

iOS客户端安全测试指南

■ 文档编号	■ 密级	内部使用
■ 版本编号 3.1	■ 日期	2015-03-20



■ 版权声明

本文中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明，版权均属**绿盟科技**所有，受到有关产权及版权法保护。任何个人、机构未经**绿盟科技**的书面授权许可，不得以任何方式复制或引用本文的任何片断。

■ 版本变更记录

时间	版本	说明	修改人
2012-9-5	1.0	创建	尚进
2013-09-09	2.0	更新文档结构，部分内容更新。添加第三章	尚进
2014-10-30	2.3	补充 hook 部分内容，添加证书相关内容	尚进，曾海涛
2015-01-09	3.0	添加威胁等级部分 添加手势密码安全性部分 添加 2.7.6 节	黄灿
2015-03-18	3.1	补充了 ios8 系统下的应用路径 ios8 下新的录屏软件使用 部分命令的更新 theos 环境配置	曾海涛

目录

IOS 客户端安全测试指南.....	1
一、 环境要求.....	4
二、 安全测试项目列表.....	5
2.1 客户端程序安全.....	5
2.1.1 *程序包签名.....	5
2.1.2 客户端程序保护.....	5
2.1.3 应用完整性检测.....	6
2.2 敏感信息安全.....	7
2.2.1 数据文件.....	8
2.2.2 系统日志.....	8
2.2.3 密钥链数据.....	9
2.3 密码软键盘安全.....	10
2.3.1 随机布局软键盘.....	10
2.3.2 键盘记录防护.....	10
2.3.3 屏幕录像.....	11
2.3.4 键盘缓存检测.....	13
2.4 安全策略设置.....	14
2.4.1 密码复杂度检测.....	14
2.4.2 帐号登录限制.....	14
2.4.3 帐号锁定策略.....	15
2.4.4 *私密问题验证.....	15
2.4.5 会话安全设置.....	15
2.4.6 界面切换保护.....	16
2.4.7 UI 信息泄露.....	16
2.4.8 验证码安全性.....	16
2.4.9 安全退出.....	17
2.4.10 密码修改验证.....	17
2.5 进程保护.....	17
2.5.1 进程内存访问.....	20
2.5.2 *动态注入.....	20
2.6 通信安全.....	21
2.6.1 通信加密.....	21
2.6.2 证书有效性检测.....	21
2.6.3 关键字段加密和校验.....	23
2.6.4 *访问控制.....	24
2.6.5 客户端更新安全性.....	25
2.7 业务功能测试.....	25

三. 测试项目风险定级	27
四. 合规性参考	29
五. iOS 应用分析	30
5.1 获得 iOS 应用程序包	30
5.1.1 导出 ipa	30
5.1.2 iOS 系统中的*.app	30
5.2 查看签名信息	33
5.2.1 查看签名	33
5.2.2 查看 entitlement	33
5.2.3 查看 requirement	34
5.3 反编译	34
5.3.1 Universal Binary 预处理	35
5.3.2 定位入口点	36
5.3.3 可视化 Mach-O 文件查看	36
5.3.4 文件解密处理	37
5.3.5 还原类定义	37
5.3.6 IDA pro	38
5.3.7 Hopper	39
5.4 处理资源文件	39
5.4.1 还原 png 图片	39
5.4.2 还原 plist	40
5.4.3 iTunes 导出	40
5.4.4 查看*.mobileprovision 文件	40
5.4.5 查看 mobileconfig 文件	42
5.4.6 分析 sqlite 数据库日志文件	43
5.5 生成 iOS 应用（越狱后）	44
5.5.1 生成证书	44
5.5.2 准备 xcode	45
5.5.3 编译 iOS 命令行程序	46
5.5.4 编译 iOS dylib	49
5.5.5 代码签名	50
5.5.6 打包 ipa	51
5.6 动态调试	52
5.6.1 使用 Xcode	52
5.6.2 使用 gdb	52
5.7 基于 hook 的测试工具	54
5.7.1 Snoop-it	54
5.7.2 Theos	56
5.7.3 cycript	57
5.8 其他命令	58
5.8.1 嗅探流量 tcpdump	58

5.8.2	网络信息查看 netstat	59
5.8.3	进程查看和监视 ps	59
5.8.4	文件列举 lsof	59
5.8.5	数据库文件查看 sqlite3	60
5.8.6	动态共享库修改 Install_name_tool	61
5.8.7	对象文件查看工具 otool	61
5.8.8	密钥链查看工具 Keychaindumper	61
5.8.9	签名工具 ldid	62
5.9	根证书安装	62
5.10	连接到 iOS 设备	63
5.10.1	Windows 平台	63
5.10.2	Mac OSX 平台	63
5.11	UI 元素查看	64
5.11.1	Reveal	64
5.11.2	Spark inspector	65
附录 A	疑难解答	68

一. 环境要求

SDK: 本文使用 Xcode 4.2 (Mac OSX 10.6.8), iOS 6.1。

Mac 工具: iTools, 010 editor, class-dump, iOSOpenDev, Hopper, MachOView, Homebrew
iPhone/iPad 工具:

1. 软件源: Telesphoreo (apt.saurik.com), Snoop-it (repo.nesolabs.de), radare (cydia.radare.org), xSellize (cydia.xsellize.com)。
2. 图形化工具: iFile, iKeyMonitor (或 iKeyGuard), DisplayRecorder (或 RecordMyScreen), snoop-it, 八门神器。
3. 命令行工具: sqlite3, tcpdump, gdb, file, open, keychain_dumper, cycrypt, clutch, lsof, openssh, AppSync, 另有命令行工具如下表所示

命令	cydia 软件包名称
ping, telnet	inetutils
netstat	Network—cmd
otool, lipo, nm	odcctools
kill, su	Core utilities
ps	adv-cmds
ldid	Link Identity Editor

注意: 安装上述软件包时, 需要在 cydia 管理->设置中将身份设置为“开发者”(Cydia 1.1.10 后的版本无需此设置)。

有用的链接:

https://www.owasp.org/index.php/IOS_Application_Security_Testing_Cheat_Sheet

<http://theiphonewiki.com/>

https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjCRuntimeGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008048

<http://resources.infosecinstitute.com/pentesting-iphone-applications/>

二. 安全测试项目列表

注：下面的测试项目中标记*的为可选测试项，在正式测试中可以不进行测试。

2.1 客户端程序安全

2.1.1 *程序包签名

检测客户端是否经过正确签名（正常情况下应用都应该是签名的，否则无法在未越狱的 iOS 上运行）。

测试方法：

参考 4.2 查看签名信息。如图，当输出结果为“valid on disk”时，表示签名正常。

```
localhost:MPBBank.app asky$ codesign -vv /Users/asky/Desktop/招商银行\ 1.4.0/Payload/MPBBank.app
/Users/asky/Desktop/招商银行 1.4.0/Payload/MPBBank.app: valid on disk
/Users/asky/Desktop/招商银行 1.4.0/Payload/MPBBank.app: satisfies its Designated Requirement
localhost:MPBBank.app asky$
```

从 Apple store 下载的应用，其签名信息中的 authority 应类似下图所示：

```
Mac-Pro:Downloads asky$ codesign -dvvv ICBCiPhoneBank.app/
Executable=/Users/asky/Downloads/ICBCiPhoneBank.app/ICBCiPhoneBank
Identifier=com.icbc.iphoneclient
Format=bundle with Mach-O universal (armv7 (12:11))
CodeDirectory v=20100 size=55690 flags=0x0(none) hashes=2776+5 location=embedded
Hash type=sha1 size=20
CDHash=7af0ae83f10fc38ab552d517f2bf8e4e71c58210
Signature size=3487
Authority=Apple iPhone OS Application Signing
Authority=Apple iPhone Certification Authority
Authority=Apple Root CA
Info.plist entries=28
Sealed Resources rules=5 files=244
Internal requirements count=1 size=104
Mac-Pro:Downloads asky$ _
```

威胁等级：

通常情况下非越狱手机只有使用开发者账号上架到 APP Store 才可下载，一般不会出现此类问题，若签名异常为低风险。

2.1.2 客户端程序保护

1. *代码混淆。

测试客户端安装程序，判断是否包含调试符号信息，是否能反编译为源代码，是否存在代码保护措施。

测试方法:

参考 4.1ipa 解包, 4.3 反编译, 对客户端程序文件进行逆向分析。

这一项一般不测。人工很难判断一个 iOS 应用是否混淆。Object-C 一般也不需要混淆。

外部参考: [Secure Mac Programming: On private methods](#) (需要开发经验)

2. * 远程调试。

检查客户端程序在 Entitlements.plist 文件或内嵌 Entitlements 中是否包含调试设置。

测试方法:

没有 Entitlements.plist 文件时, 参考 4.2 查看签名信息查看内嵌的数据。通过 codesign 查看应用中是否内嵌有相关 Entitlements 设置, 如图所示:

```
localhost:- asky$ codesign -d --entitlement - /Users/asky/Library/Developer/Xcode/
e/DerivedData/hello_world-awiqoyxttrskiaftkftknknxdlem/Build/Products/Debug-iphon
eos/hello\ world.app/
Executable=/Users/asky/Library/Developer/Xcode/DerivedData/hello_world-awiqoyxtt
rskiaftkftknknxdlem/Build/Products/Debug-iphoneos/hello world.app/hello world
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Can be debugged</key>
    <true/>
    <key>get-task-allow</key>
    <true/>
</dict>
</plist>
localhost:- asky$
```

看Entitlements信息中“Can be debugged”属性值是否为“NO”, “get-task-allow”属性值是否为“NO”。为“NO”时无法通过xcode调试。(gdb本地调试不受影响; 如需防护gdb调试可参考 [此处](#))

其他敏感属性(一般应用不应该有)有: run-unsigned-code, task_for_pid-allow, dynamic-codesigning等等。可参考Apple的开发 [文档](#)。

威胁等级:

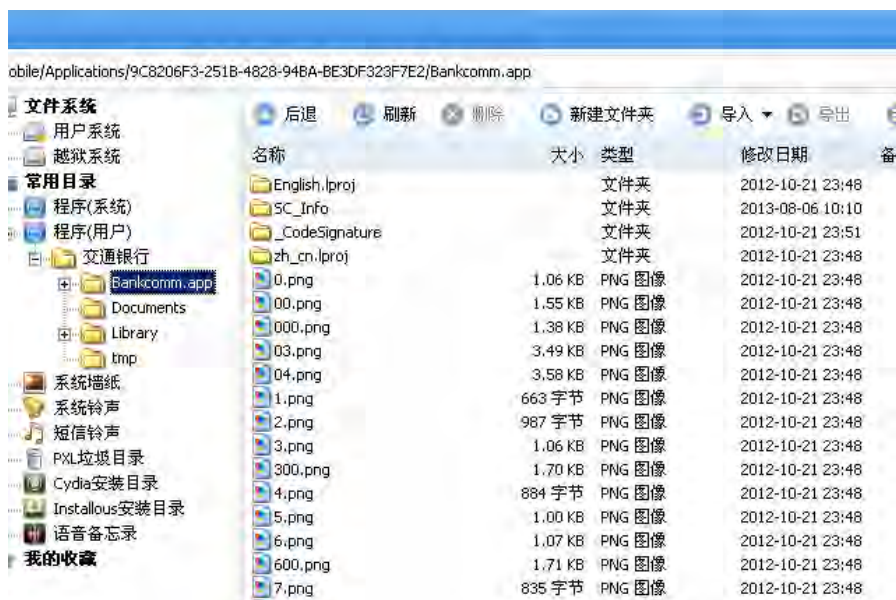
若客户端未进行代码混淆低风险(Objective-C 允许不混淆代码), 不存在时无风险。

2.1.3 应用完整性检测

测试客户端程序是否对自身进行完整性校验。客户端程序如果没有自校验机制的话, 攻击者有可能通过篡改客户端程序, 显示钓鱼信息欺骗用户, 窃取用户的隐私信息。

测试方法:

人工检测。可参考 5.1 获得 iOS 应用程序包和 5.4 处理资源文件，修改客户端程序文件或其他资源文件，看客户端是否能够运行。（推荐修改配置文件或其他文本文件或图片，使客户端显示钓鱼信息）



客户端程序文件均保存在应用私有目录的*.app 文件夹下。可以寻找文件夹中的配置文件和文本文件，对能够影响程序运行的文件进行修改（可以通过文件名和文件类型进行推测，首选修改对象是 html、js 脚本和配置文件等）。修改后需要重新运行客户端，即把已运行的客户端程序进程杀死，然后再次运行。

注意：应用在提交给 Apple Store 后其可执行文件会被修改，所以开发时不能将自身 Hash 硬编码进程序中。建议应用通过 Apple 的数字签名机制来判断是否被篡改。

(TODO:

跨应用调用

CFBundleURLTypes

)

威胁等级：

若客户端启动时不进行应用完整性检测高风险；若只对原有文件进行检测而忽略是否存在新增文件时中风险，不存在上述情况无风险。

2.2 敏感信息安全

检测客户端是否保存敏感信息，能否防止用户敏感信息的非授权访问。

2.2.1 数据文件

检查客户端程序存储在手机中的配置文件，防止账号密码等敏感信息保存在本地。

测试方法：

对客户端私有目录（具体位置可参考 5.1.2 iOS 系统中的*.app）下的私有数据文件内容进行检查，看是否包含敏感信息。iOS 客户端通常会保存配置文件、web 缓存和切出时的截图。

检查保存的敏感信息是否进行了加密，加密算法是否安全（例如，通常认为采用 base64 的编码方法或是使用硬编码密钥的加密是不安全的）。

对 sqlite 数据库文件（通常文件名以.db、.localstorage、.sqlite 等结尾），可参考 4.8.5 数据库文件查看 sqlite3，检查数据库中是否包含敏感信息，也可以使用 SQLite Studio（SQLite Database Browser 版本较老，不推荐使用）查看和修改 sqlite 文件。如果数据库文件包含以 wal 和 shm 结尾的文件，还可以使用二进制编辑器查看文件中的数据，或是参考 5.4.6 分析 sqlite 数据库日志文件尝试恢复删除的数据。

威胁等级：

当客户端存在明文用户敏感信息或服务器信息时高风险，若存在安全性低的敏感信息时高风险，不存在上述情况无风险。

2.2.2 系统日志

检查客户端程序存储在手机中的日志。

测试方法：

使用 iTools 高级功能中的系统日志（实时日志）功能查看系统日志，检查日志是否包含敏感信息。如图所示：



威胁等级:

当日志中会显示用户输入的信息（包括用户名、明文密码或单次哈希的密码）、用户访问服务器的 URL 和端口等核心敏感信息时为高风险；当日志中会显示调用逻辑或一些可供攻击者猜测逻辑的报错时为中风险；当日志中会打出除上述外的一些开发商的调试信息时为低风险；如果不存在上述情况则无风险。

2.2.3 密钥链数据

检查客户端是否在密钥链（Keychain）中存放明文密码。

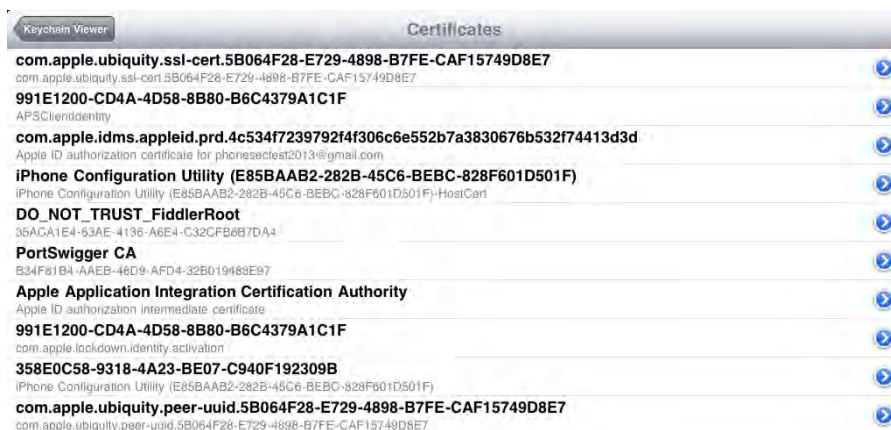
测试方法:

可以参考 4.7.1 Snoop-it 工具使用方法，在 Keychain 功能中查看客户端对密钥链的访问；或者参考 4.8.8 密钥链查看工具 Keychaindumper，导出所有密钥链中的数据。如图所示：

```
Generic Password
-----
Service: com.apple.itunesstored.token
Account: phonesectest2013@gmail.com
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: AwIAAFWAAQxgAAAAABs+xxmecaERj7BvRZhC8oUUm3tHn5FJA=

Generic Password
-----
Service: com.jflan.MiniKeePass.passwords
Account: tttt.kdb
Entitlement Group: 4BWNF7HW3R.com.jflan.MiniKeePass
Label: (null)
Generic Field: (null)
Keychain Data: tttt
```

也可以使用 [KeychainViewer](#) 工具查看，如图：



注意：Keychain数据保存在/private/var/Keychains文件夹下。关于keychain中数据的讲解可参考 [此处](#)。

威胁等级：

当密钥链中保存了用户或设备敏感信息时高风险，否则无风险。

2.3 密码软键盘安全

2.3.1 随机布局软键盘

测试客户端程序在登录 / 交易密码等输入框是否使用自定义软键盘，是否满足键位随机布放要求。

测试方法：

人工检测。

威胁等级：

当客户端软键盘未进行随机化处理时为低风险；当客户端软键盘只在某一个页面载入时初始化一次而不是在点击输入框时重新进行随机化也为低风险。

2.3.2 键盘记录防护

测试客户端能否防止键盘记录工具记录密码。（当使用自定义软键盘时，即可防护）

测试方法：

安装 iKeyGuard/iKeyMonitor 工具。运行客户端，输入数据进行测试。如果客户端不能防键盘记录，则输入会被记录下来。如图所示：



注意：未注册的 iKeyGuard 只能记录每个应用的第一个输入字符串（需要清除已有记录才能再次记录新的输入）。从 cydia 安装的第三方输入法有可能导致 iKeyGuard 无法正常工作（如搜狗输入法）。

（iOS 的键盘记录通过注入动态库实现，也许可以通过检测加载的动态库来判断是否存在键盘记录器）

威胁等级：

当客户端可以抵抗键盘记录攻击时无风险，不能抵抗时高风险。

2.3.3 屏幕录像

客户端使用的随机布局软键盘是否会对用户点击产生视觉响应。当随机布局软键盘对用户点击产生视觉响应时，木马可以通过连续截屏的方式，对用户击键进行记录，从而获得用户输入。

测试方法：

按住锁屏键，然后按下 Home 键，即可截图。将设备连接到电脑上，在“我的电脑”的 iPhone 设备里可以找到图片。

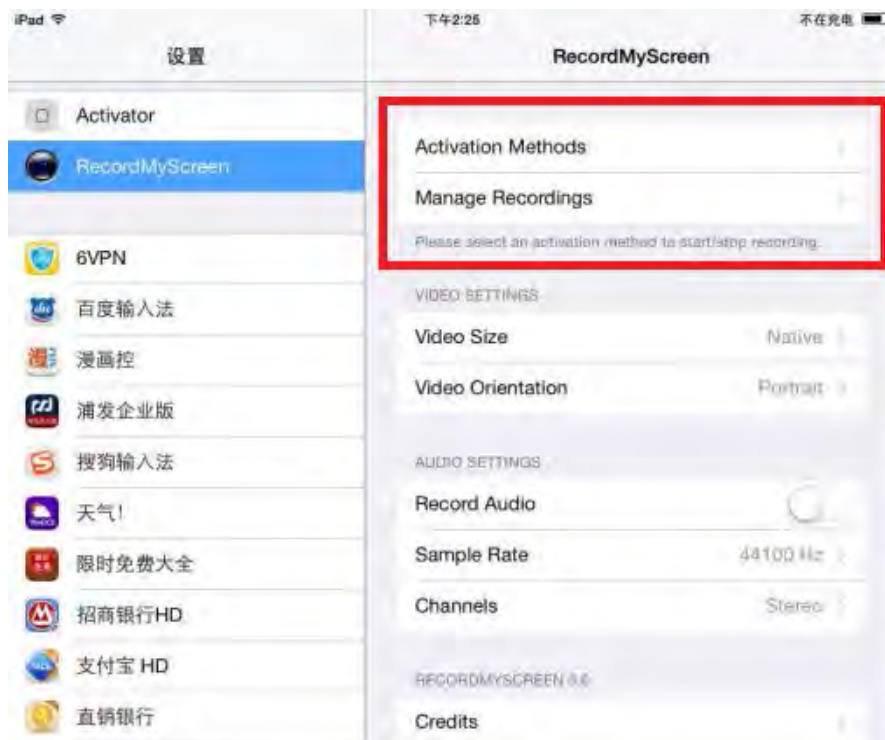
安装 Display Recorder（或者 RecordMyScreen），记录软键盘的输入过程。如图是按下按键时的截屏。（录像文件保存在/var/mobile/Library/Keyboard/DisplayRecorder/或/var/mobile/Documents/目录下）



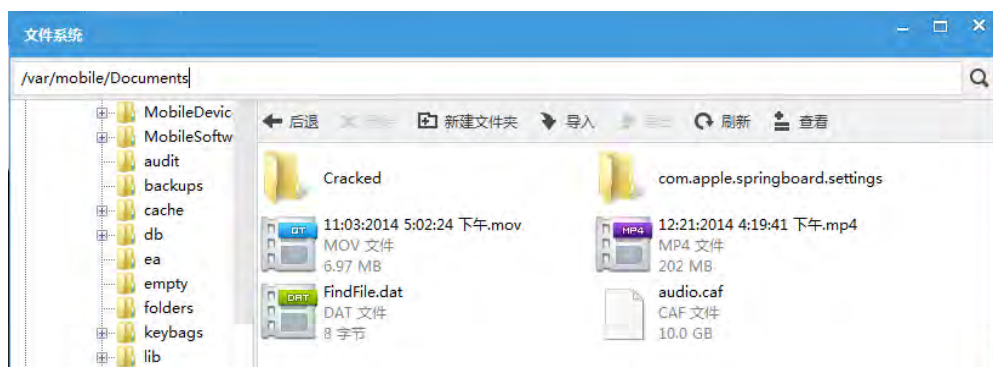
在 ios8+ 的系统中，推荐使用 RecordMyScreen。



安装之后，在设置选项下点击 RecordMyScreen，即可对工具进行配置，其中 Activation Methods 可设置该 tweak 的手势唤醒选项，Manage Recording 可浏览录屏的记录。



录屏文件目录



威胁等级：

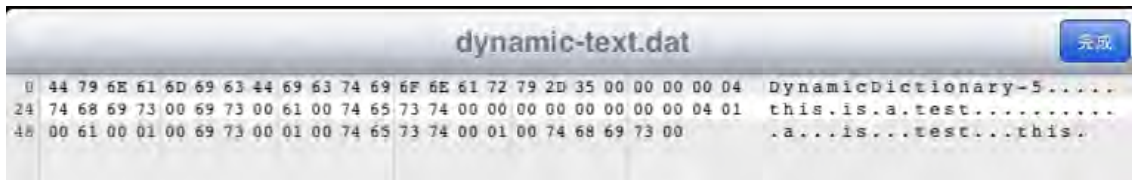
当使用第三方案序（或系统截屏）可以对客户端内容进行截屏时，为中风险；当客户端未检测手机是否越狱并进行用户提示时低风险；当客户端会对截屏操作进行有效抵抗时（无法截屏或截屏结果为黑屏等无意义图片）无风险。

2.3.4 键盘缓存检测

当客户端使用系统键盘时，有可能会在键盘缓存中保存用户输入（如当“自动改正”打开时）。如果用户输入没被保护，有可能造成敏感信息泄漏。

测试方法：

首先在设置->通用->还原中还原键盘字典，以免过去的的数据干扰测试结果。然后检查 /var/mobile/Library/Keyboard/ 中的数据文件。对于 ios6，可检查 dynamic-text.dat；对于 ios8，可检查 PhraseLearning_zh_Hans.db.bundle 或 en-dynamic.lm 等目录中的文件。下图是用二进制编辑器打开 dynamic-text.dat 所看到的输入记录：



威胁等级：

当客户端使用系统键盘并且键盘缓存中保存用户敏感信息时高风险，上述两者满足其一
时中风险，都不存在时无风险。

2.4 安全策略设置

2.4.1 密码复杂度检测

测试客户端是否检查用户输入密码的强度，是否覆盖常见的弱口令，以免木马通过字典攻击破解用户密码。手机银行的登录密码是否与交易密码不同，是否与银行卡交易密码不同。

测试方法：

人工测试，尝试将密码修改为弱口令，如：123456，654321，121212，888888 等，查看客户端是否提示或拒绝弱口令。查看客户端登录使用的是否是专门设定的登录密码，且与交易密码不同。（6.3.2.1）

威胁等级：

当系统允许用户设置弱密钥时为低风险，如果存在系统存在一定的安全策略（密码使用数字和字母组成，至少为 8 位）时无风险。

2.4.2 帐号登录限制

测试一个帐号是否可以同时在多个设备上成功登录客户端，进行操作。

测试方法：

人工测试。

威胁等级：

若同一个账号可以同时多台移动终端设备上登陆时为低风险，若不可以则无风险。

2.4.3 帐号锁定策略

测试客户端是否限制帐号错误登录的尝试次数。防止木马使用穷举法暴力破解用户密码。

测试方法：

人工测试。错误尝试次数不能超过 10 次。建议最后测试此项，先通过多次错误尝试锁定帐号后，再使用正确的密码登录。如能登录，即存在安全风险。（6.3.2.1）

威胁等级：

当系统不存在账户锁定策略时为中风险，若存在则无风险。

2.4.4 *私密问题验证

如果对账号某些信息（如单次支付限额）的修改需要私密问题验证，测试私密问题验证是否将问题和答案一一对应。

测试方法：

人工测试。选择一个私密问题，回答另一个问题的正确答案，测试是否可以成功验证。

威胁等级：

当用户进行忘记密码操作时，在发送邮件给用户邮箱前是否进行私密问题的验证，若验证则无风险；若不验证则低风险。

2.4.5 会话安全设置

测试客户端在一定时间内无操作后，是否会使会话超时并要求重新登录。超时时间设置是否合理。

测试方法：

人工测试。一般在 20 分钟内会话超时的就算安全。

威胁等级：

当系统不存在会话超时逻辑判断时为低风险，若存在则无风险。

2.4.6 界面切换保护

检查客户端程序在切换到其他应用时，已经填写的账号密码等敏感信息是否清空，防止用户敏感信息泄露。如果切换前处于已登录状态，切换后一定时间内是否会自动退出当前会话。

测试方法：

人工测试。在登录界面（或者转账界面等涉及密码的功能）填写登录名和密码，然后切出，再进入客户端，看输入的登录名和密码是否清除。登录后切出，5 分钟内自动退出为安全。

注意：某些应用可以在设置中配置“后台保持在线时长”。

威胁等级：

当移动终端设备进行进程切换操作，显示界面不为客户端页面时，若客户端提示用户确认是否为本人操作，则无风险；若无相应提示则为低风险。

2.4.7 UI 信息泄露

检查客户端的各种功能，看是否存在敏感信息泄露问题。

测试方法：

人工测试。使用错误的登录名或密码登录，看客户端提示是否不同。在显示卡号等敏感信息时是否进行部分遮挡。

威胁等级：

若在用户名输入错误和密码输入错误时提示信息不同则存在 UI 信息泄露问题，此时为低风险，否则无风险。

2.4.8 验证码安全性

测试客户端在登录和交易时是否使用图形验证码。验证码是否符合如下要求：由数字和字母等字符混合组成；采取图片底纹干扰、颜色变换、设置非连续性及旋转图片字体、变异字体显示样式等有效方式，防范恶意代码自动识别图片上的信息；具有使用时间限制并仅能使用一次；验证码由服务器生成，客户端文件中不包含图形验证码文本内容。

测试方法：

人工测试。（6.1.4.4）

威胁等级：

当图形验证码由本地生成而不是从服务器获取时为中风险；当验证码安全性低或不存在验证码时为中风险；不存在以上两个问题时无风险。

2.4.9 安全退出

测试客户端退出时是否正常终止会话。

测试方法：

检查客户端在退出时，是否向服务端发送终止会话请求。客户端退出后，还能否使用退出前的会话 id 访问登录后才能访问的页面。（6.1.4.4）

威胁等级：

若客户端退出登录时不会和服务器进行Logout的相关通信则为中风险，否则无风险。

2.4.10 密码修改验证

测试客户端在修改密码时是否验证旧密码正确性。

测试方法：

人工测试。

威胁等级：

当进行密码修改时是否要求输入原密码已验证其正确性，若需要输入则无风险；如不需输入原密码则中风险。

2.5 手势密码安全性

2.5.1 手势密码复杂度

测试客户端手势密码复杂度，观察是否有点位数量判断逻辑。

测试方法：

1. 进入客户端设置手势密码的页面进行手势密码设置。
2. 进行手势密码设置，观察客户端手势密码设置逻辑是否存在最少点位的判断。

威胁等级：

当用户设置或修改手势密码时服务器会对手势密码安全性（使用点数）进行判断时无风险，否则低风险。

2.5.2 手势密码修改和取消

检测客户端在取消手势密码时是否会验证之前设置的手势密码，检测是否存在其他导致手势密码取消的逻辑问题。

测试方法：

1. 进入客户端设置手势密码的位置，一般在个人设置或安全中心等地方。
2. 进行手势密码修改或取消操作，观察进行此类操作时是否需要输入之前的手势密码或普通密码。
3. 观察在忘记手势密码等其他客户端业务逻辑中是否存在无需原始手势或普通密码即可修改或取消手势密码的情况。
4. 多次尝试客户端各类业务，观察是否存在客户端逻辑缺陷使得客户端可以跳转回之前业务流程所对应页面。若存在此类逻辑（例如手势密码设置），观察能否修改或取消手势密码。

威胁等级：

当取消或修改手势密码时，如果不会验证之前的手势密码则为中风险；若存在验证则无风险。

2.5.3 手势密码本地信息保存

检测在输入手势密码以后客户端是否会在本地记录一些相关信息，例如明文或加密过的手势密码。

测试方法：

1. 首先通过正常的操作流程设置一个手势密码并完整一次完整的登陆过程。
2. 寻找私有目录下是否存在手势密码对应敏感文件，若进行了相关的信息保存，基本在此目录下。（关键词为 gesture, key 等）

3. 若找到对应的文件，观察其存储方式，为明文还是二进制形式存储，若为二进制形式，观察其具体位数是否对应进行 MD5（二进制 128 位，十六进制 32 位或 16 位）、SHA-1（二进制 160 位，十六进制 40 位）等散列后的位数。
4. 通过代码定位确认其是否进行了除单项哈希散列之外的加密算法，若客户端未将手势密码进行加密或变形直接进行散列处理可认为其不安全，一是因为现阶段 MD5、SHA-1 等常用的哈希算法已被发现碰撞漏洞，二是网络中存在 www.cmd5.com 等散列值查询网站可以通过大数据查询的方式获取散列前的明文手势密码。

安全建议：

建议不在客户端保存任何与手势密码相关的敏感信息（最好也不要加密存储）。建议在保证通信安全的情况每次由服务器对手势密码正确性进行验证。

威胁等级：

当本地保存了明文存储（数组形式）的手势密码时为高风险；当本地保存了只进行单项哈希散列的手势密码时为中风险。

2.5.4 手势密码锁定策略

测试客户端是否存在手势密码多次输入错误被锁定的安全策略。防止木马使用穷举法暴力破解用户密码。因为手势密码的存储容量非常小，一共只有 $9! = 362880$ 种不同手势，若手势密码不存在锁定策略，木马可以轻易跑出手势密码结果。

手势密码在输入时通常以 `a[2][2]` 这种 3×3 的二维数组方式保存，在进行客户端同服务器的数据交互时通常将此二维数组中数字转化为类似手机数字键盘的 `b[8]` 这种一维形式，之后进行一系列的处理进行发送。

测试方法：

1. 首先通过正常的操作流程设置一个手势密码。
2. 输入不同于步骤 1 中的手势密码，观察客户端的登陆状态及相应提示。若连续输入多次手势密码错误，观察当用户处于登陆状态时是否退出当前的登陆状态并关闭客户端；当客户未处于登录状态时是否关闭客户端并进行一定时间的输入锁定。

威胁等级：

当服务器不会验证手势密码输入错误次数时为中风险，会进行验证时无风险。

2.6 进程保护

2.6.1 进程内存访问

通过读取客户端内存，寻找内存中可能存在的敏感信息（卡号、明文密码等等）。

测试方法：

运行被测客户端，然后使用 SSH 隧道连接设备（如果在设备上使用 MobileTerminal，需要先切出客户端应用，可能会使被测客户端退出）。参考 4.8.3 进程查看和监视 ps，获得被测客户端的 PID。然后使用内存搜索工具搜索字符串，如图所示（目前内存搜索工具只支持搜索 ASCII 字符串，可以使用八门神器搜索数字）。

```
std:/var/mobile root# mem-check -p 1979 -s test
try open mem
Done open
search for test
found data : test_synchronised (object)
found data : test. Flags: %0
found data : test
found data : test
found data : tester.unity3d.com
found data : testerbeta.unity3d.com
found data : tester address, you must be connected to the internet before performing this or set the address to somet
hing accessible to you.
found data : test
found data : tester is not accepting new connections, test finished.
found data : test finished.
found data : tester at %s
found data : tester has banned this connection.
found data : tester.
found data : tester.
```

或参考 4.6.2 使用 gdb。使用 gdb 查看和修改客户端内存数据。

威胁等级：

当进行敏感操作后在内存中可以搜索到用户输入的敏感信息时为高风险，否则无风险。

2.6.2 *动态注入

测试能否通过向应用中注入动态库并 hook 函数，获取用户敏感信息。

测试方法：

针对 C 函数，可以检测 DYLD_INSERT_LIBRARIES 环境变量。iOS dyld 库会自动加载此环境变量中包含的动态共享库。如图所示：（可能还需要设置

DYLD_FORCE_FLAT_NAMESPACE=1，<http://hacetheplanet.com/blog/80>）

```
hellonihaojilimato-iPhone:~/Applications/mem.app mobile$ export DYLD_INSERT_LIBRARIES="./testc22.dylib"
jilimato-iPhone:~/Applications/mem.app mobile$ ./mem
hello hook
hellonihaojilimato-iPhone:~/Applications/mem.app mobile$
```

针对Object-C函数，可以使用 [MobileSubstrate](#) 框架注入并Hook函数，具体用法可参考 [此处](#)。（需要有一定的ios开发经验）

还可以使用 `cycrypt`，动态注入应用。具体用法可参考 5.7.3cycrypt。

2.7 通信安全

2.7.1 通信加密

客户端和服务端通信是否强制采用 `https` 加密。

测试方法：

参考 4.8.1 嗅探流量 `tcpdump`，使用 `tcpdump` 嗅探客户端提交的数据，将其保存为 `pcap` 文件。使用 `Wireshark` 打开 `pcap` 文件，检查交互数据是否是 `https`。

如果可能，将客户端链接到的地址改为 `http`（将所有 URL 开头的 `https` 改为 `http`），查看客户端是否会提示连接错误。

威胁等级：

当客户端和服务器的通信不经过 `SSL` 加密（或没有参考 `TLS` 协议，`RFC4346` 等实现加密信道）时为高风险；当自实现通信算法存在漏洞可被解析或绕过时为高风险；使用低版本 `SSL` 协议（`SSLV2`，`SSLV3` 均存在漏洞，至少使用 `TLSV1.1` 以上算法）时为高风险；以上问题均不存在时无风险。

2.7.2 证书有效性检测

（注：测试项 1 和 2 分别测试对证书本身有效性的验证以及对证书中 `CN` 的检查，目前 `android` `java` 和 `iOS openssl` 库中这两项检测的实现都是独立的）

1. 检查服务端证书是否是 `CA` 签发，客户端程序和服务器端的 `SSL` 通信是否严格检查服务器端证书的有效性。避免手机银行用户受到中间人攻击后，密码等敏感信息被嗅探到。

测试方法：

通过 `wifi` 将手机和测试 `PC` 连接到同一子网。然后在手机中设置代理，`Host` 中填入测试 `PC` `IP` 地址，如图所示。将客户端流量转发给测试 `PC` 上的代理工具（如 `fiddler`）。然后使用客户端访问服务端，查看客户端是否会提示证书无效。



2. *测试客户端程序是否严格核对服务器端证书信息，避免手机银行用户访问钓鱼网站后，泄露密码等敏感信息。

测试方法：

通过修改 DNS 的方式(修改/etc/hosts)，将客户端链接到的地址改为 <https://mail.qq.com/>（或其他拥有有效证书的 https URL），然后使用客户端访问服务端，查看客户端是否会提示连接错误。此项测试主要针对客户端是否对 SSL 证书中的域名（CN）以及自己正在连接的域名进行确认。

外部参考：OWASP针对证书验证问题，提出的[Certificate Pinning解决方案](#)。

3. *SSL 协议安全性。检测客户端使用的 SSL 版本号是否不小于 3.0（或 TLS 1.0），加密算法是否安全。（安全规范要求）

测试方法：

使用openssl，指定域名和端口，可以看到SSL连接的类型和版本。如下图所示，使用了 TLSv1，加密算法为AES 256 位密钥。（也可以使用 [SSL labs网站](#)检测）（对称加密算法中 RC4，DES等被认为是不安全的，采用的密钥长度应大于等于 128bits；非对称算法推荐RSA / DSA，密钥长度大于等于 2048bits。参考 [OWASP](#)）

```
asky@shangjin:~$ openssl s_client -connect login.yahoo.com:443 -CApath /etc/ssl/certs/
CONNECTED(00000003)
depth=3 C = US, O = GTE Corporation, OU = "GTE CyberTrust Solutions, Inc.", CN = GTE CyberTrust Global Root
verify return:1
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance CA-3
verify return:1
depth=0 C = US, ST = CA, L = Sunnyvale, O = Yahoo! Inc., CN = login.yahoo.com
verify return:1

Certificate chain
 0 s:/C=US/ST=CA/L=Sunnyvale/O=Yahoo! Inc./CN=login.yahoo.com
  i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance CA-3
 1 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance CA-3
  i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance EV Root CA
 2 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance EV Root CA
  i:/C=US/O=GTE Corporation/OU=GTE CyberTrust Solutions, Inc./CN=GTE CyberTrust Global Root

Server certificate
-----BEGIN CERTIFICATE-----
MIIGVzCCBbOgAwIBAgIQDAcE1cvEJwsSIUWeAyB69zANBgkqhkiG9w0BAQUFADBm
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Sunnyvale/O=Yahoo! Inc./CN=login.yahoo.com
issuer=/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance CA-3

No client certificate CA names sent

SSL handshake has read 4813 bytes and written 646 bytes

New, TLSv1/SSLv3, Cipher is AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS supported
Compression: zlib compression
Expansion: zlib compression
SSL-Session:
    Protocol : TLSv1
    Cipher   : AES256-SHA
    Session-ID: DED3A0D5639B156B071B071FF6FD247BD942A72473BADE0D80B5FEFD87ED3C78
    Session-ID-ctx:
    Master-Key: 617AC12CA13D7CEDC5EA9F6657320D2F22D8201EB6CE58CAA94743FD2BE62E2525407EDA7CAEE
    Key-Arg : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    TLS session ticket:
```

威胁等级:

当客户端和服务端互相不验证证书时高风险，当只有客户端验证服务器证书时为中风险；当服务器不通过白名单的方式验证客户端时为中风险；当客户端和服务端进行双向认证，并且服务器通过白名单方式验证客户端证书时无风险。

2.7.3 关键字段加密和校验

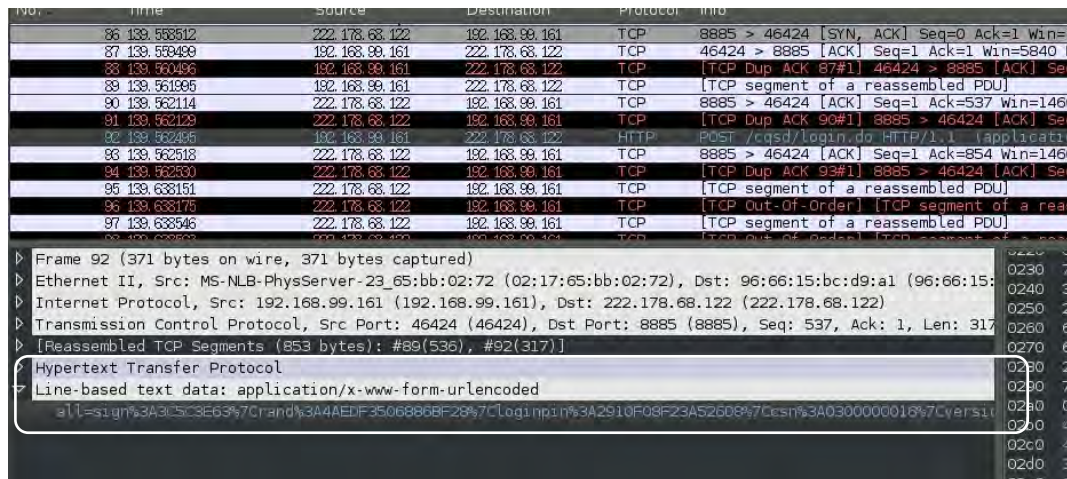
1. 测试客户端程序提交数据给服务端时，密码等关键字段是否进行了加密，防止恶意用户嗅探到用户数据包中的密码等敏感信息。

测试方法:

参考 2.6.2 证书有效性检测一节中配置代理的方式，使用代理工具截获通信数据。如果客户端采用 HTTPS 通信，并通过 iOS 系统框架对 SSL 证书进行验证，则需要先在 iOS 设备上

安装代理的根证书。对自带根证书的客户，则可以参考 2.1.2 客户端程序保护用代理根证书替换客户端的根证书。

或参考 4.8.1 嗅探流量 tcpdump, 使用 tcpdump 嗅探客户端提交的数据, 将其保存为 pcap 文件。使用 Wireshark 打开 pcap 文件, 对交互数据进行分析。检查关键字端是否加密, 是否包含签名。如下图所示:



2. 测试客户端和服务端的交互数据是否进行签名, 防止提交和接收的数据被恶意篡改。

测试方法:

参考 2.6.2 证书有效性检测一节中配置代理的方式。在代理工具中篡改交互数据, 检查服务端和客户端是否能检测到篡改。对资金交易等高风险业务, 需检查签名数据是否包含随机信息 (即同样的交易数据每次签名不同)。

注: 当客户端使用了 Cert Pinning, 无法直接使用代理时, 可以尝试使用 [ios-ssl-kill-switch](#) 工具来绕过证书检测。

威胁等级:

当账号, 密码, 卡号等数据明文传输, 未进行二次加密时为高风险; 当密码只进行了单项散列而未经过加密时为高风险; 当返回数据中包含更新的 URL 且数据不加密时为高风险; 当校验字段删除后服务器仍会处理所发送的数据包时为高风险; 当校验字段的散列中不包含随机因子时为高风险。以上问题均不存在时无风险。

2.7.4 *访问控制

测试 iOS 客户端访问的 URL 是否仅能由手机客户端访问。是否可以绕过登录限制直接访问登录后才能访问的页面, 对需要二次验证的页面 (如私密问题验证), 能否绕过验证。

测试方法:

人工测试。在 PC 机的浏览器里输入 URL，尝试访问手机银行页面。

威胁等级：

当 PC 端也可访问手机页面时低风险，当可以绕过登陆限制访问登陆后才能访问的页面时中风险。

2.7.5 客户端更新安全性

测试客户端自动更新机制是否安全。如果客户端更新没有使用 iOS 应用商店的更新方式，就可能导致用户下载恶意应用，从而引入安全风险。

测试方法：

使用代理抓取检测更新的数据包，尝试将服务器返回的更新 url 替换为恶意链接。看客户端是否会直接打开此链接并下载应用。

威胁等级：

当客户端返回明文 URL 地址并可以通过篡改的方式控制用户下载恶意软件包进行安装，则高风险；若返回数据包经过二次加密则无风险。

2.7.6 短信重放攻击

检测应用中是否存在数据包重放攻击的安全问题。是否会对客户端用户造成短信轰炸的困扰。

测试方法：

尝试重放短信验证码数据包是否可以进行了短信轰炸攻击。

威胁等级：

当存在短信轰炸的情况时为中风险，若短信网关会检测短时间内发送给某一手机号的短信数量则无风险。

2.8 业务功能测试

对于可以通过代理的方式对交互数据进行分析的客户端，可以对涉及到敏感信息操作的具体业务功能进行测试。

测试方法:

根据客户端的业务流程，使用代理截获客户端每个功能的通信数据，测试对交互数据的篡改或重放所导致的问题。

具体测试内容包括但不限于：篡改造成的越权操作（如跨帐户查询），交易篡改（如修改金额，转帐金额为负值），特殊数据提交（如各种注入问题），重放导致的多次交易，以及用户枚举和暴力密码破解，等等。

（有条件的话可以使用扫描器工具来进行测试）

（下面参考安全规范 6.3.2.2 交易流程测试）

1. 资金类交易中，如果客户端对交易数据签名，签名数据除流水号、交易金额、转入账号、交易日期和时间等要素外，还应包含由服务器生成的随机数据。对于从网上银行客户端提交的交易数据，服务器应验证签名的有效性并安全存储签名。
2. 通过移动终端提交交易请求时，金融机构应采取有效措施鉴别客户身份，保证敏感信息和交易数据的机密性、完整性，并设置与安全防护能力相适应的交易限额以控制交易风险。通过移动终端客户端程序提交交易请求时，应上送终端相关信息，例如，IMEI、IMSI 等。后台服务器应对编号信息和登记信息进行一致性验证。如果对交易数据签名，签名数据应包含此类信息。
3. 在客户确认交易信息后，再次提交交易信息（例如收款方、交易金额）时，应检查客户确认的信息与最终提交交易信息之间的一致性，防止在客户确认后交易信息被非法篡改或替换。

三. 测试项目风险定级

此表格列举了第二章中各个测试项在存在安全风险时，对其风险等级定级的参考标准。

测试分类	测试项目	风险等级
客户端程序安全	*安装包签名	--
	应用完整性检测	高
	*远程调试	--
敏感信息安全	数据文件	高：保存明文或简单编码的密码、cookies、包含敏感信息的网页缓存、用户其他敏感信息等； 中：保存登录用户名、其他非敏感信息。
	系统日志	
	密钥链数据	
密码软键盘安全性	随机布局软键盘	高：使用系统软键盘；中：使用自带软键盘，但键盘布局不随机。
	键盘记录防护	高：可以记录密码。
	屏幕录像	高：输入时有视觉回显；低：无视觉回显，客户端针对越狱设备有风险提示。
进程保护	进程内存访问	高：内存中可以搜索到明文密码
	*动态注入	--
安全策略	密码复杂度检测	中：没有复杂度检测；低：有检测，但不全面
	帐号登录限制	高
	帐户锁定策略	高
	*私密问题验证	中
	会话安全设置	高
	界面切换保护	中
	UI 信息泄露	中
	验证码安全性	中
	安全退出	高
	密码修改验证	高
手势密码安全性	手势密码复杂度	中：没有复杂度检测
	手势密码修改和取消	高
	手势密码本地信息保存	高

	手势密码锁定策略	高
通信安全	通信加密	高：未使用 HTTPS 且没有加密 中：未使用 HTTPS 但通信数据全部加密
	证书有效性检测	高
	关键字段加密和校验	高
	*访问控制	中
	客户端更新安全性	高
	短信重放攻击	中
业务功能测试	越权查询 / 修改账户信息	高
	其他	

四. 合规性参考

下表是《网上银行系统信息安全通用规范》（2012）中关于客户端安全方面的要求。

分类	要求项	对应编号
客户端程序（基本要求）	a) 金融机构应采取有效技术措施保证客户端处理的敏感信息、客户端与服务交互的重要信息的机密性和完整性；应保证所提供的客户端程序的真实性和完整性，以及敏感程序逻辑的机密性	2.1.3, 2.6.3
	b) 客户端程序上线前应进行严格的代码安全测试，如果客户端程序是外包给第三方机构开发的，金融机构应要求开发商进行代码安全测试。金融机构应建立定期对客户端程序进行安全检测的机制	--
	c) 客户端程序应通过指定的第三方中立测试机构的安全检测，每年至少开展一次。	--
	d) 应对客户端程序进行签名，标识客户端程序的来源和发布者，保证客户所下载的客户端程序来源于所信任的机构。	2.1.1
	e) 客户端程序在启动和更新时应进行真实性和完整性校验，防范客户端程序被篡改或替换。	2.1.3, 2.6.5
	f) 客户端程序的临时文件中不应出现敏感信息，临时文件包括但不限于 Cookies。客户端程序应禁止在身份认证结束后存储敏感信息，防止敏感信息的泄露。	2.2.1
	g) 客户端程序应提供客户输入敏感信息的即时加密功能，例如采用密码保护控件。	2.3
	h) 客户端程序应具有抗逆向分析、抗反汇编等安全性防护措施，防范攻击者对客户端程序的调试、分析和篡改。	2.1.2
	i) 客户端程序应防范恶意程序获取或篡改敏感信息，例如使用浏览器接口保护控件进行防范。	（针对 PC 客户端）
	j) 客户端程序应防范键盘窃听敏感信息，例如防范采用挂钩 Windows 键盘消息等方式进行键盘窃听，并应具有对通过挂钩窃听键盘信息进行预警的功能。	（针对 PC 客户端）
	k) 客户端程序应提供敏感信息机密性、完整性保护功能，例如采取随机布放按键位置、防范键盘窃听技术、计算 MAC 校验码等措施。	2.3
客户端程序（增强要求）	a) 客户端程序应保护在客户端启动的用于访问网上银行的进程，防止非法程序获取该进程的访问权限。	2.5.1
	b) 客户端程序应采用反屏幕录像技术，防范非法程序获取敏感信息。	2.3.3
	c) 客户端程序应采取代码混淆等技术手段，防范攻击者对客户端程序的调试、分析和篡改。	2.1.2
	d) 客户端程序开发设计过程中应注意规避各终端平台存在的安全漏洞，例如，按键输入记录、自动拷屏机制、文档显示缓存等。	2.2, 2.3

五. iOS 应用分析

5.1 获得 iOS 应用程序包

5.1.1 导出 ipa

可以使用 iTunes 或 iTools 等工具导出 iOS 应用，导出后的文件以 ipa 结尾。ipa 文件是使用 zip 算法的压缩包，可以使用任何支持 zip 格式的工具（winRAR，7zip 等等）解压缩。Payload 中的 *.app 文件夹就是应用程序包了。

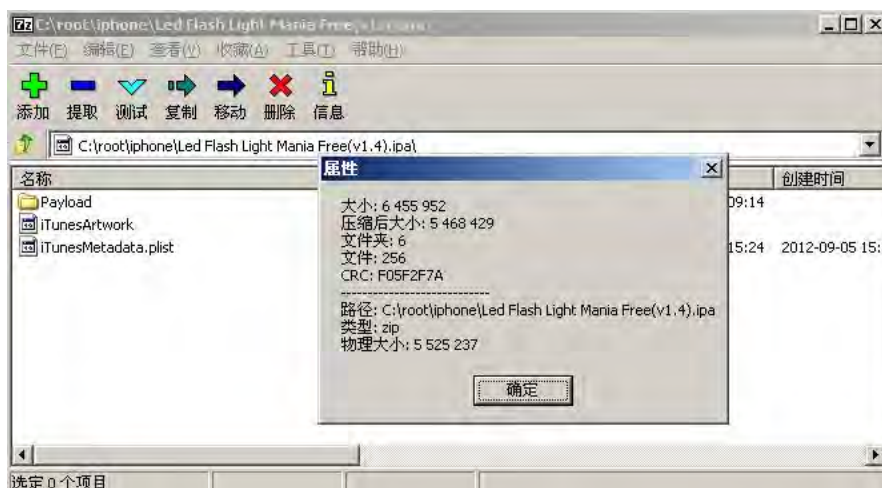
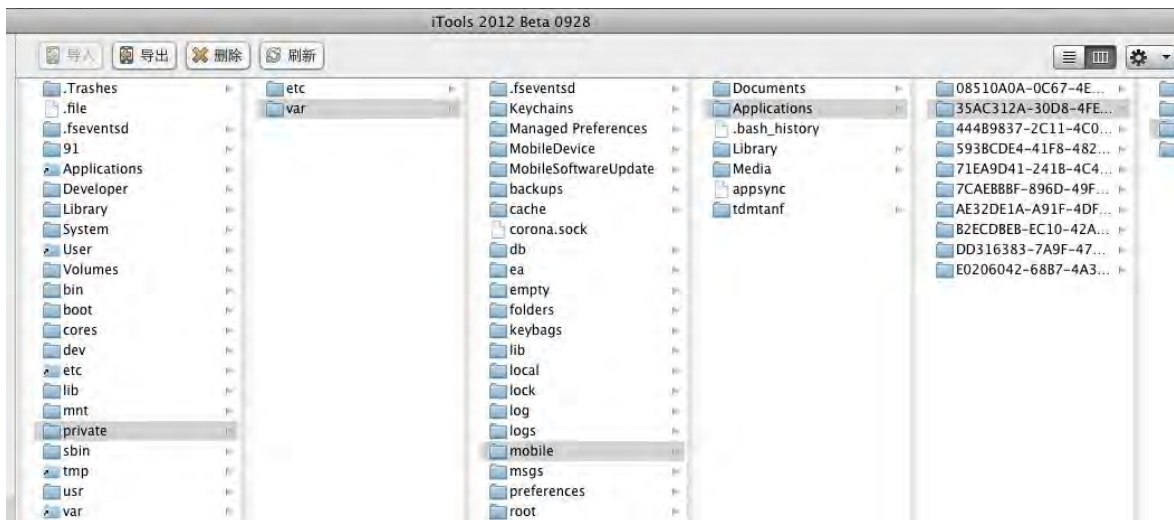


图 1.1 使用 7zip 打开 ipa 文件

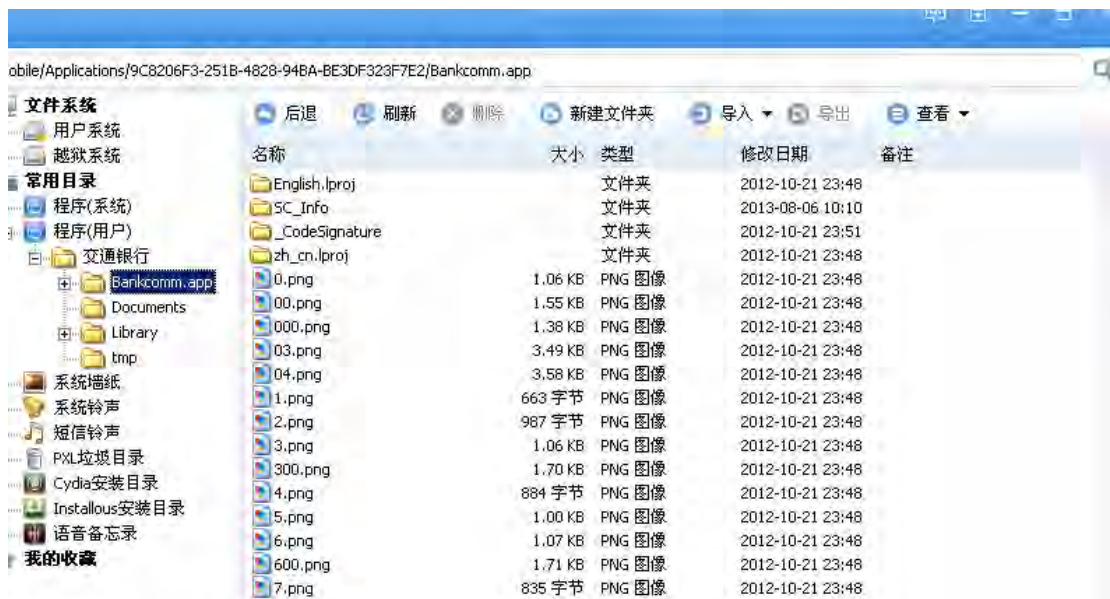
5.1.2 iOS 系统中的*.app

越狱后，可以直接查看和修改安装在 iOS 系统中的应用。

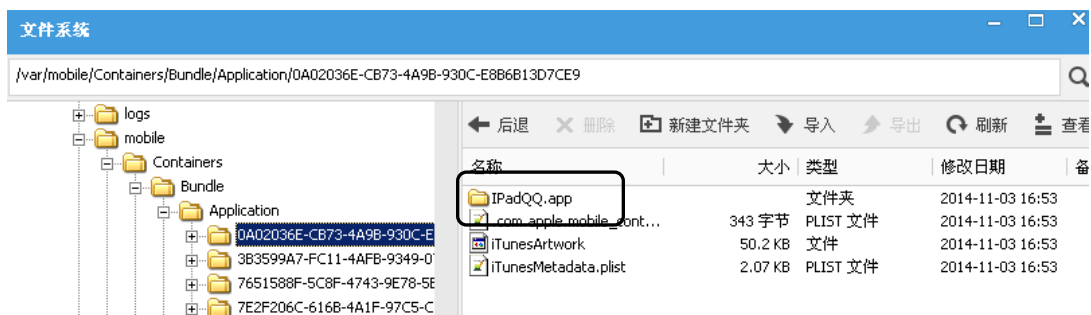
1. 对于 ios 8 之前的系统，可以在/var/mobile/Applications 文件夹中找到所有用户安装的应用，每一个安装的应用都会在其中创建一个子目录，作为自己的私有目录。私有目录名是一串随机数字，每个应用均不相同。从文件/private/var/mobile/Library/Caches/com.apple.mobile.installation.plist 中可以确定每个应用所在的路径。如图所示：



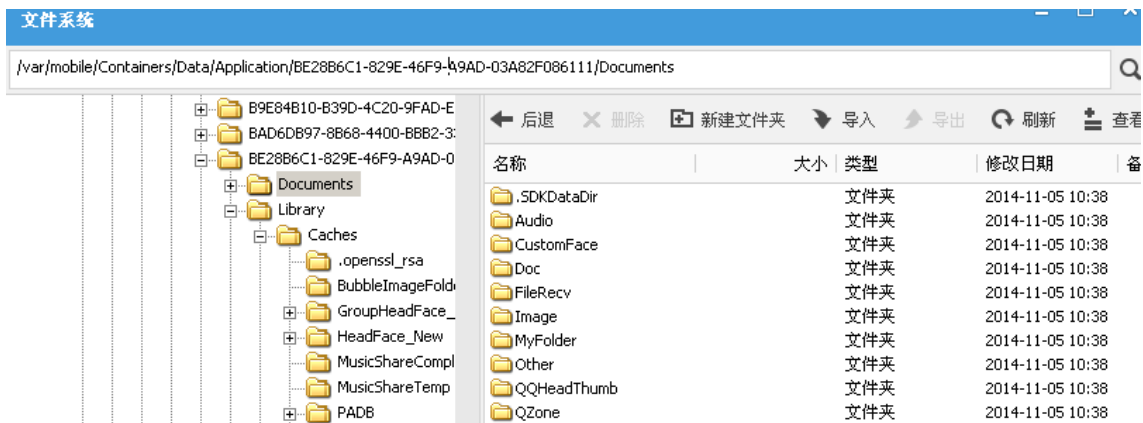
应用程序包和运行时生成并保存的私有数据文件都位于私有目录中。程序包安装为私有目录的*.app 子目录，私有目录下的其他子目录为应用保存私有数据的位置，如图所示：



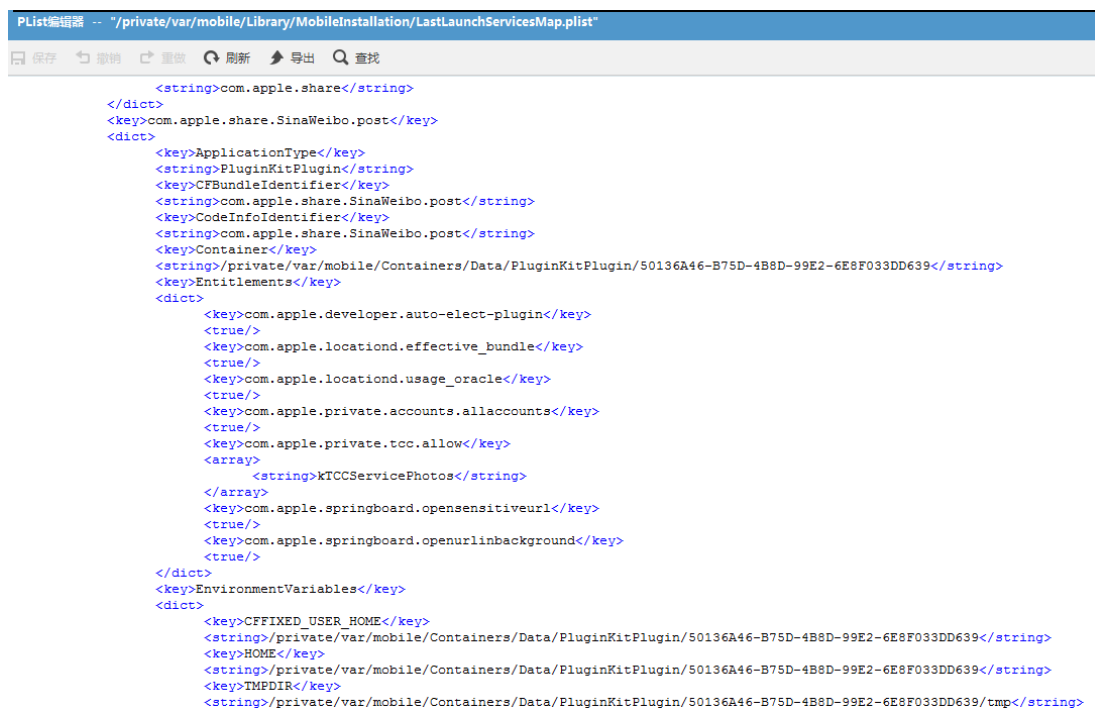
2. 对于 ios8，目录结构有所不同。所有用户应用程序包均安装在/var/mobile/Containers/Bundle/Application/目录下，如下图所示。



应用保存私有数据的目录则建立在/var/mobile/Containers/Data/Application/目录下，如下图所示。注意，程序包所在的文件夹名称和私有目录名称是不相同的。



为了方便查找，可以使用/private/var/mobile/Library/MobileInstallation/LastLaunchServicesMap.plist 文件寻找应用对应的安装目录和私有目录（此文件不一定是最新的，新安装的应用可能找不到，此时可以查看/var/mobile/Library/Caches/com.apple.LaunchServices-102.csstore 缓存文件）。



注：关于iOS文件系统结构的详细说明，可参考Apple [File System Basics](#)。

5.2 查看签名信息

可以使用codesign查看签名信息。Apple在对ios应用签名的过程中，嵌入了多种信息。（关于codesign命令行的说明，参见man手册。关于签名，可参考[此处](#)。Apple提供了一个[Troubleshooting Failed Signature Verification](#)，检查签名时可供参考）

5.2.1 查看签名

1. 验证应用签名，确定应用在签名后是否被篡改，使用命令行：“codesign -vv APP_NAME.app”，如图：

```
localhost:MPBBank.app asky$ codesign -vv /Users/asky/Desktop/招商银行\ 1.4.0/Payload/MPBBank.app
/Users/asky/Desktop/招商银行 1.4.0/Payload/MPBBank.app: valid on disk
/Users/asky/Desktop/招商银行 1.4.0/Payload/MPBBank.app: satisfies its Designated Requirement
localhost:MPBBank.app asky$
```

2. 查看签名数字证书详细信息，使用命令行：“codesign -dvvv APP_NAME.app”。如图所示（下图是开发中的应用，会显示开发者名称；商店里下载的应用显示不同）。

```
sj-askymatoiMac:- asky$ codesign -dvvv /Users/asky/Desktop/Payload\ 2/ynrcbmbank.app
Executable=/Users/asky/Desktop/Payload 2/ynrcbmbank.app/ynrcbmbank
Identifier=com.ynrcb.ynrcbmbank
Format=bundle with Mach-O thin (armv7)
CodeDirectory v=20100 size=2889 flags=0x0(none) hashes=136+5 location=embedded
CDHash=c314d22029392afacc278ed461cec83ac5db3ba1
Signature size=4354
Authority=iPhone Developer: lilin ruan (7MP7JBCWBB)
Authority=Apple Worldwide Developer Relations Certification Authority
Authority=Apple Root CA
Signed Time=2012-11-20 下午 12:02:35
Info.plist entries=28
Sealed Resources rules=3 files=130
Internal requirements count=1 size=180
sj-askymatoiMac:- asky$
```

5.2.2 查看 entitlement

查看应用的 entitlement。关于 entitlement 的说明，请参考（TODO）

1. 使用命令行：“codesign -d —entitlement - APP_NAME.app”，如图所示：

```
localhost:~ asky$ codesign -d --entitlement - /Users/asky/Library/Developer/Xcode/
e/DerivedData/hello_world-awiqoyxttrskiaftkftknknxdlem/Build/Products/Debug-iphon
e/hello\ world.app/
Executable=/Users/asky/Library/Developer/Xcode/DerivedData/hello_world-awiqoyxtt
rskiaftkftknknxdlem/Build/Products/Debug-iphon
e/hello world.app/hello world
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Can be debugged</key>
  <true/>
  <key>get-task-allow</key>
  <true/>
</dict>
</plist>
localhost:~ asky$
```

2. 使用“`ldid -e file`”命令，见 5.8.9 签名工具 `ldid`。

5.2.3 查看 requirement

查看嵌入的 requirement，使用命令行：“`codesign -d -r - APP_NAME.app`”，如图所示：
(关于 requirement 语法，可参考 [此处](#))

```
localhost:Applications asky$ codesign -d -r - AppCleaner.app/
Executable=/Applications/AppCleaner.app/Contents/MacOS/AppCleaner
designated => anchor apple generic and identifier "com.freemacsoft.AppCleaner" and (certificate leaf[field.1.2.840.113
635.100.6.1.9] /* exists */ or certificate 1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate leaf[field.1
.2.840.113635.100.6.1.13] /* exists */ and certificate leaf[subject.OU] = X85Zx835W9)
localhost:Applications asky$
```

5.3 反编译

ipa 文件解压缩后，payload 目录下的 *.app 子目录存放有应用程序，其中包含入口可执行程序文件，通常文件名和略去 .app 后缀的目录名相同，如图 1.2 所示。

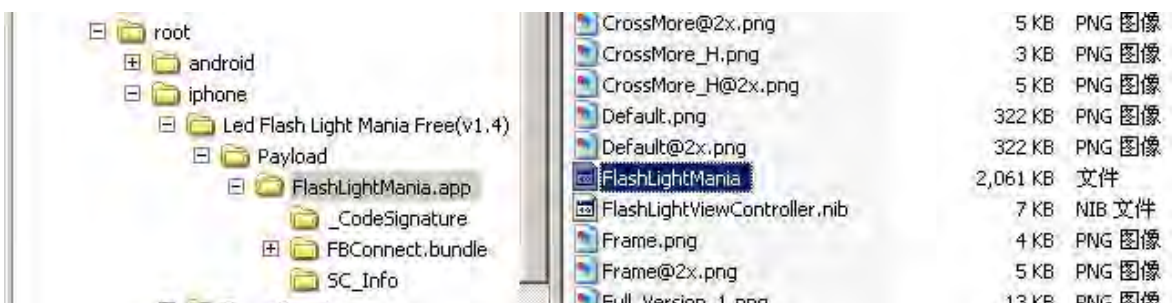
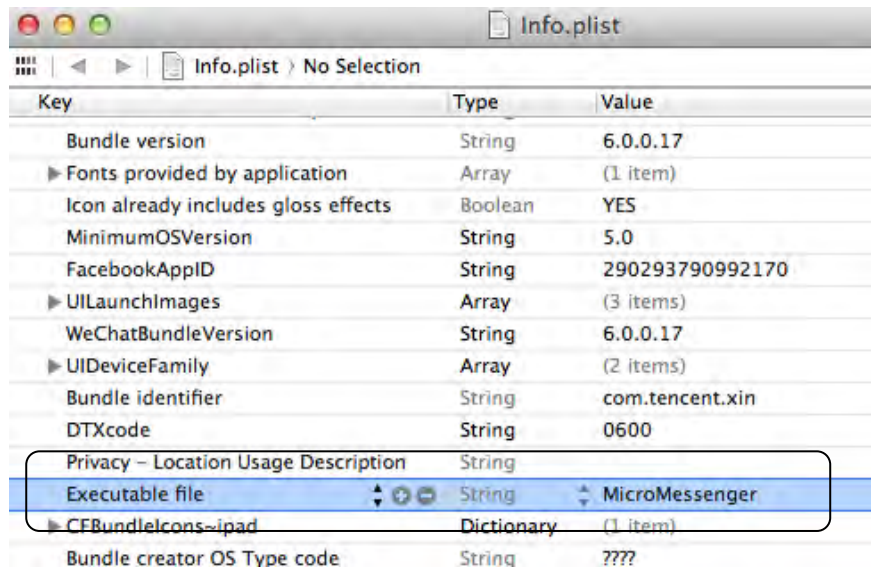


图 1.2 ipa 文件解压后的目录结构

如果找不到同名文件，可在 Info.plist 中查找“Executable file”或“CFBundleExecutable”字段，对应的就是可执行文件名，如图：



5.3.1 Universal Binary 预处理

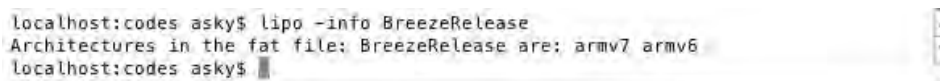
iOS 下（也包括 Mac OSX）的可执行文件为 Mach-O 格式，会存在所谓的“fat binary”。这种可执行文件能同时包含多种架构的代码，所以称为通用二进制代码（Universal Binary）文件。注意：如果不处理 fat binary 的话，可能会对接下来的逆向分析造成一定影响。

1. 如图 1.3，可以使用 `otool -f` 查看文件中包含的架构：



图 1.3 otool 显示同一文件中包含两种架构代码

2. 使用 `lipo` 提取其中某架构的程序文件。先列举一下，如图所示：



3. 然后执行“lipo -thin armv7 BreezeRelease -output BreezeRelease_v7”，将其中 armv7 架构的代码提取出来，保存到 BreezeRelease_v7。

5.3.2 定位入口点

1. 使用 otool -l，pc 指定第一条执行的指令地址，如图所示。

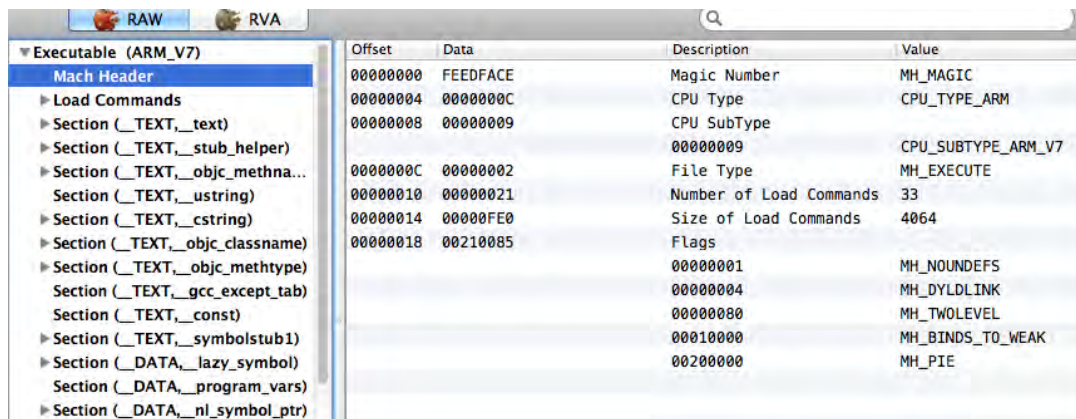
```
Load command 10
  cmd LC_UNIXTHREAD
  cmdsize 84
  flavor ARM_THREAD_STATE
  count ARM_THREAD_STATE_COUNT
    r0 0x00000000 r1 0x00000000 r2 0x00000000 r3 0x00000000
    r4 0x00000000 r5 0x00000000 r6 0x00000000 r7 0x00000000
    r8 0x00000000 r9 0x00000000 r10 0x00000000 r11 0x00000000
    r12 0x00000000 sp 0x00000000 lr 0x00000000 pc 0x00002f5c
  cpsr 0x00000000
Load command 11
```

2. 显示符号表 nm，start 为入口符号，如图所示。（strip 之后 start 会被删除）

```
sj-askymatoMac:~ asky$ nm /Users/asky/Library/Developer/Xcode/DerivedData/testr
un-asjvalyodeyberddzuigtqpxynffn/Build/Products/Debug/testrun
00000000100001050 D _NXArgc
00000000100001058 D _NXArgv
00000000100001068 D __progname
00000000100000000 A _mh_execute_header
00000000100001060 D _environ
U _exit
U _getchar
00000000100000ed0 T _main
U _printf
U dyld_stub_hinder
00000000100000e90 T start
```

5.3.3 可视化 Mach-O 文件查看

可以使用 MachOView 工具查看可执行文件，包括文件的头部和各个 section 的内容（类似于 Windows 上的 Exescope）。如图是某个可执行文件的 Mach Header 内容：



Offset	Data	Description	Value
00000000	FEEDFACE	Magic Number	MH_MAGIC
00000004	0000000C	CPU Type	CPU_TYPE_ARM
00000008	00000009	CPU SubType	CPU_SUBTYPE_ARM_V7
0000000C	00000002	File Type	MH_EXECUTE
00000010	00000021	Number of Load Commands	33
00000014	0000FE0	Size of Load Commands	4064
00000018	00210085	Flags	
	00000001		MH_NOUNDEFS
	00000004		MH_DYLDLINK
	00000080		MH_TWOLEVEL
	00010000		MH_BINDS_TO_WEAK
	00200000		MH_PIE

MachOView 支持直接修改文件，在需要修改的数据上双击即可。

5.3.4 文件解密处理

直接从App store下载的应用是经过加密的（可以通过文件的Crypt ID值判断是否加密，如下图所示，为1时是加密的），逆向分析前需要先对其解密。解密可以使用clutch工具（最新版可在[此处](#)下载。cydia安装的版本较老，不支持fat binary）。

	Offset	Data	Description	Value
LC_ENCRYPTION_INFO	00004A68	00000021	Command	LC_ENCRYPTION_INFO
Command Size	00004A6C	00000014	Command Size	20
Crypt Offset	00004A70	00004000	Crypt Offset	16384
Crypt Size	00004A74	006B0000	Crypt Size	7012352
Crypt ID	00004A78	00000001	Crypt ID	1

1. 在设备的命令行中运行 clutch，如图所示，clutch 会列出可解密的应用：

```
std:/var/mobile/Applications/21550808-6218-4B91-B8C5-35644214FE5C/Tapped Out.app root# clutch
usage: clutch [application name] [...]
Applications available: Alipay AngryBirdsSpace AngryBirdsStarWars BaiduMapPad Bankcomm BlocNotes cai
chengyu CCBMBC CheatGame CheatGame CMBCForIpad com.spdb.retail.bank.hd Compass DBS CN EGB-iPad flapb
ird frozensagachina HSBC IpadForTDX IpadPerbank IpadQQ2014 4.0.0.72 201401201618 KWPlayerHD MicroMes
senger MiniKeePass MobileTicket MojiWeatherHD MPBBank MPBBankHD NinJumpDeluxeHD PadBank-OnlinePay Po
rtal PvZ2 QQMusicPad QR Reader RSAS HD SmartEye SmartEye_iOS Taobao4iIpad Tapped Out ting U17 WheresM
yWater wpsoffice 凤凰知音汇 波克斗地主
```

2. 以 IpadPerbank 为例，在命令行输入“clutch IpadPerbank”，即可对其解密：

```
std:~ root# clutch IpadPerbank
Cracking IpadPerbank...
/var/root/Documents/Cracked/IpadPerbank_v1.7.ipa
std:~ root#
```

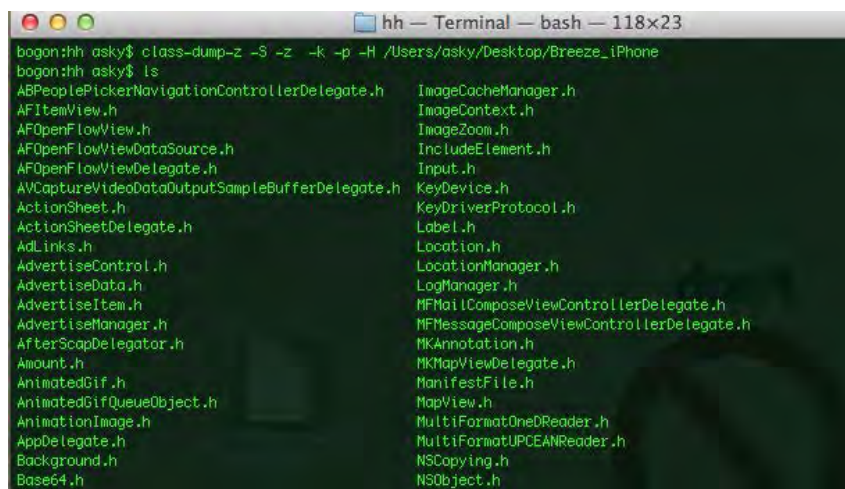
3. 解密后的应用如上图所示，会打包后保存在一个 ipa 中。解包后就可以找到解密后的可执行文件。

外部参考：想自己动手解密，可以参考[此处](#)或[此处](#)步骤。

5.3.5 还原类定义

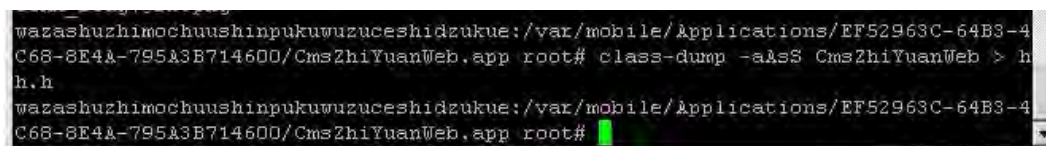
在 iOS 应用的可执行程序文件中包含有很多内部运行时信息。可以通过工具对其进行解析，帮助逆向分析。对于加密的应用，需要参考 5.3.4 文件解密处理先解密。

1. 使用 class-dump(或 class-dump-z)可以得到可执行程序所使用的大部分类信息(与 otool -ov 类似)。如图是使用 class-dump-z 将可执行文件中的类定义导出为头文件。



```
hh -- Terminal -- bash -- 118x23
bagon:hh asky$ class-dump-z -S -z -k -p -H /Users/asky/Desktop/Breeze_iPhone
bagon:hh asky$ ls
ABPeoplePickerNavigationControllerDelegate.h  ImageCacheManager.h
UIImageView.h                                ImageContext.h
AFOpenFlowView.h                             ImageZoom.h
AFOpenFlowViewDataSource.h                   IncludeElement.h
AFOpenFlowViewDelegate.h                    Input.h
AVCaptureVideoDataOutputSampleBufferDelegate.h  KeyDevice.h
ActionSheet.h                               KeyDriverProtocol.h
ActionSheetDelegate.h                       Label.h
AdLinks.h                                  Location.h
AdvertiseControl.h                         LocationManager.h
AdvertiseData.h                           LogManager.h
AdvertiseItem.h                           MFMailComposeViewControllerDelegate.h
AdvertiseManager.h                       MFMessageComposeViewControllerDelegate.h
AfterScapDelegator.h                     MKAnnotation.h
Amount.h                                  MKMapViewDelegate.h
AnimatedGif.h                             ManifestFile.h
AnimatedGifQueueObject.h                  MapView.h
AnimationImage.h                         MultiFormatOneDReader.h
AppDelegate.h                           MultiFormatUPCEANReader.h
Background.h                             NSCopying.h
Base64.h                                 NSObject.h
```

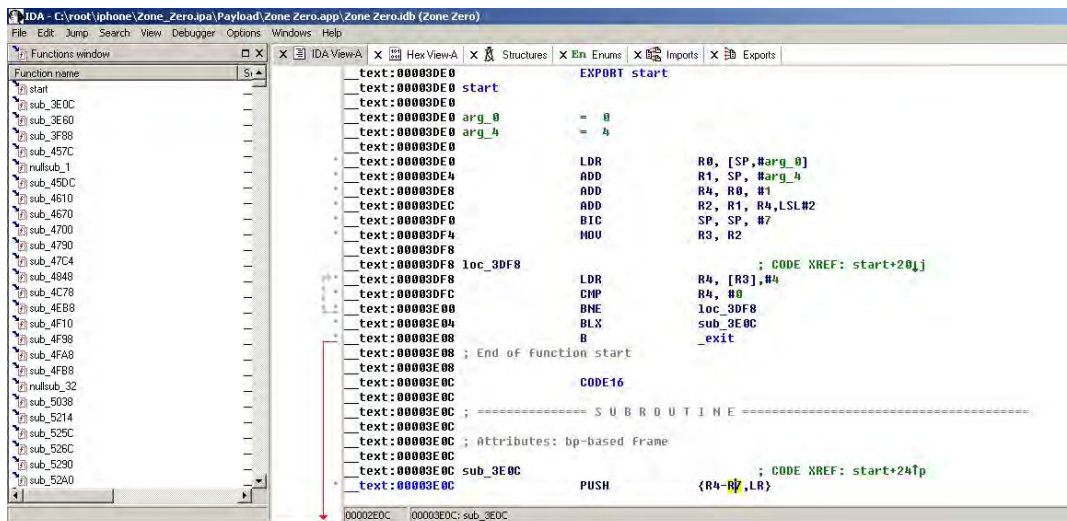
使用 class-dump 将所有类定义导出到一个头文件中，如图：



```
wazashuzhimochuushinpukuwuzuceshidzuke:/var/mobile/Applications/EF52963C-64B3-4C68-8E4A-795A3B714600/CmsZhiYuanWeb.app root# class-dump -a&sS CmsZhiYuanWeb > h.h
wazashuzhimochuushinpukuwuzuceshidzuke:/var/mobile/Applications/EF52963C-64B3-4C68-8E4A-795A3B714600/CmsZhiYuanWeb.app root#
```

5.3.6 IDA pro

可执行文件为 Mach-O 格式，可以用 IDA 反汇编，如图 1.4 所示。



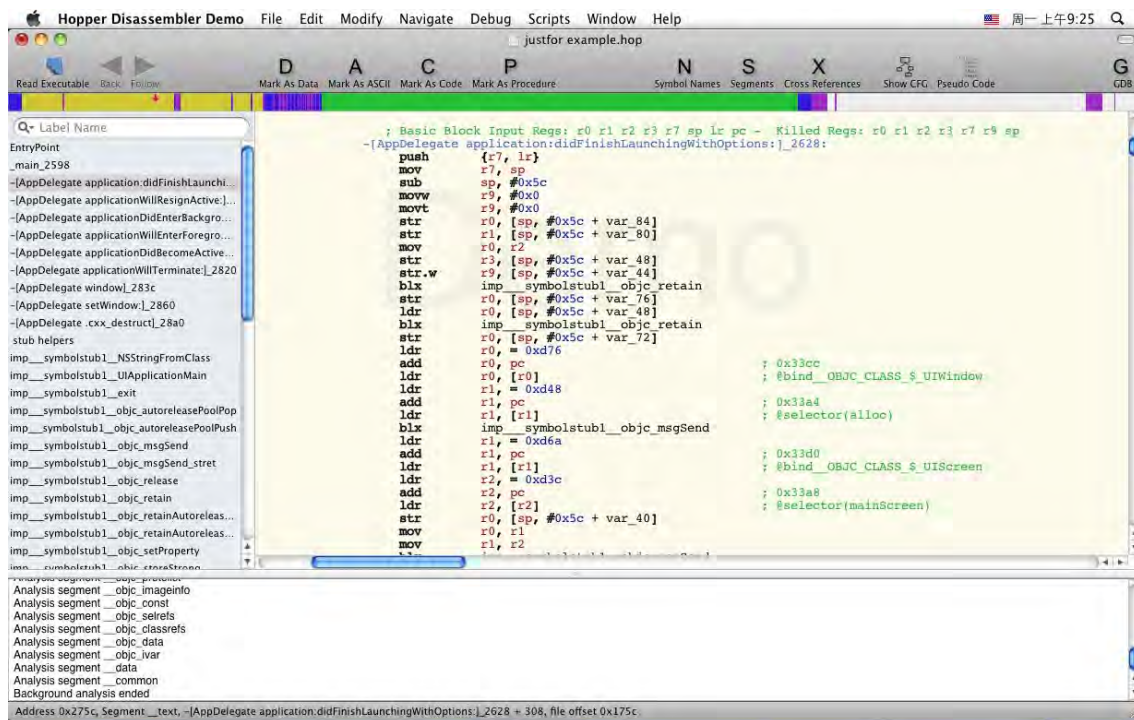
```
IDA - C:\root\phone\Zone_Zero.ipa\Payload\Zone Zero.app\Zone Zero.idb (Zone Zero)
File Edit Jump Search View Debugger Options Windows Help
Functions window: start, sub_3E0C, sub_3E60, sub_3F88, sub_457C, nullsub_1, sub_45D0, sub_4610, sub_4670, sub_4700, sub_4790, sub_47C4, sub_4848, sub_4C78, sub_4EB8, sub_4F10, sub_4F98, sub_4FA8, sub_4FB8, nullsub_32, sub_5038, sub_5214, sub_525C, sub_526C, sub_5290, sub_52A0
IDA View: text:0003DE0 start, text:0003DE0 arg_0 = 0, text:0003DE0 arg_4 = 4, text:0003DE0 LDR R0, [SP, #arg_0], ADD R1, SP, #arg_4, text:0003DE4 ADD R4, R0, #1, text:0003DEC ADD R2, R1, R4, LSL#2, text:0003DF0 BIC SP, SP, #7, text:0003DF4 MOV R3, R2, text:0003DF8 loc_3DF8, text:0003DF8 LDR R4, [R3], #4, text:0003DFC CMP R4, #0, text:0003E00 BNE loc_3DF8, text:0003E04 BLX sub_3E0C, text:0003E08 B _exit, text:0003E08 ; End of function start, text:0003E08 CODE16, text:0003E0C ; ===== S U B R O U T I N E =====, text:0003E0C ; attributes: bp-based frame, text:0003E0C sub_3E0C, text:0003E0C PUSH {R4-R7, LR}, text:0003E0C
```

图 1.4 IDA 中的反汇编代码

注意：iTool 启动时可能会强制杀掉 IDA 进程，注意保存。

5.3.7 Hopper

Hopper 是 Mac OS 下的反汇编工具（也有 Windows 版本）。与 IDA 相比，Hopper 能够更好的分析 Mac 和 iOS 下的 Object-C 程序，而且可以形成 Object-C 格式的伪代码（可能有 bug，会对某些跳转代码处理不正确）。如图：



5.4 处理资源文件

iOS 应用在编译过程中，会对程序文件进行优化（如 png 图片和 plist 文件）。在分析打包好的应用时，需要将优化后的文件还原。

5.4.1 还原 png 图片

还原 png 图片可以使用如下命令行，其中 a.png 是优化过的文件，a_org.png 是输出的优化前 png 文件。（PATH_TO_XCODE 替换为 xcode 的安装目录）

```
PATH_TO_XCODE/Platforms/iPhoneOS.platform/Developer/usr/bin/pngcrush  
-revert-iphone-optimizations a.png a_org.png
```

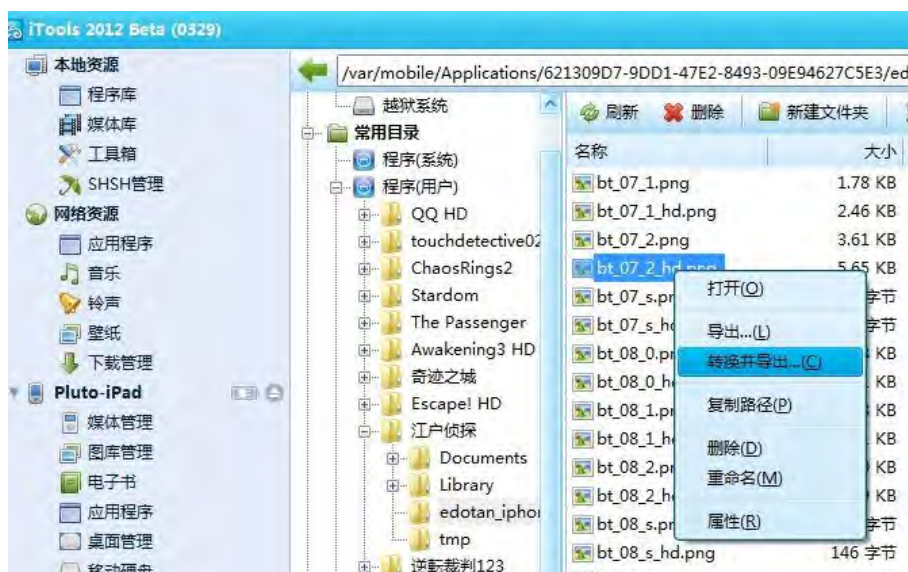
5.4.2 还原 plist

对于经过编码的 plist 文件，可以使用如下命令行还原。

```
plutil -convert xml1 b.plist
```

5.4.3 iTunes 导出

使用 iTunes 可以直接转换并导出经过优化的程序文件，如图。



5.4.4 查看*.mobileprovision 文件

此文件为预置描述文件。通常处于开发状态，使用 Ad Hoc 方式分发的应用中会包含此类文件，通常命名为 embedded.mobileprovision。此种文件使用 ASN.1 格式，其中包含文本信息和证书签名。可使用 Xcode Organizer 导出此种描述文件。

1. 使用文本编辑器，也可以查看其中的 plist 文本内容。
2. 要查看全部信息可使用 openssl 命令：“openssl asn1parse -inform der -in file”。如图所示：



5. 其他关于openssl的操作可参考此 [博客](#)。

(描述文件保存在/private/var/MobileDevice/ProvisioningProfiles)

5.4.5 查看 mobileconfig 文件

此类文件为配置描述文件（安装在设备/通用/描述文件，通常不会放在应用安装包中），在Mac上可使用 [iPhone配置实用工具](#)生成。使用ASN.1 格式，其中包含文本信息和证书签名。查看方式与.mobileprovision文件类似，可参考查看*.mobileprovision文件。

1. 使用 openssl 查看：

```
asky@shangjin:~$ openssl asn1parse -i -dump -inform der -in '/media/linux/shared/Share/tl.mobileconfig'
0:d=0 hl=2 l=inf cons: SEQUENCE
2:d=1 hl=2 l=9 prim: OBJECT :pkcs7-signedData
13:d=1 hl=2 l=inf cons: cont [ 0 ]
15:d=2 hl=2 l=inf cons: SEQUENCE
17:d=3 hl=2 l=1 prim: INTEGER :01
20:d=3 hl=2 l=11 cons: SET
22:d=4 hl=2 l=9 cons: SEQUENCE
24:d=5 hl=2 l=5 prim: OBJECT :sha1
31:d=5 hl=2 l=0 prim: NULL
33:d=3 hl=2 l=inf cons: SEQUENCE
35:d=4 hl=2 l=9 prim: OBJECT :pkcs7-data
46:d=4 hl=2 l=inf cons: cont [ 0 ]
48:d=5 hl=2 l=inf cons: OCTET STRING
50:d=6 hl=4 l=2673 prim: OCTET STRING
0000 - 3c 3f 78 6d 6c 20 76 65-72 73 69 6f 6e 3d 22 31 <?xml version="1
0010 - 2e 30 22 20 65 6e 63 6f-64 69 6e 67 3d 22 55 54 .0" encoding="UT
0020 - 46 2d 38 22 3f 3e 0a 3c-21 44 4f 43 54 59 50 45 F-8"?><!DOCTYPE
0030 - 20 70 6c 69 73 74 20 50-55 42 4c 49 43 20 22 2d plist PUBLIC "~
0040 - 2f 2f 41 70 70 6c 65 2f-2f 44 54 44 20 50 4c 49 //Apple//DTD PLI
0050 - 53 54 20 31 2e 30 2f 2f-45 4e 22 20 22 68 74 74 ST 1.0//EN" htt
0060 - 70 3a 2f 2f 77 77 77 2e-61 70 70 6c 65 2e 63 6f p://www.apple.co
0070 - 6d 2f 44 54 44 73 2f 50-72 6f 70 65 72 74 79 4c m/DTDs/PropertyL
0080 - 69 73 74 2d 31 2e 30 2e-64 74 64 22 3e 0a 3c 70 ist-1.0.dtd">.<p
```

2. Mac 下使用使用 security 查看，“security cms -D -i file”。
3. 关于配置描述文件所能实现的具体功能，可参考 [iPhone配置实用工具](#)。
(TODO 移动设备管理)

5.4.6 分析 sqlite 数据库日志文件

如果数据库引擎使用了 [Write Ahead Logging](#) (WAL)，那么有可能在wal文件中发现已删除的数据。（当sqlite数据库没有正常关闭时，会出现这种情况）

注意：当使用 iFile 或是 SQLite Studio 等工具查看数据库时，WAL 文件可能被清空。

1. 使用编辑器或其他工具查看文件中的字符串，如图，可以看到数据库中查不到的信息。

```
localhost:~ asky$ strings /Users/asky/Downloads/DB/WCDB_OpLog.sqlite-wal
SQLite format 3
Ytablessqlite_sequencesqlite_sequence
CREATE TABLE sqlite_sequence(name,seq)
etableOplogOplog
CREATE TABLE Oplog(OplogID integer primary key on conflict replace autoincrement, CmdID integer, OpBuf blob)
wxid_kq3yutrng1wv22
aaa
13
Beijingb
Haidianp
Oplog
wxid_kq3yutrng1wv22
aaa
13
```

2. 使用python脚本，下载地址在 [此处](#)。解析的输出如图：

```
localhost:forensics-sqlite asky$ python test.py /Users/asky/Downloads/DB/WCDB_OpLog\ 2.sqlite
Version 2de218
Page size 1000
Sequence 00
Salt1 921c0c07
Salt2 34632b2b
Checksum1 dbf31fa5
Checksum2 bf065a7e
Current position 4152
Page number 02
Salt1 921c0c07
Salt2 34632b2b
Checksum1 6ceb5eeb
Checksum2 6cdee628
???? ??
wxid_kq3yutrng1wv22
aaa *2
```

关于对sqlite-wal的分析，可参考此 [链接](#)。对sqlite文件格式的描述见 [此处](#)。

5.5 生成 iOS 应用（越狱后）

5.5.1 生成证书

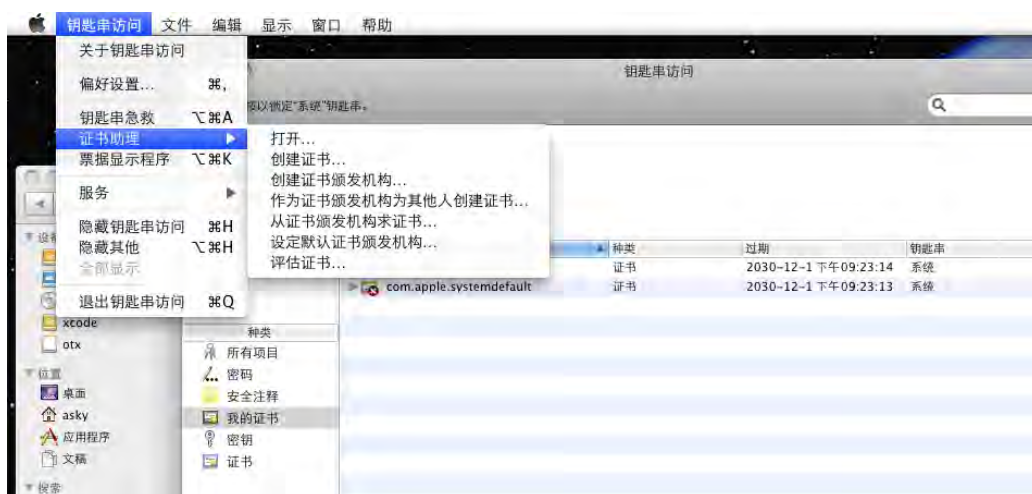
注意：当设备从 Cydia 安装了 AppSync 后，可以运行没有签名的应用。此步骤和下面用于证书签名的步骤均可以省略。

iOS 上的程序必须经过签名才能运行。苹果的证书需要付费。在修改 Xcode 后，可以使用自己生成的证书给代码签名。证书生成过程如下：

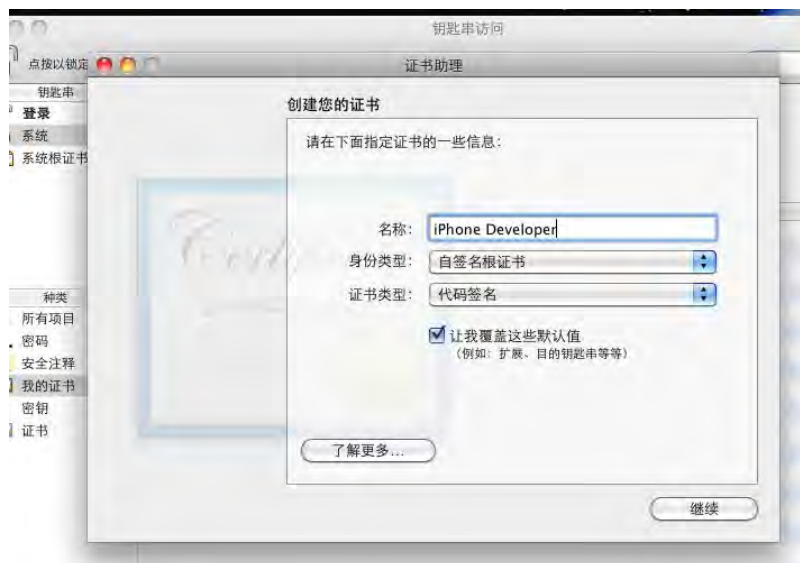
1. 在实用工具里选择“钥匙串访问”。



2. 选择“创建证书”。



3. 名称为“iPhone Developer”，也可以为其他名称。类型：代码签名。



4. 按默认选项继续，直至完成。如图所示。



5.5.2 准备 xcode

注意：下面几节演示的步骤适用于 Xcode 4.x。Xcode 5 的界面有所不同，某些步骤可能会不太一样，请勿生搬硬套。

xcode 默认无法使用上述方法生成的证书签名应用，在编译时会报错。需要修改配置文件解决这一问题，有两种方法：（`PATH_TO_XCODE` 替换为 xcode 的安装目录）

1. 在

`PATH_TO_XCODE/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS5.0.sdk/`

（xcode4.6 默认路径为

`/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/SDKs/iPhoneOS6.1.sdk`）中，编辑 `SDKSettings.plist`，修改下面的 YES 为 NO：

```
<key>CODE_SIGNING_REQUIRED</key> <string>YES</string>
```

```
<key>ENTITLEMENTS_REQUIRED</key> <string>YES</string>
```

然后在 `PATH_TO_XCODE/Platforms/iPhoneOS.platform/`（xcodes 4.6 默认路径为 `/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform`）中，编辑 `Info.plist`，将全部的 `XCiPhoneOSCodeSignContext` 修改成 `XCCodeSignContext`。

2. 可以通过安装 [iOSSignDev](#) 自动化完成上述过程。

外部参考：[Xcode 4.1/4.2/4.3 免证书\(iDP\)开发 真机调试 生成IPA全攻略](#)

[Xcode 4.1/4.2/4.3/4.4/4.5 + iOS 5.1.1 免证书\(iDP\)开发+真机调试+生成IPA全攻略](#)

5.5.3 编译 iOS 命令行程序

注意：推荐安装 [iOSSignDev](#)，可以简化下面的设置步骤。

1. 新建项目。选择“Empty Application”。下一步，填写项目相关信息。完毕后如图 1.6 所示。

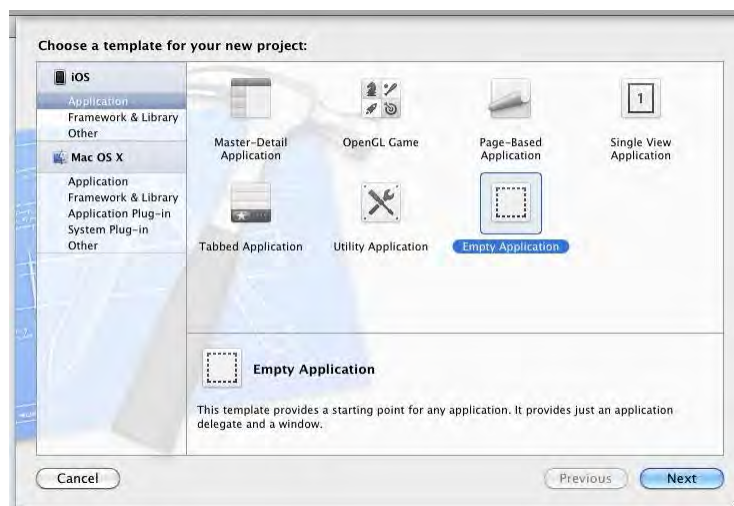


图 1.5 选择项目模版

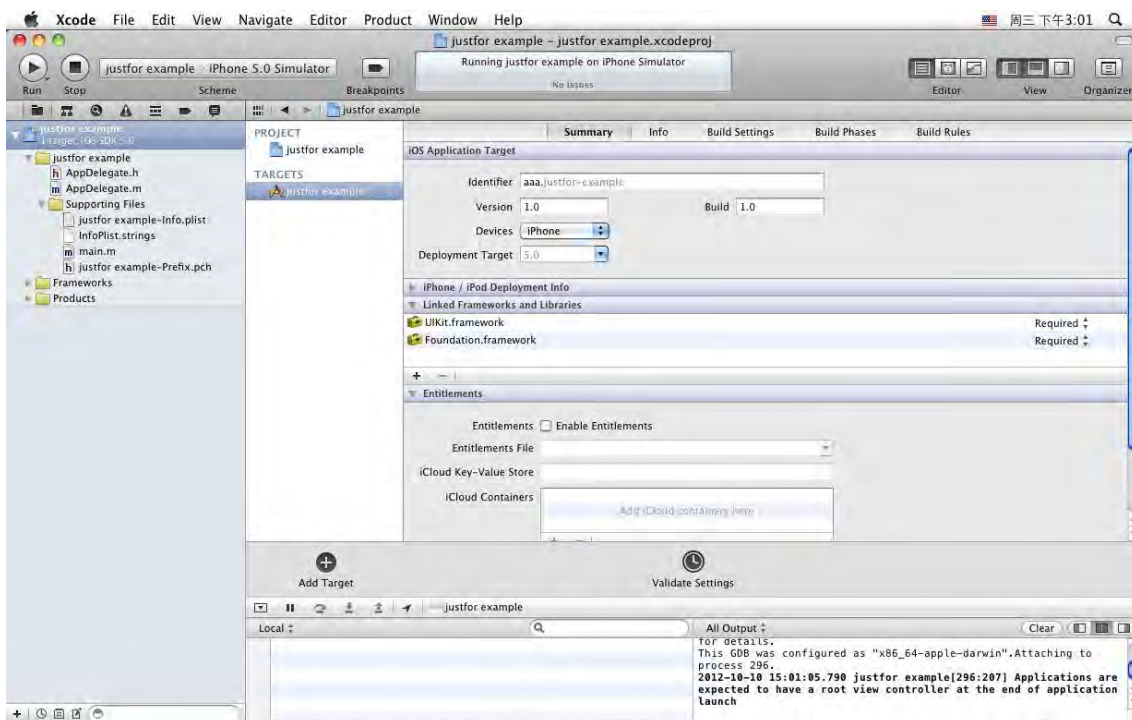


图 1.6 新建好的 iOS 项目

在新建好的项目里，需要修改“Code Signing”选项，将所有设置均修改为“Don't Code Sign”。

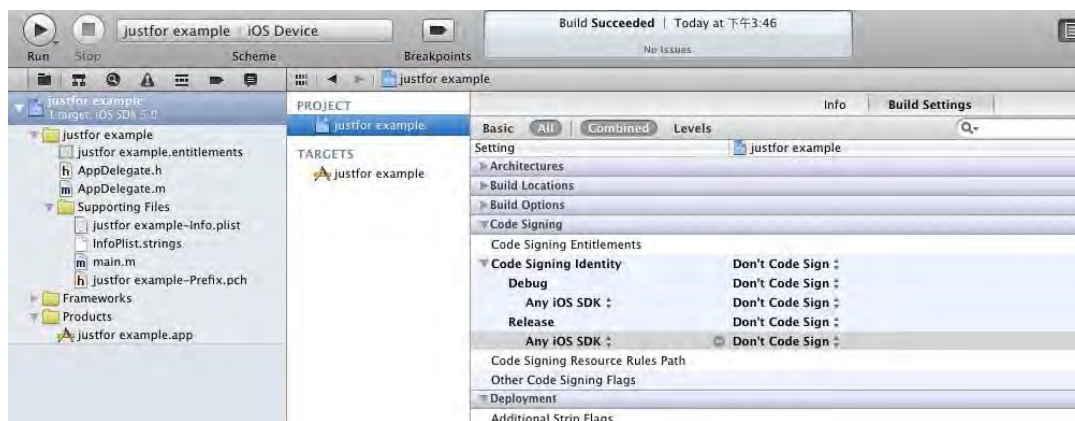


图 1.7 修改代码签名设置

2. 如果需要动态调试，需要建立 Code Signing Entitlements 文件。如图 1.8 选择“Enable Entitlements”，此时会自动生成一个扩展名为 entitlements 的文件。然后去掉“Enable Entitlements”选择（如能正常编译可以不去掉），按图 1.9 所示编辑文件，将 “Can be debugged”属性值改为 “YES”，将“get-task-allow”属性值改为“YES”，这两个选项主要用于xcode 调试。“application-identifier”设置为“iPhone Developer”（没有此项可手动添加）。（此步骤主要是为 4.5.5 代码签名做准备）

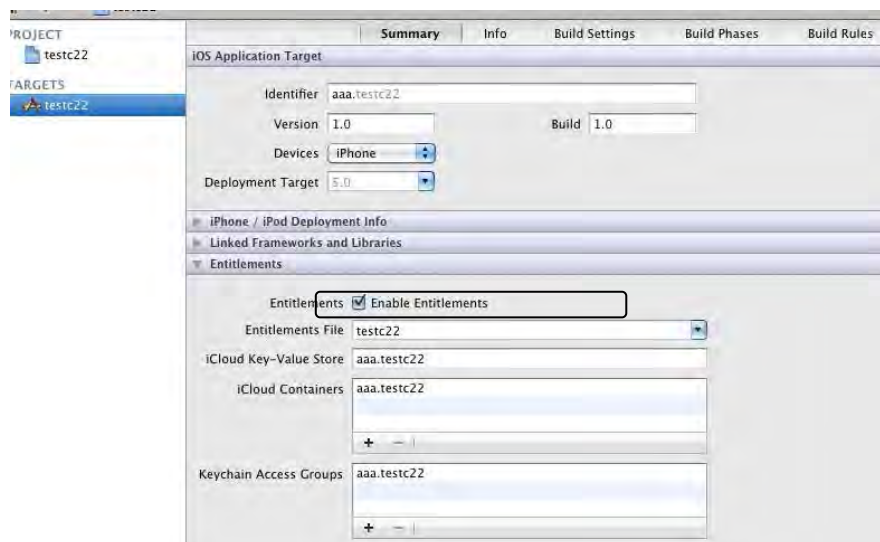


图 1.8 使能 Entitlements

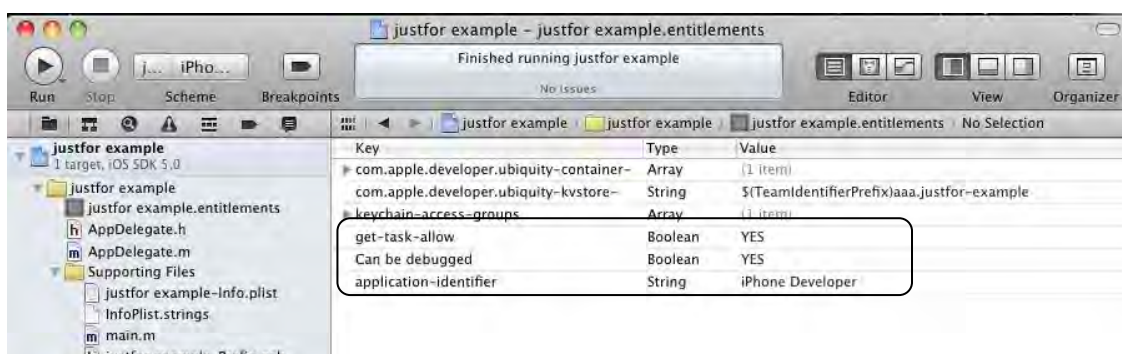


图 1.9 编辑 entitlements 文件

3. 编译目标（左上角）选择“iOS Device”。菜单栏选择“Product”->“Build”，编译代码。然后在 products 下的 app 点右键，选择“Show in Finder”，如图 1.10 所示。

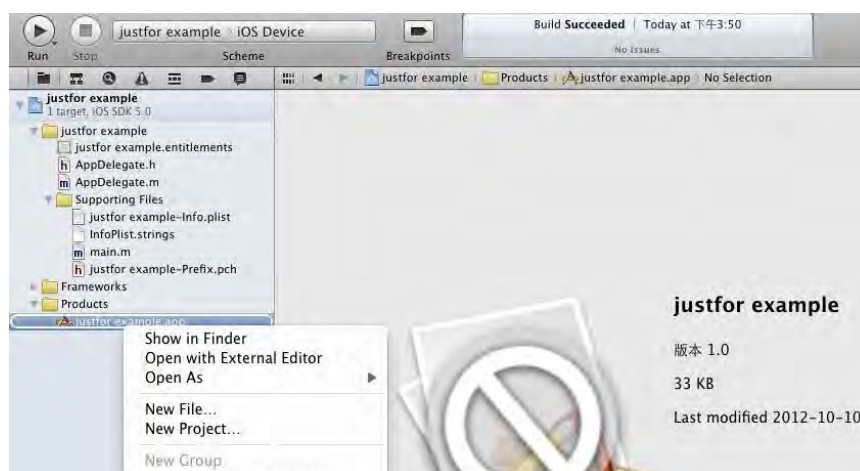
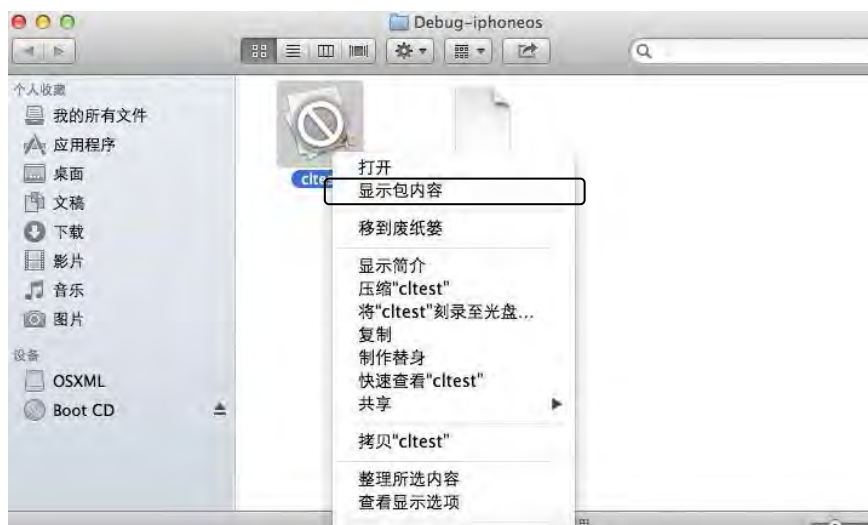


图 1.10 打开编译好的 app

4. 在 Finder 里显示包内容，找到编译好的命令程序文件。



5.5.4 编译 iOS dylib

1. 参考 4.5.3 编译 iOS 命令程序，新建一个命令程序项目。
2. 在“Build Settings”中进行如图 1.11 的设置。其中 Dynamic Library Install Name 的设置 为“@executable_path/LIBNAME.dylib”，这样放置在可执行程序所在的文件夹里的 dylib 文件 就允许被加载了。
3. 如图 1.11，Initialization Routine 设置项设置动态库的加载初始化函数（c 函数注意要在 函数名前加“_”）。

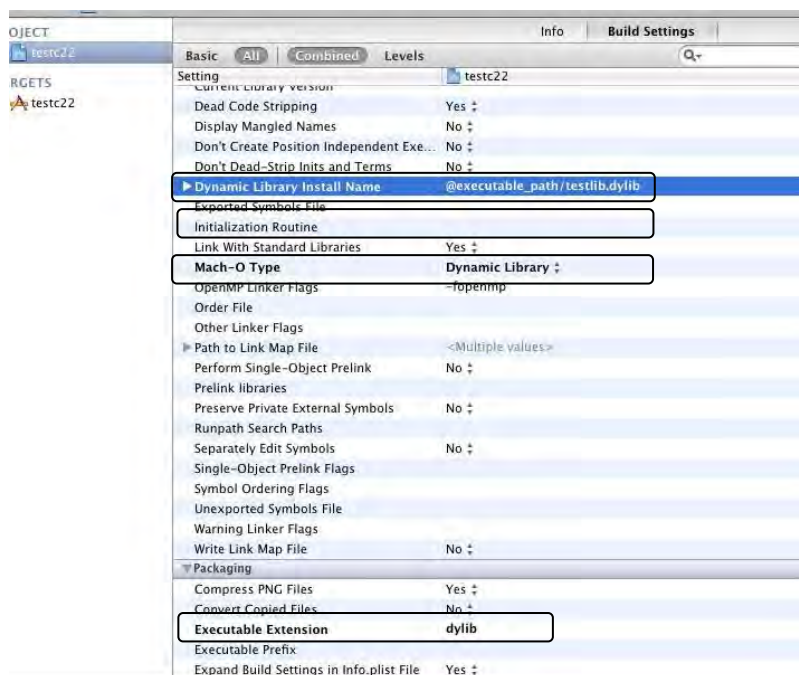


图 1.11 dylib 编译设置

4. 将加载 *LIBNAME.dylib* 的程序和 *LIBNAME.dylib* 放在同一个目录下，就可以通过编程动态加载 dylib 了。

外部参考：[Build and use dylib on iOS](#)

5.5.5 代码签名

在 iOS 运行的程序如需要签名（越狱后的 iOS 系统有些也需要签名，视版本而定），可以使用以下几种方法：（下面的 *PATH_TO_XCODE* 为 xcode 安装根目录，*PATH_TO_ENTM* 为 entitlement 文件路径）

1. 完成步骤 4.5.1 生成证书后，在命令行输入如下命令：

```
export  
CODESIGN_ALLOCATE=PATH_TO_XCODE/Platforms/iPhoneOS.platform/Developer/usr/bin/codesign_allocate  
  
Codesign -fs "iPhone Developer" --entitlements "PATH_TO_ENTM" path_to_app
```

2. 在手机里使用 ldid 工具（注意，ldid 生成的签名和 codesign 的可能不兼容）：

```
ldid -S PATH_TO_ENTM path_to_app
```

3. 在 Mac 上使用 ldid 工具。

4. 完成步骤 4.5.1 生成证书后，xcode 添加自定义的生成脚本。在 Build Phases 中添加一个 Phase，右下角的 Add Build Phase，然后单击 Add Run Script。如图 1.12 所示，添加如下内容：

```
export
CODESIGN_ALLOCATE=PATH_TO_XCODE/Platforms/iPhoneOS.platform/Developer/usr/bin/codesign_allocate

codesign -fs "iPhone Developer" --entitlements
"${PROJECT_DIR}/${PROJECT_NAME}/${PROJECT_NAME}.entitlements"
"${BUILT_PRODUCTS_DIR}/${WRAPPER_NAME}"
```

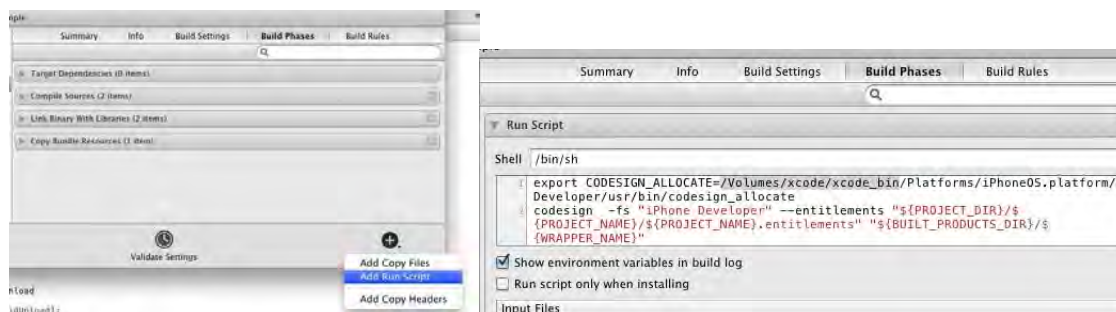


图 1.12 添加编译脚本

4. 取消内核中的代码签名校验：（在 iOS 55.1.1 中无效，iOS 8.1 中为只读，其它版本没有测过）

```
sysctl -w security.mac.proc_enforce=0
sysctl -w security.mac.vnode_enforce=0
```

注：对代码签名的深入解释，可参考 [此处](#)。

5.5.6 打包 ipa

Xcode编译成功的程序包一般位于~/Library/Developer/Xcode/DerivedData/“项目名”/Build/Products/Release-iphoneos。使用Xcode导出ipa可参考 [此处](#)。

当成功生成“项目名.app”文件夹后，也可以将其拖到 iTunes 中，它会出现现在应用程序列表中，然后再把它从 iTunes 的那个列表中拖出来（比如拖到桌面），拖出来的就是相应的 ipa 文件。

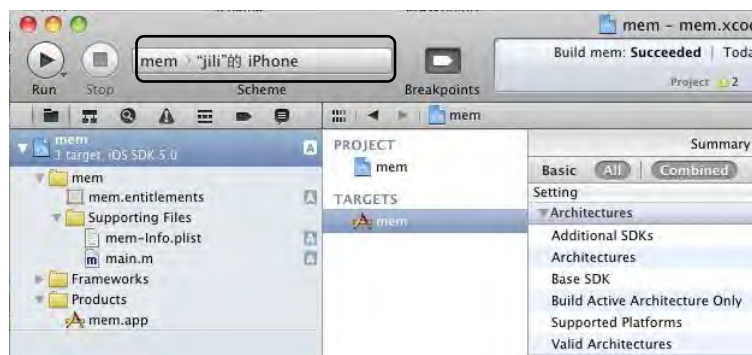
5.6 动态调试

5.6.1 使用 Xcode

1. 当使用 USB 接入的设备 iOS 版本是 xcode 所支持的时候，在 Organizer 面板中可以看到关于设备的一些信息。如图所示：



2. 打开一个 iOS 项目，在 xcode 界面左上就能看到设备名被显示出来。此时就可以远程调试 iOS 项目了。



3. 注意，如果无法调试，可以把项目的 Entitlements.plist 添加“Can be debugged”属性，值设置为“YES”，添加“get-task-allow”属性，值设置为“YES”。

5.6.2 使用 gdb

在 iOS 中，可以使用 gdb 调试程序（gdb 详细用法请参考 gdb 手册）。为了调试方便，可以参考 4.3.3 可视化 Mach-O 文件查看工具，修改 PIE 标志位，固定程序加载地址。

1. 使用 `gdb -p pid` 调试正在运行的应用。（pid 的获取参考 4.8.3 进程查看和监视 ps）
2. 使用 gdb 获取进程内存信息。使用 `info mach-regions` 命令，如图：

```

Undefined command: "mach-region". Try "help".
(gdb) info mach-region
Argument required (expression to compute).
(gdb) info mach-regions
Region from 0x100000000 to 0x100001000 (r-x, max rwx; copy, private, not-reserved)
... from 0x100001000 to 0x100002000 (rw-, max rwx; copy, private, not-reserved)
... from 0x100002000 to 0x100003000 (r--, max rwx; copy, private, not-reserved)
... from 0x100003000 to 0x100004000 (---, max rwx; copy, private, not-reserved)
... from 0x100004000 to 0x100005000 (rw-, max rwx; copy, private, not-reserved)
... from 0x100005000 to 0x100007000 (---, max rwx; copy, private, not-reserved) (2 sub-regions)
... from 0x100007000 to 0x10001c000 (rw-, max rwx; copy, private, not-reserved)
... from 0x10001c000 to 0x10001e000 (---, max rwx; copy, private, not-reserved) (2 sub-regions)
... from 0x10001e000 to 0x100033000 (rw-, max rwx; copy, private, not-reserved)
... from 0x100033000 to 0x100034000 (---, max rwx; copy, private, not-reserved)
... from 0x100034000 to 0x100035000 (r--, max rwx; copy, private, not-reserved)
... from 0x100100000 to 0x100200000 (rw-, max rwx; copy, private, not-reserved)
... from 0x100200000 to 0x101000000 (rw-, max rwx; copy, private, not-reserved)
... from 0x7fff5bc00000 to 0x7fff5f400000 (---, max rwx; copy, private, not-reserved)
... from 0x7fff5f400000 to 0x7fff5fc00000 (rw-, max rwx; copy, private, not-reserved)
... from 0x7fff5fc00000 to 0x7fff5fc3c000 (r-x, max rwx; copy, private, not-reserved)
... from 0x7fff5fc3c000 to 0x7fff5fc7b000 (rw-, max rwx; copy, private, not-reserved) (5 sub-regions)
... from 0x7fff5fc7b000 to 0x7fff5fc8f000 (r--, max rwx; copy, private, not-reserved)
... from 0x7fff70000000 to 0x7fff70200000 (rw-, max rwx; copy, private, not-reserved)
... from 0x7fff70200000 to 0x7fff80000000 (r--, max rwx; share, private, reserved)
... from 0x7fff80000000 to 0x7fffc0000000 (r--, max rwx; share, private, reserved)
... from 0x7fffc0000000 to 0x7fffffe00000 (r--, max rwx; share, private, reserved)
... from 0x7fffffe00000 to 0x800000000000 (r-x, max r-x; share, private, reserved)
(gdb) info mach-region 0x100003000
Region from 0x100003000 to 0x100004000 (---, max rwx; copy, private, not-reserved)

```

3. 将指定内存区域转储到文件，使用 `memory dump` 命令。如图是将 `0x40000—0x97000` 的内存数据转储到 `aa.dmp` 文件中。修改内存数据，直接用 `set` 即可。

```

Invalid number: "0x0x97000".
(gdb) dump memory aa.dmp 0x40000 0x97000
(gdb) set *0x40000 = 0x90
(gdb) x /x 0x40000
0x40000: 0x00000090

```

4. 使用 `ni (si)` 单步执行指令。`info r` 查看寄存器信息。
5. 可以从 <https://github.com/gdnano/binit/Gdbinit> 下载一个 `gdbinit` 配置文件，对 `gdb` 的命令和响应进行自定义配置，方便测试。如图是使用配置文件后，`gdb` 加载程序完毕的显示：

```

warning: Could not find object file "/Users/_X_/Library/Developer/Xcode/DerivedData/BlockAds-ecfymacacwpanfsjkqjgyhlfihs/Build/Intermediates/BlockAds.build/Debug-iphones/AdBlocker.build/Objects-normal/armv6/BlockerURLProtocol.o" - no debug information available for "/Users/_X_/Dropbox/Coding/AdBlocker_1056/Classes/BlockerURLProtocol.m".

..... done
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
bfd_mach_o_scan: unknown architecture 0x100000c/0x0
Reading symbols for shared libraries + done
0x3c812eb4 in mach_msg_trap ()

[regs]
R0: 0x10004005 R1: 0x07000006 R2: 0x00000000 R3: 0x00000000
R4: 0x00002003 R5: 0xffffffff R6: 0x00000000 R7: 0x2fdd3ea0
R8: 0x00000000 R9: 0x001aee00 R10: 0xffffffff R11: 0x00000000
R12: 0xffffffff SP: 0x2fdd3e60 LR: 0x3c813040 PC: 0x3c812eb4 nZcYqjeoift

[code]
0x3c812eb4: 70 01 bd e8      e8bd0170 pop {r4, r5, r6, r8}
0x3c812eb8: 1e ff 2f e1      e12ffffe bx lr
0x3c812ebc: 0d c8 a0 e1      e1a0c80d mov r12, sp
0x3c812ec0: 70 01 2d e9      e92d0170 push {r4, r5, r6, r8}
0x3c812ec4: 70 01 9c e8      e89c0170 ldm r12, {r4, r5, r6, r8}
0x3c812ec8: 1f c8 e0 e3      e3e0c81f mvn r12, #31 ; 0x1f
0x3c812ecc: 80 00 00 ef      ef000080 svc 0x00000080
0x3c812ed0: 70 01 bd e8      e8bd0170 pop {r4, r5, r6, r8}

gdb$ _

```

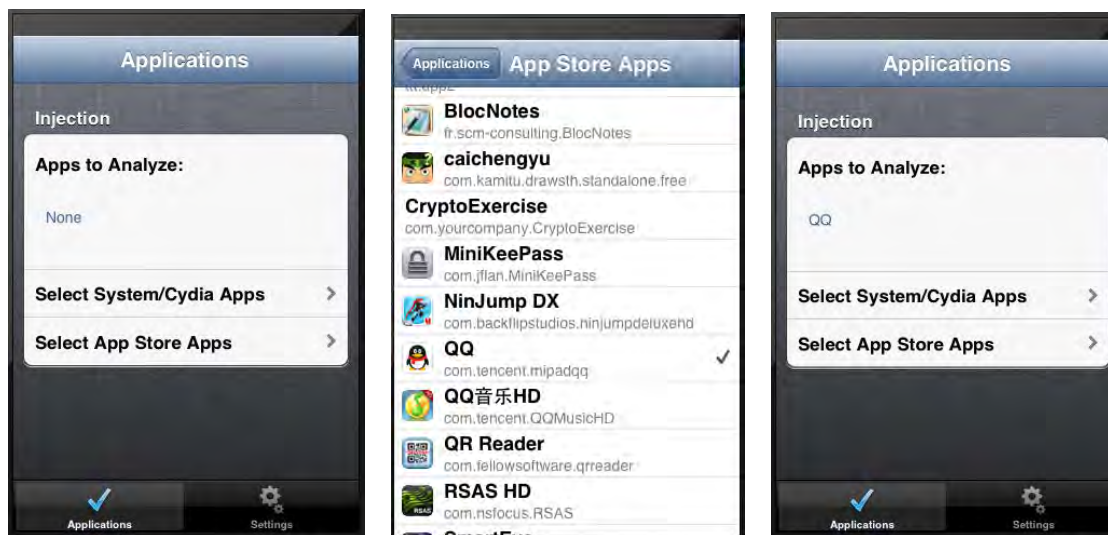
有用的链接：[这里](#)。

5.7 基于 hook 的测试工具

5.7.1 Snoop-it

Snoop-it 是一个 iOS 应用动态测试工具，其监控功能包括：监控文件访问、Keychain 访问、HTTP(S)连接（NSURLConnection）、敏感 API 访问（地址簿，图片等）等等，另外还可以对客户端的函数调用进行跟踪记录。

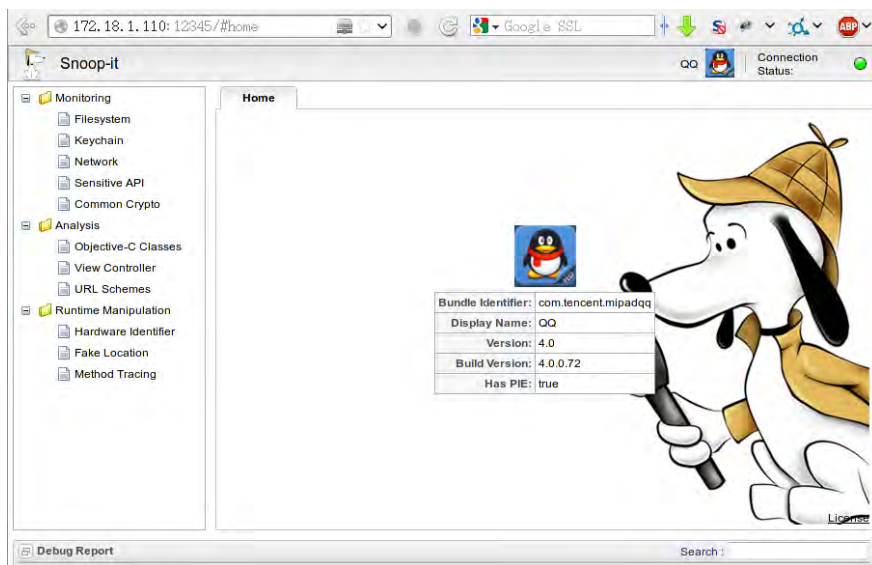
1. 首先在Cydia中添加snoop-it的软件源repo.nesolabs.de（[官方说明](#)）。从Cydia中安装snoop-it，然后在snoop-it的界面中选择要测试的客户端应用，此处可以一次选择多个应用。如图所示，这里选择的是QQ ipad客户端。



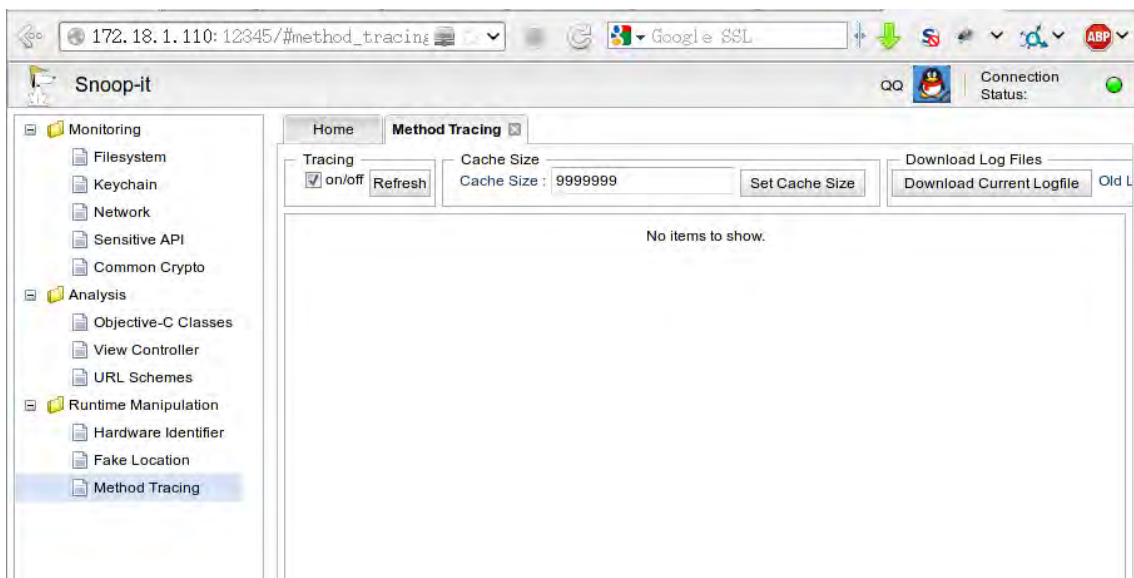
2. 在设置（settings）界面，可以设置监听端口和其他选项。



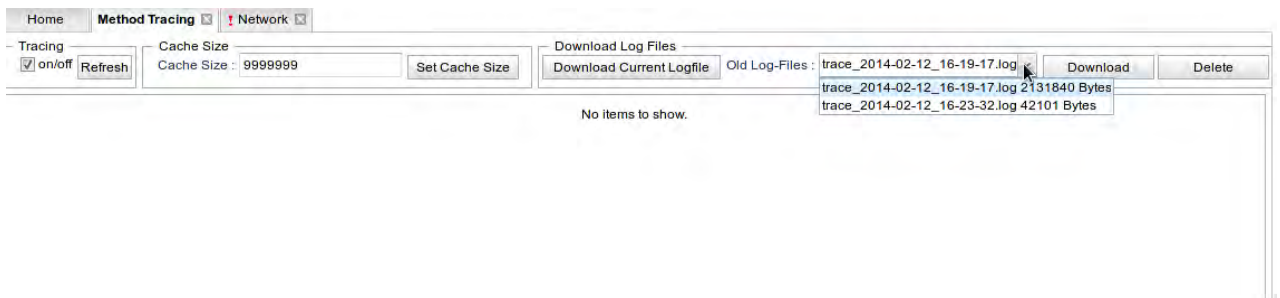
3. 运行 QQ 客户端。在 pc 上连接上面的地址，就可以看到 snoop-it 的 web 界面了。



4. 然后就可以正常使用各项功能了。这里重点说一下函数调用跟踪功能。如图是其界面，默认是关闭的。在“on/off”前打勾可以打开此项功能。由于各类应用的编码差异，这个功能有时无法正常显示记录，如图所示。



5. 虽然没有显示，但记录是有的。此时可以选择右边“old log-files”，将日志记录直接下载下来。如图：



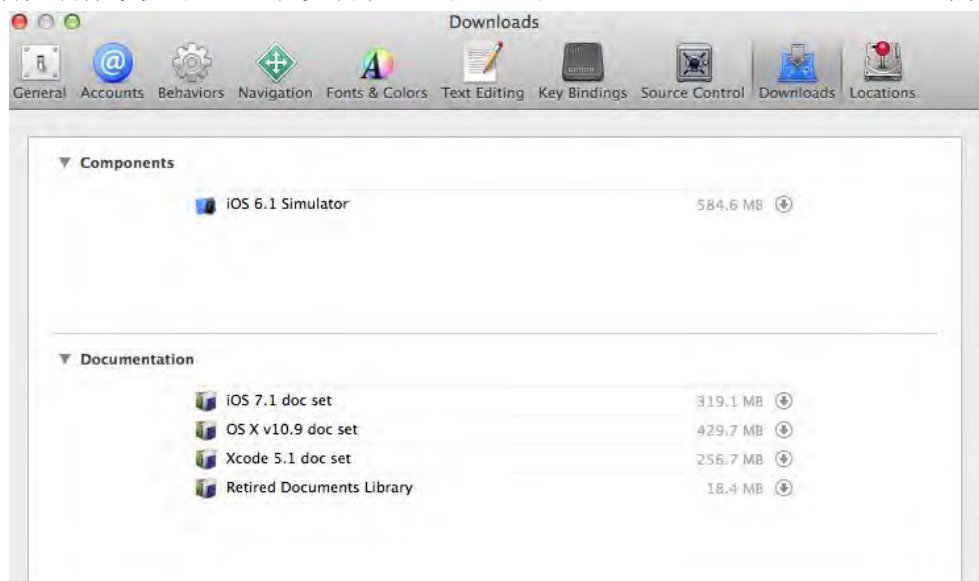
6. 从下载的日志中，就可以找到 qq 号和密码的明文。如图：

```
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQNavigationController(0xfa46510) childModalViewController]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQNavigationController(0xfa46510) loadViewIfNeeded]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQMessageView(0x1e2c5800) _systemGestureStateChanged:], args:
<0xf969880>
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQAboutScrollView(0xfa57a40) _systemGestureStateChanged:], args:
<0xf969880>
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQLoginViewController(0xfa46390)
performSelector:withObject:withObject:], args: @selector(doLogin), <0xf9421c0>, <0x1dda6510>
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQLoginViewController(0xfa46390) doLogin]
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountEditCellID(0xfa4dc20) idEdit]
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountEditCellPW(0xfa56580) pwEdit]
Wed Feb 12 16:23:56 2014 (Thread 0): - [PwTextField(0xfa57090) text]
Wed Feb 12 16:23:56 2014 (Thread 0): - [PwTextField(0xfa57090) _text]
Wed Feb 12 16:23:56 2014 (Thread 0): - [PwTextField(0xfa57090) textInputTraits]
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountHeaderView(0xfa48d10) changeAccountWith:], args:
<_NSString 0xf769560: 2486343802>
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountHeaderView(0xfa48d10) setUin:], args: <_NSString
0xf769560: 2486343802>
Wed Feb 12 16:23:56 2014 (Thread 0): - [QASynHeadImageView(0xf95f3d0) loadHeadImage:], args:
<_NSString 0xf769560: 2486343802>
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountHeaderView(0xfa48d10) setNeedsDisplay]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQLoginViewController(0xfa46390) doLogin:needPass:pass:], args:
<_NSString 0xf769560: 2486343802>, 1, <_NSString 0xf9d4350: 1234567890>
Wed Feb 12 16:23:56 2014 (Thread 0): + [AKNetworkReachability(0x2140450) networkStatus]
Wed Feb 12 16:23:56 2014 (Thread 0): + [QQDataCenter(0x213fe10) GetInstance]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQDataCenter(0xf20eaf0) accountsManager]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQAccountsManager(0xf2edd30) allAccounts]
Wed Feb 12 16:23:56 2014 (Thread 0): - [QQLoginViewController(0xfa46390) StartToLogin:needPass:pass:],
args: <_NSString 0xf769560: 2486343802>, 1, <_NSString 0xf9d4350: 1234567890>
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountEditCellID(0xfa4dc20) idEdit]
Wed Feb 12 16:23:56 2014 (Thread 0): - [AccountEditCellPW(0xfa56580) pwEdit]
Wed Feb 12 16:23:56 2014 (Thread 0): - [PwTextField(0xfa57090) resignFirstResponder]
```

5.7.2 Theos

Theos 是一个跨平台的 ios 开发环境。这里讲一下如何使用 theos 开发 tweak 组件。

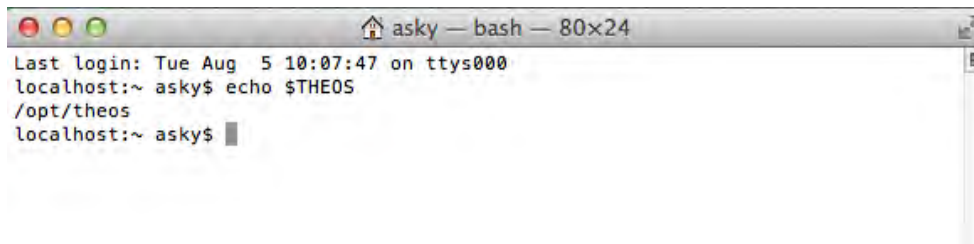
1. 首先确保本机的 xcode 以安装了 ios 的 sdk，在 xcode->Preference->Downloads 路径下



2. 配置 theos 的环境变量

建议将 theos 安装在 /opt/theos 下，并在 terminal 中输入一下命令
export THEOS = /opt/theos

可以通过 `echo $THEOS` 确认是否设置成功



```
asky — bash — 80x24
Last login: Tue Aug 5 10:07:47 on ttys000
localhost:~ asky$ echo $THEOS
/opt/theos
localhost:~ asky$
```

3. 下载 theos

使用如下命令，可直接将 theos 下到指定目录

```
svn co http://svn.howett.net/svn/theos/trunk $THEOS
```

4. 安装 ldid

ldid 的作用是模拟 iPhone 签名的流程，使得你能够在真实的设备上安装越狱的 apps/hacks，可使用以下命令。

```
curl -s http://dl.dropbox.com/u/3157793/ldid > ~/Desktop/ldidchmod +x ~/Desktop/ldid mv
~/Desktop/ldid $THEOS/bin/ldid
```

5. 安装 dpkg

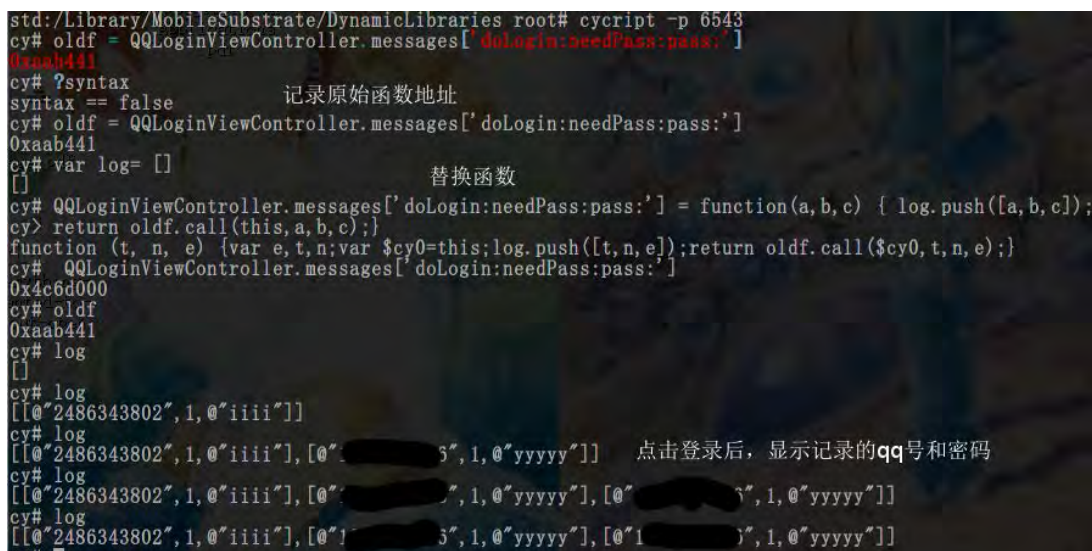
Dpkg 是一个专门用来制作 deb 的工具，theos 开发后的插件都是以 deb 的格式发布可在该链接下载 <http://www.macports.org/install.php>.

安装之后，在 terminal 中运行 `sudo port selfupdate`，以确保 MacPorts 升级到最新的版本，运行 `sudo port install dpkg`，该命令会安装不少 dpkg 的依赖软件，所以时间较长。

6. 至此 theos 的环境配置完毕。

5.7.3 cycrypt

[cycrypt](#) 是基于 [MobileSubstrate](#) 的脚本编程工具，可以通过脚本动态注入应用并修改应用。如图是使用 cycrypt 修改 QQ HD 应用的登录函数，获取用户输入的过程：



```
std:/Library/MobileSubstrate/DynamicLibraries root# cycrypt -p 6543
cy# oldf = QQLoginViewController.messages['doLogin:needPass:pass:']
0xaab441
cy# ?syntax
syntax == false
cy# oldf = QQLoginViewController.messages['doLogin:needPass:pass:']
0xaab441
cy# var log= []
[]
cy# QQLoginViewController.messages['doLogin:needPass:pass:'] = function(a,b,c) { log.push([a,b,c]);
cy> return oldf.call(this,a,b,c);}
function (t, n, e) {var e,t,n;var $cy0=this;log.push([t,n,e]);return oldf.call($cy0,t,n,e);}
cy# QQLoginViewController.messages['doLogin:needPass:pass:']
0x4c6d000
cy# oldf
0xaab441
cy# log
[]
cy# log
[[@2486343802",1,@"iiii"]
cy# log
[[@2486343802",1,@"iiii",[0" 3",1,@"yyyy"]
cy# log
[[@2486343802",1,@"iiii",[0" 3",1,@"yyyy"],[0" 3",1,@"yyyy"]
cy# log
[[@2486343802",1,@"iiii",[0" 3",1,@"yyyy"],[0" 1",1,@"yyyy"]
cy#
```

注：关于更多cycrypt使用示例，可参看 [此处](#)，[此处](#)。

5.8 其他命令

5.8.1 grep

用于对命令的输出进行过滤。

```
Std:/usr/bin root# netstat -anp tcp
Active Internet connections (including servers)
Proto Recv-Q Send-Q Local Address Foreign Address (state)
tcp4 0 0 192.168.31.185.22 192.168.31.159.53053 ESTABLISHED
tcp4 0 0 127.0.0.1.50771 127.0.0.1.50772 ESTABLISHED
tcp4 0 0 127.0.0.1.50772 127.0.0.1.50771 ESTABLISHED
tcp4 0 0 127.0.0.1.50737 127.0.0.1.50738 ESTABLISHED
tcp4 0 0 127.0.0.1.50738 127.0.0.1.50737 ESTABLISHED
tcp4 0 0 127.0.0.1.50735 127.0.0.1.50736 ESTABLISHED
tcp4 0 0 127.0.0.1.50736 127.0.0.1.50735 ESTABLISHED
tcp4 0 0 127.0.0.1.50733 127.0.0.1.50734 ESTABLISHED
tcp4 0 0 127.0.0.1.50734 127.0.0.1.50733 ESTABLISHED
tcp4 0 0 192.168.31.185.50717 17.143.163.155.5223 ESTABLISHED
tcp4 0 0 *.22 *. * LISTEN
tcp4 0 0 *.62078 *. * LISTEN
tcp4 0 0 127.0.0.1.8021 *. * LISTEN
tcp4 0 0 127.0.0.1.1082 *. * LISTEN
tcp4 0 0 127.0.0.1.1081 *. * LISTEN
tcp4 0 0 127.0.0.1.1084 *. * LISTEN
tcp4 0 0 127.0.0.1.1080 *. * LISTEN
tcp4 0 0 127.0.0.1.1083 *. * LISTEN
tcp4 0 0 127.0.0.1.62078 127.0.0.1.51033 TIME_WAIT
Std:/usr/bin root# netstat -anp tcp | grep LIST
tcp4 0 0 *.22 *. * LISTEN
tcp4 0 0 *.62078 *. * LISTEN
tcp4 0 0 127.0.0.1.8021 *. * LISTEN
tcp4 0 0 127.0.0.1.1082 *. * LISTEN
tcp4 0 0 127.0.0.1.1081 *. * LISTEN
tcp4 0 0 127.0.0.1.1084 *. * LISTEN
tcp4 0 0 127.0.0.1.1080 *. * LISTEN
tcp4 0 0 127.0.0.1.1083 *. * LISTEN
```

5.8.2 嗅探流量 tcpdump

tcpdump 程序可以嗅探指定网络接口的所有网络流量。在抓包前，可以使用“tcpdump -D”命令列出当前的网络接口。在抓包时，一般使用 en0。tcpdump 选项-i 指定监听的网络接口，-s 0 指定捕获完整数据包，-w 指定输出文件。命令使用如图所示。程序运行后用 Ctrl+C 结束。得到 capture.pcap 文件后，可以在 pc 上使用 Wireshark 进一步分析。

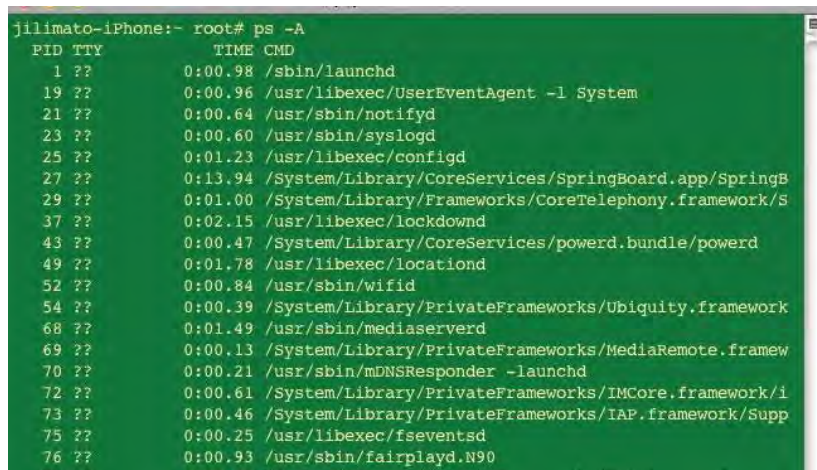
```
jilimato-iPhone:- root# tcpdump -D
1.pdp_ip0
2.en0
3.pdp_ip1
4.pdp_ip2
5.pdp_ip3
6.lo0
jilimato-iPhone:- root# tcpdump -i en0 -w capture.pcap -s 0
tcpdump: listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C604 packets captured
605 packets received by filter
0 packets dropped by kernel
jilimato-iPhone:- root#
```

5.8.3 网络信息查看 netstat

使用 netstat 查看网络连接情况。netstat -r, 可以查看 route table。

5.8.4 进程查看和监视 ps

ps -A 列举所有进程，如下图所示，最后一列为程序路径名，第一列是 PID。



```
jilimato-iPhone:- root# ps -A
PID TTY          TIME CMD
  1 ??            0:00.98 /sbin/launchd
 19 ??            0:00.96 /usr/libexec/UserEventAgent -l System
 21 ??            0:00.64 /usr/sbin/notifyd
 23 ??            0:00.60 /usr/sbin/syslogd
 25 ??            0:01.23 /usr/libexec/configd
 27 ??            0:13.94 /System/Library/CoreServices/SpringBoard.app/SpringB
 29 ??            0:01.00 /System/Library/Frameworks/CoreTelephony.framework/S
 37 ??            0:02.15 /usr/libexec/lockdownd
 43 ??            0:00.47 /System/Library/CoreServices/powerd.bundle/powerd
 49 ??            0:01.78 /usr/libexec/locationd
 52 ??            0:00.84 /usr/sbin/wifid
 54 ??            0:00.39 /System/Library/PrivateFrameworks/Ubiquity.framework
 68 ??            0:01.49 /usr/sbin/mediaserverd
 69 ??            0:00.13 /System/Library/PrivateFrameworks/MediaRemote.framew
 70 ??            0:00.21 /usr/sbin/mDNSResponder -launchd
 72 ??            0:00.61 /System/Library/PrivateFrameworks/IMCore.framework/i
 73 ??            0:00.46 /System/Library/PrivateFrameworks/IAP.framework/Supp
 75 ??            0:00.25 /usr/libexec/fsevents
 76 ??            0:00.93 /usr/sbin/fairplayd.N90
```

5.8.5 文件列举 lsof

lsof 命令可以显示所有进程打开的文件。其中第二列为打开文件的进程 pid，第一列是进程名，最后一列是打开的文件名。

```
jllimato-lPhone:~ root# ./lsnf-arm7-10S4.2
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
launchd 1 root cwd DIR 14,2 952 2 /
launchd 1 root txt REG 14,2 146640 40339 /sbin/launchd
launchd 1 root txt REG 14,2 64240 42501 /Library/Frameworks/CydiaSubstrate.framework/Librar
ies/SubstrateLauncher.dylib
launchd 1 root txt REG 14,2 202288 39565 /usr/lib/dyld
launchd 1 root txt REG 14,2 200301298 41413 /System/Library/Caches/com.apple.dyld/dyld_shared_c
ache_armv7
launchd 1 root 0r CHR 3,2 0t0 71 /dev/null
launchd 1 root 1r CHR 3,2 0t0 71 /dev/null
launchd 1 root 2r CHR 3,2 0t0 71 /dev/null
launchd 1 root 3u KQUEUE 0,0 0t75 69 /dev/console
launchd 1 root 4w CHR 0,0 0t0 71 /dev/null
launchd 1 root 5u system 0x8031cb04 0t0 68 /dev
launchd 1 root 6r DIR 129,12206052 1667 68 /dev
launchd 1 root 7u unix 0x83d7de40 0t0 43 /private/var/mobile/Library/Preferences
launchd 1 root 8r DIR 14,3 2856 43 /private/var/mobile/Library/Preferences
launchd 1 root 9u IPV6 0x849f8da0 0t0 TCP *:ssh (LISTEN)
launchd 1 root 10u IPV4 0x849b9d98 0t0 TCP *:ssh (LISTEN)
launchd 1 root 11u IPV4 0x849b9868 0t0 TCP localhost:sun-as-jpda (LISTEN)
launchd 1 root 12u unix 0x83d7dc78 0t0 /var/run/hostapd.socket
launchd 1 root 13u unix 0x83d7dd10 0t0 /var/run/mdnsResponder
launchd 1 root 14u unix 0x83d7dbe0 0t0 /var/run/lockbot
launchd 1 root 15u IPV4 0x849b9338 0t0 TCP *:62078 (LISTEN)
launchd 1 root 16u IPV6 0x849f8a80 0t0 TCP *:62078 (LISTEN)
launchd 1 root 17u unix 0x83d7db48 0t0 /var/run/lockdown.sock
launchd 1 root 18u unix 0x83d7dab0 0t0 /var/run/printd
launchd 1 root 19u unix 0x83d7da18 0t0 /var/run/vpncontrol.sock
launchd 1 root 20u unix 0x83d7d980 0t0 /var/run/syslog
launchd 1 root 21u unix 0x83d7d8e8 0t0 /var/run/asl_input
launchd 1 root 22u IPV6 0x849f8760 0t0 TCP localhost:intu-ec-client (LISTEN)
```

查看系统网络连接的详细信息，包括进程，PID，用户，连接类型，端口等

```
Std:/usr/bin root# lsof -Pnl +M -i4
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
launchd 1 0 7u IPV4 0x232b9bb7 0t0 TCP 127.0.0.1:1083 (LISTEN)
launchd 1 0 8u IPV4 0x232b9517 0t0 TCP 127.0.0.1:1080 (LISTEN)
launchd 1 0 9u IPV4 0x232b8e77 0t0 TCP 127.0.0.1:1084 (LISTEN)
launchd 1 0 10u IPV4 0x232b87d7 0t0 TCP 127.0.0.1:1081 (LISTEN)
launchd 1 0 11u IPV4 0x232b8137 0t0 TCP 127.0.0.1:1082 (LISTEN)
launchd 1 0 14u IPV4 0x232b9517 0t0 TCP 127.0.0.1:1080 (LISTEN)
launchd 1 0 15u IPV4 0x232b9bb7 0t0 TCP 127.0.0.1:1083 (LISTEN)
launchd 1 0 16u IPV4 0x232b8137 0t0 TCP 127.0.0.1:1082 (LISTEN)
launchd 1 0 17u IPV4 0x232b87d7 0t0 TCP 127.0.0.1:1081 (LISTEN)
launchd 1 0 18u IPV4 0x232b8e77 0t0 TCP 127.0.0.1:1084 (LISTEN)
launchd 1 0 19u IPV4 0x232b7a97 0t0 TCP 127.0.0.1:8021 (LISTEN)
launchd 1 0 24u IPV4 0x232b7a97 0t0 TCP 127.0.0.1:8021 (LISTEN)
launchd 1 0 27u IPV4 0x232b73f7 0t0 TCP *:62078 (LISTEN)
launchd 1 0 28u IPV4 0x23da0a97 0t0 TCP 192.168.31.185:22->192.168.31.159:53053 (ESTABLISHED)
launchd 1 0 34u IPV4 0x232b6d57 0t0 TCP *:22 (LISTEN)
launchd 1 0 37u IPV4 0x23da0a97 0t0 TCP 192.168.31.185:22->192.168.31.159:53053 (ESTABLISHED)
launchd 1 0 40u IPV4 0x232b6d57 0t0 TCP *:22 (LISTEN)
lockdown 43 0 6u IPV4 0x232b73f7 0t0 TCP *:62078 (LISTEN)
lockdown 43 0 9u IPV4 0x238da517 0t0 TCP 127.0.0.1:62078->127.0.0.1:51017 (ESTABLISHED)
discovery 123 65 8u IPV4 0x230d894b 0t0 UDP *:5353
discovery 123 65 28u IPV4 0x230d53db 0t0 UDP *:5353
discovery 123 65 32u IPV4 0x230d777b 0t0 UDP *:5353
discovery 123 65 36u IPV4 0x238c85a3 0t0 UDP *:5353
wifid 127 0 4u IPV4 0x230d7b0b 0t0 UDP *:5353
wifid 127 0 8u IPV4 0x230d85bb 0t0 UDP *:5353
apsd 200 501 16u IPV4 0x232b66b7 0t0 TCP 192.168.31.185:50717->17.143.163.155:5223 (ESTABLISHED)
apsd 200 501 17u IPV4 0x232b66b7 0t0 TCP 192.168.31.185:50717->17.143.163.155:5223 (ESTABLISHED)
afcd 893 501 8u IPV4 0x238ef3f7 0t0 TCP 127.0.0.1:50733->127.0.0.1:50734 (ESTABLISHED)
notificat 899 501 6u IPV4 0x23beabb7 0t0 TCP 127.0.0.1:50737->127.0.0.1:50738 (ESTABLISHED)
afcd 1497 0 6u IPV4 0x238ee017 0t0 TCP 127.0.0.1:50735->127.0.0.1:50736 (ESTABLISHED)
springboa 1549 501 5u IPV4 0x23abb517 0t0 TCP 127.0.0.1:50771->127.0.0.1:50772 (ESTABLISHED)
sshd 1566 0 4u IPV4 0x23da0a97 0t0 TCP 192.168.31.185:22->192.168.31.159:53053 (ESTABLISHED)
sshd 1566 0 5u IPV4 0x23da0a97 0t0 TCP 192.168.31.185:22->192.168.31.159:53053 (ESTABLISHED)
```

查看某一端口的运行情况

```
Std:/usr/bin root# lsof -i :50771
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
springboa 1549 mobile 5u IPV4 0x23abb517 0t0 TCP localhost:50771->localhost:50772 (ESTABLISHED)
```

如果Cydia中的lsnf不能正常运行，可以尝试从[此处](#)下载新的lsnf（相关[讨论](#)）。

5.8.6 数据库文件查看 sqlite3

sqlite3 的命令可以让用户手工输入并执行面向SQLite数据库的SQL命令。系统命令以.开头，可以通过.help命令详细查看（查看所有数据可使用.dump）。SQL命令须以;结尾。首先以db文件为参数进入sqlite3 命令行，然后就可以对挂载的数据库进行操作。查看手册点[此处](#)。

5.8.7 动态共享库修改 Install_name_tool

使用 install_name_tool，可以修改程序加载的动态共享库位置。（参见 man 手册）

Install_name_tool -change old_path new_path file

外部参考：Apple的[手册页](#)，[dyld手册页](#)。

5.8.8 对象文件查看工具 otool

otool 是 xcode 自带的命令行工具。类似于 Linux 上的 objdump。otool -L 可以查看加载库的信息。如图：

```
-Q use otool(1)'s disassembler
Mac-Pro:bin asky$ ./otool -L /Users/asky/Library/Developer/Xcode/DerivedData/hook-dtthcyxxwqyghxasgeiorkakljzl/Build/P
roducts/Debug-iphonesimulator/hook.dylib
/Users/asky/Library/Developer/Xcode/DerivedData/hook-dtthcyxxwqyghxasgeiorkakljzl/Build/Products/Debug-iphonesimulator
/hook.dylib:
    /Library/MobileSubstrate/DynamicLibraries/hook.dylib (compatibility version 1.0.0, current version 1.0.0)
    /System/Library/Frameworks/Foundation.framework/Foundation (compatibility version 300.0.0, current version 993
.0.0)
    /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current version 227.0.0)
    /usr/lib/libstdc++.6.dylib (compatibility version 7.0.0, current version 56.0.0)
    /usr/lib/libSystem.dylib (compatibility version 1.0.0, current version 125.0.0)
Mac-Pro:bin asky$ _
```

外部参考：Apple的[otool手册](#)。

5.8.9 密钥链查看工具 Keychaindumper

keychaindumper是用于查看iOS设备上的Keychain（密钥链）内容的工具。推荐从此[地址下载](#)工具。Cydia中的keychaindumper功能较少，不建议使用。

在命令行输入 keychain_dumper -a 就可以显示所有类型的数据，如图所示：

```
asky — Terminal — ssh — 118x23
std:~ mobile$ ./keychain_dumper -a
Generic Password
-----
Service: BluetoothGlobal
Account: Identity Root
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>KEY</key>
  <data>
    UBrSRXMHAF8KeHBDi6imQg==
  </data>
</dict>
</plist>
Generic Password
-----
Service: BluetoothGlobal
```

5.8.10 签名工具 ldid

ldid 工具有三个功能，给未签名程序签名（-S），程序修改后更新程序签名散列（-s），以及显示程序内嵌的 entitlement（-e）。下图是某个应用 entitlement 的显示：

```
std:/var/mobile/Applications/0C58C300-2284-4A49-AA0F-7A5DDF0B2401/com.spdb.retail.bank.hd.app root# ldi
d -e PufaBankForIpad
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
  <dict>
    <key>keychain-access-groups</key>
    <array>
      <string>ES79Y6ELVC.com.spdb.retail.bank.hd</string>
    </array>
    <key>application-identifier</key>
    <string>ES79Y6ELVC.com.spdb.retail.bank.hd</string>
  </dict>
</plist><?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

5.9 根证书安装

1. 将需要的根证书复制到手机上。（可以通过邮件附件方式）
2. 打开证书，按照 iOS 提示安装。
3. 在设置->通用->描述文件中，可以看到已安装的证书。



5.10 连接到 iOS 设备

设备越狱后，安装OpenSSH软件包，就可以通过ssh登录到设备上，然后运行命令行命令。当电脑和设备在同一子网时，可以直接使用设备的ip连接。其他方法还可以参考 [此处](#)。

5.10.1 Windows 平台

1. 当设备使用 usb 线与电脑连接时，可使用多种方法建立隧道，进行 ssh 连接。可以使用 iTools（3.x）工具箱->高级功能->SSH 隧道，建立 ssh 隧道。
2. 可以使用 [itunnel_mux](#)建立隧道，如图所示：

```
C:\Documents and Settings\Administrator>C:\root\iphone\itunnel_mux.exe --lport 22 --lport 22
[ERROR] locate_AMRecoveryModeDeviceSendFileToDevice: Could not locate function p
rolog!
[INFO] Waiting for new TCP connection on port 22
[INFO] Waiting for device...
[INFO] Device connected: 495a787df496ba46ff94bc0be564bfef2a767003
[INFO] Info: New connection...
[INFO] Device connected
[INFO] USBMuxConnectByPort OK
```

5.10.2 Mac OSX 平台

1. 在Mac上，也可以使用 [itunnel_mux](#)建立隧道，如图所示：

```
localhost:itnl_rev8 asky$ ./itnl --lport 22 --lport 2022
[INFO] Waiting for new TCP connection on port 2022
[INFO] Waiting for device...
[INFO] Device connected: 495a787df496ba46ff94bc0be564bfef2a767003
[INFO] Info: New connection...
[INFO] Device connected
[INFO] USBMuxConnectByPort OK
```

2. 或者使用 [iPhone Tunnel](#)，运行后在Mac的菜单栏会有提示图标，选择“Turn Tunnel On”，然后选择Tools中的SSH，就可以了。

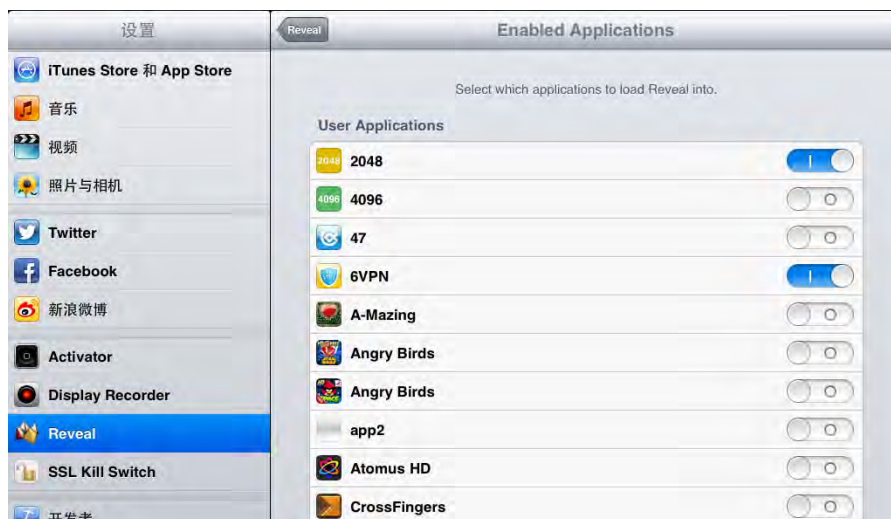


5.11 UI 元素查看

5.11.1 Reveal

安装 Reveal 的 Mac 和移动设备需要在同一网段内，在移动设备 Cydia 中安装 Reveal Loader。

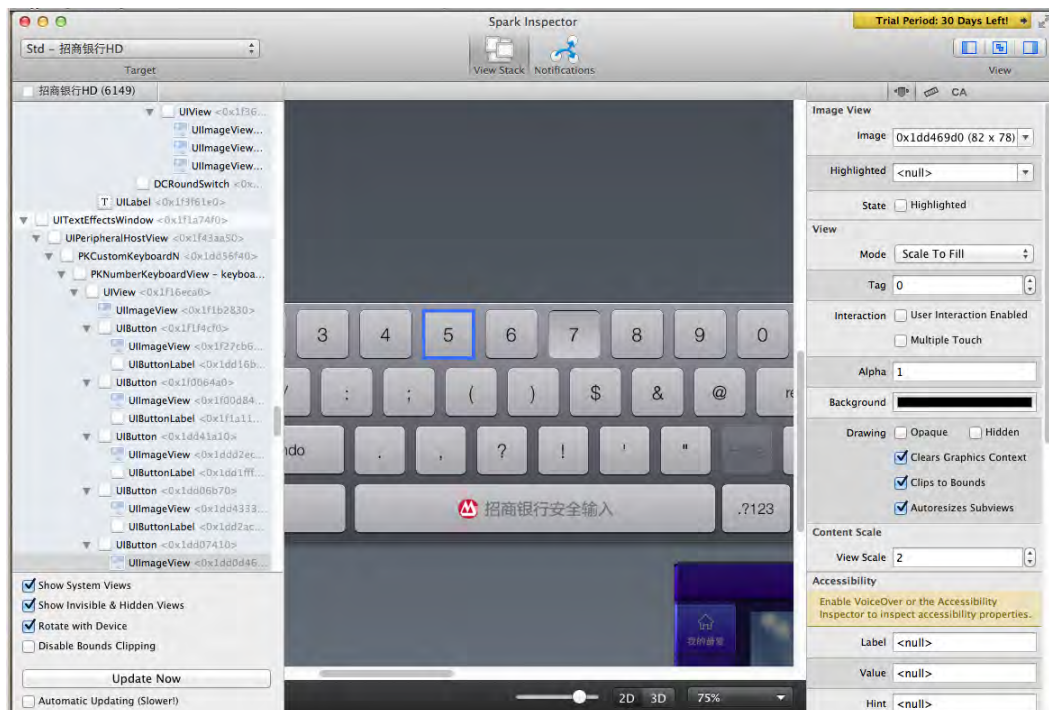
在设备中设置需要查看的应用：



运行要查看的应用，在 Mac 中打开 Reveal，就可以看到 UI 的结构了（vmware 中看不到中间的图像，因为没有 3D 加速）



运行目标应用后，显示如图所示，对于较为复杂的应用，程序运行会比较慢。



About Mach-O

About signature ,hash...

keychain-access-groups

```
<plist version="1.0">
  <dict>
    <key>keychain-access-groups</key>
    <array>
      <string>YX26KRYDDN.com.app.vpn</string>
    </array>
  </dict>
</plist>
```

使用方法: <http://useyourloaf.com/blog/2010/04/03/keychain-group-access.html>

Programming with Objective-C

<https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html>

iPhone Configuration Utility (Mac) <https://support.apple.com/kb/DL1465>

(windows) <https://support.apple.com/kb/DL1466>

<https://itunes.apple.com/us/app/apple-configurator/id434433123?mt=12>

http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=iPhonDbg_Toolkit

<https://github.com/jmoody>

Vpn plugin??

附录A 疑难解答

以下是一些在实际测试中所遇到的一些小问题，在此记录，以供方便查阅。

1. 关于手机上的 ssh server，如果运行出现：

Could not load host key: /etc/ssh/ssh_host_rsa_key

Could not load host key: /etc/ssh/ssh_host_dsa_key

Disabling protocol version 2. Could not load host key

sshd: no hostkeys available — exiting.

可以在手机上运行如下命令：

```
ssh-keygen -t rsa1 -f /etc/ssh/ssh_host_key -N ""
```

```
ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key -N ""
```

```
ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key -N ""
```