD             R0, PC ; classRef_NSString
MOVT.W          R2, #(:upper16:(aSourcecacheM_0 - 0x10A68)) ; "/SourceCache/MobilePhone/MobilePhone-12"...
LDR             R1, [R1] ; "stringWithUTF8String:"
LDR             R0, [R0] ; _OBJC_CLASS_$_NSString
ADD             R2, PC  ; "/SourceCache/MobilePhone/MobilePhone-12"...
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x10A7E))
MOV.W           R3, #0x3EC
MOVT.W          R1, #(:upper16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x10A7E))
MOVW            R2, #(:lower16:(cfstr_SubclassOfMp_2 - 0x10A86)) ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetAnswer"
ADD             R1, PC ; selRef_handleFailureInMethod_object_file_lineNumber_description_
MOVT.W          R2, #(:upper16:(cfstr_SubclassOfMp_2 - 0x10A86)) ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetAnswer"
LDR             R1, [R1] ; "handleFailureInMethod:object:file:lineN"...
ADD             R2, PC  ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetAnswer"
STMEA.W         SP, {R0,R3}
MOV             R0, R6
MOV             R3, R5
STR             R2, [SP,#0x18+var_10]
MOV             R2, R4
BLX             _objc_msgSend
ADD             SP, SP, #0xC
POP             {R4-R7,PC}
; End of function -[MPIncomingCallController handleOptionsSheetAnswer]
```

图 10-42 handleOptionsSheetAnswer 片段

```
; MPIncomingPhoneCallController - (void)handleOptionsSheetAnswer

; void __cdecl -[MPIncomingPhoneCallController handleOptionsSheetAnswer](struct MPIncomingPhoneCa
__MPIncomingPhoneCallController_handleOptionsSheetAnswer_
MOV             R1, #(_OBJC_IVAR_$_SBUISlidingFullscreenAlertController._bottomBar_ptr - 0x44F4)
ADD             R1, PC ; _OBJC_IVAR_$_SBUISlidingFullscreenAlertController._bottomBar_ptr
LDR             R1, [R1] ; _OBJC_IVAR_$_SBUISlidingFullscreenAlertController._bottomBar
LDR             R2, [R1]
MOV             R1, #(selRef_answer_ - 0x4502) ; selRef_answer_
ADD             R1, PC ; selRef_answer_
LDR             R2, [R0,R2]
LDR             R1, [R1] ; "answer:"
B.W             j__objc_msgSend
; End of function -[MPIncomingPhoneCallController handleOptionsSheetAnswer]
```

图 10-43 handleOptionsSheetAnswer 片段

函数的逻辑简单清晰，难度并不大，笔者也没必要手把手带着读者一步步分析了，只要稍作浏览，就会发现这个函数的绝大部分工作是在布置来电界面，其最核心的部分就体现在图 10-45 中。

转到 answerCall: 的实现，如图 10-46 所示。

果不其然，还记得图 10-34 的 grep 结果吗？绕了一大圈，还是回到了原点，其实当时只要用 IDA 打开 CoreTelephony，搜索含有 answer 关键词的函数，应该就能快速解决战斗。虽然走了弯路，但得到的回报是整理出了 iOS 处理来电的逻辑：当 iOS 接到一个来电时，TelephonyUI.framework 只是显示界面，实际操作交由 IncomingCall.servicebundle 处理；而 IncomingCall.servicebundle 最终还是要调用 CoreTelephony 来接听电话；之前用 Reveal 捕获来电界面失败的原因也大白于天下了：来电界面是由 IncomingCall.servicebundle 生成的，而 libReveal.dylib 却被加载进了 MobilePhone.app，自然捕获不到来电界面了！从图 10-46

就可以看出，CTCallAnswer 的原型是 void CTCallAnswer (struct __CTCall *call)；再对应到 CoreTelephony-Structs.h（如图 10-47 所示），相当于 void CTCallAnswer(CTCallRef call)。

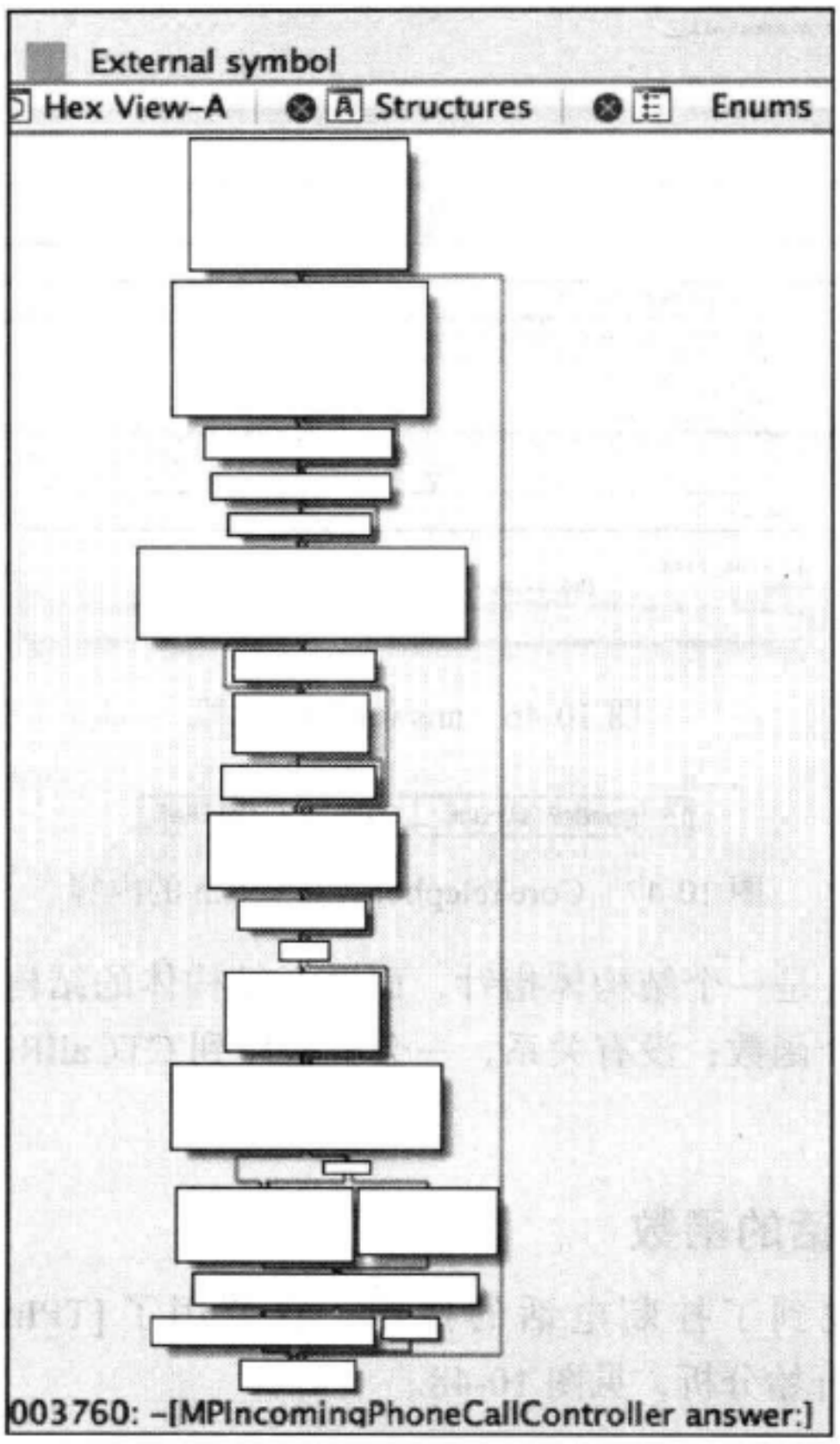


图 10-44　answer: 片段

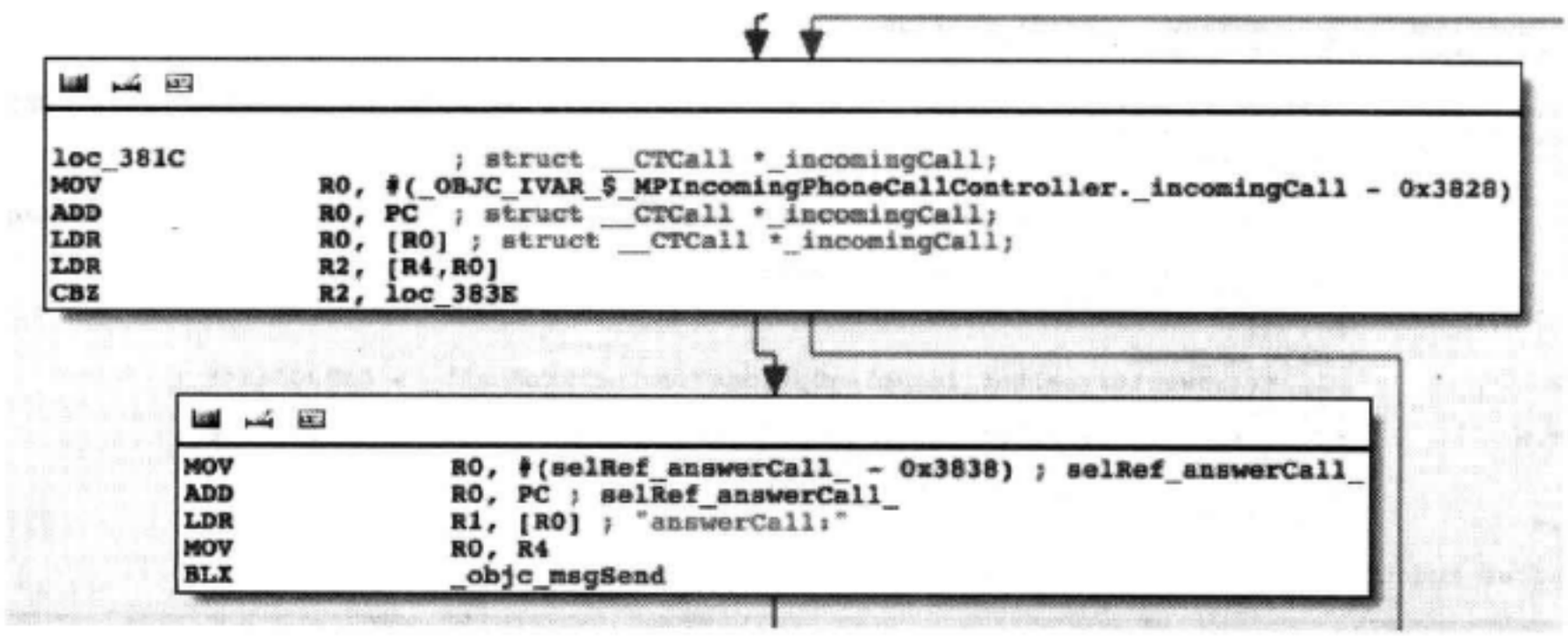


图 10-45　answer: 片段

```
; MPIncomingPhoneCallController - (void)answerCall:(struct __CTCall *)
; Attributes: bp-based frame

; void __cdecl -[MPIncomingPhoneCallController answerCall:](struct MPIncomingPhoneCallController *self, SEL, struct __CTCall *)
__MPIncomingPhoneCallController_answerCall__
PUSH            {R4,R7,LR}
MOV             R4, R2
ADD             R7, SP, #4
MOV             R0, R4
BLX             _CTCallGetStatus
CMP             R0, #4
BNE             locret_374E
```

```
BLX             _SBUIGetUserAgent
MOV             R1, #(selRef_prepareToAnswerCall - 0x3746) ; selRef_prepareToAnswerCall
ADD             R1, PC ; selRef_prepareToAnswerCall
LDR             R1, [R1] ; "prepareToAnswerCall"
BLX             _objc_msgSend
MOV             R0, R4
BLX             _CTCallAnswer
```

```
locret_374E
POP             {R4,R7,PC}
; End of function -[MPIncomingPhoneCallController answerCall:]
```

图 10-46 answerCall: 片段

```
15 typedef struct __CTCall* CTCallRef;
```

图 10-47 CoreTelephony-Structs.h 的内容

因为这里的参数 call 是一个结构体指针，而这个结构体的结构我们不知道，没法构造，所以目前还无法测试这个函数；没有关系，一会就会碰到 CTCallRef 类的对象，到时候再测试也不迟。

10.2.4 寻找挂断电话的函数

在上一节中已经找到了挂断电话的按钮，它调用了 [TPIncomingCallOptionsView _decline]，下面就从这里开始分析，见图 10-48。

```
; TPIncomingCallOptionsView - (void)_decline
; Attributes: bp-based frame

; void __cdecl -[TPIncomingCallOptionsView _decline](struct TPIncomingCallOptionsView *self, SEL)
__TPIncomingCallOptionsView__decline_
PUSH            {R4,R7,LR}
MOV             R4, R0
MOV             R0, #(selRef_incomingOptionsDelegate - 0x33056332) ; selRef_incomingOptionsDelegate
ADD             R7, SP, #4
ADD             R0, PC ; selRef_incomingOptionsDelegate
LDR             R1, [R0] ; "incomingOptionsDelegate"
MOV             R0, R4
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_incomingOptionsViewDeclinedCall_ - 0x33056346))
MOV             R2, R4
MOVT.W          R1, #(:upper16:(selRef_incomingOptionsViewDeclinedCall_ - 0x33056346))
ADD             R1, PC ; selRef_incomingOptionsViewDeclinedCall_
LDR             R1, [R1] ; "incomingOptionsViewDeclinedCall:"
POP.W           {R4,R7,LR}
B.W             j__objc_msgSend
; End of function -[TPIncomingCallOptionsView _decline]
```

图 10-48 _decline 片段

其分析套路与 _answer 如出一辙，咱们又回到了 IncomingCall.servicebundle 中，如图 10-49 所示。

```
; MPIncomingCallController - (void)incomingOptionsViewDeclinedCall:(id)

; void __cdecl -[MPIncomingCallController incomingOptionsViewDeclinedCall:](struct MPIncomingCallController *self, SEL, id)
__MPIncomingCallController_incomingOptionsViewDeclinedCall__
MOVW            R1, #(:lower16:(selRef_handleOptionsSheetIgnoreWithDismiss_ - 0xE6FE))
MOVS            R2, #1
MOVT.W          R1, #(:upper16:(selRef_handleOptionsSheetIgnoreWithDismiss_ - 0xE6FE))
ADD             R1, PC ; selRef_handleOptionsSheetIgnoreWithDismiss_
LDR             R1, [R1] ; "handleOptionsSheetIgnoreWithDismiss:"
B.W             j__objc_msgSend
; End of function -[MPIncomingCallController incomingOptionsViewDeclinedCall:]
```

图 10-49　incomingOptionsViewDeclinedCall: 片段

该片断没什么好说的，继续看图 10-50。

```
; void __cdecl -[MPIncomingCallController handleOptionsSheetIgnoreWithDismiss:](struct MPIncomingCallController *self, SEL, char)
__MPIncomingCallController_handleOptionsSheetIgnoreWithDismiss__

var_10= -0x10

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
SUB             SP, SP, #0xC
MOV             R4, R1
MOV             R1, #(selRef_currentHandler - 0x10AB2) ; selRef_currentHandler
MOV             R5, R0
MOVW            R0, #(:lower16:(classRef_NSAssertionHandler - 0x10ABA))
ADD             R1, PC ; selRef_currentHandler
MOVT.W          R0, #(:upper16:(classRef_NSAssertionHandler - 0x10ABA))
LDR             R1, [R1] ; "currentHandler"
ADD             R0, PC ; classRef_NSAssertionHandler
LDR             R0, [R0] ; _OBJC_CLASS_$_NSAssertionHandler
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_stringWithUTF8String_ - 0x10AD8))
MOV             R6, R0
MOVT.W          R1, #(:upper16:(selRef_stringWithUTF8String_ - 0x10AD8))
MOV             R0, #(classRef_NSString - 0x10ADA) ; classRef_NSString
MOVW            R2, #(:lower16:(aSourcecacheM_0 - 0x10AE4)) ; "/SourceCache/MobilePhone/MobilePhone-12"...
ADD             R1, PC ; selRef_stringWithUTF8String_
ADD             R0, PC ; classRef_NSString
MOVT.W          R2, #(:upper16:(aSourcecacheM_0 - 0x10AE4)) ; "/SourceCache/MobilePhone/MobilePhone-12"...
LDR             R1, [R1] ; "stringWithUTF8String:"
LDR             R0, [R0] ; _OBJC_CLASS_$_NSString
ADD             R2, PC  ; "/SourceCache/MobilePhone/MobilePhone-12"...
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x10AFA))
MOV.W           R3, #0x370
MOVT.W          R1, #(:upper16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x10AFA))
MOVW            R2, #(:lower16:(cfstr_SubclassOfMp_3 - 0x10B02)) ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetIgnoreWithDismiss:"
ADD             R1, PC ; selRef_handleFailureInMethod_object_file_lineNumber_description_
MOVT.W          R2, #(:upper16:(cfstr_SubclassOfMp_3 - 0x10B02)) ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetIgnoreWithDismiss:"
LDR             R1, [R1] ; "handleFailureInMethod:object:file:lineN"...
ADD             R2, PC  ; "Subclass of MPIncomingCallController should implement -handleOptionsSheetIgnoreWithDismiss:"
STMEA.W         SP, {R0,R3}
MOV             R0, R6
MOV             R3, R5
STR             R2, [SP,#0x18+var_10]
MOV             R2, R4
BLX             _objc_msgSend
ADD             SP, SP, #0xC
POP             {R4-R7,PC}
; End of function -[MPIncomingCallController handleOptionsSheetIgnoreWithDismiss:]
```

图 10-50　handleOptionsSheetIgnoreWithDismiss: 片段

这里也没有需要进一步分析的内容，继续看图 10-51。

```
; MPIncomingPhoneCallController - (void)handleOptionsSheetIgnoreWithDismiss:(char)

; void __cdecl -[MPIncomingPhoneCallController handleOptionsSheetIgnoreWithDismiss:](struct MPIncomingPhoneCallController *self, SEL, char)
__MPIncomingPhoneCallController_handleOptionsSheetIgnoreWithDismiss__
MOV             R1, #(selRef_ignoreWithDismiss_ - 0x4514) ; selRef_ignoreWithDismiss_
ADD             R1, PC ; selRef_ignoreWithDismiss_
LDR             R1, [R1] ; "ignoreWithDismiss:"
B.W             j__objc_msgSend
; End of function -[MPIncomingPhoneCallController handleOptionsSheetIgnoreWithDismiss:]
```

图 10-51　handleOptionsSheetIgnoreWithDismiss: 片段

在图 10-51 所示的片断中，ignoreWithDismiss: 一开始就是个 CTCallDisconnect() 函数，如图 10-52 所示。

CTCallDisconnect 的原型是 void CTCallDisconnect(CTCallRef call)，整个流程与上一节几乎一模一样，笔者就不再啰唆了，因为与上节相同的原因，等会再测试这个函数。

```
; void __cdecl -[MPIncomingPhoneCallController ignoreWithDismiss:](struct MPIncomingPhoneCallController *self, SEL, char)
__MPIncomingPhoneCallController_ignoreWithDismiss__

var_24= -0x24
var_20= -0x20
var_1C= -0x1C
var_18= -0x18
var_C= -0xC

PUSH            {R4,R5,R7,LR}
ADD             R7, SP, #8
SUB             SP, SP, #0x1C
MOV             R5, R2
MOV             R4, R0
BLX             _CTTelephonyCenterGetDefault
MOV             R1, R4
BLX             _CTTelephonyCenterRemoveEveryObserver
MOV             R1, #(selRef_defaultCenter - 0x3B50) ; selRef_defaultCenter
MOV             R0, #(classRef_NSNotificationCenter - 0x3B52) ; classRef_NSNotificationCenter
ADD             R1, PC ; selRef_defaultCenter
ADD             R0, PC ; classRef_NSNotificationCenter
LDR             R1, [R1] ; "defaultCenter"
LDR             R0, [R0] ; _OBJC_CLASS_$_NSNotificationCenter
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_removeObserver_ - 0x3B66))
MOV             R2, R4
MOVT.W          R1, #(:upper16:(selRef_removeObserver_ - 0x3B66))
ADD             R1, PC ; selRef_removeObserver_
LDR             R1, [R1] ; "removeObserver:"
BLX             _objc_msgSend
MOVW            R0, #(:lower16:(_OBJC_IVAR_$_MPIncomingPhoneCallController._observingRingerChanged - 0x3B78)) ; char _observingRingerChanged;
MOVS            R2, #0
MOVT.W          R0, #(:upper16:(_OBJC_IVAR_$_MPIncomingPhoneCallController._observingRingerChanged - 0x3B78)) ; char _observingRingerChanged;
ADD             R0, PC  ; char _observingRingerChanged;
LDR             R1, [R0] ; char _observingRingerChanged;
MOV             R0, #(_OBJC_IVAR_$_MPIncomingPhoneCallController._incomingCall - 0x3B84) ; struct __CTCall *_incomingCall;
ADD             R0, PC  ; struct __CTCall *_incomingCall;
LDR             R0, [R0] ; struct __CTCall *_incomingCall;
STRB            R2, [R4,R1]
LDR             R0, [R4,R0]
CMP             R0, #0
IT NE
BLXNE           _CTCallDisconnect
TST.W           R5, #0xFF
BEQ             loc_3C3A
```

图 10-52 ignoreWithDismiss: 片段

10.2.5 寻找删除通话记录的函数

如果读者跟上了前几节的进度，本节的任务应该可以轻松地独立完成，因此本节的文字解说不会太多，尽量“点到为止”，希望能激发读者的思维，与笔者一起完成本节的任务。

iOS 里删除通话记录的操作需要在“最近通话”界面进行，如图 10-53 所示。

其实绝大多数 iOS 开发者应该对这个界面非常熟悉了，它是一个典型的 UITableView，按下右上角的 Edit 按钮，点击出现的红色减号，再选择 Delete，就完成了一条记录的删除操作，而这正是 UITableViewDataSource 中定义的删除 UITableView 某一行的标准方法，对应的函数是 tableView:commitEditingStyle:forRowAtIndexPath:。直接在 MobilePhone 的所有头文件里搜索 commitEditingStyle，结果如图 10-54 所示。



图 10-53 “最近通话”界面

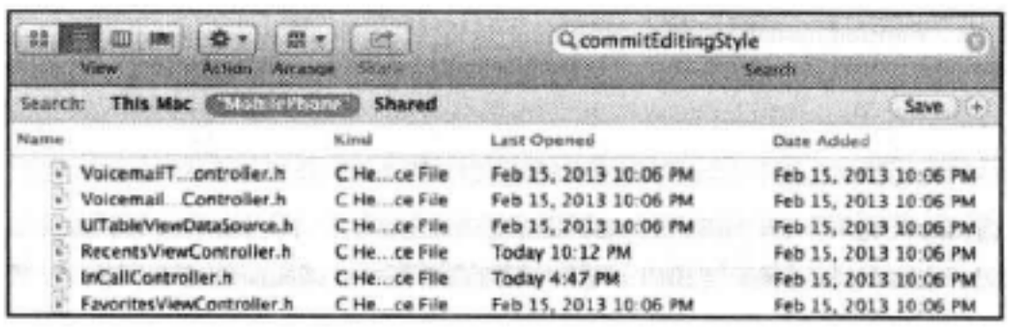


图 10-54 Finder 搜索关键词 commitEditingStyle

在 IDA 里打开 MobilePhone，定位到 [RecentsViewController tableView:commitEditingStyle:forRowAtIndexPath:]，如图 10-55 所示。

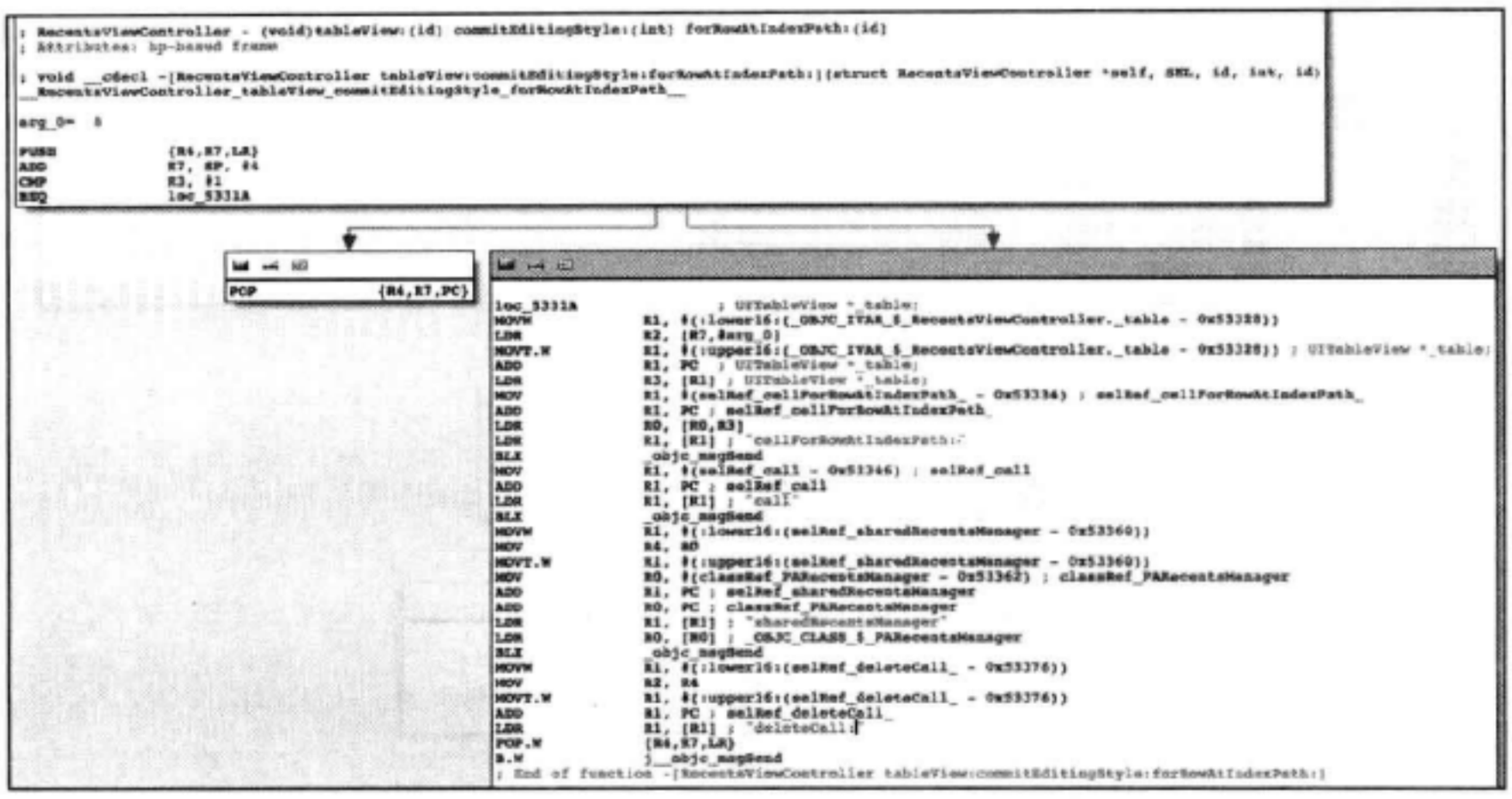


图 10-55　tableView:commitEditingStyle:forRowAtIndexPath: 片段

目标还能再明确点吗？转到 [PARecentsManager deleteCall:]，简单浏览一下，定位到图 10-56。



图 10-56　deleteCall: 片段

这段代码的关键词都是 delete，所以几乎不需要关注上下文就可以“武断”地确定目标。这里，removeObject: 只是将某一条记录从所有记录的数组中移除，而这条记录仍存在于通话

记录的数据库中；最终的绝杀是由 deleteUnderlyingCTCall 完成的，如图 10-57 所示。

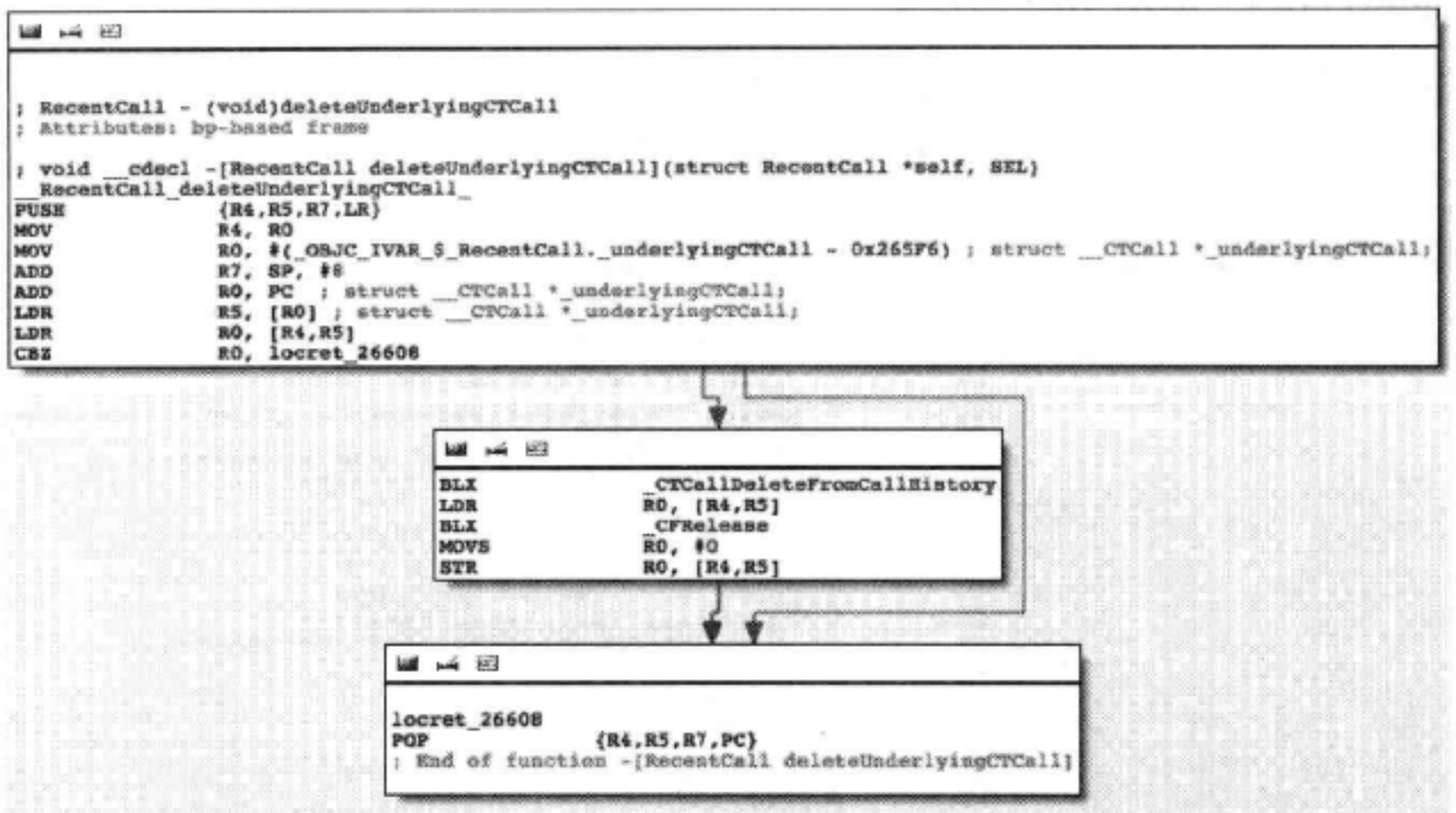


图 10-57 deleteUnderlyingCTCall 片段

最后得出结论，删除通话记录的函数是 void CTCallDeleteFromCallHistory(CTCallRef call)，留待后面测试。

10.2.6 寻找拦截电话的方法

前面针对拨打电话、接听电话、挂断电话、删除通话记录等四个功能进行了查找，相信读者已经非常熟悉这个套路了。对于系统原生的功能来说，我们总能寻找到蛛丝马迹，然后顺藤摸瓜，在 IDA、GDB 和 grep 的帮助下定位目标，随着熟练程度越来越高，所需要的时间会越来越短。但是，有时我们想要的功能是 iOS 没有提供的，例如拦截电话，它们根本无迹可寻，要想实现这类功能，又该从何下手呢?

既然这类功能没法“山寨”，那就拿出一些创新精神，开动我们的脑筋，抛弃思维定势，大胆幻想并进行尝试。笔者无法给出一个通用的公式，希望读者从“拦截电话”的实现过程中抽丝剥茧，最终达到“无招胜有招”的境界。

其实说到这里，很多读者可能都没有仔细考虑过一个问题——什么叫“拦截电话”？有人会说：“某个号码打来电话时被直接挂断，iPhone 上没有这个号码来电的提示，就叫拦截电话。”这个定义一点也没错，但太平民化了，也就是说，这是一名 iPhone 用户眼中的“拦截电话”，对于程序员来说，这样的概念太抽象，提供的信息不足以让我们把它转换成代码。那么从程序员或 iOS 逆向工程师的角度来看，什么是“拦截电话”呢?

在第 5 章里，笔者曾提到过，iOS 是由无数个“对象”构成的，那么“对象”之间的关系是如何产生的呢？答案是“方法”和“实现”。当一个对象的方法得到调用时，对象要去实现这个方法——一个巴掌拍不响，一个篱笆三个桩，一个好汉三个帮，在实现这个方法的过程中，对象可能会用到其他对象的方法和各种 C 函数，这个过程不断嵌套，将各个对象之

间的关系紧紧耦合在一起。它们的关系可以看做是这样的一种模型，如图 10-58 所示。

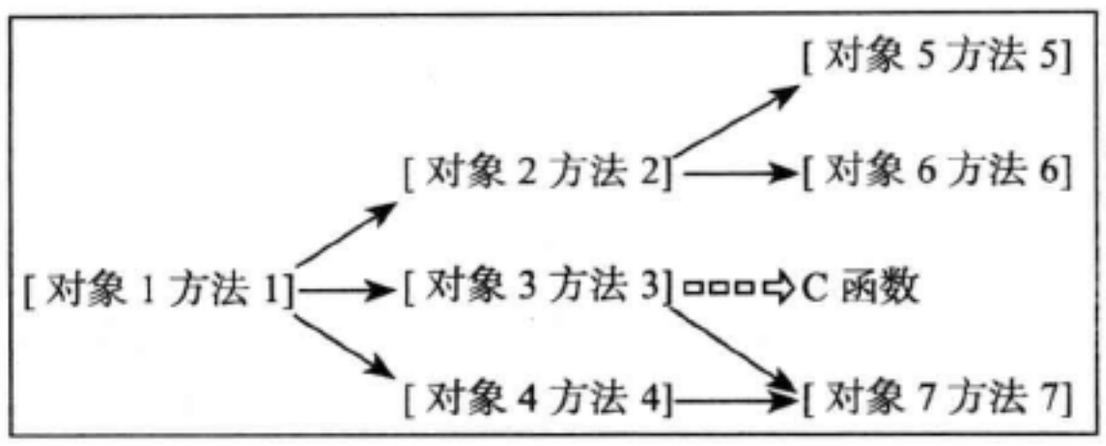


图 10-58　对象关系模型

当 iOS 感知到一个来电，用“方法”来描述它的一种典型反应，可以是这样的：

- 暂停当前工作。
- 显示来电界面。
- 响铃。
- 振动。

当挂断来电后，则是：

- 退出来电界面。
- 恢复来电之前的工作。
- 添加一条通话记录。

模仿上面的方式，“某个号码打来电话时被直接挂断，iPhone 上没有这个号码来电的提示”可以翻译成：

当 iOS 感知到一个来电：

- 不要暂停当前工作。
- 不要显示来电界面。
- 不要响铃。
- 不要振动。
- 挂断来电。

当挂断来电后，则是：

- 不要添加一条通话记录。

也就是说，在正常情况下，iOS 来电时会调用多种方法，而“拦截电话”就是让它不要调用这些方法。你意识到什么了吗？是不是可以通过下面的代码来表示呢？

```
%hook 对象类 1
- (void) 暂停当前工作
{
    if （正常情况）
        %orig;
    else （拦截电话）
        return;
}
%end

%hook 对象类 2
- (void) 显示来电界面
{
    if （正常情况）
        %orig;
    else （拦截电话）
```

```
            return;
    }
    %end

    ...// 继续 hook
```

可以通过 hook 掉 iOS 来电调用的 7 种方法来达到“拦截电话”的目的，而这 7 种方法就好像图 10-58 中的方法 5～方法 7，需要消耗多个 hook；如果我们能够倒推回去，找到“感知来电”的方法 1，岂不是可以减少 hook 的数量，更优雅地解决战斗？

```
%hook 对象类 6
- (void)iOS 感知到来电
{
    if （正常情况）
        %orig;
    else （拦截电话）
        return;
}
%end
```

综上所述，我们要做的就是找到 iOS 来电的其中一种表现，然后找出这种表现的发起者、发起者的发起者、发起者的发起者的发起者……最后揪出幕后黑手，也就是“感知来电”函数，一举拿下，擒贼擒王。看起来很难？其实用到的都是我们已经掌握的知识，操作起来你就会知道，一点都不难。正式开始吧！

经过 10.2.3 节的分析，我们已经知道当 iOS 收到一个来电时，会显示 TPIncomingCallOptionsView，即“显示来电界面”；那么，一定有某个对象会在感知到来电后初始化 TPIncomingCallOptionsView 对象并把它显示在屏幕上，我们的目标是找到这个对象，即“显示来电界面”的发起者。打开 TPIncomingCallOptionsView.h，定位到它的初始化函数，如图 10-59 所示。

```
66 - (void)dealloc;
67 - (id)initWithFrame:(struct CGRect)arg1;
68 - (id)initForIncomingFaceTimeWithFrame:(struct CGRect)arg1;
69 - (id)initForIncomingCallWaitingWithFrame:(struct CGRect)arg1;
70 - (id)initForIncomingCallWithFrame:(struct CGRect)arg1;
71 - (id)_initForType:(int)arg1 frame:(struct CGRect)arg2;
```

图 10-59 TPIncomingCallOptionsView.h 的内容

这么多 init，哪一个才是我们要找的函数呢？显然，第 68 行和第 69 行分别是 FaceTime 和来电等待时的界面，剩下的 3 个都很可疑，但 initForIncomingCallWithFrame: 最像。不要猜了，在 IDA 里看看这 3 个函数的实现，首先是 initWithFrame:，如图 10-60 所示。

跟前面碰到的情况一样，这个函数并不起实际作用，再看看 initForIncomingCallWithFrame: 吧，如图 10-61 所示。

连 _initForType:frame: 都不用再看了，应该就是它了。接下来用 GDB 在它的结尾处下个断点，然后 ni 到它的调用者，看看谁是发起者。这个过程已经重复多遍，如图 10-62～图 10-66 所示。

```
; id __cdecl -[TPIncomingCallOptionsView initWithFrame:](struct TPIncomingCallOptionsView *self, SEL, struct CGRect)
__TPIncomingCallOptionsView_initWithFrame__

var_10= -0x10

PUSH            {R4-R7,LR}
ADD             R7, SP, #0xC
SUB             SP, SP, #0xC
MOV             R4, R1
MOV             R1, #(selRef_currentHandler - 0x33055BB6) ; selRef_currentHandler
MOV             R5, R0
MOVW            R0, #(:lower16:(classRef_NSAssertionHandler - 0x33055BBE))
ADD             R1, PC ; selRef_currentHandler
MOVT.W          R0, #(:upper16:(classRef_NSAssertionHandler - 0x33055BBE))
LDR             R1, [R1] ; "currentHandler"
ADD             R0, PC ; classRef_NSAssertionHandler
LDR             R0, [R0] ; _OBJC_CLASS_$_NSAssertionHandler
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_stringWithUTF8String_ - 0x33055BDC))
MOV             R6, R0
MOVT.W          R1, #(:upper16:(selRef_stringWithUTF8String_ - 0x33055BDC))
MOV             R0, #(classRef_NSString - 0x33055BDE) ; classRef_NSString
MOVW            R2, #(:lower16:(aSourcecacheTel - 0x33055BE8)) ; "/SourceCache/TelephonyUI/TelephonyUI-41"...
ADD             R1, PC ; selRef_stringWithUTF8String_
ADD             R0, PC ; classRef_NSString
MOVT.W          R2, #(:upper16:(aSourcecacheTel - 0x33055BE8)) ; "/SourceCache/TelephonyUI/TelephonyUI-41"...
LDR             R1, [R1] ; "stringWithUTF8String:"
LDR             R0, [R0] ; _OBJC_CLASS_$_NSString
ADD             R2, PC  ; "/SourceCache/TelephonyUI/TelephonyUI-41"...
BLX             _objc_msgSend
MOVW            R1, #(:lower16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x33055BFC))
MOVS            R3, #0x7D
MOVT.W          R1, #(:upper16:(selRef_handleFailureInMethod_object_file_lineNumber_description_ - 0x33055BFC))
MOVW            R2, #(:lower16:(cfstr_DoNotCallTpinc - 0x33055C04)) ; "Do not call -[TPIncomingCallOptionsView initWithFrame: directly"
ADD             R1, PC ; selRef_handleFailureInMethod_object_file_lineNumber_description_
MOVT.W          R2, #(:upper16:(cfstr_DoNotCallTpinc - 0x33055C04)) ; "Do not call -[TPIncomingCallOptionsView initWithFrame: directly"
LDR             R1, [R1] ; "handleFailureInMethod:object:file:lineN"...
ADD             R2, PC  ; "Do not call -[TPIncomingCallOptionsView initWithFrame: directly"
STMEA.W         SP, {R0,R3}
MOV             R0, R6
MOV             R3, R5
STR             R2, [SP,#0x18+var_10]
MOV             R2, R4
BLX             _objc_msgSend
MOVS            R0, #0
ADD             SP, SP, #0xC
POP             {R4-R7,PC}
; End of function -[TPIncomingCallOptionsView initWithFrame:]
```

图 10-60　initWithFrame: 片段

```
; TPIncomingCallOptionsView - (id)initForIncomingCallWithFrame:(struct CGRect)

; id __cdecl -[TPIncomingCallOptionsView initForIncomingCallWithFrame:](struct TPIncomingCallOptionsView *self, SEL, struct CGRect)
__TPIncomingCallOptionsView_initForIncomingCallWithFrame__

var_14= -0x14
var_10= -0x10
var_C= -0xC
var_8= -8

PUSH            {R7,LR}
MOV             R7, SP
SUB             SP, SP, #0xC
MOVW            R1, #(:lower16:(selRef__initForType_frame_ - 0x33055B26))
MOV             R9, R2
LDR             R2, [R7,#0x14+var_C]
MOVT.W          R1, #(:upper16:(selRef__initForType_frame_ - 0x33055B26))
LDR.W           R12, [R7,#0x14+var_8]
ADD             R1, PC ; selRef__initForType_frame_
STR             R3, [SP,#0x14+var_14]
MOV             R3, R9
LDR             R1, [R1] ; "_initForType:frame:"
STR             R2, [SP,#0x14+var_10]
MOVS            R2, #0
STR.W           R12, [SP,#0x14+var_C]
BLX             _objc_msgSend
ADD             SP, SP, #0xC
POP             {R7,PC}
; End of function -[TPIncomingCallOptionsView initForIncomingCallWithFrame:]
```

图 10-61　initForIncomingCallWithFrame: 片段

```
[redacted]-FunMaker-5:~ root# gdb
warning: unrecognized host cpusubtype 11, defaulting to host--armv7.
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4) (
Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
gdb$ attach SpringBoard
Attaching to process 83580.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ..........................................
..............................................................................
```

图 10-62　用 GDB 调试 SpringBoard

注意，根据前几节的分析，TelephonyUI 的调用者是 IncomingCall.servicebundle，因为它是个 SpringBoardPlugin，所以这里 attach 进 SpringBoard，而不是 MobilePhone。

```
 38 CoreTelephony                          F 0x3627c000        dyld Y Y /System/Li
brary/Frameworks/CoreTelephony.framework/CoreTelephony at 0x3627c000 (offset 0xf
d8000)
                                                              (objfile is) [memory ob
ject "/System/Library/Frameworks/CoreTelephony.framework/CoreTelephony" at 0x362
7c000]
 39 TelephonyUI                            F 0x3401b000        dyld Y Y /System/Li
brary/PrivateFrameworks/TelephonyUI.framework/TelephonyUI at 0x3401b000 (offset
0xfd8000)
                                                              (objfile is) [memory ob
ject "/System/Library/PrivateFrameworks/TelephonyUI.framework/TelephonyUI" at 0x
3401b000]
 40 Preferences                            F 0x31bed000        dyld Y Y /System/Li
brary/PrivateFrameworks/Preferences.framework/Preferences at 0x31bed000 (offset
0xfd8000)
                                                              (objfile is) [memory ob
ject "/System/Library/PrivateFrameworks/Preferences.framework/Preferences" at 0x
31bed000]
```

图 10-63　info sh 的结果

```
__text:33055B1E                 LDR.W           R12, [R7,#0x14+var_8]
__text:33055B22                 ADD             R1, PC ; selRef__initForType_frame_
__text:33055B24                 STR             R3, [SP,#0x14+var_14]
__text:33055B26                 MOV             R3, R9
__text:33055B28                 LDR             R1, [R1] ; "_initForType:frame:"
__text:33055B2A                 STR             R2, [SP,#0x14+var_10]
__text:33055B2C                 MOVS            R2, #0
__text:33055B2E                 STR.W           R12, [SP,#0x14+var_C]
__text:33055B32                 BLX             _objc_msgSend
__text:33055B36                 ADD             SP, SP, #0xC
__text:33055B38                 POP             {R7,PC}
__text:33055B38 ; End of function -[TPIncomingCallOptionsView initForIncomingCallWithFrame:]
```

图 10-64　initForIncomingCallWithFrame: 片段

```
gdb$ b *(0xfd8000 + 0x33055B38)
Breakpoint 1 at 0x3402db38
gdb$ c
```

图 10-65　用 GDB 下断点

```
Breakpoint 1, 0x3402db38 in <redacted> ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x0ce8426e in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0ce84272 in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0ce84274 in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
```

图 10-66　触发断点

很显然，这里的地址变化太大，0xce8426e 已经不在 TelephonyUI 里了。又该 grep 出场了，如图 10-67 和图 10-68 所示。

```
[redacted]-MacBook-Pro:~ snakeninny$ grep -r "initForIncomingCallWithFrame" /Users/
snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/Caches/com.apple.dyld/dyld_shared_cache_armv7s matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/Conference.framework/Conference matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/PrivateFrameworks/TelephonyUI.framework/TelephonyUI matches
Binary file /Users/snakeninny/Code/iOSSystemBinaries/6.0.1_iPhone5/System/Librar
y/SpringBoardPlugins/IncomingCall.servicebundle/IncomingCall matches
[redacted]-MacBook-Pro:~ snakeninny$
```

图 10-67　grep 关键词“initForIncomingCallWithFrame”

```
279 EKCalendar                              - 0x96d1000         dyld Y Y /System/Li
brary/SpringBoardPlugins/EKCalendar.servicebundle/EKCalendar at 0x96d1000 (offse
t 0x96d1000)
280 IncomingCall                            - 0xce80000         dyld Y Y /System/Li
brary/SpringBoardPlugins/IncomingCall.servicebundle/IncomingCall at 0xce80000 (o
ffset 0xce80000)
281 BackgroundTaskAgent                     F 0x3937e000        dyld Y Y /System/Li
brary/PrivateFrameworks/BackgroundTaskAgent.framework/BackgroundTaskAgent at 0x3
937e000 (offset 0xfd8000)
                                                                (objfile is) [memory ob
ject "/System/Library/PrivateFrameworks/BackgroundTaskAgent.framework/Background
TaskAgent" at 0x3937e000]
```

图 10-68　info sh 的结果

直接看看 IncomingCall 的 0xce8426e – ce80000 = 0x426e 有什么，如图 10-69 所示。

```
__text:00004256 ADD     R1, PC ; selRef_initForIncomingCallWithFrame_
__text:00004258 LDR     R2, [SP,#0x24+var_14]
__text:0000425A LDR     R1, [R1] ; "initForIncomingCallWithFrame:"
__text:0000425C STMEA.W SP, {R2,R5}
__text:00004260 VSUB.F32 D0, D0, D16
__text:00004264 MOVS    R2, #0
__text:00004266 VMOV    R3, S0
__text:0000426A BLX     _objc_msgSend
__text:0000426E MOVW    R2, #(:lower16:(classRef_NSBundle - 0x4292))
```

图 10-69　newBottomBar 片段

上面的片段是 [MPIncomingPhoneCallController newBottomBar] 的一部分，也就是说 TPIncomingCallOptionsView 是 newBottomBar 的一部分，而且看样子，MPIncomingPhoneCallController 还是一个 view controller。看看 MPIncomingPhone CallController.h 里有什么玄机，如图 10-70 和图 10-71 所示。

```
43 -(id)newBottomBar;
44 -(id)newTopBar;
45 -(void)ringerChanged;
46 -(void)dismiss:(BOOL)dismiss;
47 -(void)dismiss;
48 -(void)lockBarUnlocked:(id)unlocked;
49 -(void)stopRingingOrVibrating;
50 -(void)ringOrVibrate;
51 -(void)_ringIfNecessary;
```

图 10-70　MPIncomingPhoneCallController.h 内容

```
72 -(id)backgroundView;
73 -(id)defaultBackgroundView;
74 -(void)updateLCDWithName:(id)name label:(id)label br
75 -(void)viewDidLoad;
76 -(id)initWithCall:(CTCallRef)call;
```

图 10-71　MPIncomingPhoneCallController.h 内容

可以看到，既有 newBottomBar，还有 newTopBar、backgroundView，甚至还有 stopRinging OrVibrating，看来我们已经找到了图 10-58 中靠近右侧的部分，试想一下，如果将它们一一 hook，会是什么效果呢？下面来试一试。

```
%hook MPIncomingPhoneCallController

-(id)newBottomBar
{
    return nil;
}

-(id)newTopBar
{
    return nil;
}

-(id)backgroundView
{
    return nil;
}

-(id)defaultBackgroundView
{
    return nil;
}

%end
```

编译后运行，发现没有任何效果；在每个 hook 里加上一条 NSLog 的话，你会发现没有任何输出，这是为什么呢？笔者最初接触这一块内容时也碰到了这个问题，最后依靠搜索解决了问题，搞清了原因：只有当某个进程启动时，MobileSubstrate 才会根据 filter 加载对应的 dylib；其余时间 MobileSubstrate 不会主动加载 dylib。如果这个进程在启动之后才加载一个库，那么这个库就躲过了 MobileSubstrate 的“魔爪”——这一现象称为“延迟加载”（late load）。

事实上，“/System/Library/SpringBoardPlugins/”的所有 .servicebundle 都是不可执行的库文件，在 SpringBoard 启动后才“延迟加载”，它们在 MobileSubstrate 加载完 dylib 后才会现身于内存中。也就是说，上面的这一个 hook 在得到执行时，MPIncomingPhoneCallController 类还根本不存在，自然 hook 不上了。解决这个问题的办法也很简单——等 MPIncoming PhoneCallController 存在后再 hook，不就行了？

Theos 提供的 %init 和 %group 预处理指令专治各种“延迟加载”，用法如下所示。

```
%group LateHook
%hook LateLoadedClass
-(void)method
{
    // 操作
}
%end
%end // end of LateHook

%hook SomeClass
```

```
-(void)loadLateLoadedClass
{
    %orig;
    %init(LateHook);
}
%end

%ctor
{
    %init;
}
```

解决问题的关键在于找到 loadLateLoadedClass，依据笔者的经验，执行“延迟加载”的函数名一般含有“load”关键词，如“- (void)_loadServices”、“-(Class)loadPluginNamed:(id)named”等。对于本例来说，因为延迟加载的是 SpringBoard 的 Plugin，所以在 SpringBoard 的 .h 文件中搜索关键词 plugin 和 load，很容易就可以定位到 4 个类，如图 10-72 所示。

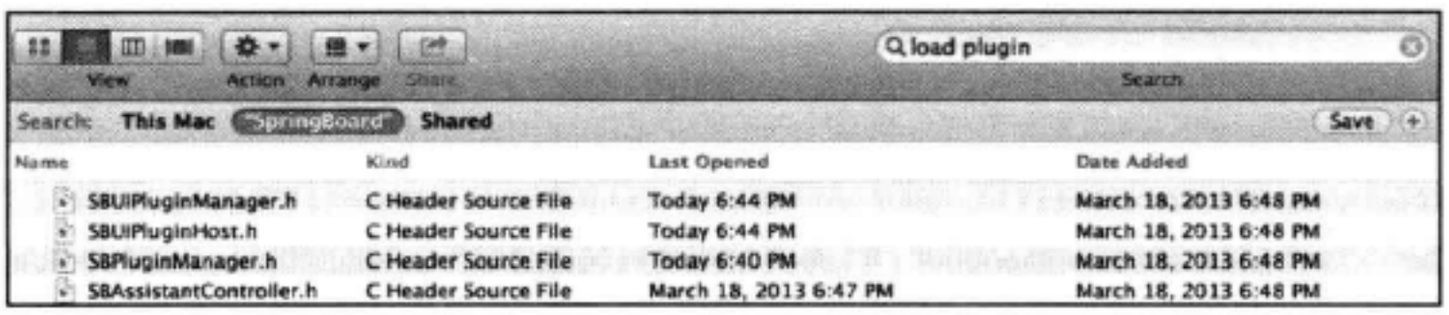


图 10-72　在 Finder 中搜索 load 和 plugin

SpringBoardPlugin 自然就是 SBPlugin，管理它们的即是 SBPluginManager。SBPluginManager.h 的内容很简单，如图 10-73 所示，似乎只干了延迟加载一件事。

```
 8 #import <XXUnknownSuperclass.h> // Unknown library
 9
10 @class NSMutableDictionary, NSString;
11
12 __attribute__((visibility("hidden")))
13 @interface SBPluginManager : XXUnknownSuperclass {
14         NSString* _pluginsDirectory;
15         NSMutableDictionary* _plugins;
16 }
17 +(id)sharedInstance;
18 -(Class)loadPluginBundle:(id)bundle;
19 -(Class)loadPluginNamed:(id)named;
20 -(void)loadAllLaunchPlugins;
21 -(id)springBoardPluginsDirectory;
22 -(void)dealloc;
23 -(id)init;
24 @end
```

图 10-73　SBPluginManager.h 的内容

模仿上面的代码示例进行调整，更改后的代码如下：

```
%group iOSHook
%hook MPIncomingPhoneCallController

-(id)newBottomBar
{
    return nil;
}

-(id)newTopBar
{
    return nil;
}

-(id)backgroundView
{
    return nil;
}

-(id)defaultBackgroundView
```

```
{
    return nil;
}

%end
%end // end of iOSHook

%hook SBPluginManager
- (Class)loadPluginBundle:(NSBundle *)bundle
{
    Class result = %orig;
    NSString *bundleIdentifier = [bundle bundleIdentifier];
    if([bundleIdentifier
    isEqualToString:@"com.apple.mobilephone.incomingcall"]) %init(iOSHook);
    return result;
}
%end

%ctor
{
    %init;
}
```

安装完成之后给自己打个电话，来电界面变成了图 10-74 这样。

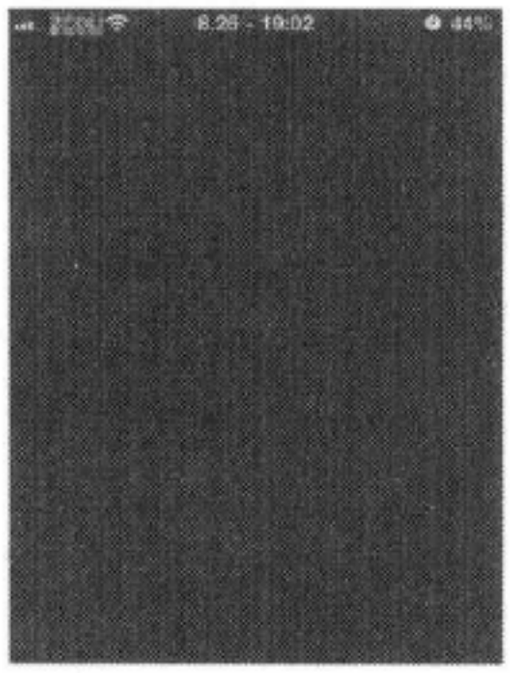


图 10-74 修改后的来电界面

显然（请允许我用这个词），来电界面上应有的视图一个也没能加载，只留了个 MPIncomingPhoneCallController 空壳。那么，如果连 MPIncomingPhoneCallController 都不加载的话，来电界面就根本不会显示，对不对？接下来看看 MPIncomingPhoneCallController 是由谁加载的，至于方法么，还是跟之前一样，用 GDB 在 MPIncomingPhoneCallController 的初始化函数末尾下断点，然后 ni 到其调用者的地址范围中去。事实上，我们在图 10-71 中已经看到了 MPIncomingPhoneCallController.h 中唯一一个带有 init 关键词的函数“ initWithCall:”，就尝试从它下手吧，操作过程如图 10-75 ～图 10-79 所示。

```
__text:0000292A loc_292A                                ; CODE XREF: -[MPIncoming
__text:0000292A                 MOV             R0, R11
__text:0000292C                 ADD             SP, SP, #0x24
__text:0000292E                 POP.W           {R8,R10,R11}
__text:00002932                 POP             {R4-R7,PC}
__text:00002932 ; End of function -[MPIncomingPhoneCallController initWithCall:]
```

图 10-75 initWithCall: 片段

一直“ ni”到 initWithCall: 执行完，进入其调用者的地址空间。0x0cc3900e – 0xcc30000 = 0x900e，如图 10-79 所示。

[MPTelephonyManager displayAlertForCall:] 应该就是我们要找的函数了。如何阻止它加载 MPIncomingPhoneCallController 呢？其实它本身就已经提供了一种方式，如图 10-80 所示。

```
[redacted]-FunMaker-5:~ root# gdb
warning: unrecognized host cpusubtype 11, defaulting to host=armv7.
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.4) (
Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
gdb$ attach SpringBoard
Attaching to process 94902.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ..........................................
.....................................................................................
```

图 10-76　用 GDB 调试 SpringBoard

```
280 EKCalendar                              - 0x9413000         dyld Y Y /System/Li
brary/SpringBoardPlugins/EKCalendar.servicebundle/EKCalendar at 0x9413000 (offse
t 0x9413000)
281 IncomingCall                            - 0xcc30000         dyld Y Y /System/Li
brary/SpringBoardPlugins/IncomingCall.servicebundle/IncomingCall at 0xcc30000 (o
ffset 0xcc30000)
282 BackgroundTaskAgent                     F 0x3937e000        dyld Y Y /System/Li
brary/PrivateFrameworks/BackgroundTaskAgent.framework/BackgroundTaskAgent at 0x3
937e000 (offset 0xfd8000)
                                                                (objfile is) [memory ob
ject "/System/Library/PrivateFrameworks/BackgroundTaskAgent.framework/Background
TaskAgent" at 0x3937e000]
```

图 10-77　info sh 的结果

```
gdb$ b *(0x2932 + 0xcc30000)
Breakpoint 2 at 0xcc32932
gdb$ c

Breakpoint 2, 0x0cc32932 in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ ni
0x01a35f90 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35f94 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35f98 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35f9c in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35fa0 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35fa4 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35fa8 in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x01a35fac in _initWithCall ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0cc3900e in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$
0x0cc39010 in PLogEnabled ()
Error while running hook_stop:
No symbol "disassembly" in current context.
gdb$ c
```

图 10-78　GDB 下断点并触发

```
__text:00009008                 LDR             R1, [R1] ; "initWithCall:"
__text:0000900A                 BLX             _objc_msgSend
__text:0000900E                 MOVS            R2, #1
__text:00009010                 MOV             R4, R0
__text:00009012                 MOVS            R0, #0
__text:00009014                 MOV             R3, R2
__text:00009016
__text:00009016 loc_9016                                ; CODE XREF: -[MPTele
__text:00009016                                         ; -[MPTelephonyManage
__text:00009016                 MOV             R1, #(selRef_requestActivatio
__text:0000901E                 STR             R0, [SP,#0x10+var_10]
__text:00009020                 MOV             R0, R4
__text:00009022                 ADD             R1, PC ; selRef_requestActiva
__text:00009024                 LDR             R1, [R1] ; "requestActivation
__text:00009026                 BLX             _objc_msgSend
__text:0000902A                 MOV             R0, #(selRef_autorelease - 0x
__text:00009032                 ADD             R0, PC ; selRef_autorelease
__text:00009034                 LDR             R1, [R0] ; "autorelease"
__text:00009036                 MOV             R0, R4
__text:00009038                 ADD             SP, SP, #4
__text:0000903A                 POP.W           {R4-R7,LR}
__text:0000903E                 B.W             j__objc_msgSend
__text:00009042 ; ---------------------------------------------------------------
__text:00009042
__text:00009042 loc_9042                                ; CODE XREF: -[MPTele
__text:00009042                                         ; -[MPTelephonyManage
__text:00009042                 ADD             SP, SP, #4
__text:00009044                 POP             {R4-R7,PC}
__text:00009044 ; End of function -[MPTelephonyManager displayAlertForCall:]
```

图 10-79 displayAlertForCall: 片段

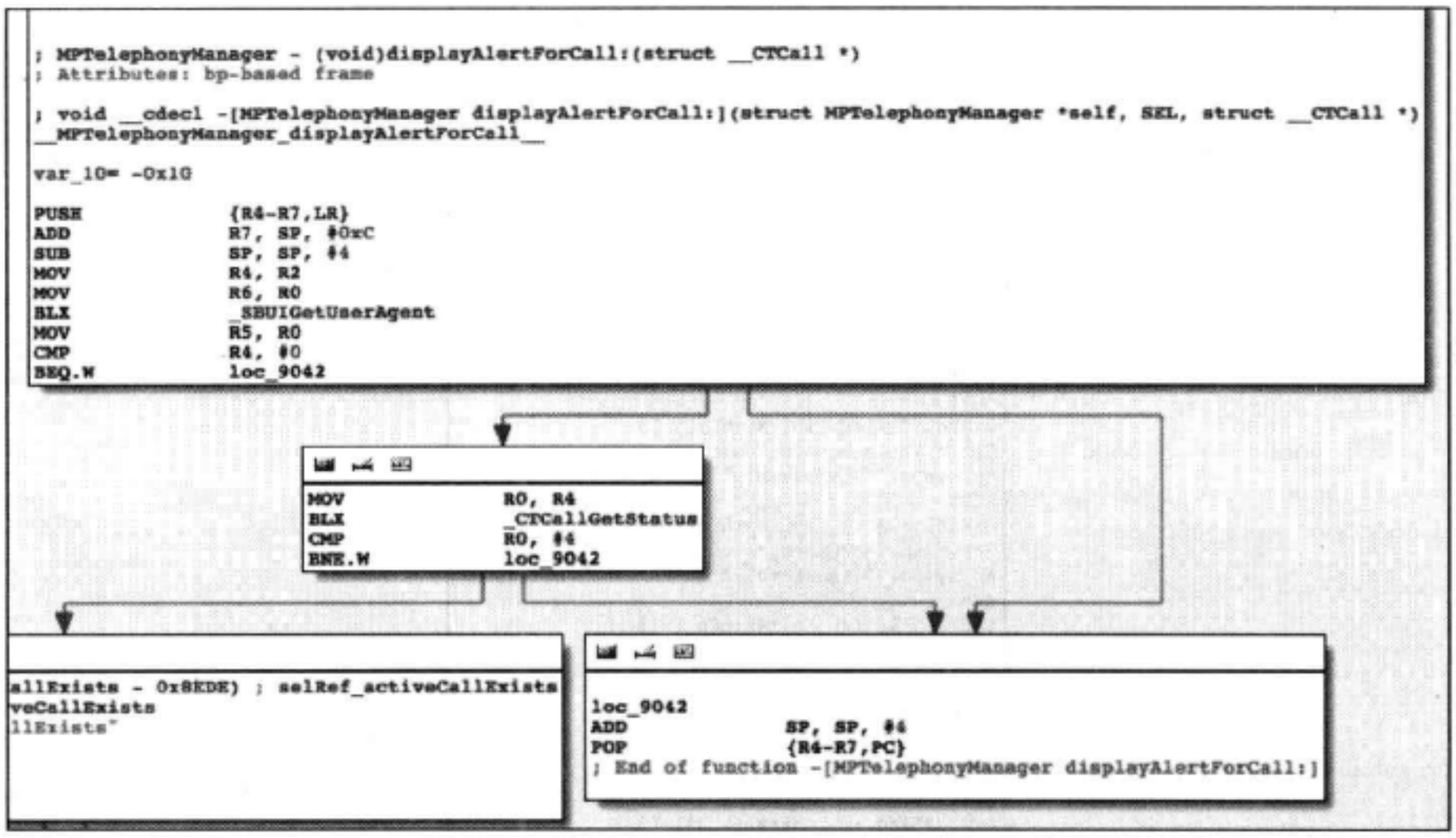


图 10-80 displayAlertForCall: 片段

注意右边的这条路，如果函数的参数是 nil，后续就不会进行一系列对参数的处理。因此可以把代码改成如下形式。

```
#import <CoreTelephony/CTCall.h>

%group iOSHook
%hook MPTelephonyManager

-(void)displayAlertForCall:(struct __CTCall *)call
```

```
{
    %orig(nil);
}
%end
%end // end of iOSHook ·

%hook SBPluginManager
- (Class)loadPluginBundle:(NSBundle *)bundle
{
    Class result = %orig;
    NSString *bundleIdentifier = [bundle bundleIdentifier];
    if([bundleIdentifier
    isEqualToString:@"com.apple.mobilephone.incomingcall"]) %init(iOSHook);
    return result;
}
%end

%ctor
{
    %init;
}
```

安装之后试试看，来电界面消失了，来电被我们拦截了。虽然目的达到了，但细心的读者会问，这个函数是“感知来电”的函数吗？打破沙锅问到底是 iOS 逆向工程中最可贵的学习精神之一！如果感兴趣，作为练习，爱动手的读者可以从 displayAlertForCall: 继续跟踪到它的“发起者”，过程不再赘述，仅提供一个结论作为参考：[MPTelephonyManager init] 注册了若干 notification observer，其中一个监听 SBIncomingCallPendingNotification 这个通知，其回调函数中调用了 [[MPTelephonyManager sharedInstance] displayAlertForCall:]，也就是说，[MPTelephonyManager init] 才是真正的感知函数，而 [MPTelephonyManager displayAlertForCall:] 则是感知函数用于处理来电的唯一函数；如果你的目标是不加选择地拦截所有来电，把 iPhone 变成 iPod，那么 hook 掉 init 是更好的选择；大多数人的目的是有选择地拦截部分来电，那么，hook 掉 displayAlertForCall: 就好。

另外，displayAlertForCall: 的参数是 CTCallRef 类型的，你反应过来了吗？也就是说，接电话和挂电话的操作可以测试了，代码如下：

```
#import <CoreTelephony/CTCall.h>

extern "C" void CTCallAnswer( struct __CTCall *call);
extern "C" void CTCallDisconnect(struct __CTCall *call);

%group iOSHook
%hook MPTelephonyManager

-(void)displayAlertForCall:(struct __CTCall *)call
{
    if (YES)
```

```
{
    // 来电界面闪一下就挂断
    %orig;
    CTCallDisconnect(call);
}
else
{
    // 来电界面不显示就挂断
    %orig(nil);
    CTCallDisconnect(call);
}

// 自己组合测试吧!
}

%end
%end // end of iOSHook

%hook SBPluginManager
- (Class)loadPluginBundle:(NSBundle *)bundle
{
    Class result = %orig;
    NSString *bundleIdentifier = [bundle bundleIdentifier];
    if([bundleIdentifier
    isEqualToString:@"com.apple.mobilephone.incomingcall"]) %init(iOSHook);
    return result;
}
%end

%ctor
{
    %init;
}
```

删除通话记录的函数在这里还不能测试：只有等这一个来电被存入通话记录，才能删除它，所以等下一节找到了通话记录更改的时机后，再测试删除通话记录的函数。

至此，拨打电话、接听电话、挂断电话、拦截电话以及删除通话记录的函数已经尽数找齐，可以开始写代码了。

10.3 编写调试代码

以上功能怎么搭配使用，完全取决于读者自己的创意；本章只简单地在电话界面上加几个按钮，调用我们逆向出的函数，作为示范。

10.3.1 编写 Tweak_PhoneOperation.xm

编写 Tweak_PhoneOperation.xm，导入 CTCallAnswer、CTCallDisconnect 等函数，并拦

截各个系统函数，代码如下：

```
/*** 每个类对应的界面元素见10.4节 ***/
#import <CoreTelephony/CTCall.h>

typedef struct __CTCall* CTCallRef;

// 接电话
extern "C" void CTCallAnswer(CTCallRef call);

// 挂电话
extern "C" void CTCallDisconnect(CTCallRef call);

// 拨打电话，第二个参数传-1
extern "C" CTCallRef CTCallDialWithID(NSString *phoneNumber, int UID);

// 删除通话记录
extern "C" void CTCallDeleteFromCallHistory(CTCallRef call);

//获取CTCallRef的号码，这个函数你自己逆向看看吧！提示：iOS感知到来电之后和显示来电之前，必
  须解析出来电号码，所以这个函数的调用位置是……
extern "C" CFStringRef CTCallCopyAddress(CFAllocatorRef allocator, CTCallRef
call);

%group iOSHook
%hook MPIncomingPhoneCallController  // 修改来电界面
%new
-(void)iOSREAnswer:(id)sender // 因为是往别人的类里添加我们的函数，所以一定要加
    namespace！以下同理
{
    CTCallRef call = MSHookIvar<CTCallRef>(self, "_incomingCall");
    CTCallAnswer(call);
}

%new
-(void)iOSREDecline:(id)sender
{
    CTCallRef call = MSHookIvar<CTCallRef>(self, "_incomingCall");
    CTCallDisconnect(call);
}

-(void)finishedAnimatingIn  // 来电界面切换动画结束
{
    %orig;

    // 还记得上面逆向出的bottomBar吗？对比它的位置，观察来电界面，你应该能猜到哪部分是
       topBar :)
    UIView *topBar = MSHookIvar<UIView *>(self, "_topBar");
    UIButton *answerButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [answerButton setTitle:@"接" forState:UIControlStateNormal];
```

```
    [answerButton setFrame:CGRectMake(6.0f, 6.0f, 36.0f, 36.0f)];
    [answerButton addTarget:self action:@selector(iOSREAnswer:)
    forControlEvents:UIControlEventTouchUpInside];
    [topBar addSubview:answerButton];

    UIButton *declineButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [declineButton setTitle:@"挂" forState:UIControlStateNormal];
    [declineButton setFrame:CGRectMake(280.0f, 6.0f, 36.0f, 36.0f)];
    [declineButton addTarget:self action:@selector(iOSREDecline:)
    forControlEvents:UIControlEventTouchUpInside];
    [topBar addSubview:declineButton];
    [topBar setUserInteractionEnabled:YES];
}
%end

%hook MPTelephonyManager  // 拦截来电
-(void)displayAlertForCall:(CTCallRef)call
{
    NSString *number = [(NSString *)CTCallCopyAddress(kCFAllocatorDefault,
    call) autorelease];
    if ([number isEqualToString:@"110"]) return;
    else %orig;
}
%end
%end // end of iOSHook

%hook SBPluginManager // late hook
- (Class)loadPluginBundle:(NSBundle *)bundle
{
    Class result = %orig;
    NSString *bundleIdentifier = [bundle bundleIdentifier];
    if ([bundleIdentifier
    isEqualToString:@"com.apple.mobilephone.incomingcall"]) %init(iOSHook);
    return result;
}
%end

%ctor
{
    %init;
}

@interface DialerLCDView : UIView
@property(retain, nonatomic) UILabel* numberLabel;
@end

@interface DialerView : UIView
@property(retain, nonatomic) DialerLCDView* lcdView;
@end
```

```
static DialerLCDView *lcdView;

%hook DialerController  // 修改拨号界面
%new
-(void)iOSRECall:(id)sender
{
    UILabel* numberLabel = [lcdView numberLabel];
    NSString *number = [numberLabel text];
    if ([number length] != 0) CTCallDialWithID(number, -1);
    else [numberLabel setText:@"请输入一个号码~"];
}

-(void)loadView
{
    %orig;

    DialerView *dialerView = MSHookIvar<DialerView *>(self, "_dialerView");
    lcdView = [dialerView lcdView];

    UIButton *callButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [callButton setTitle:@"拨" forState:UIControlStateNormal];
    [callButton setFrame:CGRectMake(6.0f, 6.0f, 36.0f, 36.0f)];
    [callButton addTarget:self action:@selector(iOSRECall:)
    forControlEvents:UIControlEventTouchUpInside];
    [lcdView addSubview:callButton];
    lcdView.userInteractionEnabled = YES;
}
%end

@interface RecentCall : NSObject
-(id)underlyingCTCalls;
@end

@interface RecentsViewController : UITableViewController
-(id)callAtTableViewIndex:(int)tableViewIndex;
@end

%hook RecentsViewController  // 修改通话记录界面
%new
-(void)iOSREDelete:(UIButton *)sender
{
    id call = [self callAtTableViewIndex:sender.tag];  // 这里的call是RecentCall
    或RecentMultiCall类型，均有underlyingCTCalls这个函数
    NSArray *ctCallArray = [call underlyingCTCalls];
    for (id ctCall in ctCallArray)
      CTCallDeleteFromCallHistory((CTCallRef)ctCall);
}

-(UITableViewCell *)tableView:(id)view cellForRowAtIndexPath:(NSIndexPath *)
indexPath
{
```

```
    UITableViewCell *result = %orig;
    UIButton *deleteButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    deleteButton.tag = indexPath.row;
    [deleteButton setTitle:@"删" forState:UIControlStateNormal];
    [deleteButton setFrame:CGRectMake(180.0f, 10.0f, 30.0f, 30.0f)];
    [deleteButton addTarget:self action:@selector(iOSREDelete:) forControlEvents:
UIControlEventTouchUpInside];
    [result addSubview:deleteButton];
    return result;
}
%end

@class PhoneCall;
@interface InCallLCDView : UIView
@end

static PhoneCall *phoneCall;
static UIButton *declineButton;

%hook InCallController  // 修改拨出电话界面
%new
-(void)iOSREDecline:(id)sender
{
    CTCallRef call = MSHookIvar<CTCallRef>(phoneCall, "_call");
    CTCallDisconnect(call);
}

-(void)_callStatusChanged:(id)changed  // 此函数会多次得到调用，hook它主要是为了在有多
    个来电的情况下定位当前通话的来电，*可能还没有考虑到所有情况，如三方通话*
{
    %orig;
    phoneCall = changed;
    InCallLCDView *lcd = MSHookIvar<InCallLCDView *>(self, "_lcd");
    if (declineButton.superview)  // 不要重复贴图，先把上一张给撕掉再贴
        [declineButton removeFromSuperview];
    declineButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
    [declineButton setTitle:@"挂" forState:UIControlStateNormal];
    [declineButton setFrame:CGRectMake(280.0f, 6.0f, 36.0f, 36.0f)];
    [declineButton addTarget:self action:@selector(iOSREDecline:)
    forControlEvents:UIControlEventTouchUpInside];
    [lcd addSubview:declineButton];
}
%end
```

10.3.2 编辑 control

编辑 control，填写 Name 和 Description，代码如下：

```
Package: com.naken.phoneoperation
Name: PhoneOperation
```

```
Depends: mobilesubstrate, firmware (>= 6.0)
Version: 1.0
Architecture: iphoneos-arm
Description: 电话操作
Maintainer: snakeninny
Author: snakeninny
Homepage: http://iosre.com
Section: Tweaks
```

这个 tweak 只在 iOS 6 上做了测试，在 iOS 5 上应该只做简单更改就可使用。

10.3.3　编辑 PhoneOperation.plist

指定 Filter 为 com.apple.mobilephone 和 com.apple.springboard，如图 10-81 所示。

Key	Type	Value
▼Root	Dictionary	(1 item)
▼Filter	Dictionary	(1 item)
▼Bundles	Array	(2 items)
Item 0	String	com.apple.mobilephone
Item 1	String	com.apple.springboard

图 10-81　PhoneOperation.plist 的内容

10.3.4　编辑 Makefile

编辑 Makefile，指定 iOSRE _FILES 以及 iOSRE_FRAMEWORKS 等参数。

```
ARCHS = armv7
TARGET = iphone:6.1:4.3

include theos/makefiles/common.mk

TWEAK_NAME = PhoneOperation
iOSRE_FILES = Tweak_PhoneOperation.xm
iOSRE_FRAMEWORKS = CoreTelephony UIKit

include $(THEOS_MAKE_PATH)/tweak.mk
after-install::
    install.exec "killall -9 MobilePhone" # 因为下面那行，本行并不必要
    install.exec "killall -9 SpringBoard"
```

10.3.5　编译 + 打包 + 安装

在 MakeFile 里加上“ THEOS_DEVICE_IP = yourDeviceIP”然后 make package install 即可，不再赘述。

10.4　调试电话操作插件

在调试之前，需要卸载与电话操作相关的其他 tweak（如 KuaiDial、搜狗号码通、360 手

机卫士等），或用 SBSettings 将其禁用，以保证测试不因 tweak 冲突而受影响。

1. 拨打电话

打开电话程序后，会发现原来的电话界面上会多出新的按钮：输入电话号码时会显示一个“拨”，拨出之后又会显示一个“挂”，如图 10-82 和图 10-83 所示。



图 10-82 拨号界面

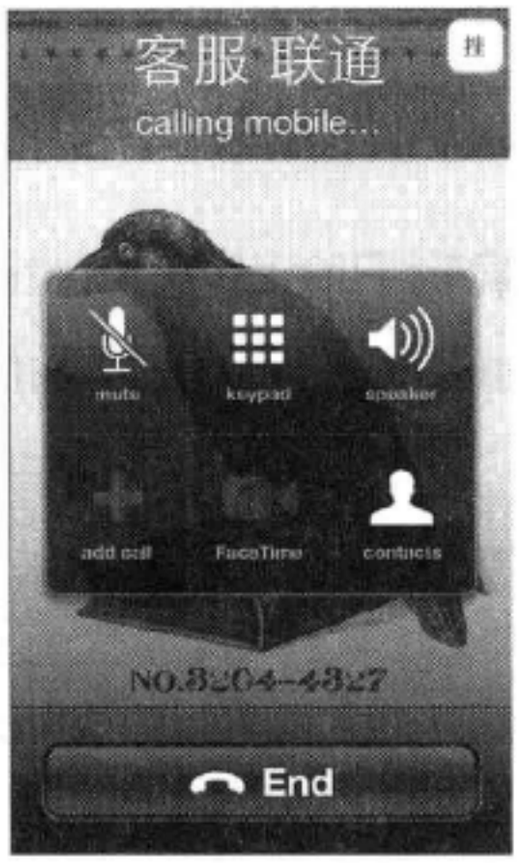


图 10-83 拨出界面

2. 接听电话

来电时会显示“接”和“挂”，如图 10-84 所示。

3. 挂断电话

可以按“挂”来结束电话，如图 10-85 所示。

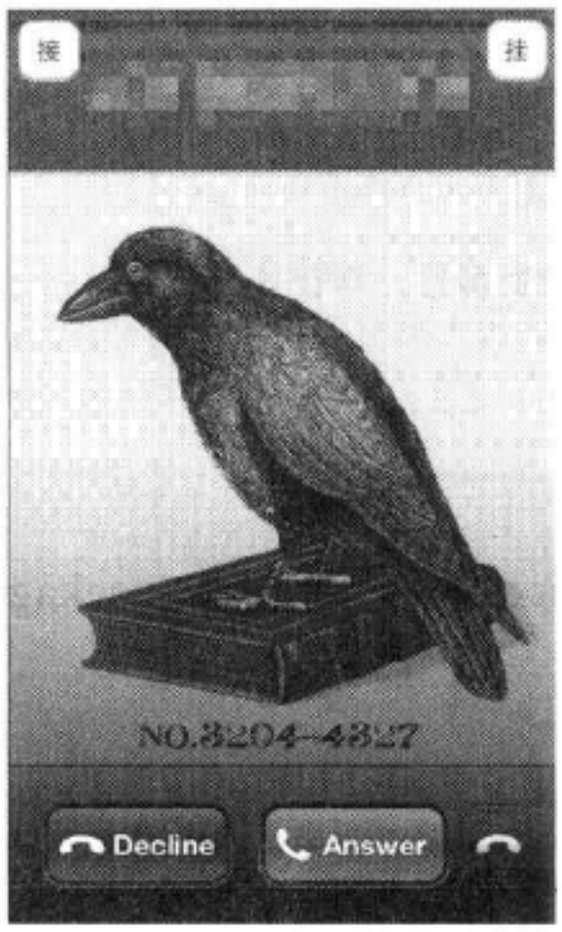


图 10-84 来电界面

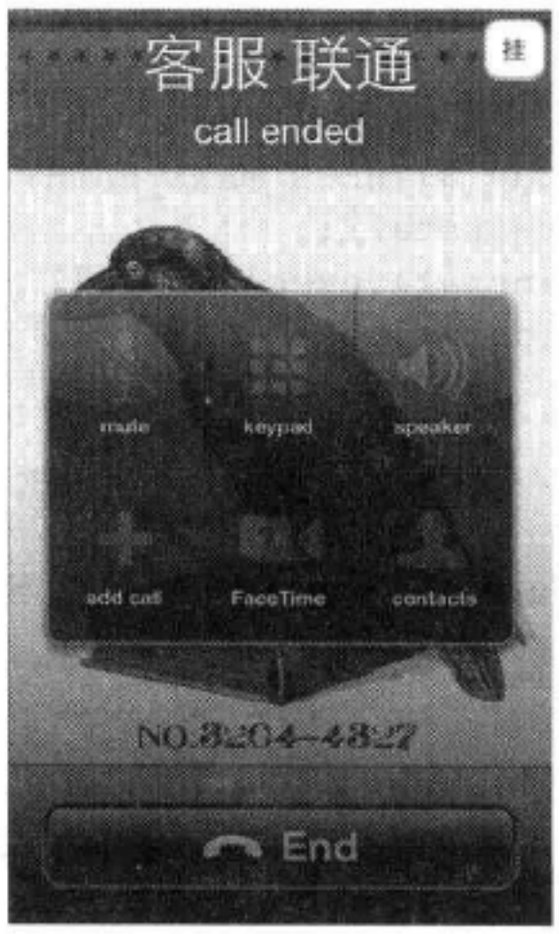


图 10-85 挂断电话

4. 删除通话记录

从通话记录里可以直接删除某条记录，如图 10-86 所示。



图 10-86　通话记录界面

10.5　代码结果整理

本章的实例是针对 iOS 系统功能的，虽然逆向整体思路不变，但复杂程度和逆向难度都要大过前面的 3 个实例。为了逆向出几个常用的电话操作，整体思路如下。

1）用 Reveal 定位拨出电话的 Call 按钮，然后用 tweak 来打印出它的响应函数，再通过 IDA 分析这个函数，直到定位 CTCallDialWithID 函数。

2）Reveal 没有能够成功捕获来电界面，但我们还是可以通过搜索关键词的方式定位接听电话的 Answer 按钮，接下来的套路跟 1）几乎一样。

3）在解决 2）时“误打误撞”找到了挂断电话的线索，一石二鸟，一箭双雕。

4）对于拦截电话的功能，我们剖析了 iOS 对来电的处理流程的一小部分，从处理链的末端倒推到了 SpringBoard 感知来电的函数，通过对其回调函数进行 hook 达到了目的。

5）删除通话记录走的是 UITableView 删除 UITableViewCell 的标准模式，可以直接定位到对应的上层函数，然后用 IDA 和 GDB 分析，这样就很容易实现了。

10.6　小结

作为 iOS 的核心功能之一，与电话相关的操作不是三言两语能够说清的，本章也只是对最常用的一些功能点到即止；不同于前 3 章的一个 binary 解决战斗，本章的 5 个电话操作分布在 MobilePhone.app、TelephonyUI.framework、IncomingCall.servicebundle 和 SpringBoard.app 中，MobilePhone.app 只是一个呈现给用户的外壳的一部分，难怪“占地面积”那么小。这些模块协同工作，逆向难度更大，经常需要 grep 整个 /System 目录才能找到下一个线索。当然，翻越更高的山峰，等待我们的将是更美的风景——你已经可以自己制作电话防火墙了，自己动手、丰衣足食的满足感是世界上最幸福的感觉之一，不是吗？

越狱开发一览

关于苹果玩的这一套软硬件通吃生意，业界已经讨论了很多遍，其封闭性可谓路人皆知。但其实这扇紧闭的大门背后有不计其数的宝藏在等待着开发者们让它们重见天日——这些宝藏就是苹果内部使用的所有 API，而让它们重见天日的过程就是越狱。这些 API 中的大多数都是使用 Objective-C——一门动态性强、语义清晰的语言——编写的。越狱之后，为了进一步扩展 iDevice 的功能，绝大多数人都会安装 CydiaSubstrate，它使我们能够在其他进程中运行自己编写的代码。它的意义非比寻常——我们不但破除了苹果的限制，还能够以其人之道，还治其人之身！因为越狱 iOS 开发程序与普通的 App Store 开发不尽相同，它需要你去摸索 API 的用法，把代码注入别的进程，修改现有的类，最终达到改变系统行为的目的。

iOS 调用的 API 可以分为 4 类：framework 使用的 Objective-C API、App 使用的 Objective-C API、C API 和 JavaScript API，其中 framework 层面的 API 是最容易使用的。一般情况下，一个程序的设计、代码、文档只为少数人所有，但经过编译的 Objective-C 二进制文件中含有所有的类、协议、方法和实例变量信息，“class-dump”大家族的工具可以把这些信息导出来，形成头文件，供那些好奇的开发者研究并调用。App 使用的 API 可以通过同样的方式导出，但不能直接调用。这时，我们可以把代码注入到对应的进程里，然后调用 Objective-C 的运行时函数 objc_getClass 来获取一个类的对象，进而调用类方法。C API 就要复杂一点，没有 class-dump 这样的工具把 C 函数的参数和数据结构给导出来，我们能找到的只有导出函数的函数名；但是，利用 OSX 自带的反汇编工具 otool，结合一些 ARM 汇编语言的知识，我们可以手动分析，还原 C 函数的原貌，这个过程比分析 Objective-C 方法要复杂得多。幸运的是，iOS 中的 C 实现与 OSX 是部分相通的，有一些 C 函数的头文件可以从 OSX 的 SDK 中找到参考，还有一些是开源的。JavaScript API 通常只是 Objective-C API 的封装，供 iTunes、App Store、iCloud 和 iAd 等应用中的网页调用。

通过逆向工程得知的 API 通常需要注入对应的进程才能调用，因为只有这些进程中含有 API 的实现。进程注入可以通过 DYLD_INSERT_LIBRARIES 方式实现，但这种方式提供的防崩溃保护不够强，很容易导致设备白苹果，只有重刷系统才能解决。更好的替代方案是

CydiaSubstrate，它为进程注入制定了一套标准，并引入了更加安全的防崩溃保护机制。安装 CydiaSubstrate 之后，只需要把一个 dylib 放到 /Library/MobileSubstrate/DynamicLibraries 下即可，CydiaSubstrate 会自动在每个进程启动时尝试把这个 dylib 加载进去；我们可以进一步通过编写一个 plist，指定 dylib 加载的对象。一旦我们的代码得到注入，就可以监听事件，调用内部 API，做这个进程本身能够做的任何事。这种注入方式可以用在任何进程上：系统 App、App Store App、iOS 上的窗口管理器 SpringBoard、UI 服务（如 MailComposer）和守护进程（如 mediaserverd）。值得一提的是，我们的代码只在注入的进程内部生效，如果要使其跨进程作用，则需要用到 sockets、fifos、mach ports 和 shared memory 等技术。

从我们能够通过简单方便的 CydiaSubstrate 更改现有的代码开始，我们能做的事就多了起来。因为 Objective-C 方法调用都是在运行时决定的，而 Objective-C 的运行时函数提供了修改类和方法的 API，所以这个过程就比较直观了，大体是通过 class_addMethod、class_getInstanceMethod、method_getImplementation 和 method_setImplementation 这 4 个运行时函数来实现 hook 的功能。这个过程并不困难，但比较繁复，许多工具就是为了自动化这个过程而存在的，其中比较简单的是 CydiaSubstrate 提供的 MSHookMessage 函数，它有 4 个参数，分别是：一个类、一个你要 hook 的方法名、一个新的实现和一个原始实现。现在越狱社区又出现了一种更简单的 Logos 预处理工具，它引入了专为 hook 设计的语法，因此在广大越狱开发者中流行了起来。Logos 语法与 Objective-C 语法十分类似，把 @implementation ClassName ... %end 的首尾替换掉，改成 %hook ClassName ... %end，把 [super...] 替换为 %orig 就行了。简单地更改系统功能往往只需要 hook 一两个独立函数，把它们的实现改掉就可以了；但多数情况下我们需要组合 hook 好几个函数并协调它们之间的调用关系。因为 iOS 的大部分功能是用 Objective-C 写的，所以大多数 tweak 仅用上面提到的语法就够了，但一旦涉及了更底层的 C 函数，hook 的方法就要复杂许多，一般需要重写一段 ARM 汇编指令，当然这种方法的危险系数极高，不建议一般开发者尝试。好在 CydiaSubstrate 也提供了一个简单的 API，即 MSHookFunction，调用者只需要把需要 hook 的函数及其新的实现作为参数传进去就可以了。越狱社区提供的这些工具把 hook 操作复杂的一面给抽象化了，封装成几个简单的接口供开发者使用，从而使我们能够把精力集中在 tweak 的编写上。

把这些越狱社区独有的技术同 App Store 开发的普遍性技术结合起来，我们得到了一套用法灵活、功能强大的工具集，利用这套工具能够打造的功能也将是前所未有的。

Ryan Petrich
牧“码”人，Activator 之父

Jailbreaking for Developers, An Overview

Much has been said about Apple's closed approach to selling devices and running an app

platform. But what few know is that behind closed doors there's a massive ecosystem of libraries and hardware features waiting to be unlocked by developers. All of the APIs Apple uses internally to build their services, applications, and widgets are available once the locks are broken via a process called jailbreaking. Most of them are written in Objective-C, a dynamic language that provides very rich introspection capabilities and has a culture of self-describing code. Further tearing down barriers, most people install something called CydiaSubstrate shortly after jailbreaking, which allows running custom code inside every existing process on the device. This is very powerful—not only have we broken out of the walled garden into the rest of the forest, all of the footpaths are already labeled. Building code that targets jailbroken iOS devices involves unique ways of inspecting APIs, injecting code into processes, and writing code that modifies existing classes and finalized behaviors of the system.

The APIs implemented on iOS can be divided into four categories: framework-level Objective-C APIs, app-level Objective-C classes, C-accessible APIs and JavaScript-accessible APIs. Objective-C frameworks are the most easily accessible. Normally the structure of a component is only accessible to the programmer and those the source code or documentation have been made available to, but compiled Objective-C binaries include method tables describing all of the classes, protocols, methods and instance variables contained in the binary. An entire family of "class-dump" tools exists to take these method tables and convert them to header-like output for easy consumption by adventurous programmers. Calling these APIs is as simple as adding the generated headers to your project and linking with the framework or library. The second category of app internal classes may be inspected via the same tools, but are not linkable via standard tools. To get to those classes, one has to have code injected into the app in question and use the Objective-C runtime function objc_getClass to get a reference to the class; from there, one can call APIs via the headers generated by the tool. Inspecting C-level functions are more difficult. No information about what the parameters or data structures are baked into the binaries, only the names of exported functions. The developer tools that ship with OS X come with a disassembler named "otool" which can dump the instructions used to implement the code in the device. Paired with knowledge of ARM assembly, the type information can be reconstructed by hand with much effort. This is much more cumbersome than with Objective-C code. Luckily, some of the components implemented in C are shared with OS X and have headers available in the OS X SDK, or are available as open-source from Apple. JavaScript-level APIs are most often facades over Objective-C level APIs to make additional functionality accessible to web pages hosted inside the iTunes, App Store, iCloud and iAd sections of the operating system.

Putting the APIs one has uncovered to use often requires having code run inside the process where their implementations are present. This can be done using the DYLD_INSERT_LIBRARIES environment variable on systems that use dyld, but this facility offers very few provisions for crash

protection and can easily leave a device in a state where a restore is necessary. Instead, the gold standard on iOS devices is a system known as Cydia Substrate, a package that standardizes process injection and offers safety features to limit the damage testing new code can do. Once Cydia Substrate is installed, one needs only to drop a dynamic library compiled for the device in /Library/MobileSubstrate/DynamicLibraries, and substrate will load it automatically in every process on the device. Filtering to only a specific process can be achieved by dropping a property list of the same name alongside it with details on which process or grouping of processes to filter to. Once inside, one can register for events, call system APIs and perform any of the same behaviors that the process normally could. This applies to apps that come preinstalled on the device, apps available from the App Store, the window manager known as SpringBoard, UI services that apps can make use of such as the mail composer, and background services such as the media decoder daemon. It is important to note that any state that the injected code has will be unique to the process it's injected into and to share state mandates use inter-process communication techniques such as sockets, fifos, mach ports and shared memory.

Modifying existing code is where it really starts to get powerful and allows tweaking existing functionality of the device in simple or even radical ways. Because Objective-C method lookup is all done at runtime and the runtime offers APIs to modify methods and classes, it is really straightforward to replace the implementations of existing methods with new ones that add new functionality, suppress the original behavior or both. This is known as method hooking and in Objective-C is done through a complicated dance of calls to the class_addMethod, class_getInstanceMethod, method_getImplementation and method_setImplementation runtime functions. This is very unwieldy; tools to automate this have been built. The simplest is Cydia Substrate's own MSHookMessage function. It takes a class, the name of the method you want to replace, the new implementation, and gives back the original implementation of the function so that the replacement can perform the original behavior if necessary. This has been further automated in the Logos Objective-C preprocessor tool, which introduces syntax specifically for method hooking and is what most tweaks are now written in. Writing Logos code is as simple as writing what would normally be an Objective-C method implementation, and sticking it inside of a %hook ClassName … %end block instead of an @implementation ClassName … %end block, and calling %orig() instead of [super …]. Simple tweaks to how the system behaves can often done by replacing a single method with a different implementation, but complicated adjustments often require assembling numerous method hooks. Since most of iOS is implemented in Objective-C, the vast majority of tweaks need only these building blocks to apply the modifications they require. For the lower levels of the system that are written in C, a more complicate hooking approach is required. The lowest level and most compatible way of doing so is to simply rewrite the assembly instructions of the victim function. This is very dangerous and does not compose well when many developers are modifying

the same parts of the system. Again, CydiaSubstrate introduces an API to automate this in form of MSHookFunction. Just like MSHookMessage, one needs only to pass in the target function, new replacement implementation function, and it applies the hook and returns the old implementation that the new replacement can call if necessary. With the tools the community has made available, the details of the very complex mechanics of hooking have been abstracted and simplified to the point where they're hidden from view and a developer can concentrate on what new features they're adding.

Combining these techniques unique to the jailbreak scene, with those present in the standard iOS and OS X development communities yields a very flexible and powerful tool chest for building features and experiences that the world hasn't seen yet.

Ryan Petrich
Code Wrangler, Father of Activator

沙箱逃脱

出于安全考虑，也为了让每个 App 运行在自己的独立空间里，iOS 引入了称为“沙箱”的安全体系。沙箱会拦截文件访问、网络套接字、Bootstrap 服务，以及对子进程的 spawn。越狱操作对沙箱作出了适量修改，使所有进程都能够加载 CydiaSubstrate，但为了用户的隐私安全，大多数沙箱限制仍保留原样。

在每一版 iOS 中，苹果公司都会增强沙箱的作用。当我们的 tweak 需要跨进程访问数据或向硬盘写数据时，沙箱会给我们的操作带来限制。绕过这些限制的方案之一是视 tweak 运行在哪些进程中而定，选择一个这些进程都能访问的路径或文件，达到共享数据的目的。这是一种常规的方法，但一旦苹果公司再次收紧沙箱的限制，这种方法就可能失效，比如 iOS 7 中已经不存在一个所有进程皆可读写的路径了。另一种更好的方案是把类似操作放在权限高的进程，如 SpringBoard、backboardd 甚至是我们自己编写的守护进程里去完成，我们的 tweak 所在的进程只需要把受限制的操作丢给这些高权限的进程，然后坐等结果就可以了。这么做的好处是，即使沙箱的限制越来越严，只要 tweak 所在的进程能够与高权限进程通信，tweak 也能够正常运行。

奇怪的是，在 iOS 7 里，苹果公司甚至限制了 App Store App 所能访问的服务[⊖]，导致几乎所有 App Store App 的进程间通信都失效了。RocketBootstrap 应运而生，它既破除了上面提到的访问限制，又较好地保留了沙箱的防护。向 RocketBootstrap 注册的服务可以被全系统进程访问，包括那些沙箱限制非常严格的进程；RocketBootstrap 是一个单独的程序，随着苹果限制规则的变化，我会不断调整 RocketBootstrap 的代码使其能够正常运行。

Ryan Petrich

⊖ 即能够通信的进程。——译注

Evading the Sandbox

As a security measure and to keep apps on the device from sharing data or interfering with each other, iOS includes a security system known as the sandbox. The sandbox blocks access to files, network sockets, bootstrap service names, and the ability to spawn subprocesses. Part of the jailbreaking process involves modifying the sandbox so that all processes can load Cydia Substrate, but much of the sandbox is left intact to respect the security and privacy of the user's data.

With each new release, Apple further improves the sandbox to improve privacy and security. When building extensions or tweaks that need to share information across processes or persist data to disk, this can be restrictive. One approach is to survey the sandbox restrictions that exist on the processes where the extension is to be run, and choose file paths and names based on them. This is common, but can leave oneself stranded when Apple tightens the tourniquet and as of iOS 7 there is no location that all processes can read and write successfully. A better approach is to do all of the interesting work inside a privileged process such as SpringBoard, backboardd or even a manually created launch daemon of your own. Child processes can then send work to the privileged service. This ensures that as the sandbox tightens, your extension will still behave properly as long as it can communicate with the service.

Oddly enough, as of iOS 7 Apple has also decided to limit which services an app store process may query. This makes nearly all forms of inter-process communication ineffective on iOS, outside of the well-defined static services that Apple has designated. RocketBootstrap was created as a way around this that simultaneously allows additional services to be registered and respects the security and privacy of the user's data. Services registered with RocketBootstrap are made globally accessible even in spite of very restrictive sandbox rules and it will serve as a single project that needs updating as the rules change.

Ryan Petrich

编写 tweak——新时代的 hacking

怎么说我都不算是一个高产的程序员，不过我的思维方式完全是程序员式的，因为在写程序时，我已经证明了自己具备解决问题的能力。虽然我以个人开发者的身份发布了几款 tweak，但是我还有很多想法没来得及实现，因为我的时间大都花在研究 iOS 内部构造上了，而且作品越多，意味着花在它们上面的时间就越多，所以……

通过使用前辈们创造的分析工具，参考 iOS 社区共享的文档和例子，我对 iOS 的了解还算深入。因为 Cocoa 和 Objective-C 语言的本质，我们有机会观察分析 iOS 及其软件的工作原理，这些都是编写 tweak 的先决条件。很多越狱 iOS 用户越狱自己设备的根本原因就是要安装那些方便好用的 tweak，这也是我们鼓励 iOS 程序员来研究 / 开发 tweak 的原因之一，而绝不是为了安装盗版软件。随着新 tweak 的出现，这种越狱生态系统也在不断蓬勃发展，同时也促成 iOS 上发生越来越多的改变。

在传统 App Store 开发之外，越狱开发提供了一种非常独特的方式来让开发者们表达自己的想法。在 MobileSubstrate 出现之前，tweak 的概念十分抽象，而现在它已经有了非常具体的定义：调试分析现有的软件，然后重写它的某个部分，并尽可能减小对其他部分的影响。重写所造成的改变并不是永久的，即使这种改变破坏了原有软件，也可以很方便地修复。用一句话概括就是：tweak 重新定义了软件的功能。在编程的历史中，这种概念前所未有。在过去，做游戏修改器、给软件打补丁是很常见的现象，但随着 iOS 和越狱平台的出现，借助 MobileSubstrate 的力量，我们可以把对软件的修改精确到函数级别，这是越狱开发的一大特色。

探索事物的工作原理是一件充满乐趣的事情，开发者编写 tweak 就是一例。在编写 tweak 之前，我们最先面对的挑战就是寻找灵感，而灵感往往是在我们对 iOS 进行重重分析之后才产生的。初学者可以从很多现成开源工程中学习，但很多简单的想法都已得到实现。如果想要原创 tweak，那么首先你要熟悉 Cydia 上现有的 tweak，然后分析它们的实现方法，直到你理清所有逻辑，并且有把握编写相同功能的 tweak。不断地重复这个过程，等你熟悉这个套路之后，也就基本具备把灵感变成 tweak 的能力了。

投入到 iOS 越狱开发并不是一件很无助的事情，因为许多最常用的工具和资源都是免费的：Apple 官方文档编写十分详尽，iphonedevwiki 上有很多有价值的信息，class-dump 可以导出详尽的头信息。而且调试和反汇编工具也有免费的，如 IDA demo 和 GDB，这些都是编写 tweak 的利器。一名合格的 App Store 开发者完全可以从这个角度入手，开始越狱开发之路；不过如果是一名纯菜鸟，你的日子可能就不那么好过了，我建议你先完整学习 Objective-C 和 Cocoa 的概念及原理，之后再考虑进入越狱开发这个领域。虽然前期的概念学习过程需要花费较长时间，但这是新手必须迈过的坎。小小提醒：Objective-C 语言的面向对象特性可不是那么好懂的。虽然 tweak 的创意构思不设门槛，普通用户即可轻松完成，但实现的过程就需要程序员往里面投入大量时间了。在这些年的经历中，我发现很多初学者都不够耐心，他们有很好的想法，但是太急于求成，一旦发现完成 tweak 所需的时间多于预期，就打退堂鼓了。殊不知，Cydia 中最出色的 tweak，往往都是由那些有耐心、能坚持的开发者完成的。

比如 Activator，它是迄今为止我眼中当之无愧的 tweak 界无冕之王，但它的创意并不高深，就是把常见的手势使用范围扩展至全系统（rpetrich 不仅将它开源，还为第三方应用提供了接口，大将风范展露无疑）。虽然这个概念看似每个 iOS 用户都想得到，但 Activator 是 iOS 越狱社区最顶尖的开发者之一 rpetrich（真名 Ryan Petrich）经年累月完成的作品，凝结了他智慧的结晶，难度可想而知。Activator 饱含 rpetrich 对 iOS 中最晦涩的 SpringBoard 和 backboard 的深入研究，如果一个 tweak 开发者想要拿一个 tweak 作为自己努力的终极目标，那么 Activator 再合适不过了。千万不要以为 Activator 完成的工作难度不大，如果你觉得自己足够强，可以试试看能不能写出同样功能的 tweak。在此向 rpetrich 致敬！

作为 TheBigBoss 源的管理员，简单地说，我的工作是积极影响、向上引导初学者。对于这些人来说，我往往是他们进入越狱开发社区后碰到的第一个圈内人。我希望所有越狱开发者都能通过各种渠道，比如聊天工具、论坛、微博等，进行交流，但很多情况下还是有些开发者不善交流，独自工作，而这时我的作用就凸显出来了：他们往 TheBigBoss 源提交作品，就必须跟我交流，而我也会从技术角度评价他们 tweak 的优劣。他们或许不熟悉这个圈子的各种规则，于是我就会对他们进行普及，让他们少走些弯路。我愿意尽可能向他们提供帮助，让他们感觉自己是这个大圈子中的一员。我为自己的所作所为感到骄傲，因为我为越狱社区的文化建设贡献了一份力量。我衷心希望越狱社区能蓬勃发展，各种创意层出不穷，各类人才百花齐放。

当你碰到难题时，不要气馁。在越狱社区里，有些程序员已身经百战，也有些才刚刚上路，不管你属于哪一类，都应该以严肃、认真、负责的态度对待自己的作品，水到方能渠成，揠苗焉能助长？你要设立的目标不应仅仅是在 Cydia 上发布什么软件，而是分享你的经验，让大家受益于此。当然，这是站在非商业角度来说的。如果你想要靠卖 Cydia 软件赚钱，就尽可能广泛地测试你的软件，确保它能正常工作。作为一个负责任的 tweak 开发者，你要记住，你是在改动别人的软件，还要和其他的 tweak 兼容，所以要慎之又慎。

tweak 开发是新时代的 hacking。你已有足够的理由来说服自己加入 tweak 开发的大部队，

越狱社区也需要 tweak 来保持旺盛的生命力。加入我们，从中学习，好好努力，保持耐心，你一定会感到由衷的快乐。

Optimo

Cydia 中最知名默认软件源 TheBigBoss 的管理员

Tweaking is the new-age hacking

I am not a prolific programmer by any means. I have a programmer's mind, and I have proven in my days I am capable of writing working solutions. I have a few tweaks in my name, and more ideas to be realized. Creating more has been about having more free time. However, my time has been spent becoming familiar with iOS-internals, because I find that I am a good learner. I have a fair understanding due to the tools we have available, made by great programmers before our time, and from documentation and examples shared by the community. Because of the nature of Cocoa and Objective-C, we can take a great adventure and introspection into the workings of third-party software, and Apple's operating system. This provides a foundation and skills for making tweaks. We want to encourage tweak making because it has been the driving initiative behind the audience that wants to have jailbroken devices, besides for the groups that wish to only have a jailbreak for pirating apps and games. The growth of this jailbreak ecosystem has gone with the proliferation of new tweaks, ever pushing the boundaries of modification while maintaining a safe environment for the end-users.

The jailbreak development scene has given a unique opportunity to developers to express themselves in a new way. In the days before Mobile Substrate, apps and games were not tweaked. This is a new concept; examining and debugging existing software and then rewriting portions of it with the least invasive tools available, the changes are nonpermanent and for the most part free of worry for breaking something with any lasting effect. Tweaks allow for a redefining of how software works and behaves. We do this with tweaks, and there has really been nothing like it before in the world of programming, even on the PC. There were opportunities throughout previous decades to make game patches, hacks and so forth, but it's only with the emergence of the audience of jailbreakers and iOS that we find our unique situation. Only recently has it become feasible to make small adjustments to existing UI and modify how things work without requiring the replacement of whole parts of the code - Mobile Substrate allows careful targeting of methods and functions.

It's a lot of fun to discover how things work, and tweak making is the embodiment of that fun time for developers. One of the challenges for tweak making is coming up with new ideas to

create, and sometimes these ideas only arise after studying the internals in some detail. If you make tweaks as a hobby, and not as a profession, you're free to do as you wish and to focus on projects that interest you. For new tweak makers, there's quite a lot of existing projects to learn from, but a lot of the easier projects have already been realized. Creating new original ideas that are unique is a task of being familiar with the available tweaks on Cydia, and then going to work discovering how the internal parts work, debugging and testing until you have a diagram or picture in your mind how it's put together. When you reach a near complete understanding, you are primed to tackle whatever challenge you make for yourself.

Some of our greatest tools and resources are free: Apple's own documentation is excellent, and for tweak makers we have a wiki and the opportunity to use class-dump to examine what methods are exposed for hooking inside the target app or process. Debugging and disassembly tools that vary from free to paid, all can be great assets for tweak makers. A well-studied programmer with some prior experience with standard projects will be in a good position to continue learning from these materials. To the contrary, a newcomer programmer, even a person with some good ideas will struggle at first with the learning curve. We recommend a core understanding of Objective-C and Cocoa principles for aspiring young tweak makers; this can be a significant investment of time, but it is really a hurdle for new tweak makers that haven't a clue where to start. To the uninitiated, the object-oriented nature of the programming involved can be a daunting thing to realize. Generating tweak ideas can be a task for amateurs, but the writing of the code for the tweak implementation is often the result of planning and research and testing for a significant time. We find that many young new programmers are impatient because their ideas for new tweaks do take more time to materialize than they were willing to invest. Patience is a virtue of course, and the best-made tweaks are all products of careful programmers.

The greatest tweak is Activator (libactivator). Based on a commonsense idea of having more triggers system-wide, activator is also a graciously open-sourced project; the product of many months and years of work by our most senior tweak maker, Ryan Petrich. His dedication and expertise shines through in Activator, which doubles as a platform for third-parties to harness the powerful triggers from anyplace to use in their own projects. It represents a lot of research and understanding of the most obscure internals on iOS: SpringBoard and backboard. If there is one shining example to point to as a goal for a tweak maker to show how much research and careful planning can go into a tweak, that is the example to look towards. It's a lofty project that none should consider as being trivial to do, however. For some aspiring developers it can be a great encouragement to see what is possible. Kudos to Ryan Petrich for making it, and for all he does to further the jailbreak development community.

As the repo maintainer for TheBigBoss, I have a job description for myself. Doing my job has given tremendous opportunity to be an influence or guidance for new tweak makers. Often

their first experience with another member of the jailbreak development community is with myself when they first contact me or submit to the repo. We wish that all developers can be involved in the social channels of this scene: chat, forums, twitter et al., however, it's not uncommon that some developers work in relative isolation from these social groups. My involvement then can be seen as important: I may be the only other voice that the programmer will hear, and I will give an opinion on the technical merits of new tweak projects; often this first encounter is invaluable because those developers that work in isolation are not wise to many of the caveats and conventions we hold as important in this community. Our documentation and wikis have improved to make these details more available, but still I am often the first time a developer has some interaction with someone with a greater expertise than their own. I try to give my wisdom and guidance to the developers because its in our best interest to support, if not groom, newcomer developers so they feel as part of the group of jailbreak developers, and they can be pointed towards ways to avoid some of the pitfalls that many newcomers make. I take some pride in doing this and helping in part to strengthen the developer community that is based around the tweak-making culture. I want the jailbreak platform to continue to grow and mature by the great ideas that are envisioned and the expertise to realize them.

Do not be discouraged when the task seems difficult. We have some developers with years and decades of programming experience, and we also have some with only a few weeks or months. I come from the school of thought that it should be well made and well tested, and not rushed or forced. If you have a goal, it should not be merely to have something of yours published on a Cydia repo, but to give something to the public, which will enrich their jailbreak experiences - that is for hobbyists like myself. If you have some commercial interest in Cydia, and for making an selling tweaks, do wide testing with users and alongside other tweaks to help assure a product that works for many users and their combinations of tweaks; your duty as a responsible tweak maker is to be careful while you modify the insides of others' programs or apps, and to be thorough in testing compatibility with others' tweaks.

Tweak making is the new-age hacking. There're already enough reasons for you to get started with tweak development, and we need tweaks to keep the jailbreak community in bloom. Join us, learn from others, work hard, be patient, and have fun.

Optimo