# Part 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.

# Chapter 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 an 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 castle. 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 implementation and design details will 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 who 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 can be 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 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’re 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 are 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 coexist for always. 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, 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 restore 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 ideas 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.

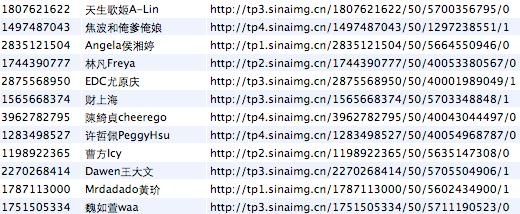


Figure 1-3 Sina Weibo database

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 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 as well as development kit, respectively.

### 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.

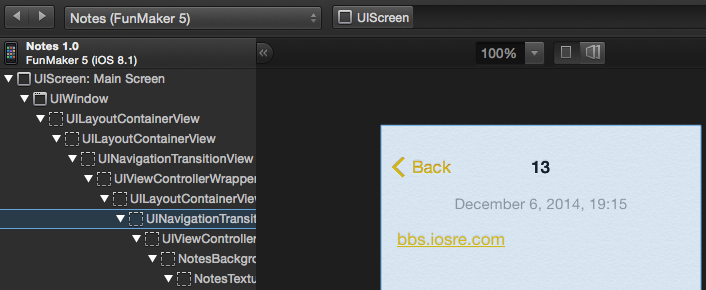


Figure 1- 4 Reveal

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.

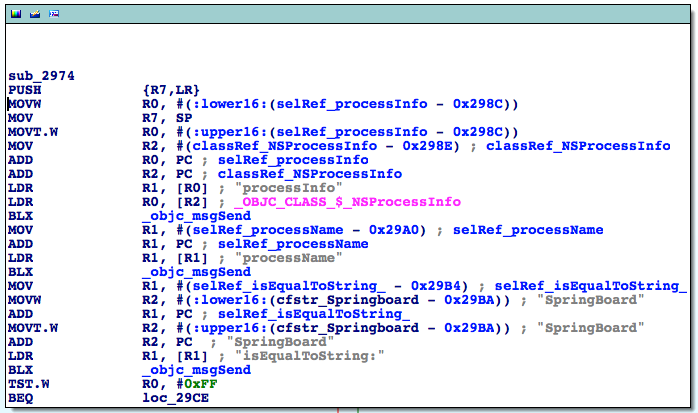


Figure 1- 5 IDA

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

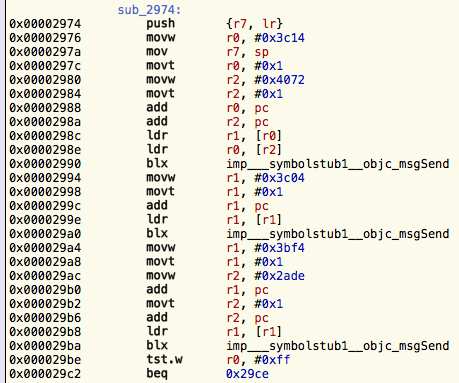


Figure 1- 6 Hopper

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 since chapter 6. 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 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.

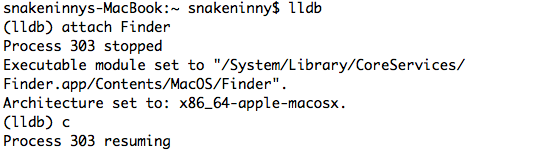


Figure 1- 7 LLDB

### 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!