BOOK**TITLE**

**SUBTITLE COULD GO HERE**

Book Author

TITLE Copyright © 2013 by Author Name.

All rights reserved. Printed in the United States of America. No part of this book may be used or reproduced in any manner whatsoever without written permission except in the case of brief quotations em- bodied in critical articles or reviews.

This book is a work of fiction. Names, characters, businesses, organiza- tions, places, events and incidents either are the product of the author’s imagination or are used fictitiously. Any resemblance to actual persons, living or dead, events, or locales is entirely coincidental.

For information contact; address www.website.com

Book and Cover design by Designer

ISBN: 123456789

First Edition: Month 2013

10 9 8 7 6 5 4 3 2 1

# CONTENTS

[Chapter Title 1 1](#_Toc382507455)

[Chapter Title 2 9](#_Toc382507457)

[Chapter Title 3 15](#_Toc382507459)

[Chapter Title 4 22](#_Toc382507461)

[Chapter Title 5 28](#_Toc382507463)

[Chapter Title 6 35](#_Toc382507465)

[About The Author 43](#_Toc382507467)

# 

1

# Recommendation

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 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

# 

1

# Preface

I'm a man who loves travelling by myself. On every vacation in university, I spend about 7 to 10 days as a backpacker, travelling around China. Since it’s a self-guiding tour, no guide will come to help me arrange anything. As a result, before travelling, my friends and I have to prepare everything by ourselves, such as scheduling, confirming the routes and buying tickets. We also need to put deep thought into our plans, and think about its dangers.

It’s a commonly held belief that travelling, especially backpacking, is a great way to expand one’s horizons. What I see during my trips can make me more knowledgeable about the world around me. More importantly, before start travelling, I need to get everything prepared for this journey. My mind has arrived at the destination, even if my body is still at the starting point. This way of thinking is good for cultivating a holistic outlook as well as making us think about problems from a wider, longer term perspective.

Before pursuing my master degree in 2009, I thought deeply about what I wanted to study. My major was computer science. From the beginning of undergraduate year, most of my classmates engaged in the study of Windows. As a student who wasn’t good at programming, there were two alternatives for me to choose - one was to continue the study of Windows, and the other was to explore something else. If I chose the former, there were at least two benefits for me. Firstly, there were lots of documents for reference. The second one was that there were numerous people engaging in the study of Windows. When I met problems, I could consult and discuss with them. However, from the other side, there were also some disadvantages. More references possibly led to less creativity, and the more people engaged in studying Windows, the more competitive pressure I would feel.

In a nutshell, if I engaged in Windows related work, I could start my career very easily. However, there was no guarantee that I could be outstanding among the researchers. If I chose to do something else, it might be very difficult. But as long as I persist with my goal, I could make something different.

Fortunately, my mentor had the same idea. He recommended me to work on mobile development. At that time, there were very few people engaging in this area in China and I had no idea about smart phones. My mobile phone was an out of date Philips phone, so that it was very hard for me to start to develop applications. Despite the difficulties, I trusted my mentor and myself. Not only because I had only chosen him after careful research and recommendations by my senior fellow students, but also that we shared the same opinions. So I started to search mobile development related information. After learning a few concepts about smart phones and mobile Internet, I faintly found that this industry was conductive to the theory that computers and Internet would become smaller, faster and more tightly related with people's lives. Many things could be done in this area. So I chose to engage in studying iOS.

Everything was hard in the beginning. There were lots of differences between iOS and Windows. For example, iOS was an UNIX-like operating system, which was a complete, but closed, ecosystem. Its main programming language Objective-C, and jailbreak, were all strange fields lacking of information at that point. So I learned by myself, week by week, in a hackintosh. And this lasted for almost a year. During this period of time, I read the book "Learn Objective-C on the Mac", input the code on the book into Xcode and checked the result by running the simulator. However, the code and the UI were hard to be associated with each other. Besides, I searched those half-UNIX concepts like backgrounding on Google and tried to understand them, but they were really hard to understand. When my classmates published their papers, I even wondered what I was doing during these several months. When they went out and play, I had to read code alone in the dormitory. When they had fallen asleep, I had to keep on working in the lab. Although these things made me feel lonely, they benefitted me a lot. I learnt a lot and became more informative during this period. As well, it made me become confident. The more knowledge I got, the less lonely I felt. A man can be excellent when he can bear the loneliness. What you pay will finally return and enrich yourself. After one-year of practice, in March 2011, the obscure code suddenly became understandable. The meaning of every word and the relationship of every sentence became clearer. All fragmented knowledge appeared to be organized in my head and the logic of the whole system became explicit.

So I sped up my research. In April 2011, I finished the prototype of my master thesis and got high praise from my mentor who didn’t keep high expectation on my iOS research. Since then, I changed from a person who felt good to a man who was really good, which signified my pass of entry level of iOS research.

In the past few years, I made friends with the author of Theos, DHowett, consulted questions with the father of Activator, rpetrich and quarreled with the admin of TheBigBoss repo, Optimo. They were the people who solved most of my problems along the way. During the development of SMSNinja, I met Hang Wu, (the second author of this book). As research continues, I met a group of people who was doing excellent things but keeping low profile and finally I realized I'm not alone—We stand alone together.

Taking a look back at the past five years, I'm glad that I made the right choice. It's hard to imagine that you can publish a book related to Windows with only 5-years of research. However, this dream comes true with iOS. The fierce competition among Apple, Microsoft and Google and the feedback from market both prove that this industry will definitely play a leading role in the next 10 years. I feel very lucky that I can be a witness and participate in it. So, don't hesitate, come and join us, right now!

When received the invitation from Hang Wu to write this book, I was a bit hesitant. Due to the large quantity of populations, there were fierce competitions in all walks of life. I summarized all accumulated knowledge from countless failures and if I shared all of them in details, would it result in more competitors? Would my advantages be handed over to others? But throughout the history of jailbreaking, from Cydia and CydiaSubstrate to Theos, all these pieces of software were open source and impressed me a lot. It was because these excellent engineers shared their "advantages” that we could absorb knowledge from and then gradually grew better. ‘TweakWeek’ led by rpetrich and ‘OpenJailbreak’ led by posixninja also shared their valuable core source code so that more fans could participate in building up the ecosystem of iOS jailbreak. They were the top developers in this area and their advantages didn’t get reduced with sharing. I was a learner who benefitted a lot from this sharing chain. Moreover, I intended to continue my research. If I didn't stop, my advantage would stay and the only competitor was myself. I believed sharing would help a lot of developers who were stuck at the entry level where I used to be. And sharing could also combine all wisdom together to make science and technology serve people better. Meanwhile, I could make more friends in this community. From this point of view, writing this book can be regarded as a long term thought, just like what I did as a backpacker.

Ok, What I said above is too serious for the preface. Let me say something about this book. The content of the book is suitable for the majority of iOS developers who are not satisfied with developing Apps. There are even more practical examples and details in this book than my master thesis. And if you want to follow up, please focus on our official website http://bbs.iosre.com and our IRC channel #Theos on irc.saurik.com. Together, let us build the jailbreak community!

Here, I want to say thank you to my mother. Without her support, I cannot focus on my research and study. Thanks to my grandpa for the enlightenment of my English studying, having good command of the English language is essential for communicating internationally. Thanks to my mentor for his guidance that helped me grew fast during the three-year master career. Thanks to DHowett, rpetrich, Optimo and those who gave me much help as well as sharp criticism. They helped me grew fast and made me realized that I still had a lot to do. Thanks to Codyd51, DHowett, Haifisch, Nexuist, Nitron, Tyilo, uroboro and yrp for suggestions and review of this book. Also, I would like to say thank you to my future girlfriend. It is the absence of you that makes me focus on my research. So, I will share half of this book’s remuneration with you :)

Career, family, friendship, love are life-long pursuit of ordinary people. However, we cannot get them all, and have to give up some of them. If I unintentionally offend someone here by pursuing and giving up something, I would like to sincerely apologize for my behaviors and thank you for your forgiveness.

At last, I want to share a poem that I like very much. Despite regrets, life is amazing.

The Road Not Taken

Robert Frost, 1874 – 1963

Two roads diverged in a yellow wood,

And sorry I could not travel both

And be one traveler, long I stood

And looked down one as far as I could

To where it bent in the undergrowth;

Then took the other, as just as fair,

And having perhaps the better claim,

Because it was grassy and wanted wear;

Though as for that the passing there

Had worn them really about the same,

And both that morning equally lay

In leaves no step had trodden black.

Oh, I kept the first for another day!

Yet knowing how way leads on to way,

I doubted if I should ever come back.

I shall be telling this with a sigh

Somewhere ages and ages hence:

Two roads diverged in a wood, and I--

I took the one less traveled by,

And that has made all the difference.

In memory of my Grandpa Hanmin Liu and Grandma Chaoyu Wu

snakeinny

# 

1

# Foreword

Why did I write this book?

Two years ago, I changed my job from network equipment work to mobile development. It was the time that mobile development was booming in China. Many startups had sprung up and social networking Apps were very popular with investors. As long as you had a good idea, you could get venture capital at scale of millions, and high salary recruitment dazzles everyone.

At that time, I had already developed some difficult enterprise Apps and I wanted to try some cooler techniques rather than developing social Apps. I thought social Apps were too easy for me. By chance, I joined the company Security Manager, built the iOS team from scratch, and took the responsibility for developing iOS Apps, including App Store and Cydia versions.

In fact, the foundation of jailbreak development is iOS reverse engineering. However, I didn't have too much experience at that time. I was totally a newbie in this area. Fortunately, I could search and learn knowledge on Google. And for iOS developers, jailbreak development and reverse engineering were not completely separate. Although the information shared on the Internet was fragmented and sometimes duplicated, they could still be organized into a complete knowledge map as long as you paid much attention.

However, studying alone makes people feel lonely, especially when you encounter a problem that no one else has encountered. Every time I had to solve problems by myself, I felt that it would be very happy if there were some skillful people that I could communicate with. Although I could email my questions to those experts like Ryan Petrich, I thought it might be some disturbance for them if my questions were very easy. So I always tried to dig into the problems and solve it by myself before I decided to ask them questions.

This embarrassing period lasted for over half a year and it ended when I met another author of this book, snakeninny, in 2012. At that time, he was a master student who faced the pressure of graduation. However, he didn't write his master thesis. Instead, he focused on the underlying iOS research and made big progress. I once asked him why not choose to develop iOS Apps since there were already lots of people engaging in it and had made large amount of money. He answered to me that his ambition was to become the leading developer in the world.

Most of time we solved problems independently. Although we just occasionally discussed questions with each other on the Internet, we still made some valuable collaborations. Before we started to write this book, we once cracked MOMO by reverse engineering and made a tweak that could show position of girls on the map. Of course, we were good developers and we submitted this bug to MOMO and they soon fixed it. This time, we cooperate again, summarize our knowledge into this book and present it to you.

During these years of research on jailbreak development and reverse engineering, the biggest payoff for me is that when I look at an iOS App, I always try to analyze it from underlying architecture and its performance. Both can directly reflect the skill level of its development team. Not only can these experiences be applied to jailbreak development, but also they are suitable for iOS App development. Of course, we must admit there are both positive and negative impacts on reverse engineering. However, we cannot deny the necessity of this area even if Apple doesn't advocate jailbreak development. If we blindly believe that the security issues exposed in this book don’t actually exist, we’re just deceiving ourselves.

Every experienced developer understands that the more knowledge you know, the more likely you have to deal with underlying technologies. For example, what aspects does the Sandbox protection mechanism apply to? Is it a pity that we only study the mechanism of runtime theoretically?

In the field of Android development, the underlying technologies are open source. However, for iOS, only the tip of the iceberg has been exposed. Although there are some books related to iOS security, such as ‘Hacking and Securing iOS Applications’, ’iOS Hacker’s Handbook’, they are too hard for most developers to understand. Even those who have some experience, like us, have difficulties in reading these books.

Since books mentioned above are too hard for most people, why not write a book consists of more details and more practical examples? So concepts, tools, theories and practices organize the contents of this book comprehensively and systematically. We illustrate our experience and knowledge from easy to hard accompanying with figures and examples, helping readers explore the internals of Apps step by step. We do not try to analyze code snippets in depth like other tech blogs. Also, we don't want to puzzle ourselves with how many similar solutions can we use to fix the same problem. What we want to do is to provide readers with a complete system of knowledge and a methodology of iOS reverse engineering. We believe that readers will gain a lot from this book.

Recently, more and more programming experts are joining the jailbreak development community. Although they keep low profile, their works, such as jailbreak tools, App assistants and Cydia tweaks, have great influence on the development of iOS. Their technique level is far beyond mine. But I'm more eager to share knowledge in the hope of helping others.

Who are our target readers?

People of the following kinds may find this book useful.

* iOS enthusiasts.
* Senior iOS developers, who have good command of App development and have the desire to understand iOS better.
* Architects. During the process of reverse engineering, they can learn architectures of those excellent Apps so that they can improve their ability of architecture design.
* Reverse engineers in other systems who’re also interested in iOS.

How to read this book?

There are four parts in this book. They are concepts, tools, theories and practices, respectively. The first three parts will introduce the background, knowledge and its associated tools as well as theories. The fourth part will consists of four examples so that readers will have a deeper understanding of previous knowledge in a practical way.

If the reader doesn’t have experience in iOS reverse engineering, we recommend you to start from the first part rather than start from the fourth part directly. Although it can be very cool visually, hacking is tasteless if you don't know how everything is working underneath.

Errata and Support

Due to our limited skills and writing schedule, it is inevitable that there are some errors or inaccuracies in the book. We plea for your correction and criticism. Also, readers can visit our official forum (http://bbs.iosre.com) and you will find iOS reverse engineers all over the world on it. Your questions will definitely get satisfied answers.

Of course, if you have any good idea or suggestion, you can get in touch with us via the forum too. We are looking forward to hearing from you and your feedback.

Acknowledgements

In the first place, I want to say thank you to evad3rs､PanguTeam､TaiG､saurik and other top teams and experts.

Also thanks to Dustin Howett. His Theos is a powerful tool that helped me to step into iOS reverse engineering.

Thanks to Security Manager for providing me with such nice atmosphere for studying reverse engineering. Although I have left this company, I do wish it could grow better.

Thanks to everyone who offers helps to me. Thanks for your support and encouragement.

This book is dedicated to my dearest family, and many friends who love iOS development.

hangcom

# 

I

# Concepts

Software reverse engineering refers to the process of deducing the implementation and design details of a program or a system by analyzing the functions, structures or behaviors of it. When we are very interested in a certain software feature while not having the access to the source code, we can try to analyze it by reverse engineering.

For iOS developers, Apps on iOS are one of the most complex but fantastic virtual items as far as we know. They are elaborate, meticulous and creative. As developers, when you see an exquisite App, not only will you be amazed by its implementation, but also you will be curious about what kind of techniques are used in this App and what we can learn from it.

# 

1

# Introduction to iOS reverse engineering

Although the recipe of Coca-Cola is highly confidential, some other companies can still copy its taste. Although we don't have access to the source code of others' Apps, we can dig into their details by reverse engineering.

## Prerequisite of iOS reverse engineering

iOS reverse engineering refers to the process of reverse analysis at software-level. If you want to have strong skills on iOS reverse engineering, you'd better be familiar with the hardware constitution of iOS and how iOS works. Also, you should have rich experiences in developing iOS Apps. If you can infer the project scale of an App after using it for a while, its related technologies, its MVC pattern, and which open source projects or frameworks it references, you can announce that you have a good ability on reverse engineering.

Sounds demanding? Aha, a bit. However, all above prerequisites are not fully necessary. As long as you can keep a strong curiosity and perseverance in iOS reverse engineering, you can also become a good iOS reverse engineer. The reason is that during the process of reverse engineering, your curiosity will drive you to study those classical Apps. And it is inevitable that you will encounter some problems that you can't answer them immediately. As a result, it takes your perseverance to support you to overcome the difficulties one by one. Trust me, you will surely get your ability improved and feel the beauty of reverse engineering after putting lots of efforts on programming, debugging and analyzing the logic of software.

## What does iOS reverse engineering do

Metaphorically speaking, we can regard iOS reverse engineering as a spear, which can break the seemingly safe protection of Apps. It is interesting and ridiculous to note that many companies that develop Apps are not aware of the existence of this spear and think their Apps are unbreakable.

For IM Apps like WeChat or WhatsApp, the core of this kind of Apps is the information they exchange. For software of bank, payment or e-commerce, the core is the monetary transaction data and customer information. All these core data have to be securely protected. So developers have to protect their Apps by combining anti-debugging, data encryption and code obfuscation together. The aim is to increase the difficulty of reverse engineering and prevent similar security issues from affecting user experience.

However, the technologies currently being used to protect Apps are not in the same dimension with those being used in iOS reverse engineering. For general App protections, they look like fortified castles. By applying the MVC architecture of Apps inside the castle with thick walls outside, we may feel that they are insurmountable, as shown in figure 1-1.



Figure 1-1 Strong fortress, taken from Assassin's Creed

But if we step onto another higher dimension and overlook into the castle where the App resides, you find that structure inside the castle is no longer a secret, as shown in figure 1-2.



Figure 1-2 Overlook the castle, taken from Assassin's Creed

All Objective-C interfaces, all properties, all exported functions, all global variables, even all logic are exposed in front of us, which means all protections have became useless. So if we are in this dimension, walls are no longer hindrances. What we should focus on is how can we find our targets inside the huge castle.

At this point, by using reverse engineering techniques, you can enter the low dimension castle from any high dimension places without damaging walls of the castle, which is definitely tricky while not laborious. By monitoring and even changing the logic of Apps, you can learn the core information and design details easily.

Sounds very incredible? But this is true. According to the experiences and achievements I’ve got from the study of iOS reverse engineering, I can say that reverse engineering can break the protection of most Apps, all their implementation and design details can be completely exposed.

The metaphor above is only my personal viewpoint. However, it vividly illustrates how powerful iOS reverse engineering is. In a nutshell, there are two major functions in iOS reverse engineering as below:

* Analyze the target App and get the core information. This can be concluded as security related reverse engineering.
* Learn from other Apps' features and then make use of them in our own Apps. This can be concluded as development related reverse engineering.



### Security related iOS reverse engineering

Security related IT industry would generally make extensive use of reverse engineering. For example, reverse engineering plays the key roles in evaluating the security level of a financial App, finding solutions of killing viruses, and setting up a spam phone call firewall on iOS, etc.

#### Evaluate security level

Apps which are consist of sensitive features like financial transactions will encrypt the data at first and then save the encrypted data locally or transfer them via network. If developers do not have strong awareness of security, it is very possible for them to save or send the sensitive information such as bank accounts and passwords without encryption, which is definitely a great security risk.

If a company with high reputation wants to release an App. In order to make the App qualified with the reputation as well as the trust from customers, the company will hire a security organization to evaluate this App before releasing it. In most cases, the security organization does not have the access to the source code so that they cannot evaluate the security level via code review. Therefore the only way they can do is reverse engineering. They try to attack the App and then evaluate the security level based on the result.

#### Reverse engineering malware

iOS is the operating system of smart devices, it has no essential difference with computer operating systems. From the first generation, iOS is capable of browsing the Internet. However, the Internet is the best medium of malware. Ikee, exposed in 2009, is the first virus in iOS. It can infect those jailbroken iOS devices which have installed ssh but have not changed the default password “alpine”. It can change the background image of the lockscreen to photo of a British singer. Another virus WireLurker appeared at the end of 2014, it can steal private information of users and spread on PC or Mac, bringing users disastrous harm.

For malware developers, by targeting system and software vulnerabilities through reverse engineering, they can penetrate into the target hosts, access to sensitive data and do whatever they want.

For anti-virus software developers, they can analyze samples of viruses through reverse engineering, observe the behaviors of viruses and then try to kill them in the infected hosts as well as summarize the methods to protect against viruses.

#### Detect software backdoors

A big advantage of open source software is its good security. Tens of thousands of developers review the code and modify the bug of open source software. As a result, the possibilities that there are backdoors inside the code are minimized, and the security related bugs would be fixed before they are disclosed. For closed source software, reverse engineering is one of the most frequently used methods to detect the backdoors in software. For example, we often install different kinds of Apps on jailbroken iPhones through third-party App Stores. All these Apps are not officially examined and reviewed by Apple so there could be unrevealed risks. Even worse, some developers will put backdoors inside their Apps on the purpose of stealing something from users. So reverse engineering is often involved in the process of detecting that kind of behaviors.

#### Remove software restriction

Selling Apps on AppStore or Cydia is one primary economic source for App developers. In the software world, piracy and anti-piracy will coexist forever. Many developers have already added protection in their software to prevent piracy. However, just like the war between spear and shield will never stop, no matter how good the protection of an App is, there will definitely be one day that the App is cracked. The endless emergency of pirated software makes it an impossible task for developers to prevent piracy. For example, the most famous share repository "xsellize" on Cydia is able to crack any App in just one day and it is notorious among the industry.

### Development related iOS reverse engineering

For iOS developers, reverse engineering is one of the most practical techniques. For example, we can do reverse engineering on system APIs to use some private functions, which are not documented. Also, we can learn good architecture and design from those classical Apps through reverse engineering.

#### Reverse System APIs

The reason that Apps developed by developers are able to run in the operating system and to provide users with a variety of functions is that these functions are already embedded in the operating system itself, what developers need to do is just reassembling them. As we all know, functions we used for developing Apps on AppStore are restricted by Apple's document and are under the strict regulation of Apple. For example, you cannot use undocumented functions like making phone calls or sending messages. However, if you’re targeting Cydia Store, absence of private functions makes your App much less competitive. If you want to use undocumented functions, the most effective reference is from reversing iOS system APIs, then you can recreate the code of corresponding functions and apply it to your own Apps.

#### Learn from other Apps

The most popular scenario for reverse engineering is to learn from other Apps. For most Apps on AppStore, although the implementations of them are not very difficult, their ingenious ideas and good business operation are the keys to success. So, if you just want to learn a function from another App, it is time-consuming and laborious to restore the code through reverse engineering; I’d suggest you write a similar App from scratch. However, reverse engineering plays a critical role in the situation when we don't know how a feature of an App is implemented. This is often seen in Cydia Apps with extensive use of private functions. For example, Audio Recorder, known as the first phone call recording App, is a closed source App. Yet it is very interesting for us to learn how it is implemented. Under this circumstance you can learn a little bit through iOS reverse engineering.

There are some classical Apps with neat code, reasonable architecture, and elegant implementation. Compared with developers of those Apps, we don't have profound technical background. So if we want to learn from those Apps while not having an idea of where to start, we can turn to reverse engineering. Through reverse engineering those Apps, we can extract the architecture design and apply it to our own projects so that we can enhance our Apps. For example, the stability and robustness of WhatsApp is so excellent that if we want to develop our own IM Apps, we can benefit a lot from learning the architecture and design of WhatsApp.

## The process of iOS reverse engineering

When we want to reverse an App, how should we think? Where should we start? The purpose of this book is to guide the beginners into the field of iOS reverse engineering, and cultivate readers to think like reversers.

Generally speaking, reverse engineering can be regarded as a combination of analysis on two stages, which are system analysis and code analysis, respectively. In the phase of system analysis, we can find our targets by observing behavioral characteristics of program and organizations of files. During code analysis, we need to restore the core code and then ultimately achieve our goals.



### System Analysis

At the stage of system analysis, we should run target Apps under different conditions, perform various operations, observe the behavioral characteristics and find out features that we are interested in, such as which option we choose leads to a popup alert? Which button makes a sound after pressing it? What is the output associated with our input, etc. Also, we can browse the filesystem, see the displayed images, find the configuration files’ locations, inspect the information stored in databases and check whether the information is encrypted.

Take Sina Weibo as an example. When we look over its Documents folder, we can find some databases:

-rw-r--r-- 1 mobile mobile 210944 Oct 26 11:34 db\_46100\_1001482703473.dat

-rw-r--r-- 1 mobile mobile 106496 Nov 16 15:31 db\_46500\_1001607406324.dat

-rw-r--r-- 1 mobile mobile 630784 Nov 28 00:43 db\_46500\_3414827754.dat

-rw-r--r-- 1 mobile mobile 6078464 Dec 6 12:09 db\_46600\_1172536511.dat

……

Open them with SQLite tools, we can find some followers’ information in it, as shown in figure 1-3.

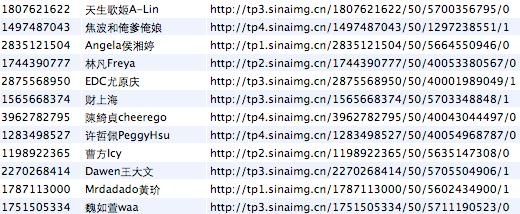


Figure 1-3 Sina Weibo database

Such information provides us with clues for reverse engineering. Database file names, Sina Weibo user IDs, URLs of user information, all can be used as cut-in points for reverse engineering. Find and organize these clues, then track down to what we are interested in, is often the first step of iOS reverse engineering.

### Code Analysis

After system analysis, we should do code analysis based on the binary of an App. Through reverse engineering, we can deduce the design pattern, internal algorithms, and the implementation details of an App. However, this is a very complex process and can be regarded as an art of deconstruction and reconstruction. To improve your reverse engineering skill level into the state of art, you must have a thorough understanding on software development, hardware principles, and iOS itself. Analyzing the low-level instructions bit by bit is not easy and cannot be fully covered in one single book.

The purpose of this book is just to introduce tools and methodologies of reverse engineering to beginners. Technologies are evolving constantly, so we cannot cover all of them. For this reason, I’ve build up a forum, <http://bbs.iosre.com>, where we can discuss and exchange ideas with each other in real time.

## Tools for iOS reverse engineering

After learning some theories about iOS reverse engineering, it is time for us to put theory into practice with some useful tools. Compare with App development, tools used in reverse engineering are not as “smart” as those in App development. Most of tasks have to be done manually, so being proficient with tools can greatly improve the efficiency of reverse engineering. Tools can be divided into 4 major categories; they are monitors, disassemblers, debuggers and development kit.



### Monitors

In the field of iOS reverse engineering, tools used for sniffing, monitoring and recording targets’ behaviors can all be concluded as monitors. These tools generally record and display certain operations performed by the target programs, such as UI changes, network activities and file accesses. Reveal, snoop-it, introspy, etc., are frequently used monitors.

Reveal, as shown in figure 1-4, is a tool to see the view hierarchy of an App in real-time.

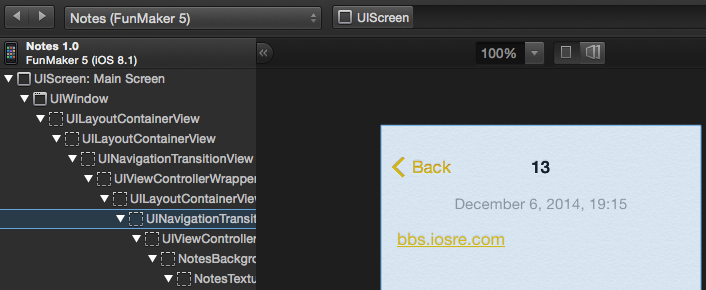


Figure 1- 4 Reveal

Reveal can assist us in locating what we are interested in an App so that we can quickly approach the code from the UI.

### Disassemblers

After approaching the code from the UI, we have to use disassembler to sort out the code. Disassemblers take binaries as input, and output assembly code after processing the files. IDA and Hopper are two major disassemblers in iOS reverse engineering.

As an evergreen disassembler, IDA is one of the most commonly used tools in reverse engineering. It supports Windows, Linux and OSX, as well as multiple processor architectures, as shown in figure 1-5.

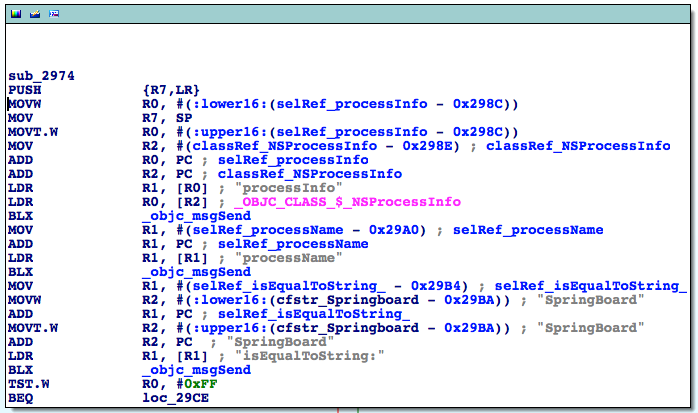


Figure 1- 5 IDA

Hopper is a disassembler that came out in recent years, which mainly targets Apple family operating systems, as shown in figure 1-6.

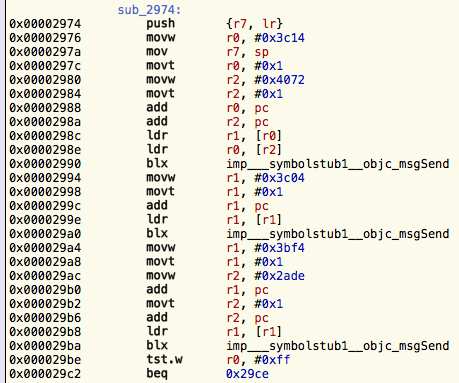


Figure 1- 6 Hopper

After disassembling binaries, we have to read the generated assembly code. This is the most challenging task as well as the most interesting part in iOS reverse engineering, which will be explained in detail in chapters 6 to 10. We will use IDA as the main disassembler in this book and you can reference the experience of Hopper on http://bbs.iosre.com.

### Debuggers

iOS developers should be familiar with debuggers because we often need to debug our own code in Xcode. We can set a breakpoint on a line of code so that process will stop at that line and display the current status of the process in real time. We constantly use LLDB for debugging during both App development and reverse engineering. Figure 1-7 is an example of debugging in LLDB.

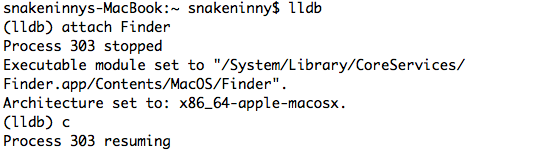


Figure 1- 7 LLDB

### Development kit

After finishing all the above steps, we can get results from analysis and start to code for now. For App developers, Xcode is the most frequently used development tool. However, if we transfer the battlefield from AppStore to jailbroken iOS, our development kit gets expanded. Not only is there an Xcode based iOSOpenDev, but also a command line based Theos. Judging from my own experiences, Theos is the most exciting development tool. Before knowing Theos, I felt like I was restricted to the AppStore. Not until I mastered the usage of Theos did I break the restriction of AppStore and completely understood the real iOS. Theos is the major development tool in this book and we’ll discuss about iOSOpenDev on our website.

## Conclusion

In this chapter, we have introduced some concepts about iOS reverse engineering in order to provide readers with a general idea of what we’ll be focusing on. More details and examples will be covered in the following chapters. Stay tuned with us!

# 

2

# Introduction to jailbroken iOS

Compared with what we see on Apps’ UI, we are more interested in their low-level implementation, which is exactly the motivation of reverse engineering. But as we know, non-jailbroken iOS is a closed blackbox, it has not been exposed to the public until dev teams like evad3rs, PanguTeam and TaiG jailbroke it, then we’re able to take a peek under the hood.

## iOS System Hierarchy

For non-jailbroken iOS, Apple provides very few APIs in the SDK to directly access the filesystem. By refering to the documents, App Store developers may have no idea of iOS system hierarchy at all.

Because of very limited permission, App Store Apps (hereafter referred to as StoreApps) cannot access most directories apart from their own. However, for jailbroken iOS, Cydia Apps can possess higher permission than StoreApps, which enables them to access the whole filesystem. For example, iFile from Cydia is a famous third-party file management App, as shown in figure 2-1.

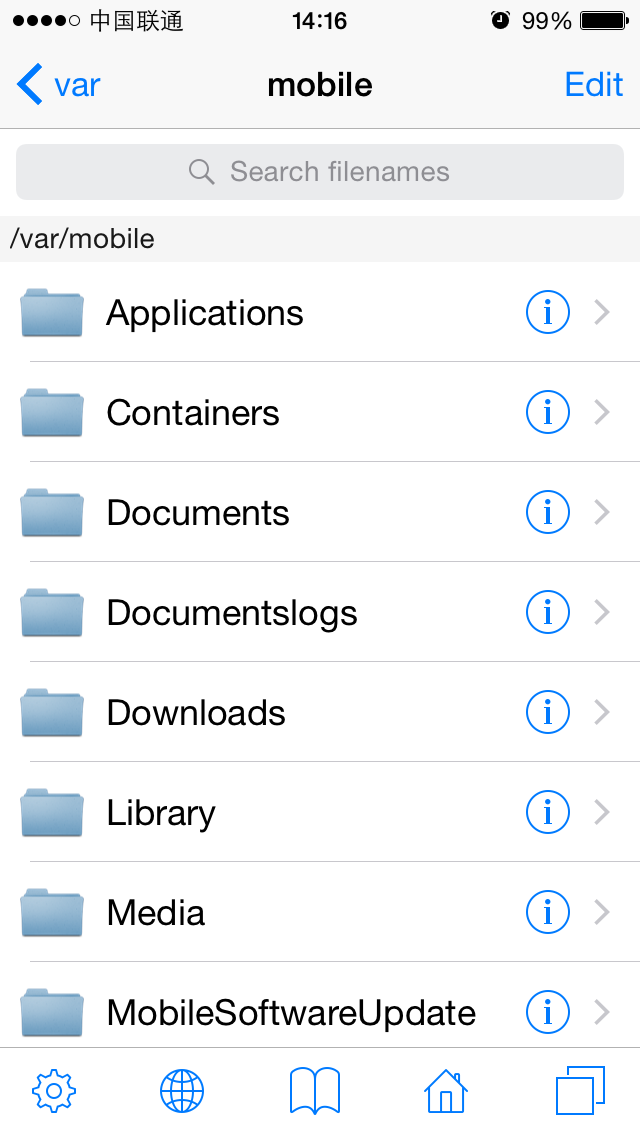


Figure 2- 1 iFile

With the help of AFC2, we can also access the whole iOS filesystem via software like iFunBox on PC, as shown in figure 2-2.

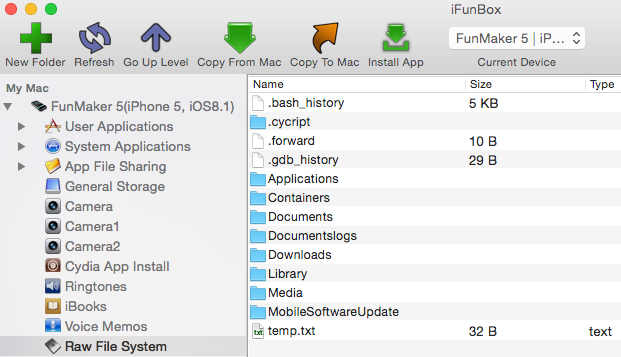


Figure 2- 2 iFunBox

Because our reverse engineering targets come right from iOS, being able to access the whole iOS filesystem is the prerequisite of our work.

### iOS filesystem

iOS comes from OSX, which is based on UNIX. Although there are huge differences among them,they are somehow related to each other. We can get some knowledge of iOS filesystem from Filesystem Hierarchy Standard and hier(7).

Filesystem Hierarchy Standard (hereafter referred to as FHS) provides a standard for all \*NIX filesystems. The intention of FHS is to make the location of files and directories predictable for users. Evolving from FHS, OSX has its own standard, called hier(7). Common \*NIX filesystem is as follows.

* /

Root directory. All other files and directories expand from here.

* /bin

Short for “binary”. Binaries that provide basic user-level functions, like ls and ps are stored here.

* /boot

Stores all necessary files for booting up. This directory is empty on iOS.

* /dev

Short for “device”, stores BSD device files. Each file represents a block device or a character device. In general, block devices transfer data in block, while character devices transfer data in character.

* /sbin

Short for “system binaries”. Binaries that provide basic system-level functions, like netstat and reboot are stored here.

* /etc

Short for “Et Cetera”. This directory stores system scripts and configuration files like passwd and hosts. On iOS, this is a symbolic link to /private/etc.

* /lib

This directory stores system-level lib files, kernel files and device drivers. This directory is empty on iOS.

* /mnt

Short for “mount”, stores temporarily mounted filesystems. On iOS, this directory is empty.

* /private

Only contains 2 subdirectories, i.e. /private/etc and /private/var.

* /tmp

Temporary directory. On iOS, this directory is a symbolic link to /private/var/tmp.

* /usr

A directory containing most user-level tools and programs. /usr/bin is used for other basic functions which are not provided in /bin or /sbin, like nm and killall. /usr/include contains all standard C headers, and /usr/lib stores lib files.

* /var

Short for “variable”, stores files that frequently change, such as log files, user data and temporary files. /var/mobile/ is for mobile user and /var/root/ is for root user, these 2 subdirectories are our main focus.

Most directories listed above are rather low-level that they’re difficult to reverse engineer. As beginners, it’s better for us to start with something much easier. As App developers, most of our daily work is dealing with iOS specific directories. Reverse engineering becomes more approachable when it comes to these familiar directories:

* /Applications

Directory for all system Apps and Cydia Apps, excluding StoreApps, as shown in figure 2-3.

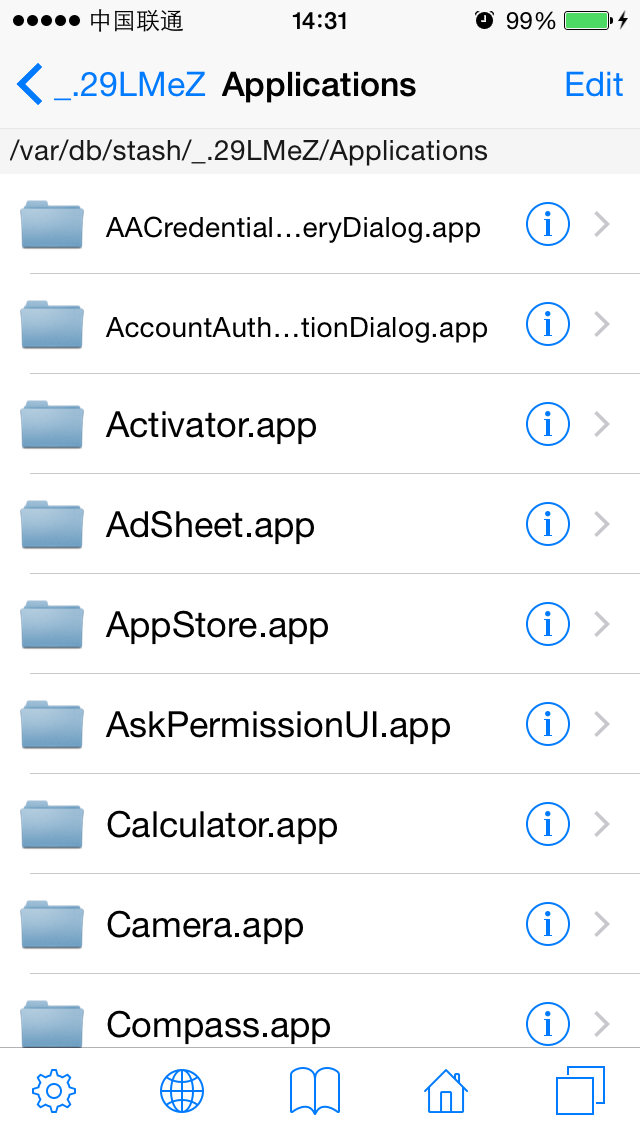


Figure 2- 3 /Applications

* /Developer

If you connect your device with Xcode and can see it in “Devices” category like figure 2-4 shows, a “/Developer” directory will be created automatically on device, as shown in figure 2-5. Inside this directory, there are some data files and tools for debugging.

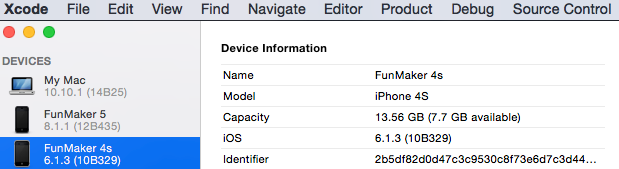


Figure 2- 4 Enable debugging on device

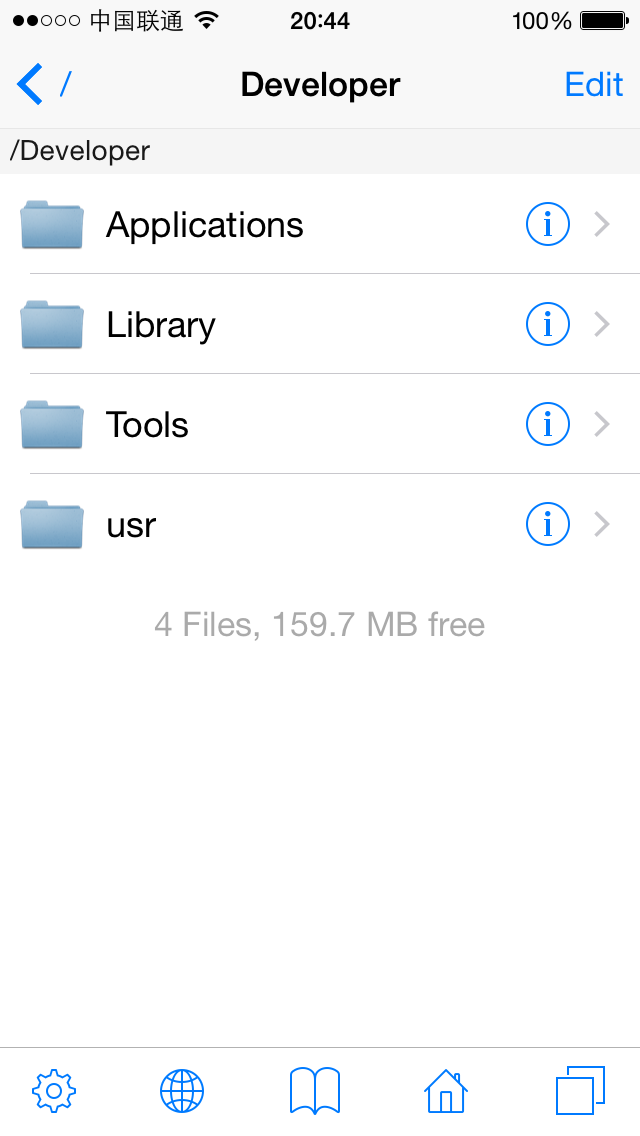


Figure 2- 5 /Developer

* /Library

This directory contains some system-supported data as shown in figure 2-6. One subdirectory of it named MobileSubstrate is where all CydiaSubstrate (formerly known as MobileSubstrate) based tweaks are.

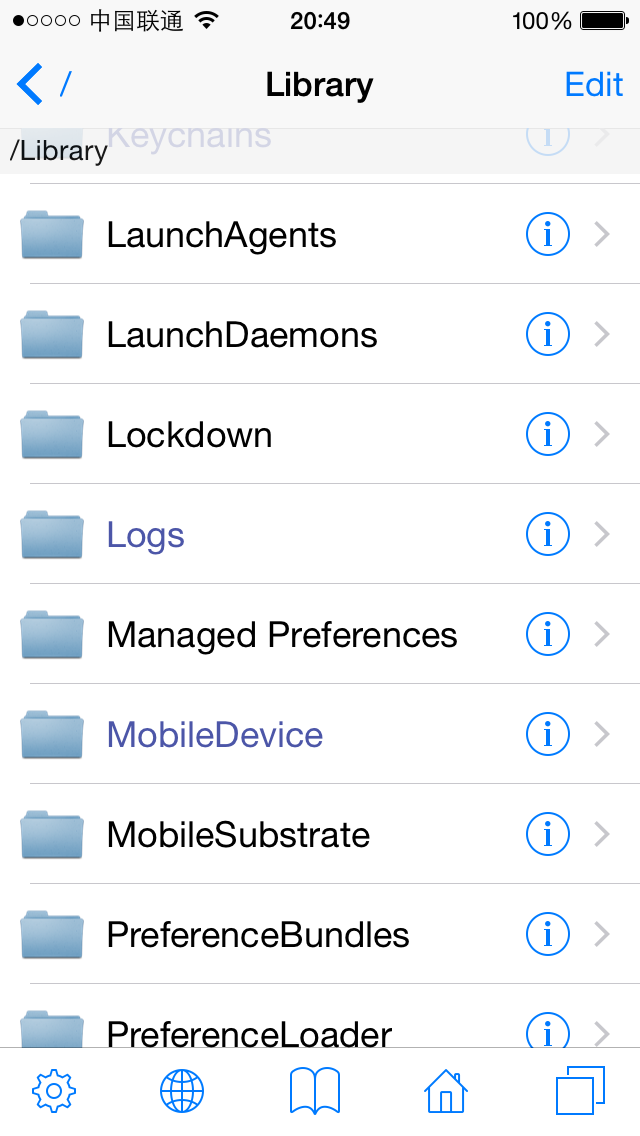


Figure 2- 6 /Library

* /System/Library

One of the most important directories on iOS, stores lots of system components, as shown in figure 2-7.

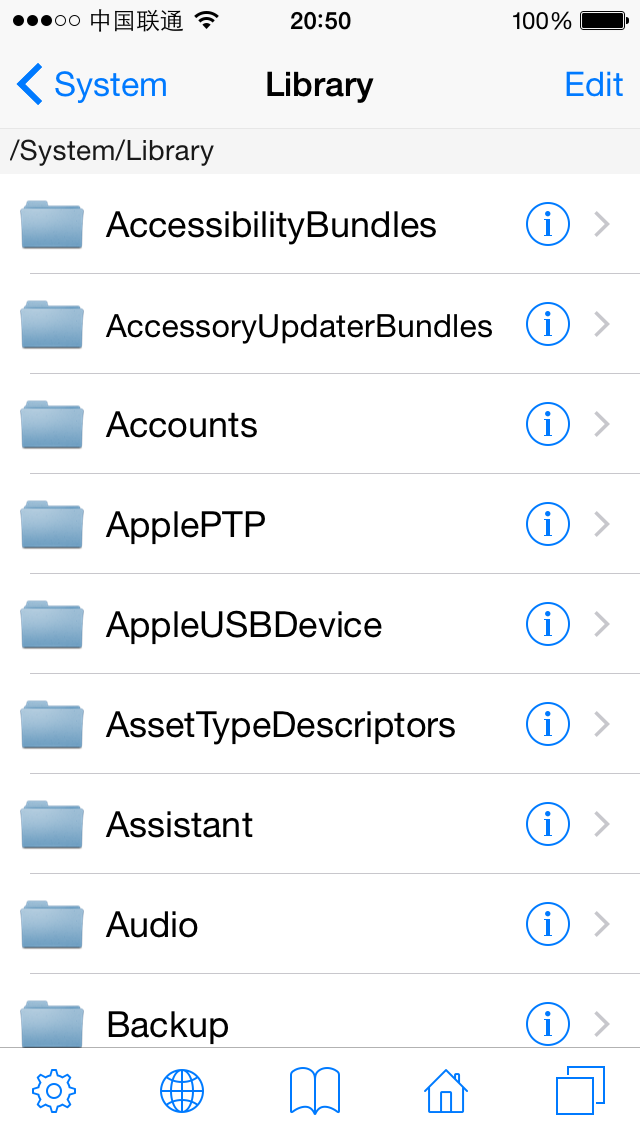


Figure2- 7 /System/Library

Under this directory, we beginners should mainly focus on these subdirectories:

* + - /System/Library/Frameworks and /System/Library/PrivateFrameworks

Stores most iOS frameworks. Documented APIs are only a tiny part of them, while countless private APIs are hidden in those frameworks.

* + - /System/Library/CoreServices/SpringBoard.app

iOS' graphical user interface, as is explorer to Windows. It is the most important intermediate between users and iOS.

More directories under “/System” deserve our attention. For more advanced contents, please visit <http://bbs.iosre.com>.

* /User

User directory, it’s a symbolic link to /var/mobile, as shown in figure 2-8.

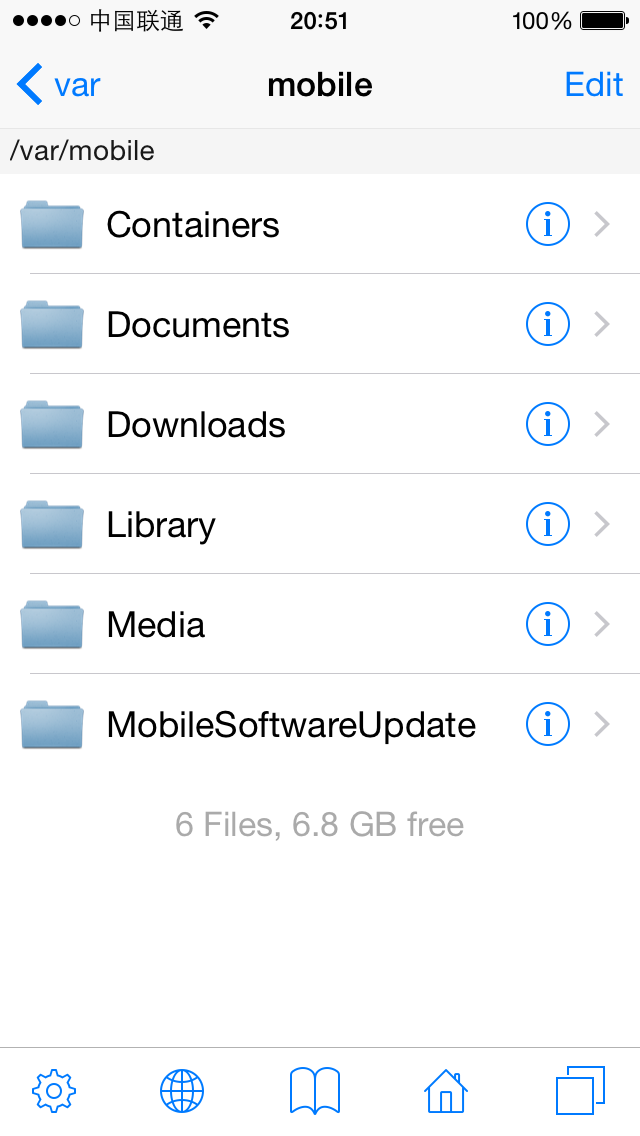


Figure 2- 8 /User

This directory contains large numbers of user data, such as:

* + - Photos are stored in /var/mobile/Media/DCIM;
    - Recording files are stored in /var/mobile/Media/Recordings;
    - SMS/iMessage databases are stored in /var/mobile/Library/SMS;
    - Email data is stored in /var/mobile/Library/Mail.

Another major subdirectory is /var/mobile/Containers, which holds StoreApps. It is noteworthy that bundles containing Apps’ executables reside in /var/mobile/Containers/Bundle, while Apps’ data files reside in /var/mobile/Containers/Data, as shown in figure 2-9.

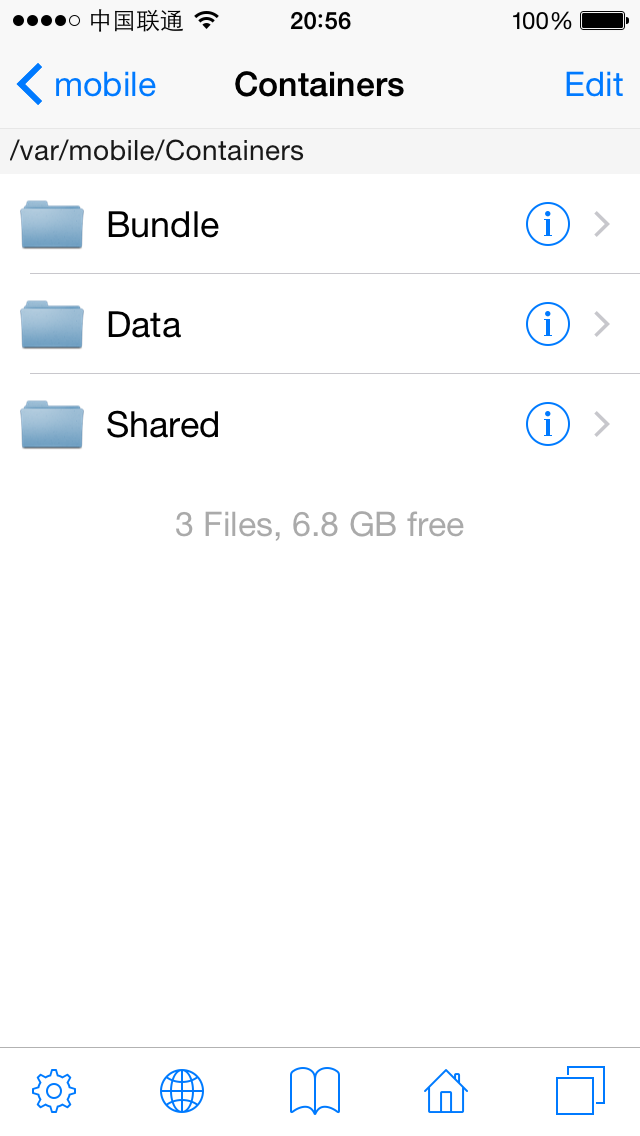


Figure 2- 9 /var/mobile/Containers

It’s helpful to have a preliminary knowledge of iOS filesystem when we discover some interesting functions and want to further locate their origins. What we’ve introduced above is only a small part of iOS filesystem. For more details, please visit <http://bbs.iosre.com>, or just type "man hier" in OSX terminal.

### iOS file permission

iOS is a multi-user system. “User” is an abstract concept, it means the ownership and accessibility in system. For example, while root user can call “reboot” command to reboot iOS, mobile user cannot. “group” is a way to organize users. One group can contain more than one user, and one user can belong to more than one group.

Every file on iOS belongs to a user and a group, or to say, this user and this group own this file. And each file has its own permission, indicating what operations can the owner, the (owner) group and others perform on this file. iOS uses 3 bits to represent a file’s permission, which are r (read), w (write) and x (execute) respectively. There are 3 possible relationships between a user and a file:

* This user is the owner of this file.
* This user is not the owner of this file, but he is a member of the (owner) group.
* This user is neither the owner nor a member of the (owner) group.

So we need 3 \* 3 bits to represent a file’s permission in all situations. If a bit is set 1, it means the corresponding permission is granted. For instance, 111101101 represents rwxr-xr-x, in other words, the owner has r, w and x permission, but the (owner) group and other users only have r and x permission. Binary number 111101101 equals to octal number 755, which is another common representation form of permission.

Actually, besides r, w, x permission, there are 3 more special permission, i.e. SUID, SGID and sticky. They are not used in most cases, so they don’t take extra permission bits, but instead reside in x permission’s bit. As beginners, there are slim chances that we will have to deal with these special permission, so don't worry if you don’t fully understand this. For those of you who are interested, <http://thegeekdiary.com/what-is-suid-sgid-and-sticky-bit/> is good to read.

## iOS file types

Rookie reverse engineers’ main targets are Application, Dynamic Library (hereafter referred to as dylib) and Daemon binaries. The more we know them, the smoother our reverse engineering will be. These 3 kinds of binaries play different roles on iOS, hence have different file hierarchies and permission.

### Application

Application, namely App, is our most familiar iOS component. Although most iOS developers deal with Apps everyday, our main focus on App is different in iOS reverse engineering. Knowing the following concepts is a prerequisite for reverse engineering.

#### bundle

The concept of bundle originates from NeXTSETP. Bundle is indeed not a single file but a well-organized directory conforming to some standards. It contains the executable binary and all running necessities. Apps and frameworks are packed as bundles. PreferenceBundles (as shown in figure 2-10), which are common in jailbroken iOS, can be seen as a kind of Settings dependent App, which is also a bundle.

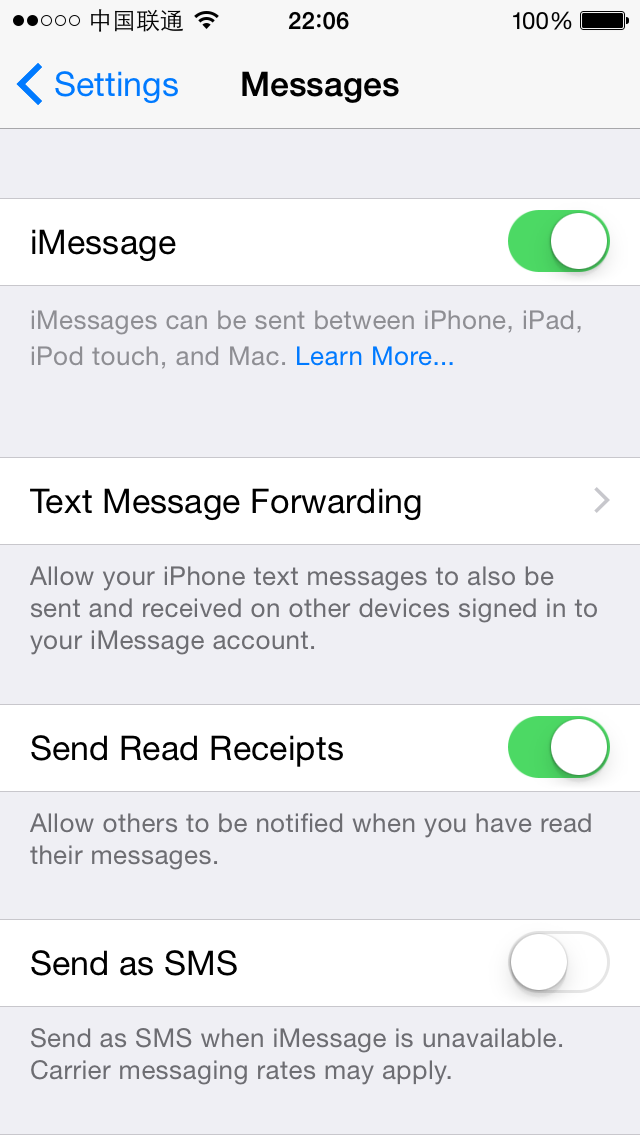


Figure 2- 10 PreferenceBundle

Frameworks are bundles too, but they contain dylibs instead of executables. Relatively speaking, frameworks are more important than Apps, because most parts of an App work by calling APIs in frameworks. When you target a bundle in reverse engineering, most of the work can be done inside the bundle, saving you significant time and energy.

#### App directory hierarchy

Being familiar with App’s directory hierarchy is a key factor of our reverse engineering efficiency. There are 3 important components in an App’s directory:

* Info.plist

Info.plist records an App’s basic information, such as its bundle identifier, executable name, icon file name and so forth. Among these, bundle identifier is the key configuration value of a tweak, which will be discussed later in CydiaSubstrate section. We can look up the bundle identifier in Info.plist with Xcode, as shown in figure 2-11.

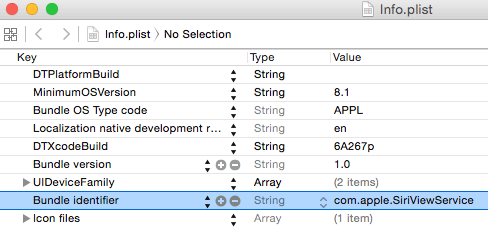


Figure 2- 11 Browse Info.plist in Xcode

Or use a command line tool, plutil, to view its value.

snakeninnysiMac:~ snakeninny$ plutil -p /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/SiriViewService.app/Info.plist | grep CFBundleIdentifier

"CFBundleIdentifier" => "com.apple.SiriViewService"

In this book, we mainly use plutil to browse plist files.

* Executable

Executable is the core of an App, as well our ultimate reverse engineering target, without doubt. We can locate the executable of an App with Xcode, as shown in figure 2-12.

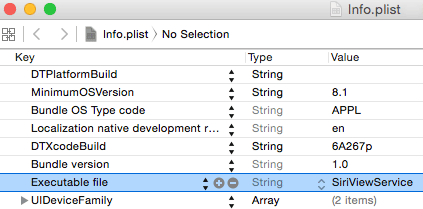


Figure 2- 12 Browse Info.plist in Xcode

Or with plutil:

snakeninnysiMac:~ snakeninny$ plutil -p /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/SiriViewService.app/Info.plist | grep CFBundleExecutable

"CFBundleExecutable" => "SiriViewService"

* lproj directories

Localized strings are saved in lproj directories. They are important clues of iOS reverse engineering. plutil tool can also parse those .string files.

snakeninnysiMac:~ snakeninny$ plutil -p /Users/snakeninny/Code/iOSSystemBinaries/8.1\_iPhone5/SiriViewService.app/en.lproj/Localizable.strings

{

"ASSISTANT\_INITIAL\_QUERY\_IPAD" => "What can I help you with?"

"ASSISTANT\_BOREALIS\_EDUCATION\_SUBHEADER\_IPAD" => "Just say “Hey Siri” to learn more."

"ASSISTANT\_FIRST\_UNLOCK\_SUBTITLE\_FORMAT" => "Your passcode is required when %@ restarts"

……

You will see how we make use of .strings in reverse engineering in chapter 5.

#### System App VS. StoreApp

/Applications contains system Apps and Cydia Apps (We treat Cydia Apps as system Apps), and /var/mobile/Containers/Bundle/Application is where StoreApps reside. Although all of them are categorized as Apps, they are different in some ways:

* Directory hierarchy

Both system Apps and StoreApps share the similar bundle hierarchy, including Info.plist files, executables and lproj directories, etc. But the path of their data directory is different, for StoreApps, their data directories are under /var/mobile/Containers/Data, while for system Apps running as mobile, their data directories are under /var/mobile; for system Apps running as root, their data directories are under /var/root.

* Installation package and permission

In most cases, Cydia Apps’ installation packages are .deb formatted while StoreApps’ are .ipa formatted. .deb files come from Debian, and are later ported to iOS. Cydia Apps’ owner and (owner) group are usually root and admin, which enables them to run as root. .ipa is the official App format, whose owner and (owner) group are both mobile, which means they can only run as mobile.

* Sandbox

Broadly speaking, sandbox is a kind of access restriction mechanism, we can see it as a form of permission. Entitlements are also a part of sandbox. Sandbox is one of the core components of iOS security, which possesses a rather complicated implementation, and we’re not going to discuss it in details. Generally, sandbox restricts an App’s file access scope inside the App itself. Most of the time, an App has no idea of the existence of other Apps, not to mention accessing them. What’s more, sandbox restricts an App’s function. For example, an App has to ask for sandbox’s permission to take iCloud related operations.

Sandbox is not suitable to be beginners’ target, it’d be enough for us to know its existence. In iOS reverse engineering, jailbreak has already removed most security protections of iOS, and reduced sandbox’s constraints in some degree, so we are likely to ignore the existence of sandbox, hence leading to some strange phenomena such as a tweak cannot write to a file, or calls a function but it’s not functioning as expected. If you can make sure your code is 100% correct, then you should recheck if the problem is because of your misunderstanding of tweak’s permission or sandbox issues. Concepts about Apps cannot be fully described in this book, so if you have any questions, feel free to raise it on <http://bbs.iosre.com>.

### Dynamic Library

Most of our developers’ daily work is writing Apps, and I guess just a few of you have ever written dylibs, so the concept of dylib is strange to most of you. In fact, you’re dealing with dylibs a lot: the frameworks and lib files we import in Xcode are all dylibs. We can verify this with ‘file’ command:

snakeninnysiMac:~ snakeninny$ file /Users/snakeninny/Code/iOSSystemBinaries/8.1.1\_iPhone5/System/Library/Frameworks/UIKit.framework/UIKit

/Users/snakeninny/Code/iOSSystemBinaries/8.1.1\_iPhone5/System/Library/Frameworks/UIKit.framework/UIKit: Mach-O dynamically linked shared library arm

If we shift our attention to jailbroken iOS, all the tweaks in Cydia work as dylibs. It is those tweaks’ existence that makes it possible for us to customize our iPhones. In reverse engineering, we’ll be dealing with all kinds of dylibs a lot, so it’d be good for us to know some basic concepts.

On iOS, libs are divided into two types, i.e. static and dynamic. Static libs will be integrated into an App’s executable during compilation, therefore increases the App’s size. Now that we have a bigger executable, iOS needs to load more data into memory during App launching, so the result is that, not surprisingly, App’s launch time increased, too. Dylibs are relatively “smart”, it doesn’t affect executable’s size, and iOS will load a dylib into memory only when an App needs it right away, then the dylib becomes part of the App.

It’s worth mentioning that, although dylibs exist everywhere on iOS, and they are the main targets of reverse engineering, they are not executables. They cannot run individually, but only serve other processes. In other words, they live in and become a part of other processes. Thus, dylibs’ permission depends on the processes they live in, the same dylib’s permission is different when it lives in a system App or a StoreApp. For instance, suppose you write an Instagram tweak to save your favorite pictures locally, if the destination path is this App’s documents directory under /var/mobile/Containers/Data, there won’t be a problem because Instagram is a StoreApp, it can write to its own documents. But if the destination path is /var/mobile/Documents, then when you save pictures happily and want to review them wistfully, you’ll find nothing under /var/mobile/Documents. All the tweak operations are banned by sandbox.

### Daemon

Since your first day doing iOS development, Apple has been telling you "There is no real backgrounding on iOS and your App can only operate with strict limitations." If you are a pure App Store developer, following Apple's rules and announcements can make the review process much easier! However, since you're reading this book you likely want to learn reverse engineering and this means straying into undocumented territory. Stay calm and follow me:

* When I’m browsing reddit or reading tweets on my iPhone, suddenly a phone call comes in. All operations are interrupted immediately, and iOS presents the call to me. If there is no real backgrounding on iOS, how can iOS handle this call in real time?
* For those who receive spam iMessages a lot, firewalls like SMSNinja are saviors. If a firewall fails to stay in the background, how could it filter every single iMessages instantaneously?
* Backgrounder is a famous tweak on iOS 5. With the help of this tweak, we can enable real backgrounding for Apps! Thanks to this tweak, we don’t have to worry about missing WhatsApp messages because of unreliable push notifications any more. If there is no real backgrounding, how could Backgrounder even exist?

All these phenomena indicate that real backgrounding does exist on iOS. Does that mean Apple lied to us? I don’t think so. For a StoreApp, when user presses the home button, this App enters background, most functions will be paused. In other words, for App Store developers, you’d better view iOS as a system without real backgrounding, because the only thing Apple allows you to do doesn’t support real backgrounding. But iOS originates from OSX, and like all \*NIX systems, OSX has daemons (The same thing is called Service on Microsoft Windows). Jailbreak opens the whole iOS to us, thus reveals all daemons.

Daemons are born to run in the background, providing all kinds of services. For example, imagent guarantees the correct sending and receiving of iMessages, mediaserverd handles almost all audios and videos, and syslogd is used to record system logs. Each daemon is consists of two parts, one executable and one plist file. The root process on iOS is launchd, which is also a daemon, checks all plist files under /System/Library/LaunchDaemons and /Library/LaunchDaemons after each reboot, then run the corresponding executable to launch the daemon. A daemons’ plist file plays a similar role as an App’s Info.plist file, it records the daemon’s basic information, as shown in the following:

snakeninnys-MacBook:~ snakeninny$ plutil -p /Users/snakeninny/Code/iOSSystemBinaries/8.1.1\_iPhone5/System/Library/LaunchDaemons/com.apple.imagent.plist

{

"WorkingDirectory" => "/tmp"

"Label" => "com.apple.imagent"

"JetsamProperties" => {

"JetsamMemoryLimit" => 3000

}

"EnvironmentVariables" => {

"NSRunningFromLaunchd" => "1"

}

"POSIXSpawnType" => "Interactive"

"MachServices" => {

"com.apple.hsa-authentication-server" => 1

"com.apple.imagent.embedded.auth" => 1

"com.apple.incoming-call-filter-server" => 1

}

"UserName" => "mobile"

"RunAtLoad" => 1

"ProgramArguments" => [

0 => "/System/Library/PrivateFrameworks/IMCore.framework/imagent.app/imagent"

]

"KeepAlive" => {

"SuccessfulExit" => 0

}

}

Compared with Apps, daemons provide much much lower level functions, accompanying with much much greater difficulties reverse engineering them. If you don’t know what you’re doing for sure, don’t even try to modify them! It may break your iOS, leading to booting failures, so you’d better stay away from daemons as reverse engineering newbies . After you get some experience reverse engineering Apps, it’d be OK for you to challenge daemons. After all, it takes more time and energy to reverse a daemon, but great rewards pay off later. The community acknowledged “first iPhone call recording App”, i.e. Audio Recorder, is accomplished by reversing mediaserverd.

## Conclusion

This chapter simply introduces iOS system hierarchy and file types, which are not necessities for App Store developers, who don’t even have an official way to learn about the concepts. This chapter’s intention is to introduce you the very important yet undocumented system level knowledge, which is essential in iOS reverse engineering.

In fact, every section in this chapter can be extended into another full chapter, but as beginners, knowing what we’re talking about and what to google when you encounter problems during iOS reverse engineering is enough. If you have anything to say, welcome to <http://bbs.iosre.com>.