Hacking iOS Applications

a detailed testing guide

SECURITY
INNOVATION

Table of Contents

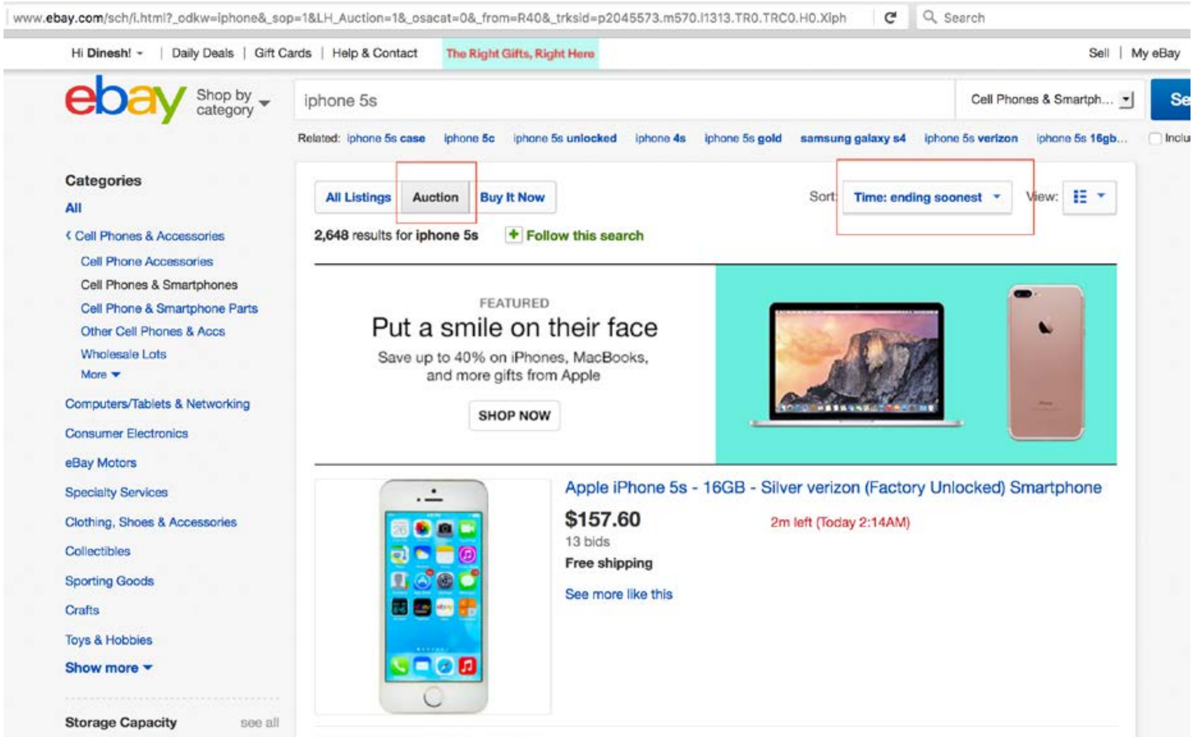
1. Setting Up iOS Pentest Lab

Setting up a device is one of the first priorities before starting a scheduled project. If setting up an iOS device for the first time, it's likely that something may break (even if the device is one that has been used previously), so it's best to test the device a couple of days before the pentest begins to ensure that the tools in it still work.

1.1 Get an iOS Device

A reliable source for iOS devices is eBay (https://www.ebay.com/). iOS updates and hardware compatibility can be an issue with Apple products, so always try to buy one of the newer devices. As of the publication of this guide, the latest iPhone in the market is Apple iPhone 7/7+ and the oldest phone recommended is the Apple iPhone 5s. An iPad Mini is also a good option. If using a new iOS device is preferable, but test cases related to network carrier usage aren't a concern, consider an iPod Touch 6th generation. They are relatively inexpensive compared other new devices that run the latest iOS releases. For best results, choose an iOS version greater than 9.0+.

NOTE: When trying to buy a device on eBay use the "Auction" functionality in conjunction with the "Time: ending soonest" filter.



www.ebay.com/sch/i.html?_odkw=iphone&_sop=1&LH_Auction=1&_osacat=0&_from=R40&_trksid=p2045573.m570.l1313.TR0.TRC0.H0.Xiph

iphone 5s

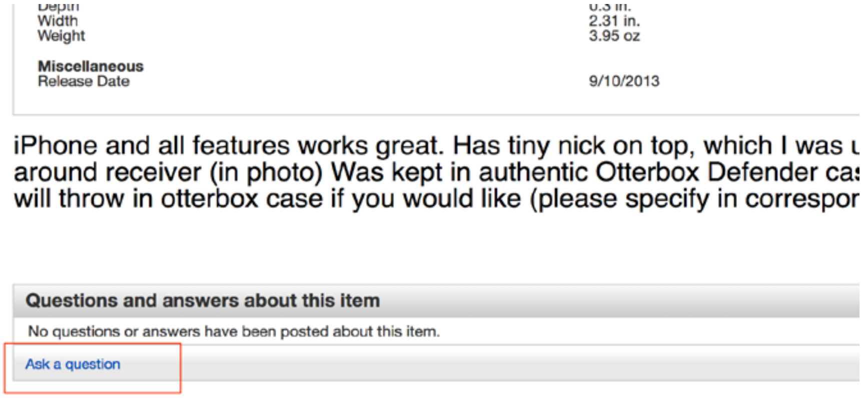
Sort: Time: ending soonest

2,648 results for iphone 5s

FEATURED
Put a smile on their face
Save up to 40% on iPhones, MacBooks, and more gifts from Apple

Apple iPhone 5s - 16GB - Silver verizon (Factory Unlocked) Smartphone
$157.60
13 bids
Free shipping
2m left (Today 2:14AM)

Unlocked devices with at least 32GB memory are preferable as they provide enough space to update the device and install all tools. Keep in mind that not all iOS versions can be jailbroken so choose a device that has a public Jailbreak available (refer to the Jailbreak section in this guide for determining if the iOS version of a device can be jailbroken). If the product description does indicate the iOS version running on the device you are considering, message the seller to confirm the iOS version. To message the seller, open the product page, go to the end of the description, and click on the link as shown below.



Depth	0.3 in.
Width	2.31 in.
Weight	3.95 oz
Miscellaneous	
Release Date	9/10/2013

iPhone and all features works great. Has tiny nick on top, which I was u
around receiver (in photo) Was kept in authentic Otterbox Defender cas
will throw in otterbox case if you would like (please specify in correspor

Questions and answers about this item

No questions or answers have been posted about this item.

Ask a question

1.2 Jailbreaking an iOS Device

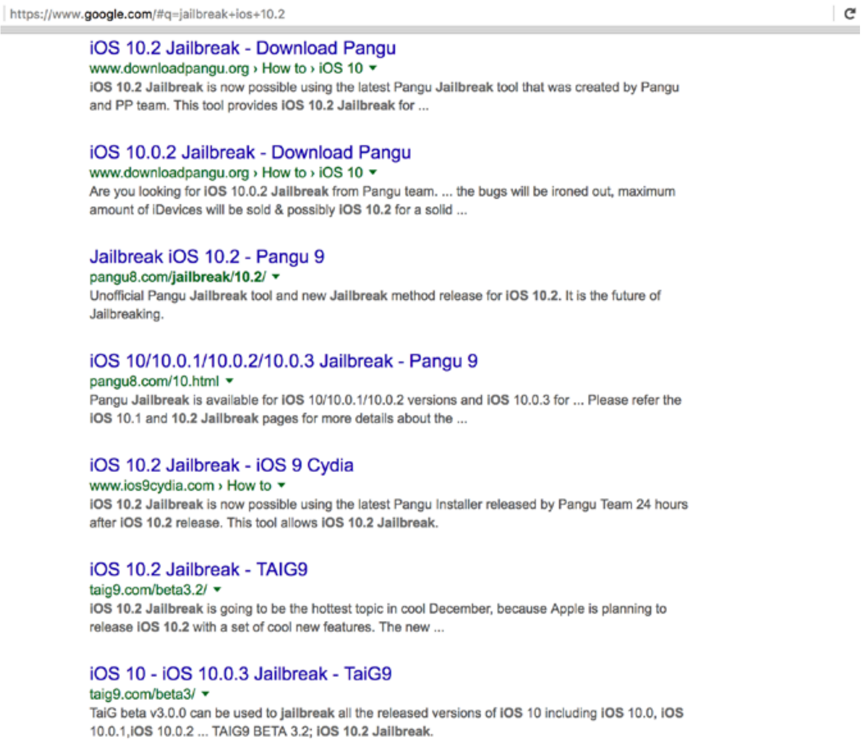
Jailbreaking is the process of gaining root access to the entire device. The best approach for security testing an application is to examine it on a jailbroken device. Jailbreaking an iOS device allows for:

- Removing the security (and other) limitations on the OS imposed by Apple
- Providing root access to the operating system
- Allowing important testing software tools to be installed
- Providing access to the Objective-C Runtime

iOS applications store data in the application sandbox which is not accessible to the public (but is available to root and the application itself). Without root access, it is not possible to access the application sandbox, see what data is being stored, and how is it stored. Also, most the system level files are owned by root.

The process for jailbreaking various iOS versions can be quite different. Instructions for jailbreaking iOS devices are found via a simple Google search. Be aware, however, that the Google links may not be legitimate even if they include names that are the same as genuine jailbreak tools.

Example:



https://www.**google.com**/#q=jailbreak+ios+10.2

iOS 10.2 Jailbreak - Download Pangu
www.downloadpangu.org › How to › iOS 10
iOS 10.2 Jailbreak is now possible using the latest Pangu **Jailbreak** tool that was created by Pangu and PP team. This tool provides **iOS 10.2 Jailbreak** for ...

iOS 10.0.2 Jailbreak - Download Pangu
www.downloadpangu.org › How to › iOS 10
Are you looking for **iOS** 10.0.2 **Jailbreak** from Pangu team. ... the bugs will be ironed out, maximum amount of iDevices will be sold & possibly **iOS 10.2** for a solid ...

Jailbreak iOS 10.2 - Pangu 9
pangu8.com/**jailbreak/10.2/**
Unofficial Pangu **Jailbreak** tool and new **Jailbreak** method release for **iOS 10.2.** It is the future of Jailbreaking.

iOS 10/10.0.1/10.0.2/10.0.3 Jailbreak - Pangu 9
pangu8.com/10.html
Pangu **Jailbreak** is available for **iOS** 10/10.0.1/10.0.2 versions and **iOS** 10.0.3 for ... Please refer the **iOS** 10.1 and **10.2 Jailbreak** pages for more details about the ...

iOS 10.2 Jailbreak - iOS 9 Cydia
www.ios9cydia.com › How to
iOS 10.2 Jailbreak is now possible using the latest Pangu Installer released by Pangu Team 24 hours after **iOS 10.2** release. This tool allows **iOS 10.2 Jailbreak.**

iOS 10.2 Jailbreak - TAIG9
taig9.com/beta3.2/
iOS 10.2 Jailbreak is going to be the hottest topic in cool December, because Apple is planning to release **iOS 10.2** with a set of cool new features. The new ...

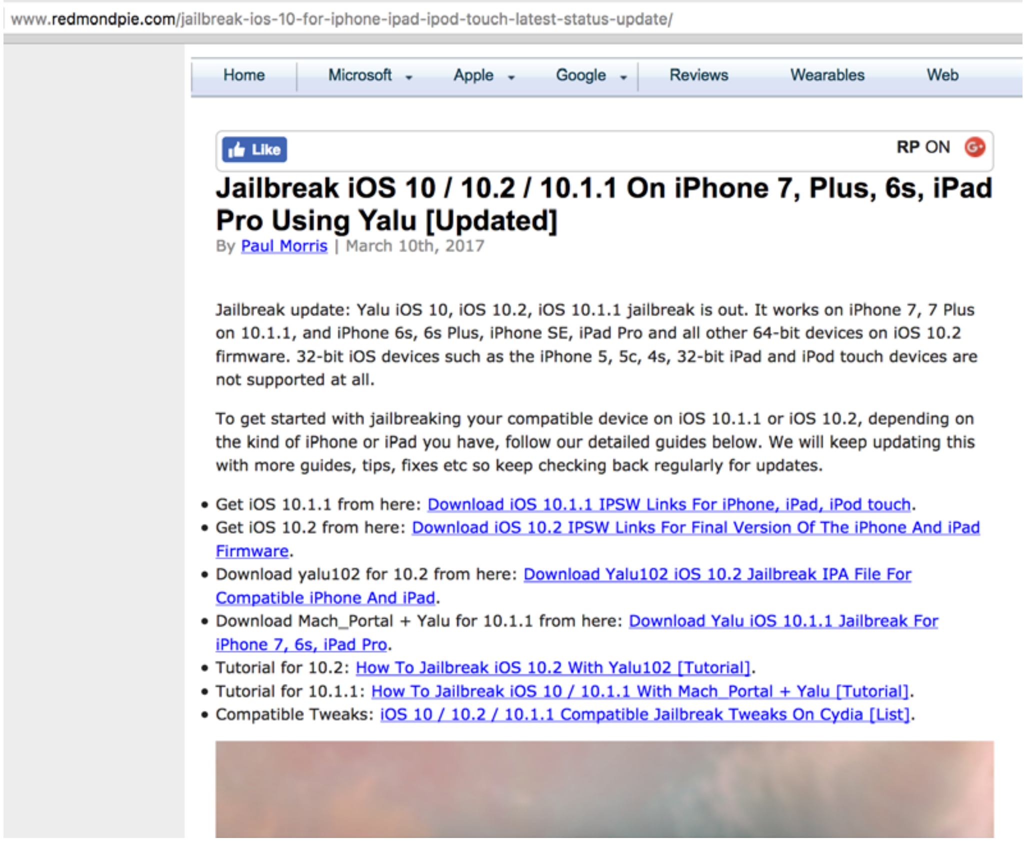
iOS 10 - iOS 10.0.3 Jailbreak - TaiG9
taig9.com/beta3/
TaiG beta v3.0.0 can be used to **jailbreak** all the released versions of **iOS** 10 including **iOS** 10.0, **iOS** 10.0.1,**iOS** 10.0.2 ... TAIG9 BETA 3.2; **iOS 10.2 Jailbreak.**

The above example shows that many of the results include “pangu” and “taig” (legitimate jailbreak tools) but none of the links for iOS 10.2 are genuine.

Recommended Websites:

- https://www.theiphonewiki.com/wiki/Jailbreak A reliable website to check if Jailbreak for an iOS device is available and what software to use
- https://www.redmondpie.com/ Includes walkthrough guides with links to the real software
- https://www.reddit.com/r/jailbreak/ Good resource to keep track of updated jailbreak events around the world (note: use with caution and double check information found on this site)

Use the guide below to jailbreak an iOS 10.2.1 device:
http://www.redmondpie.com/jailbreak-ios-10-for-iphone-ipad-ipod-touch-latest-status-update/


www.redmondpie.com/jailbreak-ios-10-for-iphone-ipad-ipod-touch-latest-status-update/
Home
Microsoft
Apple
Google
Reviews
Wearables
Web
Like
RP ON

Jailbreak iOS 10 / 10.2 / 10.1.1 On iPhone 7, Plus, 6s, iPad Pro Using Yalu [Updated]

By Paul Morris | March 10th, 2017

Jailbreak update: Yalu iOS 10, iOS 10.2, iOS 10.1.1 jailbreak is out. It works on iPhone 7, 7 Plus on 10.1.1, and iPhone 6s, 6s Plus, iPhone SE, iPad Pro and all other 64-bit devices on iOS 10.2 firmware. 32-bit iOS devices such as the iPhone 5, 5c, 4s, 32-bit iPad and iPod touch devices are not supported at all.

To get started with jailbreaking your compatible device on iOS 10.1.1 or iOS 10.2, depending on the kind of iPhone or iPad you have, follow our detailed guides below. We will keep updating this with more guides, tips, fixes etc so keep checking back regularly for updates.

- Get iOS 10.1.1 from here: Download iOS 10.1.1 IPSW Links For iPhone, iPad, iPod touch.
- Get iOS 10.2 from here: Download iOS 10.2 IPSW Links For Final Version Of The iPhone And iPad Firmware.
- Download yalu102 for 10.2 from here: Download Yalu102 iOS 10.2 Jailbreak IPA File For Compatible iPhone And iPad.
- Download Mach_Portal + Yalu for 10.1.1 from here: Download Yalu iOS 10.1.1 Jailbreak For iPhone 7, 6s, iPad Pro.
- Tutorial for 10.2: How To Jailbreak iOS 10.2 With Yalu102 [Tutorial].
- Tutorial for 10.1.1: How To Jailbreak iOS 10 / 10.1.1 With Mach_Portal + Yalu [Tutorial].
- Compatible Tweaks: iOS 10 / 10.2 / 10.1.1 Compatible Jailbreak Tweaks On Cydia [List].

Since this is a legitimate site, these links may be used to download the proper IPA or source code for the jailbreak application. This site also includes helpful walkthrough guides.

A quick Redmond Pie search will confirm whether there are jailbreak steps for various IOS versions, what they are, and how to implement them.

NOTE: Never use the "reset all content and settings" option on a jailbroken iOS device as it will ALWAYS get stuck in a reboot loop. When this happens, the device will need to be restored (to latest version most likely). If a reboot loop occurs, try the steps mentioned in the links below to fix:

- https://www.qdtricks.net/how-to-fix-iphone-stuck-on-apple-logo/

- https://support.apple.com/en-in/HT201263
- http://www.ikream.com/2016/02/how-to-fix-apple-iphone-6-boot-loop-blod-and-other-power-related-issues-troubleshooting-guide-23912 http://www.iphonehacks.com/2016/08/fix-boot-loop-jailbreak-ios-9-3-3-iphone-ipad-ipod-touch.html

1.3 Installing Required Software and Utilities

After jailbreaking an iOS device, the following utilities will need to be installed. The majority of the tools, if not all, can be installed from Cydia. Cydia is a GUI wrapper for apt and, once apt is installed, the rest can be installed via command line. Cydia is preferred due to the ease of use.
Installation steps for many of these tools are covered elsewhere in this guide.

- OpenSSH
    - A utility to provide users the ability to connect remotely to the iOS FileSystem. OpenSSH utility is broken in the iOS 10.2 jailbreak released by Luca, however there is a default DropBear SSH service running on the device to make sure that SSH access isn't missed.
        - Connect to DropBear using the same steps as mentioned in Method 8 (Reading Application Data using SSH over USB)
        - IMPORTANT: change the OpenSSH password as soon as OpenSSH is installed.
- BigBoss Recommended Tools
    - A collection of all the recommended hacker CLI tools like wget, tar, vim etc., that do not come pre-installed with the Cydia repo.
- Cydia substrate
    - An important requirement for many of the tweaks and tools included in this guide. Required for modifying the software during the runtime on the device without access to the source code. Tools like Cycript need Cydia Substrate installed.
    - Be wary of installing third-party patches on latest iOS. Patches by Ijapija00 for iOS 10 and 10.1.1 were found to cause devices to break.
- APT 0.6 transitional (apt-get command)
    - Packaging tools for iOS
- Class-dump-z, class-dump, classdump-dyld
    - A reverse engineering tool for iOS that helps dump declarations for the classes, categories and protocols.
- Cycript
    - A utility that provides a mechanism to modify applications during runtime using a combination of Objective-C++ and JavaScript syntax.
- IPA installer console

    - A command-line utility to install third party applications on a jailbroken iOS device.
- AppSync
    - An iOS tweak that allows for the installation of a modified and fake signed IPA package on the iOS device.
    - Make sure whether Jailbreak supports this tool or the device might end up in reboot loop.
        - AppSync is temporarily broken in iOS 10.2 jailbreak so installation is not recommended.
- Clutch from the iphonecake repo (com.iphonecake.clutch2)
    - A utility that allows users to dump decrypted iOS binaries from a jailbroken device.
- GDB from the repo cydia.radare.org
    - The GNU Debugger for jailbroken IOS on arm64.
- MTerminal
    - An on-device terminal for running commands on the iOS device without the need for a separate laptop.
- Filemon
    - A real-time iOS Filesystem Monitoring software.
    - Can be downloaded from www.newosxbook.com
- Introspy-iOS
    - A tool to help security researchers profile the iOS applications using a blackbox approach
    - Can be downloaded from https://github.com/iSECPartners/Introspy-iOS
- SSL Kill Switch 2
    - A tool to help bypass SSL validation and SSL pinning in iOS applications
    - Can be downloaded from https://github.com/nabla-c0d3/ssl-kill-switch2

On a laptop, the software below will need to be installed:

- Hopper
    - An inexpensive, but useful, reverse engineering tool to help disassemble, decompile and debug iOS applications.
- IDA Pro
    - An expensive, but advanced, tool to aid iOS reverse engineering.
- Burp Suite
    - An interception proxy to perform MITM on iOS applications.
- idb
    - A tool to aid many of the commonly seen iOS application test cases.
- FileDP
    - A tool to help extraction of data protection class from files on iOS device.
    - Can be downloaded from http://www.securitylearn.net/wp-content/uploads/tools/iOS/FileDP.zip

- Libimobiledevice
  - An excellent cross-platform protocol library to access iOS devices.
  - Can be downloaded from https://github.com/libimobiledevice/

2. Acquiring iOS Binaries

Customers will not always provide an .IPA file for a pentest. Below are some alternative ways to acquire iOS Binaries for analyzing.

1. Open iTunes App Store on Mac. Download the application from the App Store using Mac Native application. Select “Apps” and select Application name in the “Library.” Right click and select “Show in Finder” to get the iPA path. Normally it is /Users/<username>/Music/iTunes/iTunes Media/Mobile Applications/

2. When the device is synced with iTunes, the .IPA file is sent to the iTunes folder. Pull the .IPA file from the iTunes folder. (Works on non-jailbroken devices)

3. Use a tool like iMazing. Launch iMazing and connect the iOS device to the laptop. Click on Apps. Select the application binary to be extracted. Click on Manage Apps at the bottom of the view. Click on Extract App - then choose a location for the app to be stored on the computer. (Works well on apps before 9.0. Versions after 9.0 do not work well)

4. Use a tool like iFunBox. Launch iFunBox and connect the iOS device. Click on iFunBox Class tab and then in the “Connected Devices” section, select the iOS device. Click on User Applications. Select the application to be extracted. Right click and select “Backup to .ipa Package.” Save the application to any location. (Works only up to iOS 8.3 or on a jailbroken device)

5. Use iTools. Connect device. Click on Apps. Select application. Right click and select archive to get the application binary. (Works only up to iOS 8.3 or on a jailbroken device)

6. With access to the source code, it is possible to compile the application binarydirectly. This is helpful when working with older jailbroken devices as it allows for compile the application to run on the older device and perform the testing.

7. Download the application from the App Store. The problem with using these binaries for testing are that they are encrypted for your protection and for digital rights management (DRM). Techniques on breaking the FairPlayDRM and perform analysis of the encrypted App Store binaries are discussed later in this guide.

8. Use "transfer purchases from device” option in iTunes.

9. Sometimes, the customer will provide you access to the application via TestFlight (https://developer.apple.com/testflight/) where you can directly log in to the account and download the IPA file.

If all of the above methods fail, which is unlikely, ask the customer for the .IPA file. Always make sure to get the mobile provision certificate along with the application binary.

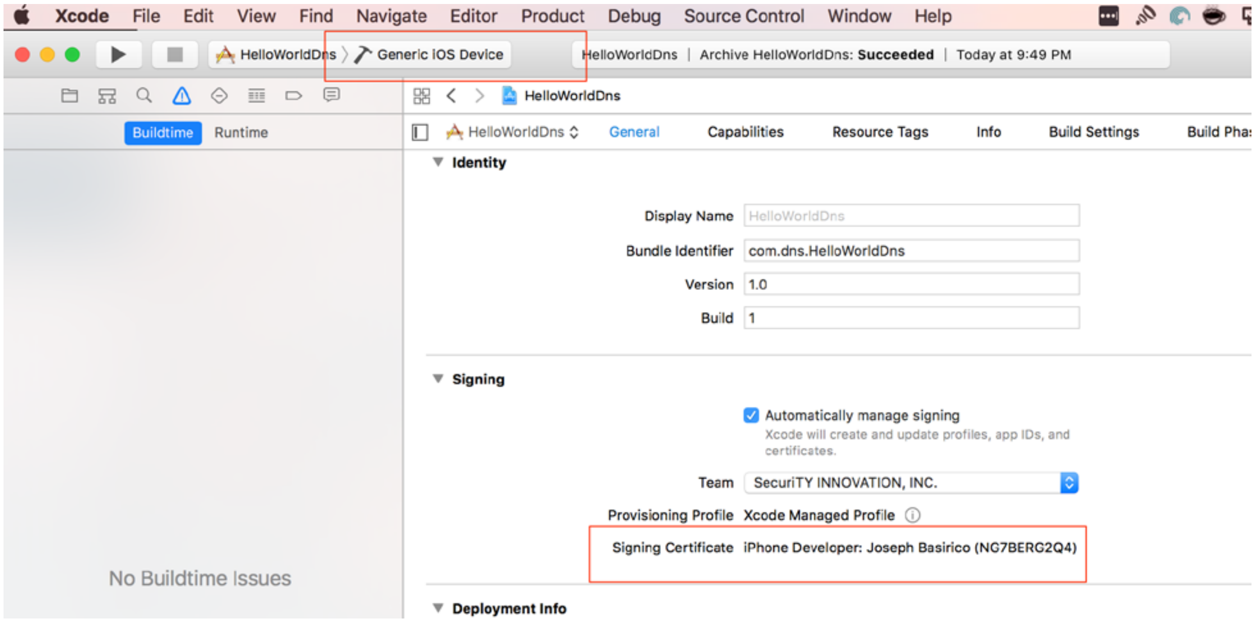
3. Generating iOS Binary (.IPA file) from Xcode Source Code:

Testing an iOS application requires access to the IPA file. Below are two ways of generating IPA files:

3.1 Method I – With A Valid Paid Developer Account.

Make sure iOS device is registered to the Developer account using the steps mentioned here:https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#//apple_ref/doc/uid/TP40012582-CH30-SW10

1. In Xcode log in to the correct developer account. Set the target device to “Generic iOS Device”.



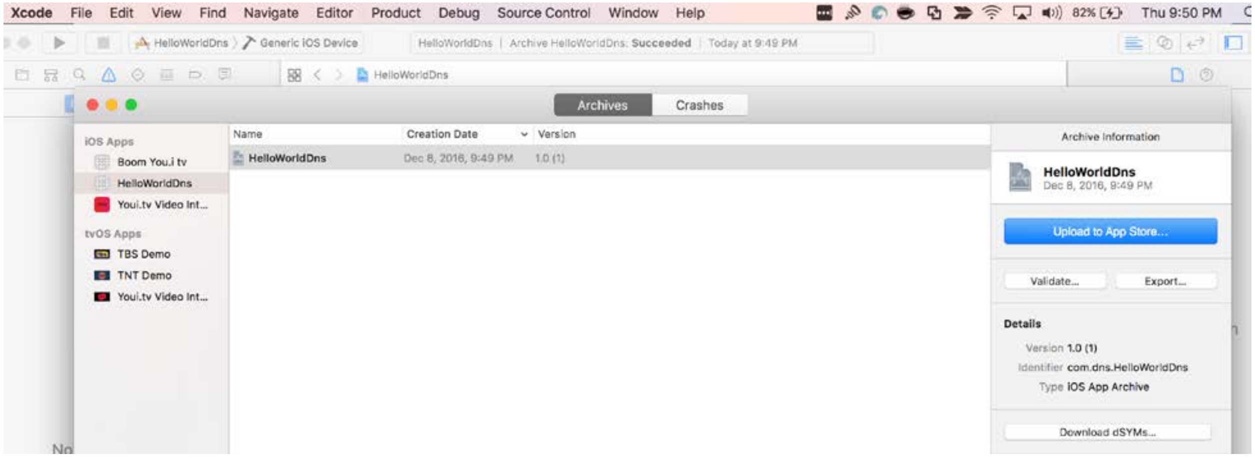
HelloWorldDns | Archive HelloWorldDns: **Succeeded** | Today at 9:49 PM

Display Name: HelloWorldDns
Bundle Identifier: com.dns.HelloWorldDns
Version: 1.0
Build: 1

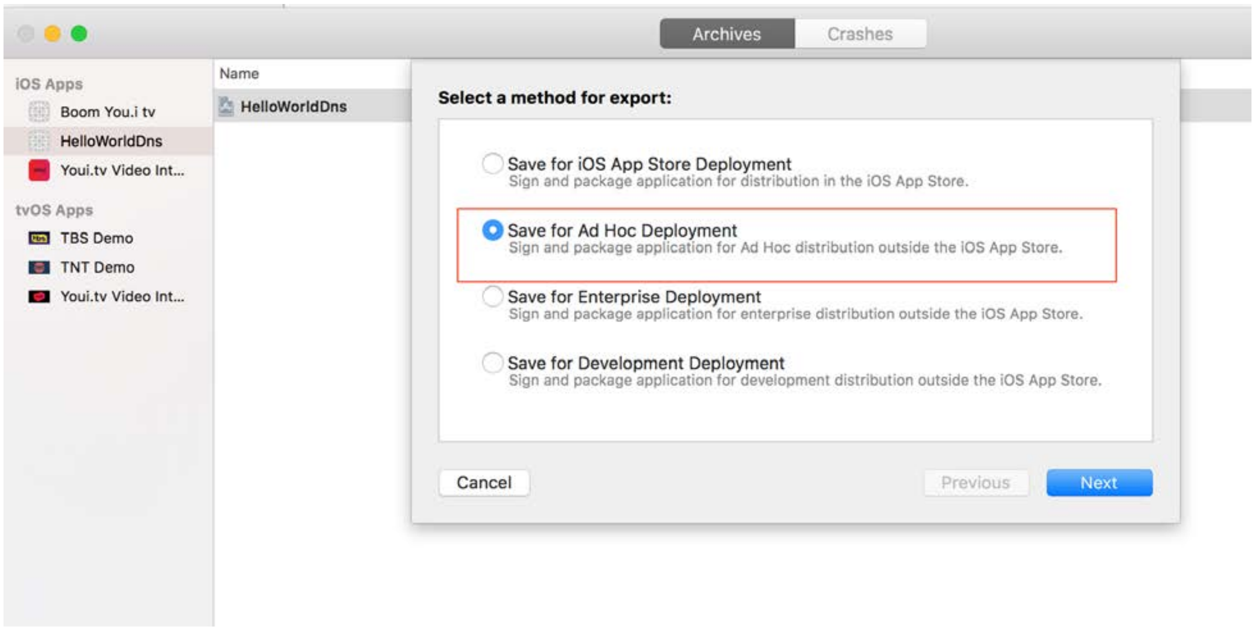
☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: SecuriTY INNOVATION, INC.
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iPhone Developer: Joseph Basirico (NG7BERG2Q4)

2. Connect the iOS device to a laptop.
3. Go to the Product menu at the top and select Archive. This will archive the current build and provide a list of Archives in the Organizer.



4. Go to Window -> Organizer. Press Export, and select the ad-hoc distribution.



Select a method for export:

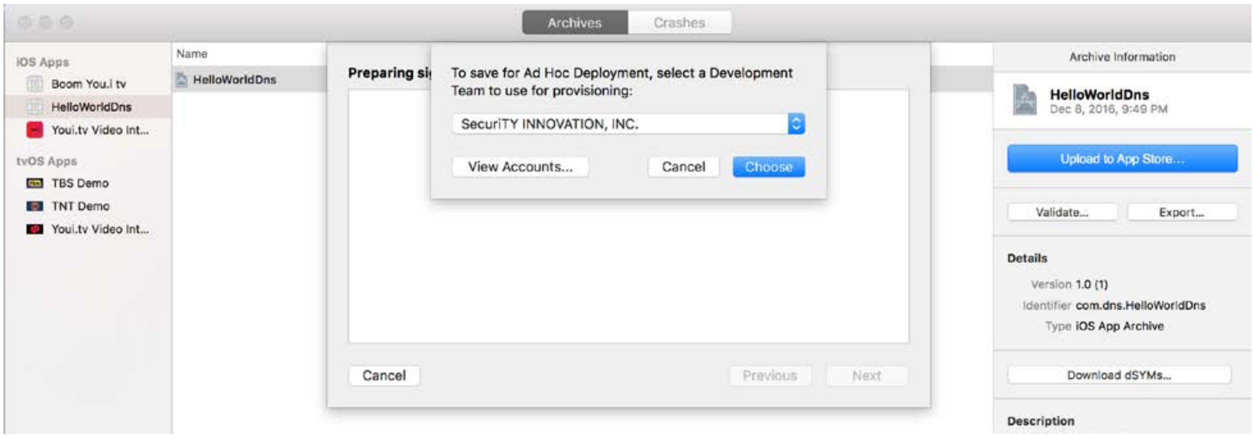
○ Save for iOS App Store Deployment
Sign and package application for distribution in the iOS App Store.

● Save for Ad Hoc Deployment
Sign and package application for Ad Hoc distribution outside the iOS App Store.

○ Save for Enterprise Deployment
Sign and package application for enterprise distribution outside the iOS App Store.

○ Save for Development Deployment
Sign and package application for development distribution outside the iOS App Store.

5. Select the proper development team for provisioning.



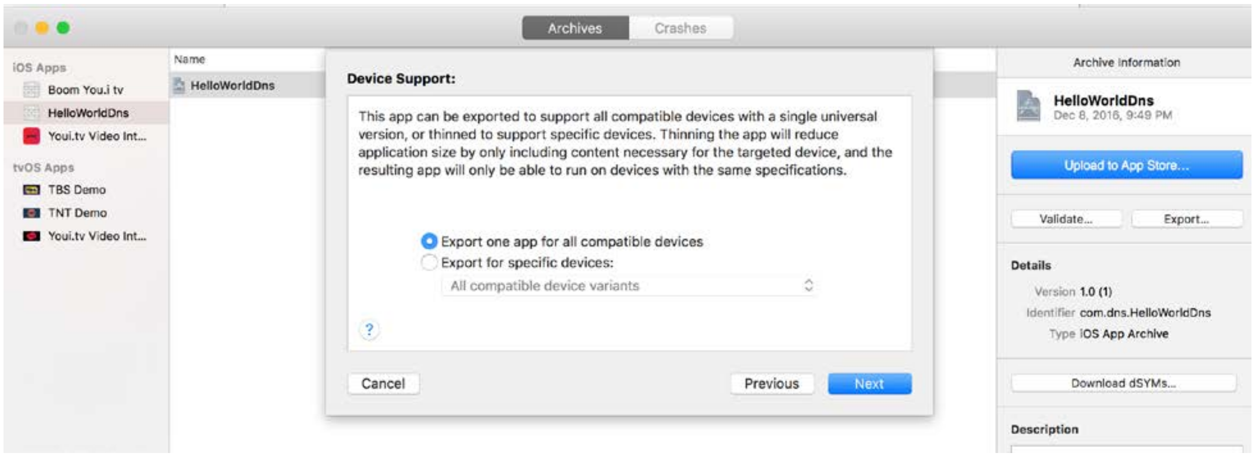
To save for Ad Hoc Deployment, select a Development Team to use for provisioning:

SecuriTY INNOVATION, INC.

Details

Version 1.0 (1)
Identifier com.dns.HelloWorldDns
Type iOS App Archive

6. Select the most suitable device support option (often this is the default option).



Device Support:

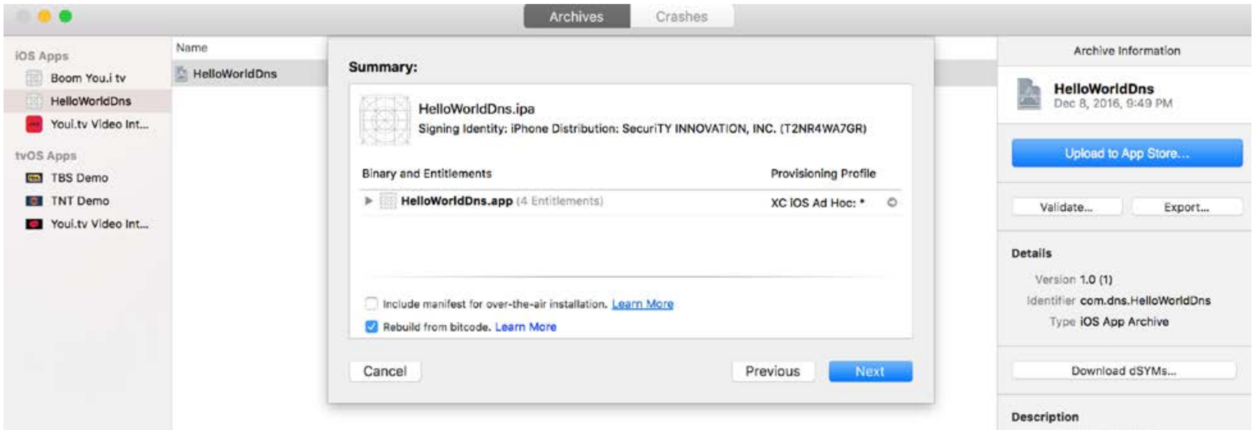
This app can be exported to support all compatible devices with a single universal version, or thinned to support specific devices. Thinning the app will reduce application size by only including content necessary for the targeted device, and the resulting app will only be able to run on devices with the same specifications.

● Export one app for all compatible devices
○ Export for specific devices:
All compatible device variants

Details

Version 1.0 (1)
Identifier com.dns.HelloWorldDns
Type iOS App Archive

7. Click Next on the Summary screen



Summary:

HelloWorldDns.ipa
Signing Identity: iPhone Distribution: SecuriTY INNOVATION, INC. (T2NR4WA7GR)

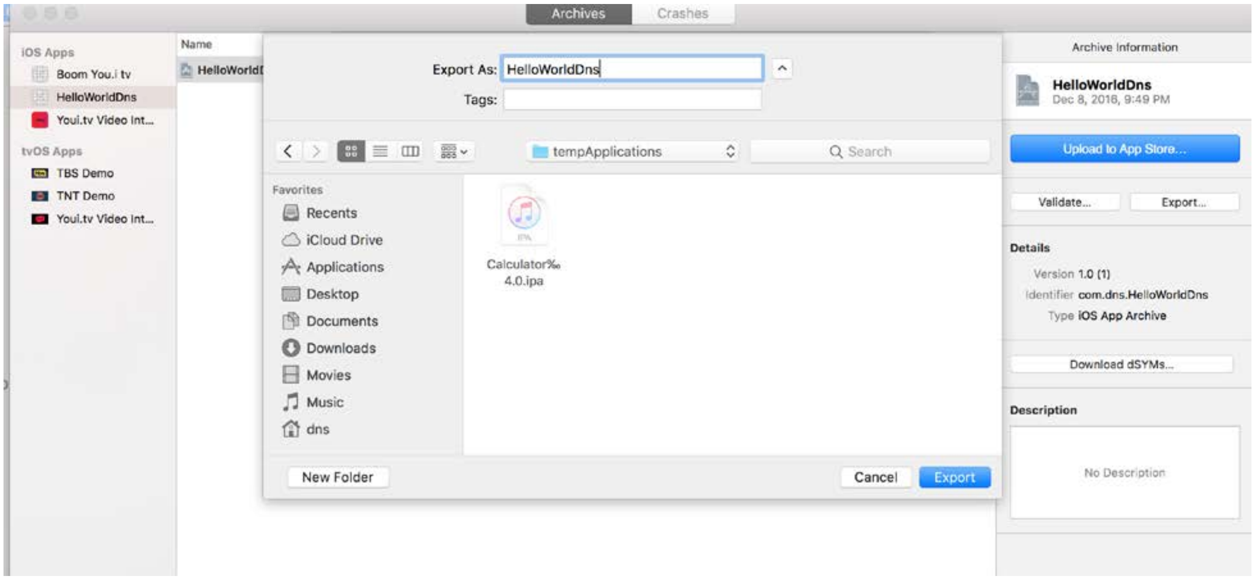
Binary and Entitlements	Provisioning Profile
HelloWorldDns.app (4 Entitlements)	XC iOS Ad Hoc: *

☐ Include manifest for over-the-air installation. Learn More
☑ Rebuild from bitcode. Learn More

Archive Information
HelloWorldDns
Dec 8, 2016, 9:49 PM

Details
Version 1.0 (1)
Identifier com.dns.HelloWorldDns
Type iOS App Archive

8. Save the .IPA file at any known location on the laptop for later use by the security testing team.



Export As: HelloWorldDns
Tags:

tempApplications

Calculator‰ 4.0.ipa

Archive Information
HelloWorldDns
Dec 8, 2016, 9:49 PM

Details
Version 1.0 (1)
Identifier com.dns.HelloWorldDns
Type iOS App Archive

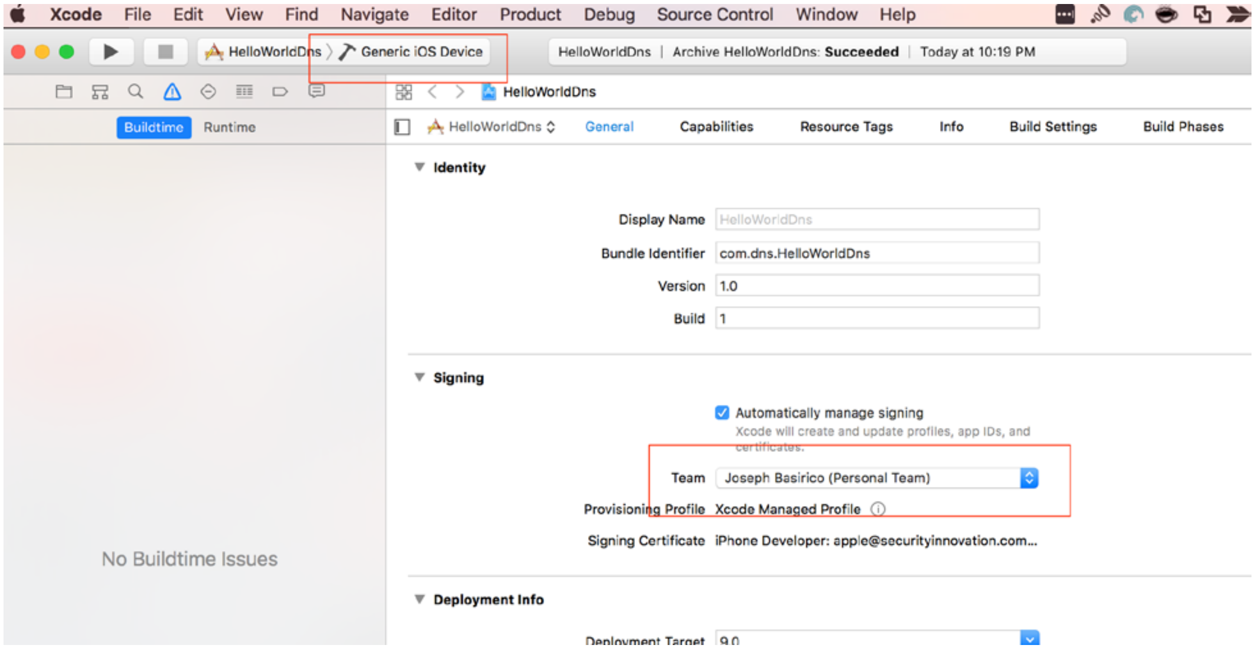
Description
No Description

3.2 Method II - Without a Valid Paid Developer Account

The steps below can be followed to generate an .IPA file when there is not a valid developer account and a personal team certificate is being used.

Make sure the iOS device is registered to the Developer account using the steps mentioned here: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#//apple_ref/doc/uid/TP40012582-CH30-SW10

1. In Xcode log in to the correct account. Set the target device to “Generic iOS Device”.


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
HelloWorldDns 〉 Generic iOS Device
Buildtime Runtime
HelloWorldDns
General Capabilities Resource Tags Info Build Settings Build Phases

HelloWorldDns | Archive HelloWorldDns: **Succeeded** | Today at 10:19 PM

No Buildtime Issues

Identity

Display Name: HelloWorldDns
Bundle Identifier: com.dns.HelloWorldDns
Version: 1.0
Build: 1

Signing

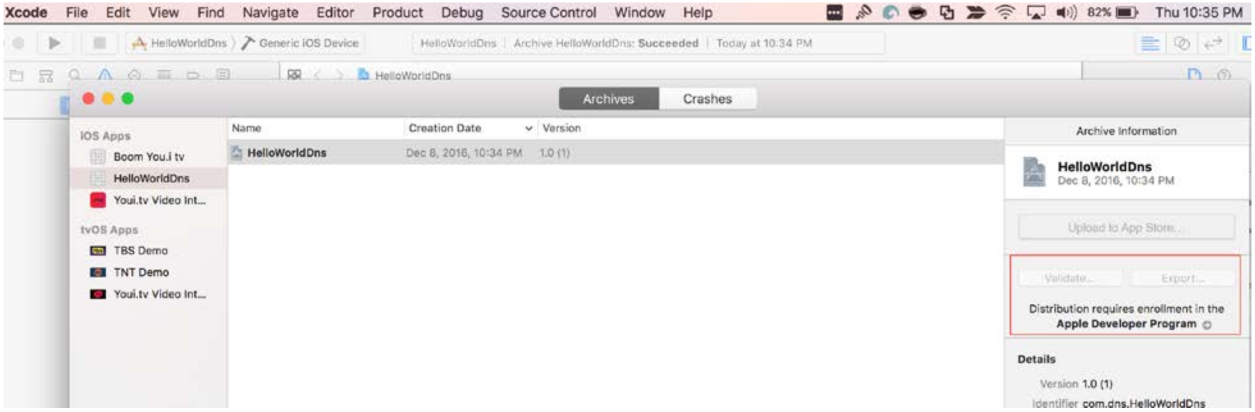
☑ Automatically manage signing
Xcode will create and update profiles, app IDs, and certificates.

Team: Joseph Basirico (Personal Team)
Provisioning Profile: Xcode Managed Profile
Signing Certificate: iPhone Developer: apple@securityinnovation.com...

Deployment Info

Deployment Target: 9.0

2. Connect the iOS device to a laptop.
3. Go to the Product menu at the top and select Archive. This will archive the current build and provide a list of Archives in the Organizer.
4. Go to Window -> Organizer. Note that Export will not work since there is no developer account.



HelloWorldDns | Archive HelloWorldDns: **Succeeded** | Today at 10:34 PM

Name	Creation Date	Version
HelloWorldDns	Dec 8, 2016, 10:34 PM	1.0 (1)

Archive Information

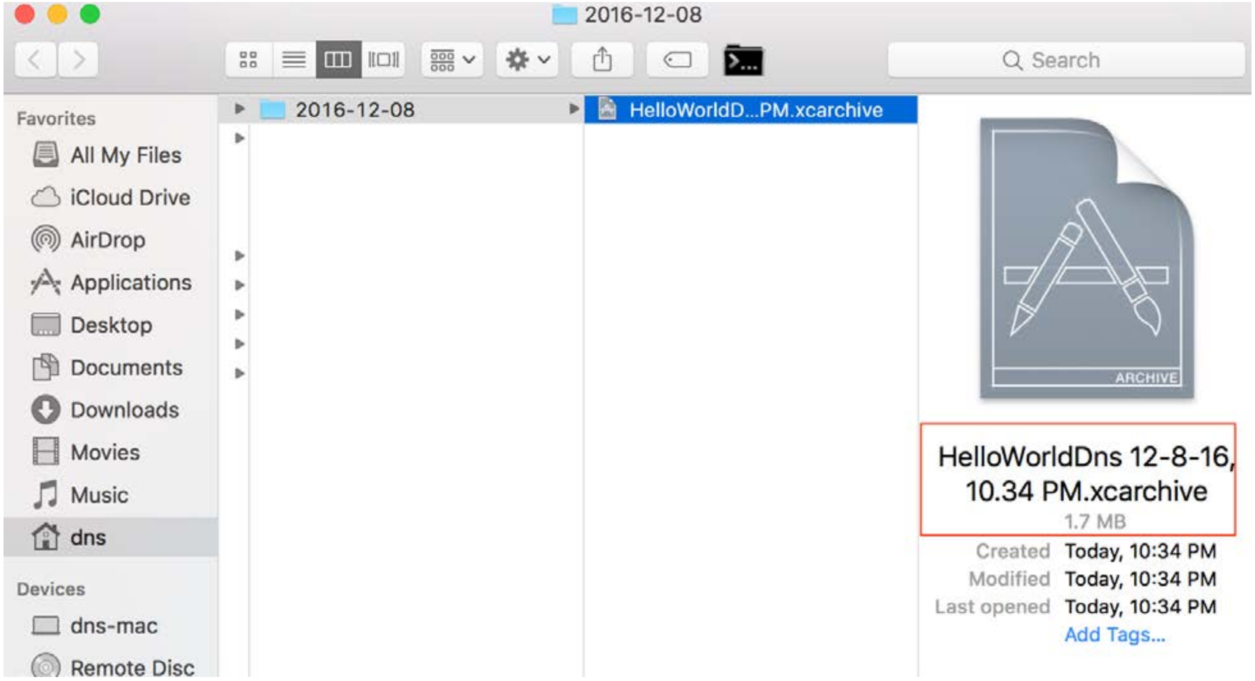
HelloWorldDns
Dec 8, 2016, 10:34 PM

Distribution requires enrollment in the **Apple Developer Program**

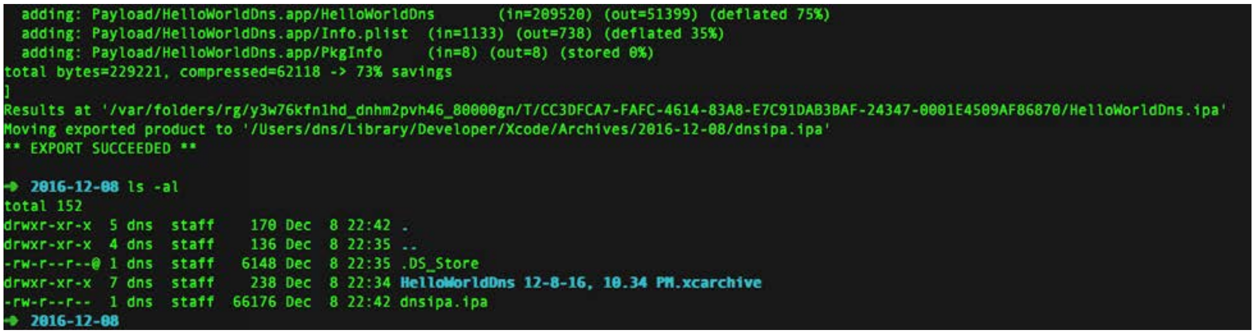
Details

Version 1.0 (1)
Identifier com.dns.HelloWorldDns

5. Right click the archive name and select “Show in Finder.”



6. Open terminal at that location and enter the below command:
    - *xcodebuild -exportArchive -exportFormat ipa -archivePath /<path-to-application.xcarchive -exportPath ~/somepath/ipatobegenerated.ipa*

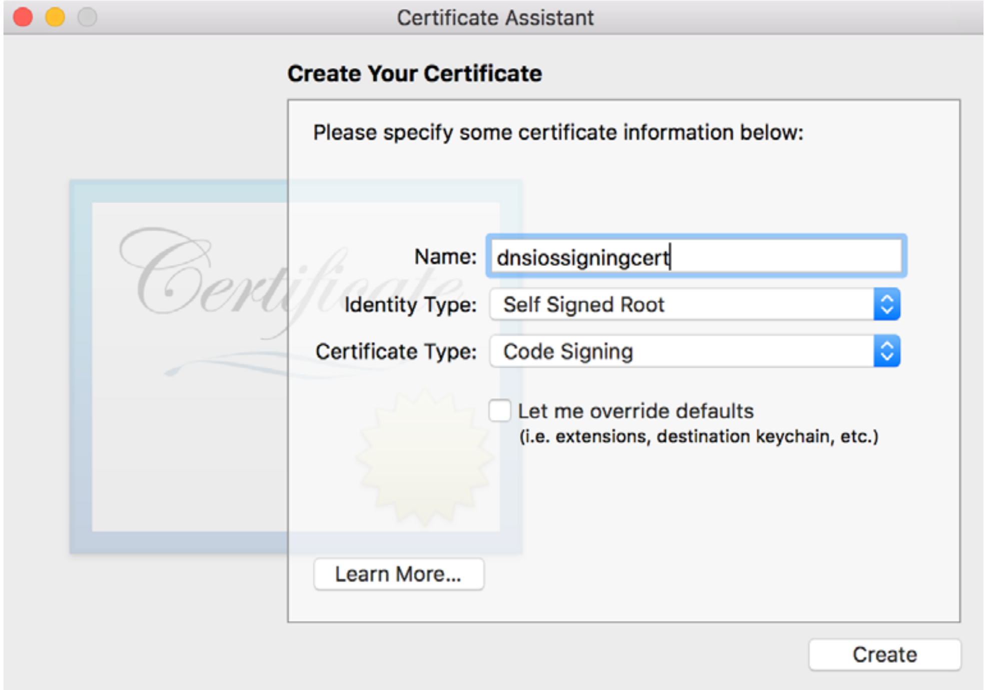


```
  adding: Payload/HelloWorldDns.app/HelloWorldDns       (in=209520) (out=51399) (deflated 75%)
  adding: Payload/HelloWorldDns.app/Info.plist  (in=1133) (out=738) (deflated 35%)
  adding: Payload/HelloWorldDns.app/PkgInfo     (in=8) (out=8) (stored 0%)
total bytes=229221, compressed=62118 -> 73% savings
]
Results at '/var/folders/rg/y3w76kfn1hd_dnhm2pvh46_80000gn/T/CC3DFCA7-FAFC-4614-83A8-E7C91DAB3BAF-24347-0001E4509AF86870/HelloWorldDns.ipa'
Moving exported product to '/Users/dns/Library/Developer/Xcode/Archives/2016-12-08/dnsipa.ipa'
** EXPORT SUCCEEDED **

➜ 2016-12-08 ls -al
total 152
drwxr-xr-x  5 dns  staff    170 Dec  8 22:42 .
drwxr-xr-x  4 dns  staff    136 Dec  8 22:35 ..
-rw-r--r--@ 1 dns  staff   6148 Dec  8 22:35 .DS_Store
drwxr-xr-x  7 dns  staff    238 Dec  8 22:34 HelloWorldDns 12-8-16, 10.34 PM.xcarchive
-rw-r--r--  1 dns  staff  66176 Dec  8 22:42 dnsipa.ipa
➜ 2016-12-08
```

This generated ipa file can be used for binary analysis but, to install it on a real device, the application will need to be re-signed.

This can be done using the steps below or by using tools like Cydia Impactor as explained in “section 4”:

1. Check the current signature used to sign the application using the below command:
    - *codesign -v -d HelloWorldDns.app*
2. Create a self-signed signature using the Certificate Assistant in Keychain Access.
    - Choose Keychain Access > Certificate Assistant > Create a Certificate.
    - Enter a name for the certificate.
    - Set Identity Type as “Self Signing Root” and the Certificate Type as “Code Sign”.



Certificate Assistant

Create Your Certificate

Please specify some certificate information below:

Name: dnsiossigningcert

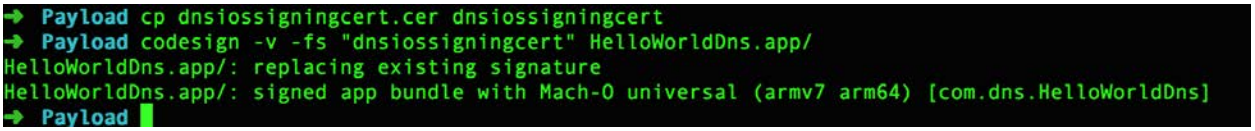
Identity Type: Self Signed Root

Certificate Type: Code Signing

☐ Let me override defaults (i.e. extensions, destination keychain, etc.)

- Click on Create.
- In Keychain Access, search for the created certificate and copy it to a known location on the laptop.

3. Modify the application signature using codesign.
   - *codesign -v -fs "<abovecreatedcertificatename>" HelloWorldDns.app/*



```
→ Payload cp dnsiossigningcert.cer dnsiossigningcert
→ Payload codesign -v -fs "dnsiossigningcert" HelloWorldDns.app/
HelloWorldDns.app/: replacing existing signature
HelloWorldDns.app/: signed app bundle with Mach-O universal (armv7 arm64) [com.dns.HelloWorldDns]
→ Payload
```

4. Resign the application using ldid on the binary inside the .app folder
   - *ldid -s <appname>*
5. Choose one of the following steps:
   - Create a new folder named Payload. Move .app folder inside it and compress the Payload folder as Payload.zip. Rename Payload.zip to <applicationname>.ipa. The application can then be installed using the steps mentioned in “Module 4” (Using installipa utility)”.

   OR
   - Copy the .app file to the /Applications directory on the device. The application can then be installed using the steps mentioned in “Module 4” (Using .app).

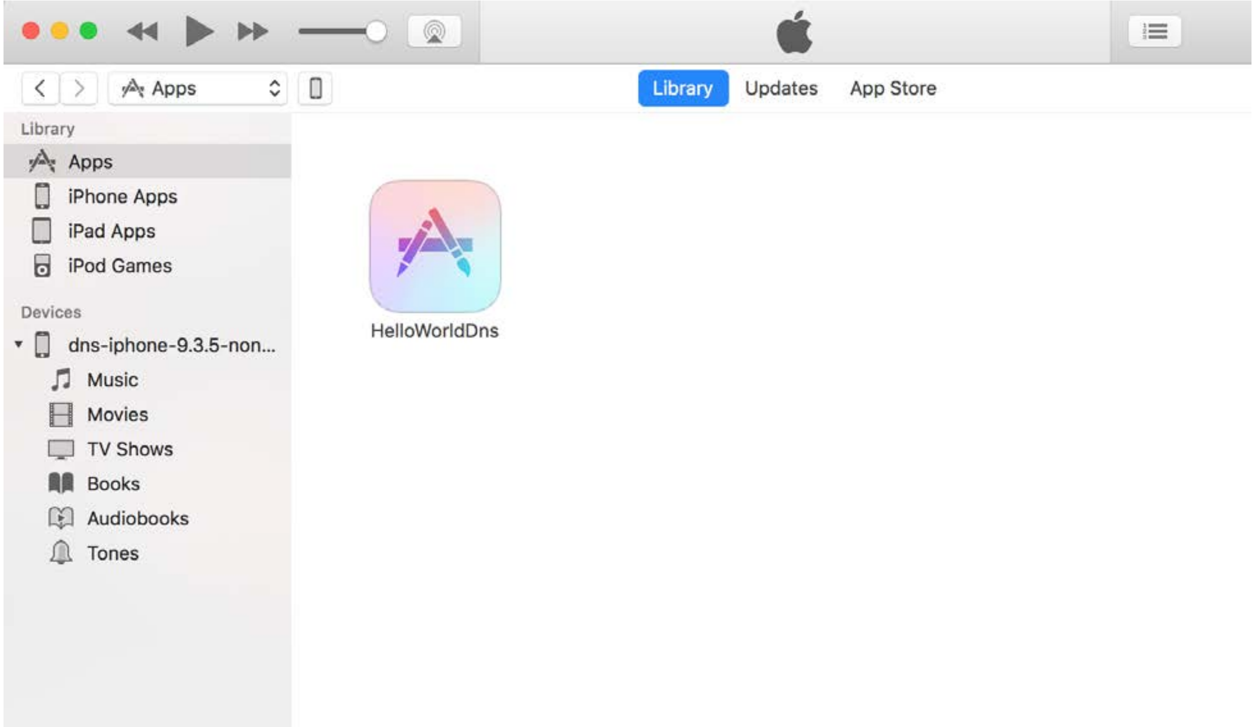
4. Installing iOS Binaries on Physical Devices

If the client provides iOS binary, below are some of the methods to install them on a physical device.

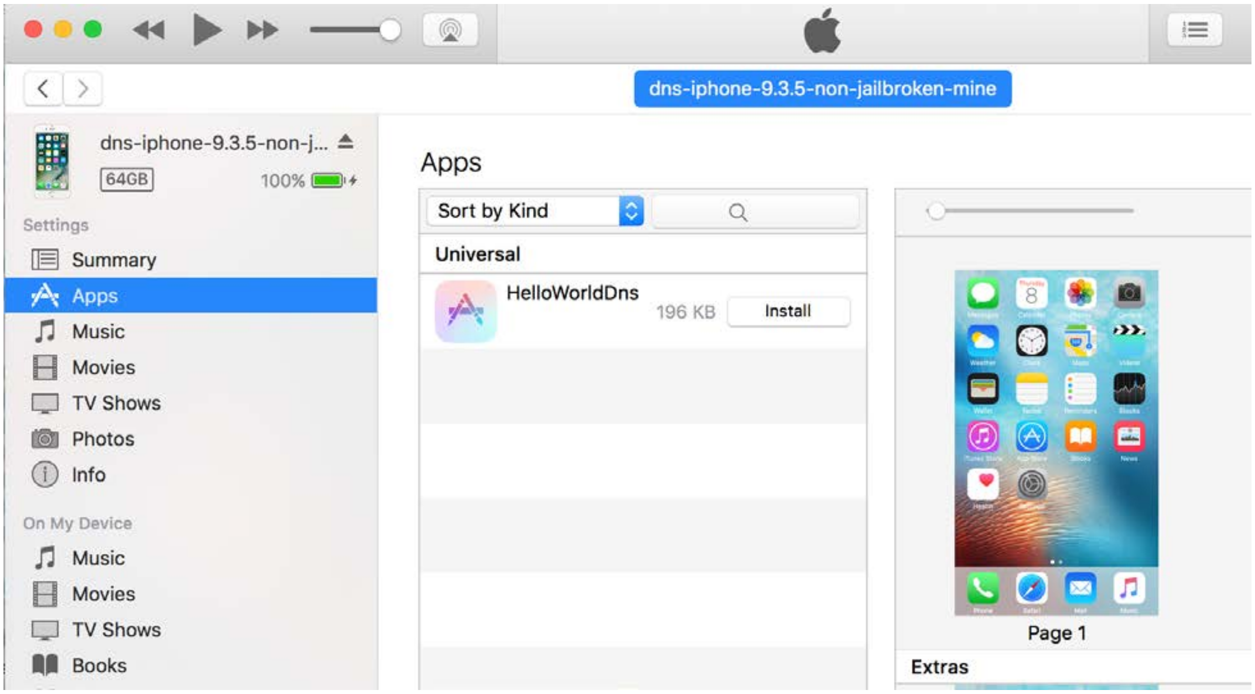
4.1 Method I - Using iTunes

The steps below can be used to install the application on a device once access is granted to the .IPA or .app file. Depending upon the circumstances, there be a need for a separate mobile provision file which is the provisional certificate for ad hoc distribution of the binary file.

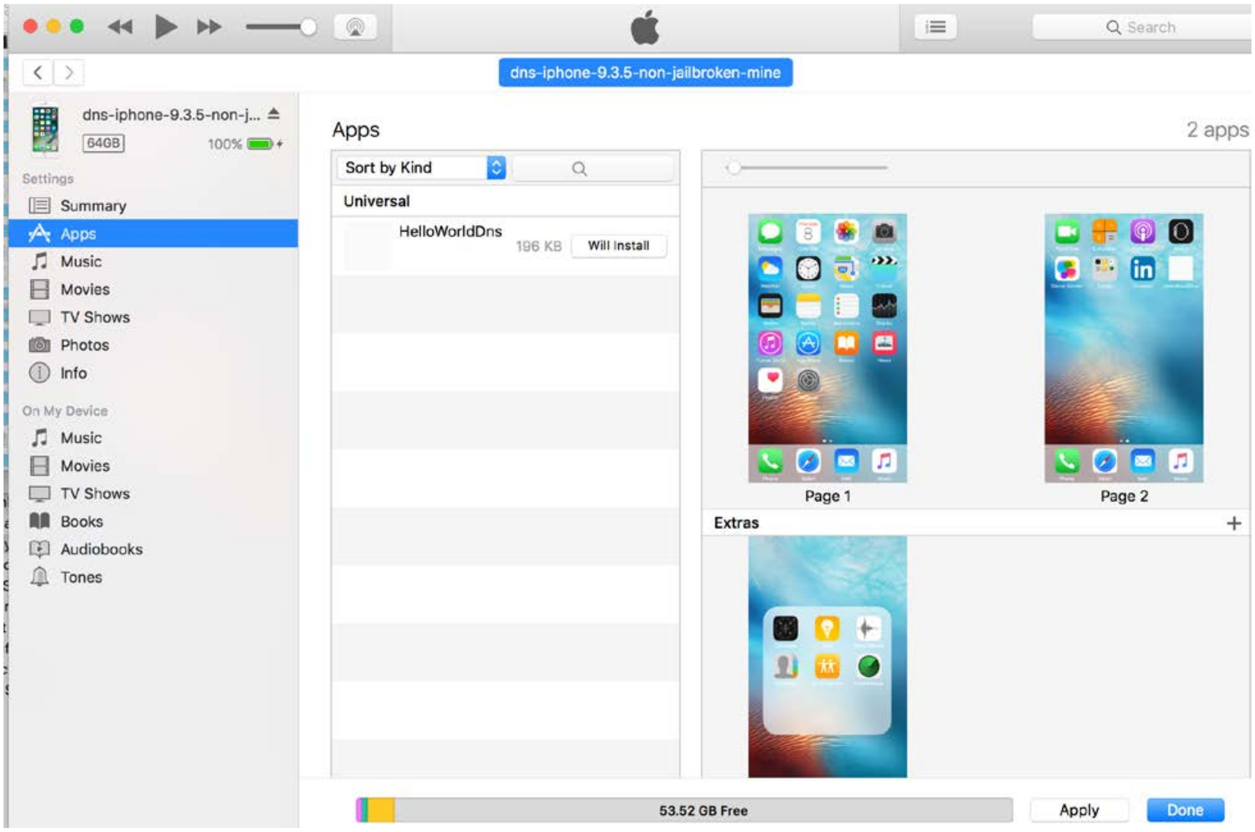
1. Launch iTunes
2. Drag drop the .app/.ipa file and the provisional certificate into the iTunes “Apps” tab in Library (Not in device Apps). If the “Apps” tab is missing, follow the steps below.
   a. On Mac: iTunes -> Music Dropdown -> Click Edit Menu -> check if Apps is selected or not. If not, click on Music Dropdown -> Click Edit Menu -> Enable Apps.
   b. On Windows: iTunes -> Edit -> check if Apps is selected or not.



3. Connect the iOS device to the laptop and the iOS device name will appear in the sidebar. Click on it, and select “Apps” from iTunes device menu. (The Apps on the device)



4. Select the iOS application to be installed, click Install, Apply for the application to Sync, and Install on the device.


Search
dns-iphone-9.3.5-non-jailbroken-mine
dns-iphone-9.3.5-non-j...
64GB
100%
Settings
Summary
Apps
Music
Movies
TV Shows
Photos
Info
On My Device
Music
Movies
TV Shows
Books
Audiobooks
Tones
Apps
2 apps
Sort by Kind
Universal
HelloWorldDns
196 KB
Will Install
Page 1
Page 2
Extras
53.52 GB Free
Apply
Done

4.2 Method II - Using Cydia Impactor

On a non-jailbroken device, Cydia Impactor can be used to install self-signed iOS binaries and install them to the device.

1. Download the tool from cydiaimpactor.com
2. Download the .deb or .IPA file
3. Install the latest version of iTunes
4. Connect the iOS device to the laptop
5. Launch Impactor and drag drop the iOS binary to the dropdown menu with the device name
6. Log in using an Apple developer account. Select the Agent/iOS Distribution certificate from the list.
    - A free account can also be used, but the certificate will expire after 7 days. Also, note that an existing iOS developer certificate will be revoked to make way for this new device certificate
7. Click OK on the Apple Developer Warning
8. In Settings > General > Profile & Device Management find the profile used to sign the application and *Trust* it

4.3 Method III - Using iOS App Signer

On a non-jailbroken device, iOS App Signer can be used to install self-signed iOS binaries and install them to the device. The binary can be a .IPA file or a .deb file.

This is a relatively straight forward application. The steps can be found here:

- http://dantheman827.github.io/ios-app-signer/

4.4 Method IV - Installing .app file

On a jailbroken device:

1. *scp -r HelloWorldApp.app/ root@10.0.1.24:/Applications/*

2. *cd /Applications/HelloWorldApp.app/*
3. *chmod +x HelloWorldApp*
4. *uicache*

4.5 Method V - Installing Modified Binary

There are a variety of ways to modify/patch a binary (find more information later in this guide). Due to code signing, these applications won't work as-is on the iOS device. Follow the steps below to make sure that the modified binary works.

1. Download the .app file from the device (Use ipa -> decrypted ipa -> app file if using encrypted binary).
2. Extract the contents of the .app file and look for the application binary in it.
3. Patch the binary file using any technique.
4. Create a self-signed signature using the Certificate Assistant in Keychain Access.
    - Choose Keychain Access > Certificate Assistant > Create a Certificate.
    - Enter a name for the certificate.
    - Set Identity Type as "Self Signing Root" and the Certificate Type as "Code Sign."
    - Click on Create.
    - In Keychain Access, search for the created certificate and copy it to a known location on the laptop.
5. Modify the application signature using codesign.
    - *codesign -v -fs "<abovecreatedcertificatename>" HelloWorldDns.app/*
6. Resign the application using ldid on the binary inside the .app folder
    - *ldid -s <appname>*
7. Copy the modified .app file not yet converted into a valid one to the device using the below command:
    - *scp -r HelloWorldApp.app/ root@10.0.1.24:/Applications/*
8. Navigate to the directory and run the below commands to clear the iOS device cache.
    - *cd /Applications/HelloWorldApp.app/*
    - *chmod +x HelloWorldApp*
    - *uicache*
9. The application now appears on the iOS device and can be used without any issues.

4.6 Method VI - Using Installipa Utility

On a jailbroken device, Installipa Utility can be used to install self-signed iOS binaries as user “mobile” or “root.” Installipa can be downloaded from Cydia.

Copy the .IPA file on the device using ssh. With AppSync installed on the device using Cydia, ssh into the device as user “mobile” and use the below command to install application as user “mobile”:

- *installipa HelloWorldApp.ipa*

4.7 Method VII - Using iPhone Configuration Utility

To install the binary .IPA file, use the iPhone configuration utility (now renamed as Apple Configurator and downloadable from Mac AppStore) from here:

- https://www.theiphonewiki.com/wiki/IPhone_Configuration_Utility

AppSync must be installed on an iPhone via Cydia for majority of the .IPA files to be installed on the device. Appsync Unified is software that allows the installation of fake signed IPA files on the device.

If have AppSync is not installed, add http://cydia.angelxwind.net as a repo & look for AppSync Unified.

Once the tool is installed on a MAC and iPhone, launch the tool on the MAC and add the IPA file to install to Apple Configurator.

4.8 Method VIII - Using iFunBox

iFunBox can also be used to install an .IPA file. This requires a jailbroken device. Follow the steps mentioned here:

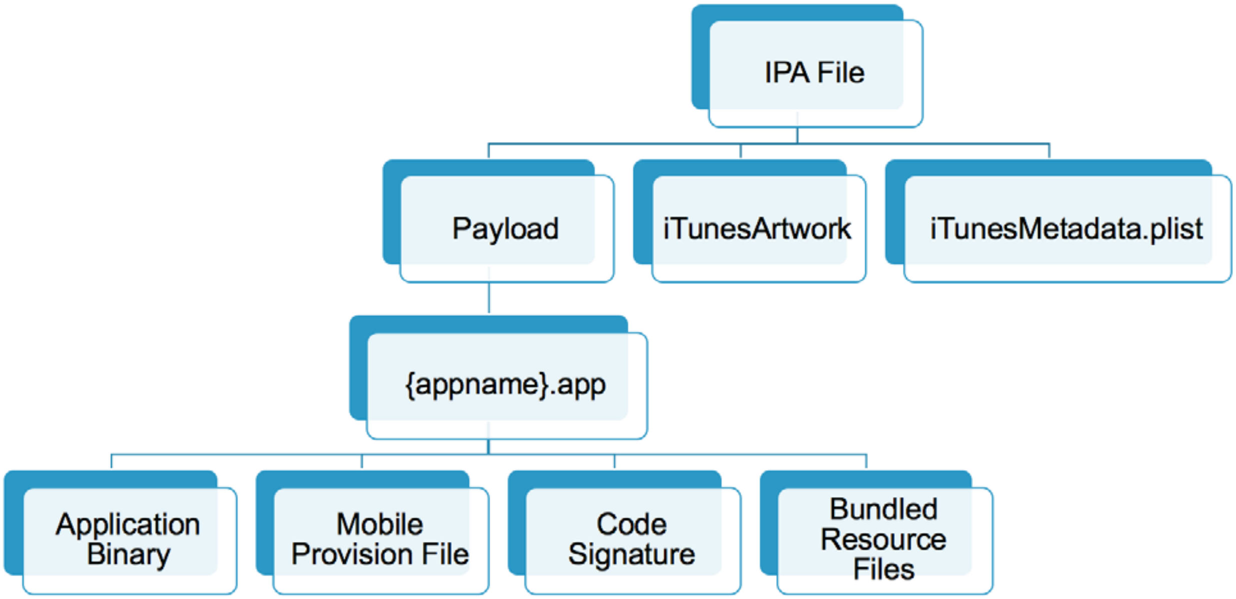
- http://iosgeeksblog.blogspot.in/2013/01/how-to-install-ipa-files-directly-on-iphone-with-ifunbox.html

5. iOS Binary Package Primer

iOS applications have a binary file format known as IPA which are basically ZIP archives. The .IPA files include a binary for the ARM architecture and can only be installed on an iOS device. There are no known ways to install the .IPA file on an iOS simulator.

The .IPA files can be uncompressed using an unzip utility.

5.1 Understanding the iOS Binary Package Structure



iTunesArtwork: A 512 x 512 pixel PNG image. It contains the applications icon that shows up on iTunes and the App Store app on the iOS device.

iTunesMetadata.plist: A property list xml file that contains developer information like developer name, ID, copyright information, application name, release information, etc.

Payload: The folder that contains the application data.

Application Binary: The executable file containing the application's code. The name of this file is always the same as the actual application name without the .app extension. During the pentest, the complete binary analysis is performed on this application binary.

Mobile Provision File: By default, applications on iOS can only be installed via the AppStore. In special cases, when the application is to be beta tested, mobile provision certificates are generated and used. This is the file which is included in the binary when ad hoc distribution of the file is to be done. A provision profile is a document that lists the digital certificates, the devices, and the IDs of the applications allowed to operate on a device. This is specifically used for beta stages (usually named ***Ad_Hoc_Distribution_Profile.mobileprovision***).
For more information, see: http://www.wikihow.com/Install-Ad-hoc-iPhone-OS-Apps

Code Signature: The purpose of the code signature is to make sure that the integrity of the .app file is maintained from when the application was released. Any kind of editing or deletion (even images having the same name) will invalidate the signature. Any changes that are made to the .app file require that the whole package be re-signed.

Bundled Resource Files: Images, Videos, Sounds, HTML, Property list files, etc. which are required by the application to be installed on the mobile device.

For further information on iOS application package structure, go here

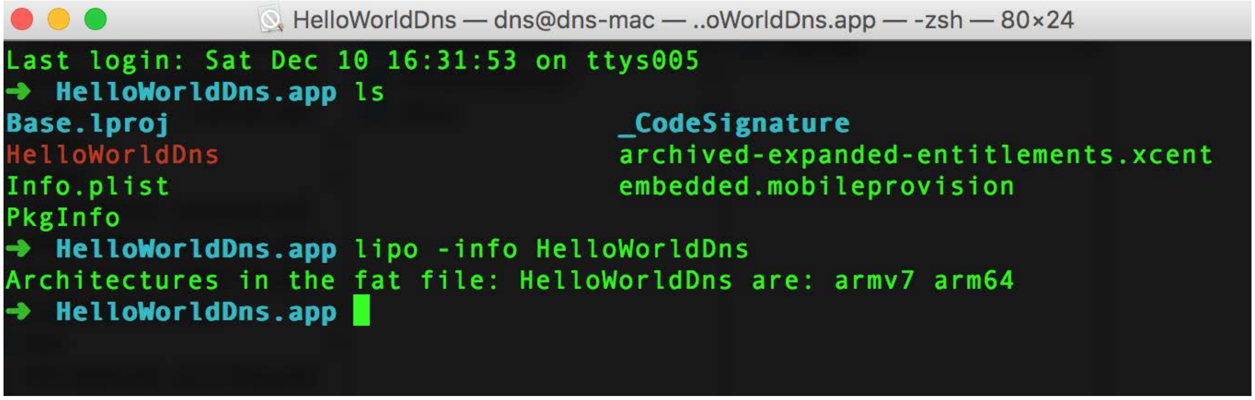
- https://developer.apple.com/library/content/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_ref/doc/uid/10000123i-CH101-SW1.

5.2 Understanding the Supported Architectures for the Provided Application

Lipo is a Mac utility that can be used to view all the architectures on which the provided application can be installed. Lipo might need to be installed on aMAC when running it for the first time. The syntax for using lipo is shown below:

lipo -info <applicationbinary>

Remember to run *lipo* on the binary *inside* the .app folder, not on the IPA file.

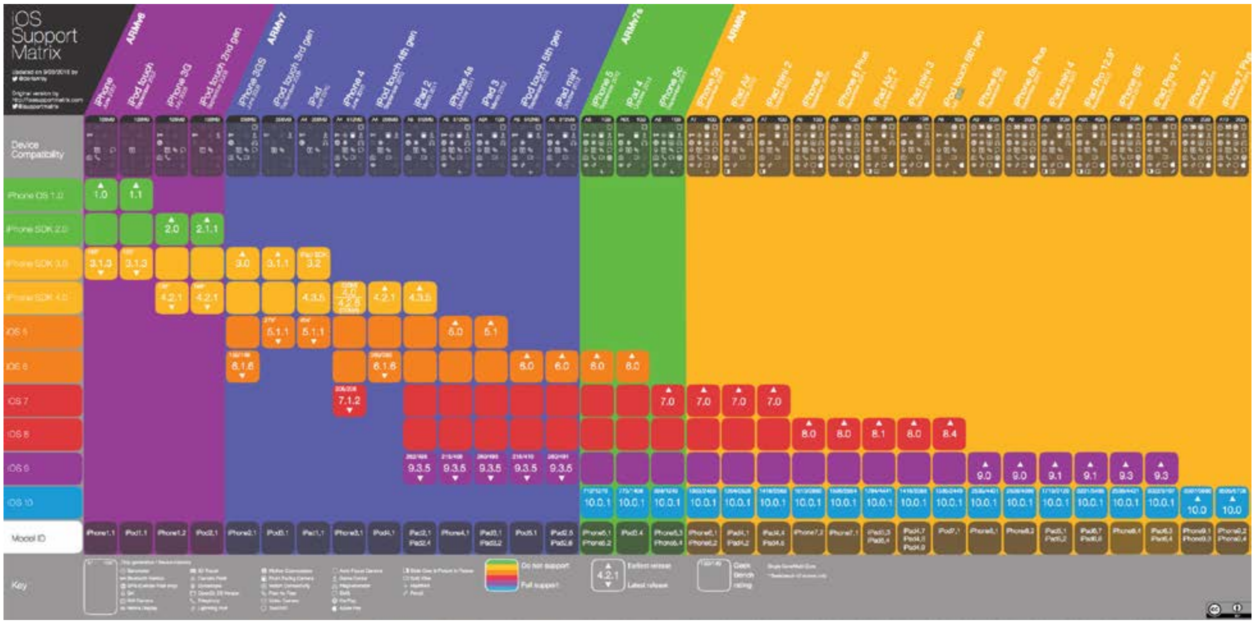


HelloWorldDns — dns@dns-mac — ..oWorldDns.app — -zsh — 80×24

```
Last login: Sat Dec 10 16:31:53 on ttys005
➜  HelloWorldDns.app ls
Base.lproj                              _CodeSignature
HelloWorldDns                           archived-expanded-entitlements.xcent
Info.plist                              embedded.mobileprovision
PkgInfo
➜  HelloWorldDns.app lipo -info HelloWorldDns
Architectures in the fat file: HelloWorldDns are: armv7 arm64
➜  HelloWorldDns.app
```

5.3 Understanding the Architecture Available on the Test Devices

The image below illustrates the iOS support matrix. A more detailed version can be found here: http://iossupportmatrix.com/



Source image: http://dorianroy.com/blog/wp-content/uploads/2016/09/iOS_Support_Matrix_v4.2.pdf

A text only version of this support matrix can be found here: https://www.innerfence.com/howto/apple-ios-devices-dates-versions-instruction-sets

iOS device identifiers can be found here: https://www.theiphonewiki.com/wiki/Models

Maximum iOS version for every iOS device can be found here: http://www.everyi.com/by-capability/maximum-supported-ios-version-for-ipod-iphone-ipad.html

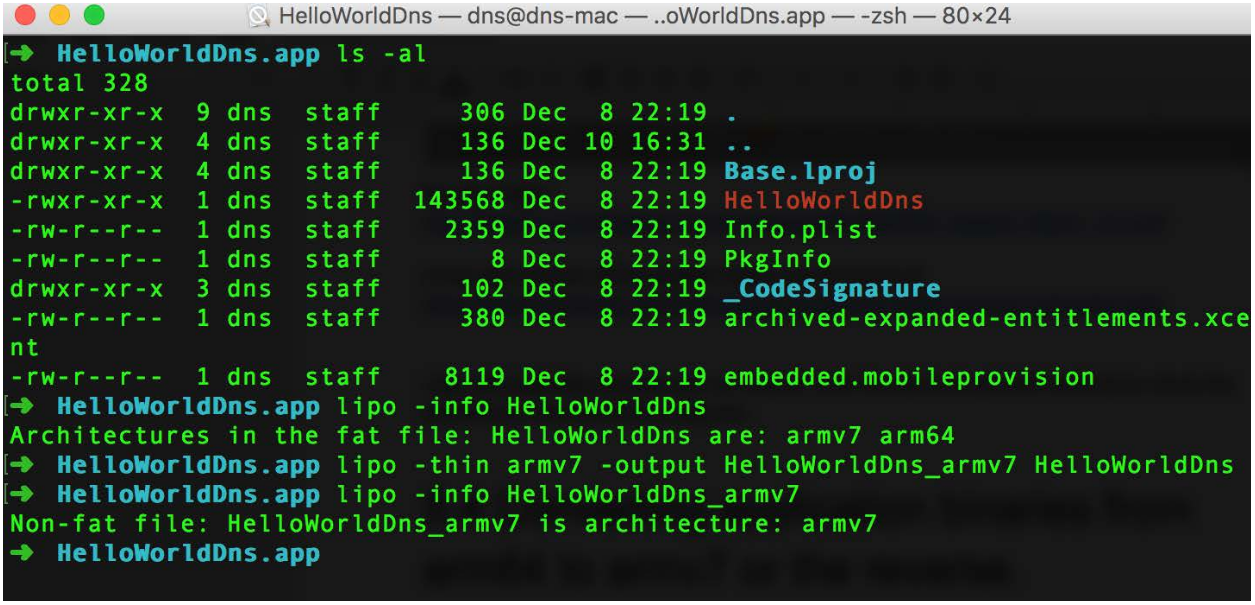
Users with jailbroken access can run *uname -a* on the device local SSH terminal to check the architecture that the device supports.

5.4 Converting Application Binaries from FAT Binary to Specific Architecture Binary

Many of the available tools do not support AArch64 binaries. To analyze these 64bit binaries, strip out a particular architecture from the fat binary.

Lipo can be used to strip out a particular architecture (*64 bit - arm64*) from the provided binary. Use the below command:

lipo -thin armv7 -output <newarmv7binaryname> <binaryname>


HelloWorldDns — dns@dns-mac — ..oWorldDns.app — -zsh — 80×24

```
[➜  HelloWorldDns.app ls -al
total 328
drwxr-xr-x  9 dns  staff     306 Dec  8 22:19 .
drwxr-xr-x  4 dns  staff     136 Dec 10 16:31 ..
drwxr-xr-x  4 dns  staff     136 Dec  8 22:19 Base.lproj
-rwxr-xr-x  1 dns  staff  143568 Dec  8 22:19 HelloWorldDns
-rw-r--r--  1 dns  staff    2359 Dec  8 22:19 Info.plist
-rw-r--r--  1 dns  staff       8 Dec  8 22:19 PkgInfo
drwxr-xr-x  3 dns  staff     102 Dec  8 22:19 _CodeSignature
-rw-r--r--  1 dns  staff     380 Dec  8 22:19 archived-expanded-entitlements.xce
nt
-rw-r--r--  1 dns  staff    8119 Dec  8 22:19 embedded.mobileprovision
[➜  HelloWorldDns.app lipo -info HelloWorldDns
Architectures in the fat file: HelloWorldDns are: armv7 arm64
[➜  HelloWorldDns.app lipo -thin armv7 -output HelloWorldDns_armv7 HelloWorldDns
[➜  HelloWorldDns.app lipo -info HelloWorldDns_armv7
Non-fat file: HelloWorldDns_armv7 is architecture: armv7
➜  HelloWorldDns.app
```

The application can then be repackaged into an .IPA file and installed on the device using steps mentioned in "Module 4 - Method VI" (Using installipa utility).

5.5 Converting Pre-iOS 9 Executables to an iOS 9 Executable

Use this tool to convert pre-iOS 9 executables to an iOS 9 executable:

- https://github.com/Starwarsfan2099/iOS-9-Executsable-Converter

5.6 Converting 32 Bit Applications into 64 Bit Applications in Xcode

This method can be used to run 32bit applications on a 64-bit device even with just a 32-bit binary. With access to the source code, use the steps below:

1. Open the application source code in Xcode
2. Update the project settings to support the latest version of iOS available
3. In Build Settings go to the Architectures section and set Architectures to “Standard architectures (arm64)”
4. Fix all the compiler warnings that have countered using steps mentioned here:http://www.chupamobile.com/blog/2015/01/19/convert-app-64-bit-requirement/ and https://developer.apple.com/library/content/documentation/General/Conceptual/CocoaTouch64BitGuide/ConvertingYourAppto64-Bit/ConvertingYourAppto64-Bit.html
5. Run the updated project code in a 64-bit iOS simulator to ensure that the application works.

6. Compiling Customer-Provided Source Code for Pentesting on Latest iOS Using Xcode

For code assisted penetration tests, the best approach is to get the source code from the customer and set it up to run locally. This gives additional debugging capabilities which could prove to be very helpful in the long run. During the kick-off call with a customer request “*build-ready code package along with all the dependencies.*” This will avoid missing library and other dependency issues that may arise if the development team just copies a folder.

Even with a build-ready source code, there could be issues getting the application up and running.

Assume that a customer has provided the source code for DVIA (Damn Vulnerable iOS application) for a code assisted penetration test.

The below steps assume Xcode 8.x is being used (the latest Xcode available as of the publication of this guide). The target device for testing can be iOS 9.x or higher.

6.1 Download the Source Code

Download the source for DVIA from https://github.com/prateek147/DVIA using this command:

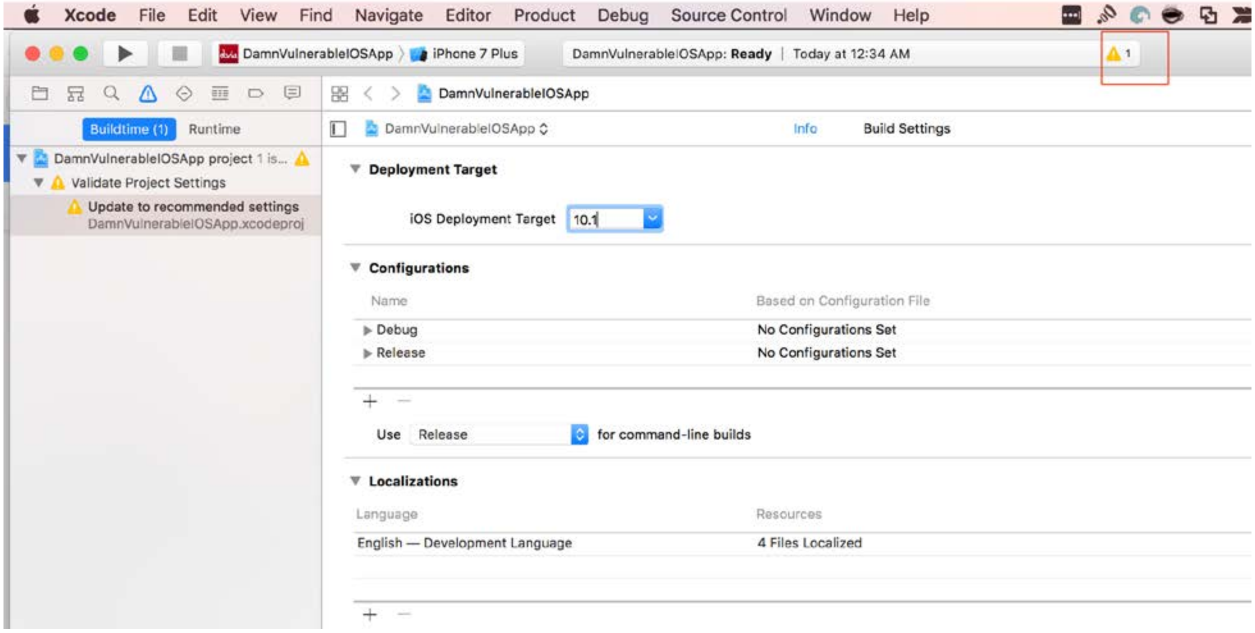
git clone https://github.com/prateek147/DVIA.git

6.2 Launch the Workspace

From the downloaded source code open the. xcworkspace file using Xcode. Allow the “Indexing” process to complete.

6.3 Application Configuration

1. Change the deployment target from iOS 8.2 to the device iOS version and click on the yellow warning sign as shown here:



Deployment Target

iOS Deployment Target 10.1

Configurations

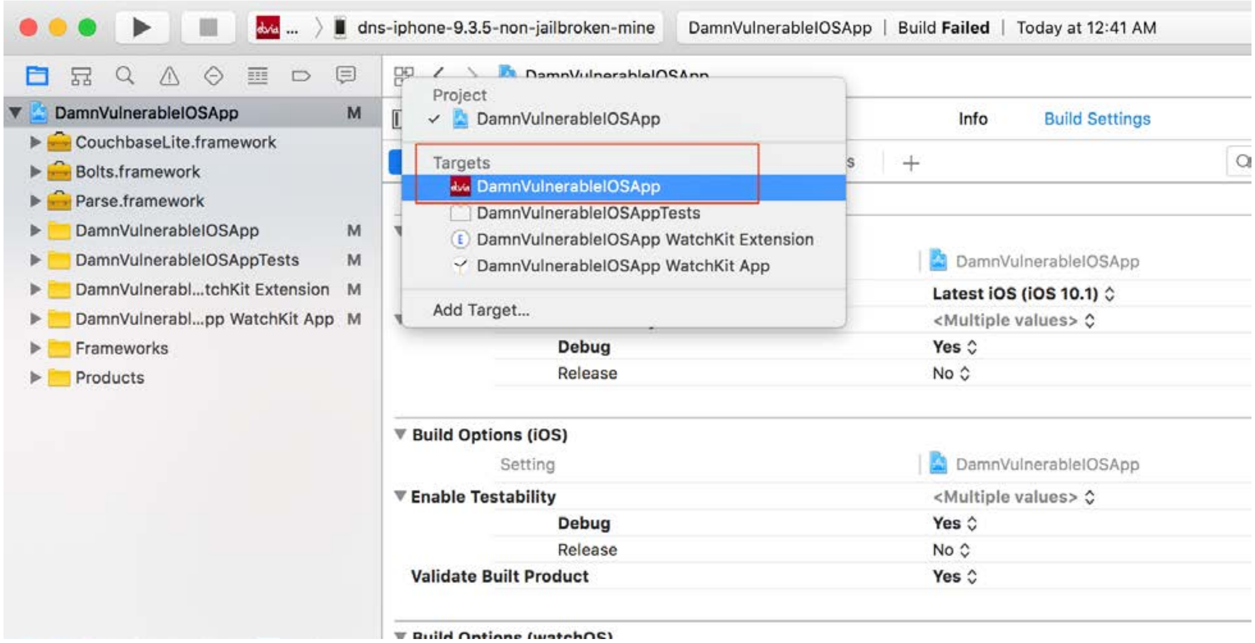
Name	Based on Configuration File
▶ Debug	No Configurations Set
▶ Release	No Configurations Set

Use Release for command-line builds

Localizations

Language	Resources
English — Development Language	4 Files Localized

2. Make sure that the Projects menu has “DamnVulnerableIOSApp” selected and the Target is set to the physical device that is connected to the laptop.

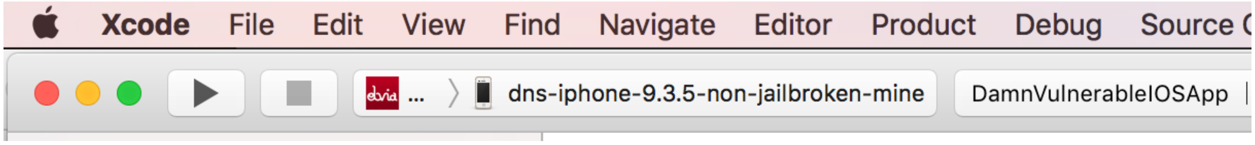


dns-iphone-9.3.5-non-jailbroken-mine
DamnVulnerableIOSApp | Build Failed | Today at 12:41 AM
DamnVulnerableIOSApp M
CouchbaseLite.framework
Bolts.framework
Parse.framework
DamnVulnerableIOSApp M
DamnVulnerableIOSAppTests M
DamnVulnerabl...tchKit Extension M
DamnVulnerabl...pp WatchKit App M
Frameworks
Products
Project
DamnVulnerableIOSApp
Targets
DamnVulnerableIOSApp
DamnVulnerableIOSAppTests
DamnVulnerableIOSApp WatchKit Extension
DamnVulnerableIOSApp WatchKit App
Add Target...
Info
Build Settings

DamnVulnerableIOSApp
Latest iOS (iOS 10.1)
<Multiple values>
Debug Yes
Release No

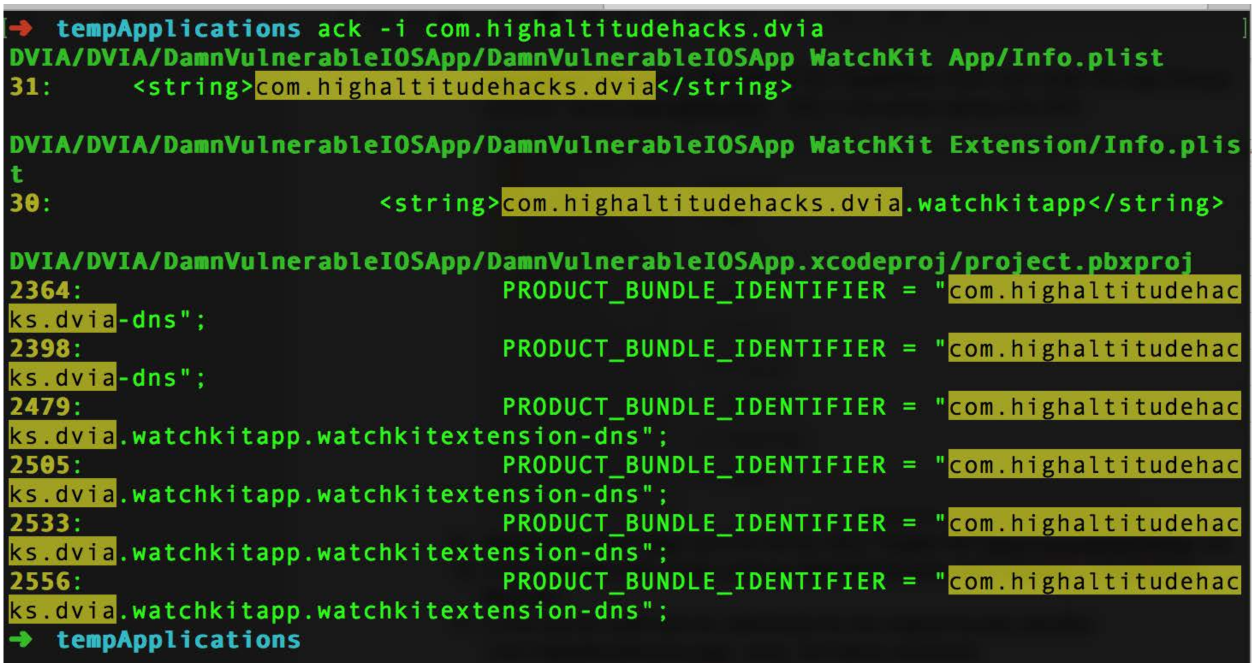
Build Options (iOS)
Setting DamnVulnerableIOSApp
Enable Testability <Multiple values>
Debug Yes
Release No
Validate Built Product Yes

Build Options (watchOS)



3. Click on General and set the Bundle Identifier to a unique value that is not already registered to any of the other developer accounts.
4. In the Signing section, enable “Automatically manage signing.”
   Enabling the “Automatically manage signing” option resets the signing build settings.
5. Change the Team Name to the Developer account.
6. Repeat the steps 4 and 5 for all the other Targets like WatchKit Extensions, etc.

7. If the Apple watch app will not be tested, disable the AppleWatch app group. Navigate to the Capabilities menu and under the App Groups uncheck “group.dvia.applewatch.” Add a new group “group.dns.dvia.”
8. Repeat step 7 for all the other Targets like DamnVulnerableiOSApp, etc.
9. In the WatchKit App target, remove WatchKit binaries from General -> Embedded Binaries.
10. In the Watchkit extensions rename *com.highaltitudehacks.dvia to com.dns.dvia* on the basis of the group name
11. In the source code look for references of the original bundle identifier “com.highaltitudehacks.dvia” using the below command
    ack -i com.highaltitudehacks.dvia



```
➜  tempApplications ack -i com.highaltitudehacks.dvia
DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit App/Info.plist
31:     <string>com.highaltitudehacks.dvia</string>

DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit Extension/Info.plis
t
30:                     <string>com.highaltitudehacks.dvia.watchkitapp</string>

DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp.xcodeproj/project.pbxproj
2364:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia-dns";
2398:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia-dns";
2479:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2505:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2533:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
2556:                           PRODUCT_BUNDLE_IDENTIFIER = "com.highaltitudehac
ks.dvia.watchkitapp.watchkitextension-dns";
➜  tempApplications
```

Make sure that all the references to original “*com.highaltitudehacks.dvia*” are replaced by the updated reference with “*com.highaltitudehacks.dvia-dns*”

In our case use open "DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit App/Info.plist" and replace “*com.highaltitudehacks.dvia*” with “*com.highaltitudehacks.dvia-dns*”

Open "DVIA/DVIA/DamnVulnerableIOSApp/DamnVulnerableIOSApp WatchKit Extension/Info.plist" and replace content with group name - *com.dns.dvia.watchkitapp*

12. In the Build Settings for all targets set *enable bitcode = no*
13. If there exists hard links in the framework search path, remove them.
14. (If using POD) Do a “pod init”
15. (If using POD) Add all required pods in the Podfile and do a “pod install”

16. (If using POD) Remove all the items that were added in Podfile from the general -> linked frameworks
17. Remove *-ObjC* from "*Other Linker Flags*" and add *$(inherited)*
18. Keep only *$(inherited)* in all the "search path"
19. Relaunch the workspace in Xcode and not the project file
20. A network connection issue means the application does not have ATS disabled. To disable it set the code below in DamnVulnerableIOSApp-Info.plist before the last dict.

    ```
    <key>NSAppTransportSecurity</key>
    <dict>
    <key>NSAllowsArbitraryLoads</key> <true/>
    </dict>
    ```
21. Make sure the iOS device is registered to a Developer account using the steps mentioned here: https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingProfiles/MaintainingProfiles.html#//apple_ref/doc/uid/TP40012582-CH30-SW10

7. iOS Security Model Primer

Below are some important features about the iOS security model.

7.1 Security Features

1. The applications need to be signed with a paid Apple developer certificate.
2. The application binaries are encrypted using FairPlayDRM, similar to what is used in iTunes Music.
3. The applications are protected by code signing.
4. Patched applications cannot be installed on non-jailbroken devices.
5. Every iOS application runs in its own sandbox. After iOS 8.3+, this sandboxed data cannot be accessed without jailbreaking the iOS device.
6. No application can access data belonging to another application. Protocol handlers like URL schemes are the only way for inter-application communication to be used for message passing between applications. The data can also be stored in keychains.
7. Whenever new files are created on the iOS device, they are assigned data protection classes as specified by the developers. This helps put access restriction on these files.
8. Applications need to specifically request for permission from the user to access resources like Camera, Maps, Contacts, etc.
9. iOS devices 5s+ have a secure hardware component called Secure Enclave. It is a highly-optimized version of ARM's TrustZone and prevents the main processor from directly accessing sensitive data. More details about the Secure Enclave can be found here:
    - https://www.blackhat.com/docs/us-16/materials/us-16-Mandt-Demystifying-The-Secure-Enclave-Processor.pdf

A detailed account of the available iOS security features can be found here:

- https://www.apple.com/business/docs/iOS_Security_Guide.pdf
- https://support.apple.com/en-us/HT207143
- https://www.apple.com/support/security/

8. Exploring iOS File System

The iOS File System can be accessed on both jailbroken and non-jailbroken phones. The amount of access varies. From iOS 8.3+ on the application sandbox can be accessed only on a jailbroken phone. Risk rating for this vulnerability should explained to customers so they can make an informed decision.

If there is a difficulty reading the file directories (except the Media and app directories on the iOS device) even after a jailbreak, make sure AFC2 (Apple File Conduit2) is installed from Cydia (https://cydia.saurik.com/package/com.saurik.afc2d/).

8.1 Reading Data Using iExplorer

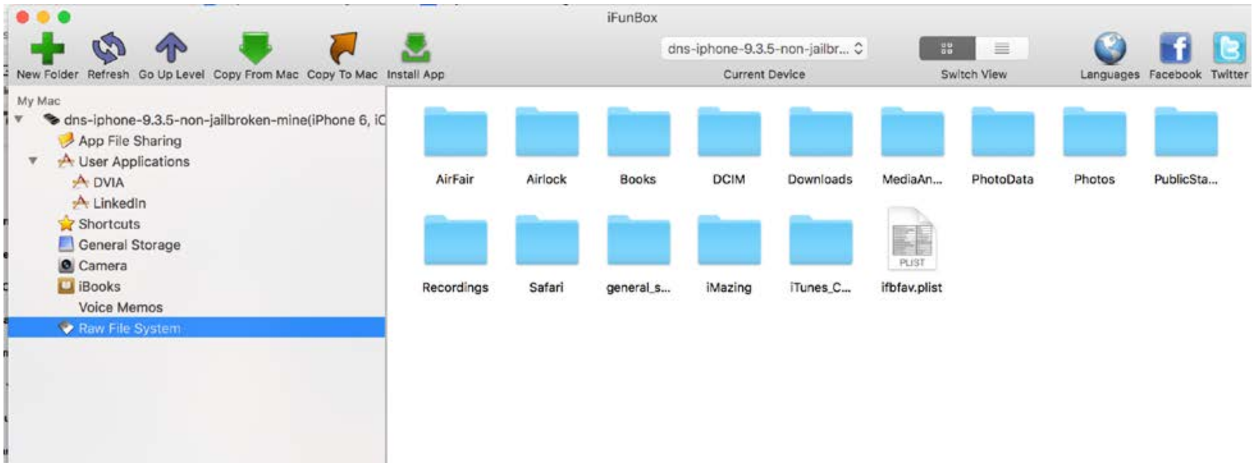
iExplorer is one of the simplest tools to view the iOS file structure use even without a jailbroken device. Before iOS 8.3, the application sandbox and its content were directly visible using iExplorer. As of iOS 8.3+ the application sandbox and root device directories are accessible only after jailbreak.

The iExplorer utility installed can be downloaded here: https://macroplant.com/downloads
Connecting the device to a laptop and launching iExplorer should enable the device file system to be read.

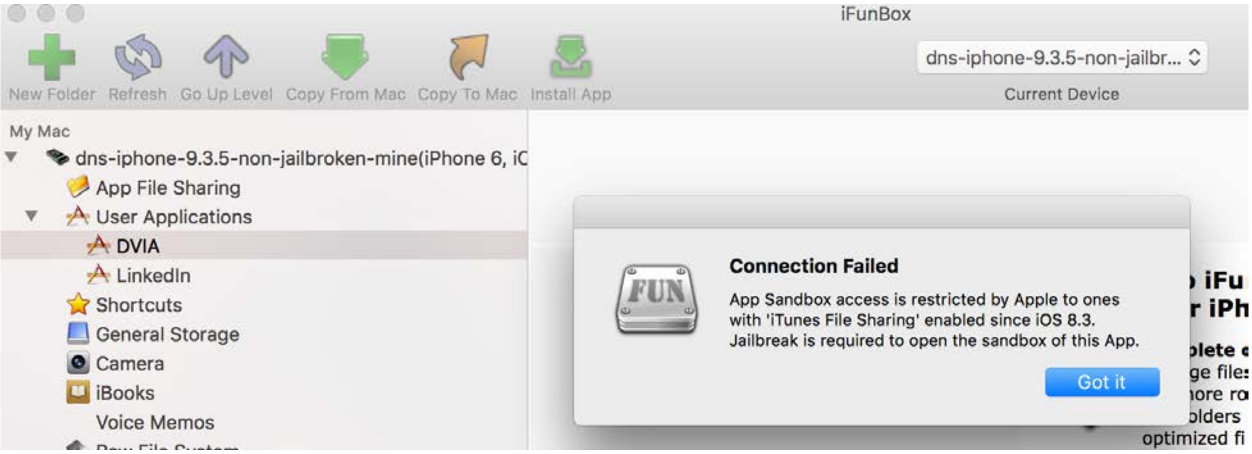
8.2 Reading Data Using iFunBox

iFunBox is another application that can be used to access the iOS File System. Before iOS 8.3, the application sandbox and its content was directly visible using iFunBox.

The following directories are accessible via iFunBox.



As of iOS 8.3+ the application sandbox and root device directories are accessible only after jailbreak.



Connection Failed

App Sandbox access is restricted by Apple to ones with 'iTunes File Sharing' enabled since iOS 8.3. Jailbreak is required to open the sandbox of this App.

iFunBox can also be used to install iOS applications on the device using the “Install App” feature. Cracked applications or applications without a provision certificate require a jailbroken device.

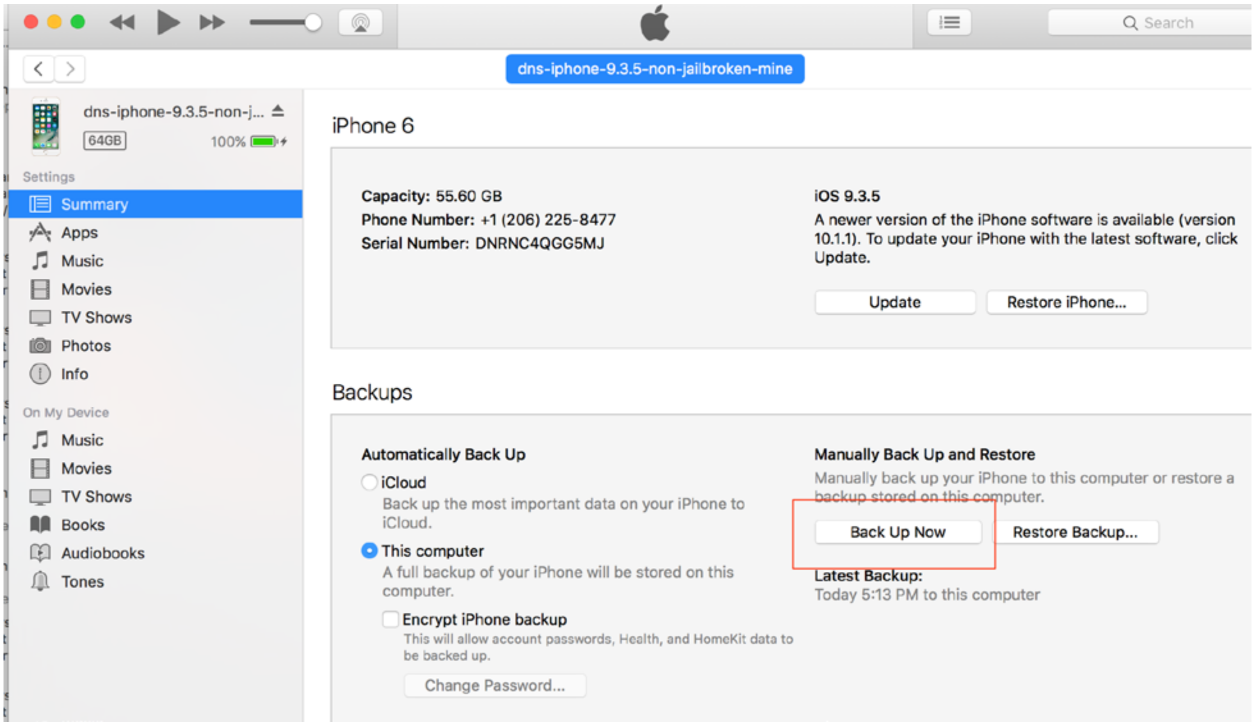
Legitimate applications can be directly installed using the iTunes method mentioned in “4.1 Method 1 - Using iTunes.”

8.3 Reading iOS > 8.3 Application SandBox Data Using Backup Method

Application sandbox data for iOS > 8.3 is not allowed by Apple, but there is a workaround that can be used to access this data. Before proceeding, take a backup of the device and the application data.

8.3.1 Backing Up the iDevice

Method 1: iTunes
Backing up the device can be performed using iTunes as shown below. Be sure to rename the device with a unique name so that it can be identified.



dns-iphone-9.3.5-non-jailbroken-mine

dns-iphone-9.3.5-non-j... 64GB 100%

iPhone 6

Capacity: 55.60 GB
Phone Number: +1 (206) 225-8477
Serial Number: DNRNC4QGG5MJ

iOS 9.3.5
A newer version of the iPhone software is available (version 10.1.1). To update your iPhone with the latest software, click Update.

Backups

Automatically Back Up

○ iCloud
Back up the most important data on your iPhone to iCloud.

● This computer
A full backup of your iPhone will be stored on this computer.

□ Encrypt iPhone backup
This will allow account passwords, Health, and HomeKit data to be backed up.

Manually Back Up and Restore
Manually back up your iPhone to this computer or restore a backup stored on this computer.

Latest Backup:
Today 5:13 PM to this computer

Method 2: LibImobiledevice
Alternatively, the backup can be performed using the *idevicebackup2* utility that can be installed from the libimobiledevice library. Install is as follows - *brew install libimobiledevice*.

The udid for the utility can be found using “*idevice_id -l*”

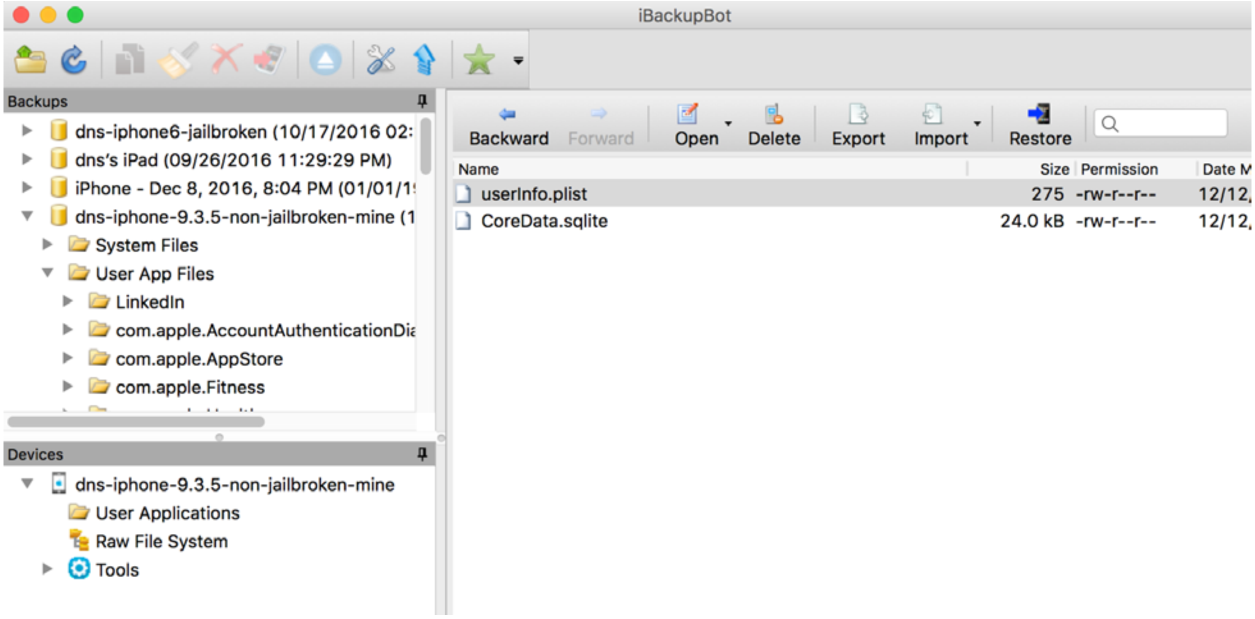
It may be necessary to run *sudo chmod -R 777 /var/db/lockdown* before backing up data.

Use the command below to back up the content:
idevicebackup2 backup --full --source <deviceudid> --udid <deviceudid> ~/Documents

Method 3: 3rd party tools
The backup of the device can also be done using iCloud content. Tools like iLoot (https://github.com/hackappcom/iloot) can be used. Legitimate login credentials linked with the target device are needed for the backup to work.
Use the command below to perform an iCloud backup of the content:
python iloot.py iCloudemailaddress iCloudpassword

Once the backup is complete, tools like iExplorer or iBackupBot can be used to view the application sandbox data.

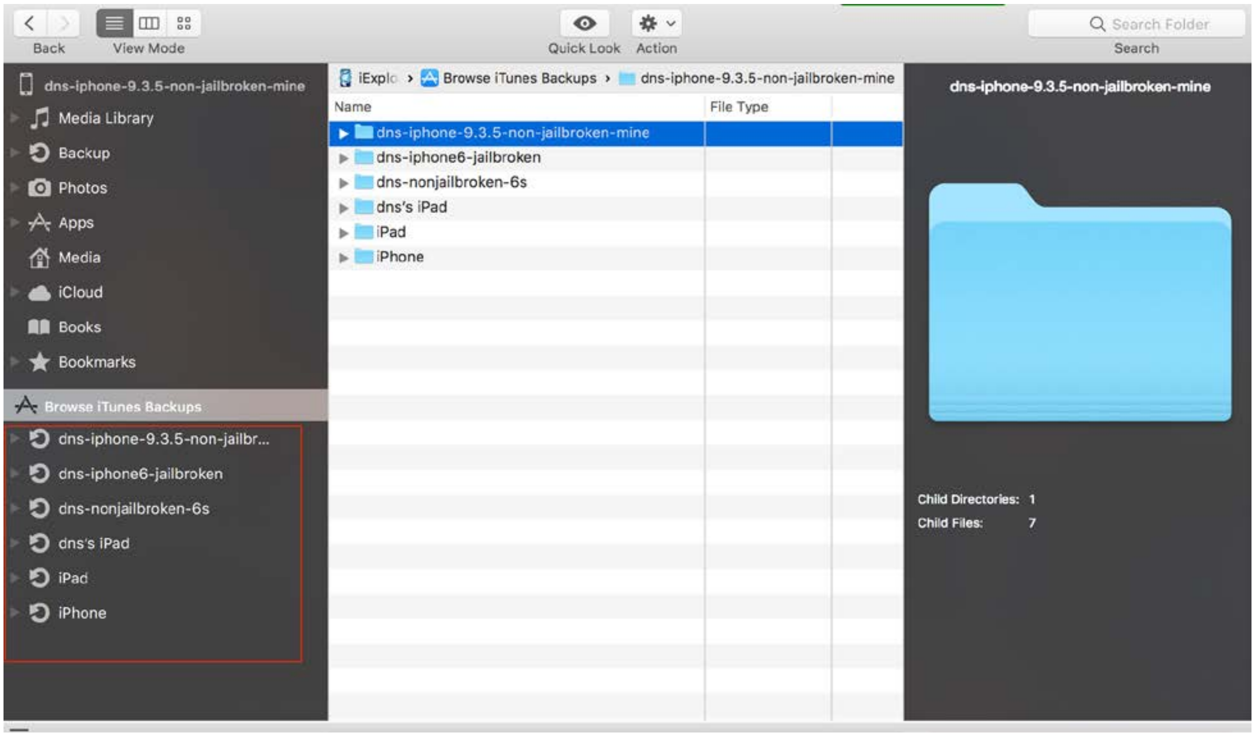
8.3.2 Using iBackupBot


iBackupBot
Backups
dns-iphone6-jailbroken (10/17/2016 02:
dns's iPad (09/26/2016 11:29:29 PM)
iPhone - Dec 8, 2016, 8:04 PM (01/01/1
dns-iphone-9.3.5-non-jailbroken-mine (1
System Files
User App Files
LinkedIn
com.apple.AccountAuthenticationDia
com.apple.AppStore
com.apple.Fitness
Devices
dns-iphone-9.3.5-non-jailbroken-mine
User Applications
Raw File System
Tools
Backward
Forward
Open
Delete
Export
Import
Restore

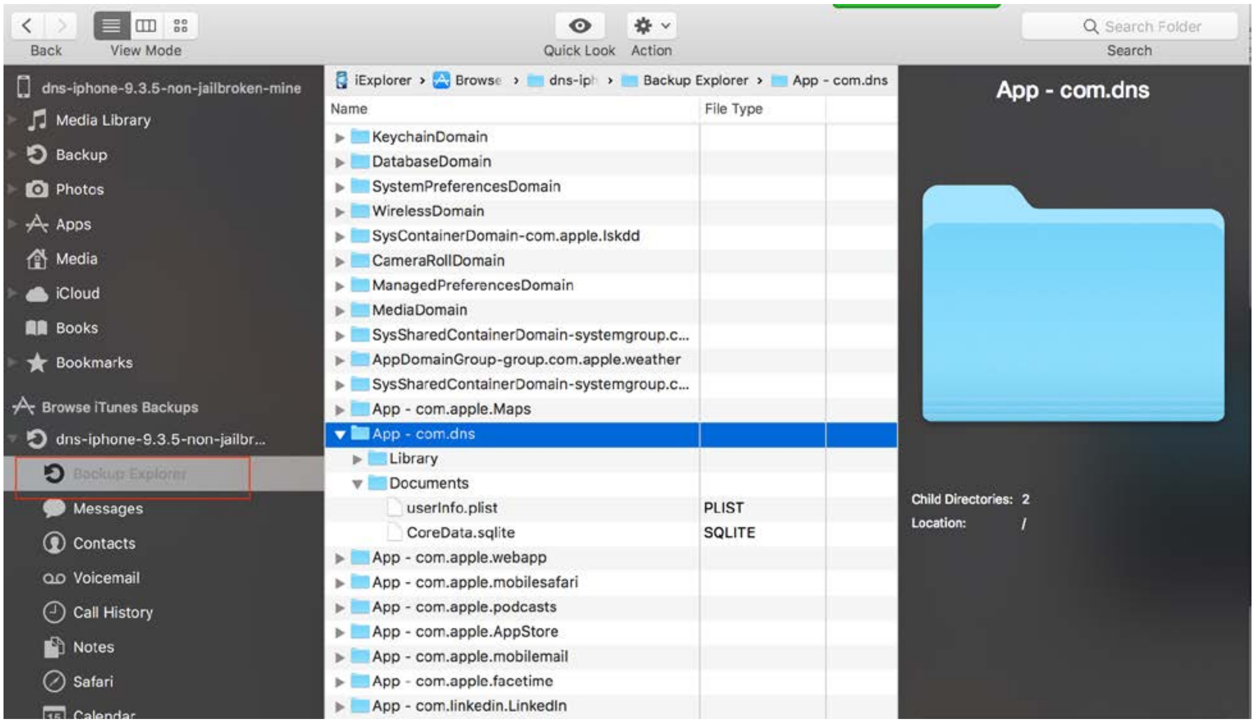
Name	Size	Permission	Date M
userInfo.plist	275	-rw-r--r--	12/12,
CoreData.sqlite	24.0 kB	-rw-r--r--	12/12,

8.3.3 Using iExplorer

Launch iExplorer and locate the backed-up device names and click on Backup Explorer to view content as shown below:



Note that the application sandbox is backed up and the application data can be viewed in the folder Backup Explorer.



Name	File Type
KeychainDomain	
DatabaseDomain	
SystemPreferencesDomain	
WirelessDomain	
SysContainerDomain-com.apple.lskdd	
CameraRollDomain	
ManagedPreferencesDomain	
MediaDomain	
SysSharedContainerDomain-systemgroup.c...	
AppDomainGroup-group.com.apple.weather	
SysSharedContainerDomain-systemgroup.c...	
App - com.apple.Maps	
App - com.dns	
Library	
Documents	
userInfo.plist	PLIST
CoreData.sqlite	SQLITE
App - com.apple.webapp	
App - com.apple.mobilesafari	
App - com.apple.podcasts	
App - com.apple.AppStore	
App - com.apple.mobilemail	
App - com.apple.facetime	
App - com.linkedin.LinkedIn	

Child Directories: 2
Location: /

8.4 Reading Application Data Using OpenSSH

OpenSSH is one of the first applications to be installed on a jailbroken device for accessing the iOS file system contents. OpenSSH can be installed from Cydia.

To install OpenSSH, navigate to Cydia, search for OpenSSH, and click on Install.

The default credentials for the SSH server hosted by OpenSSH locally on the device is “***alpine”*** (alpine was actually the codename for iOS 1.0).

After installing OpenSSH, choose a utility to access the iOS file system. For a GUI of the filesystem, use FileZilla on Mac or Linux. On windows, use WinSCP to connect to the iOS device.

Make sure to change the OpenSSH password after first login (***ssh root@IPAD_IP_ADDRESS***) to the SSH using simple “passwd” via a CLI.

For more information, go to: http://lifehacker.com/5760626/how-to-install-and-set-up-ssh-on-your-jailbroken-ios-device

Root access allows access to the complete iOS file system. Look over the file system to understand the various important directories and locations where the application can store data. This location can be the application sandbox or the iOS OS cache.

NOTE: OpenSSH does not need to be installed on the latest iOS 10.2 jailbreak because the jailbreak comes with a Dropbear instance out of the box.

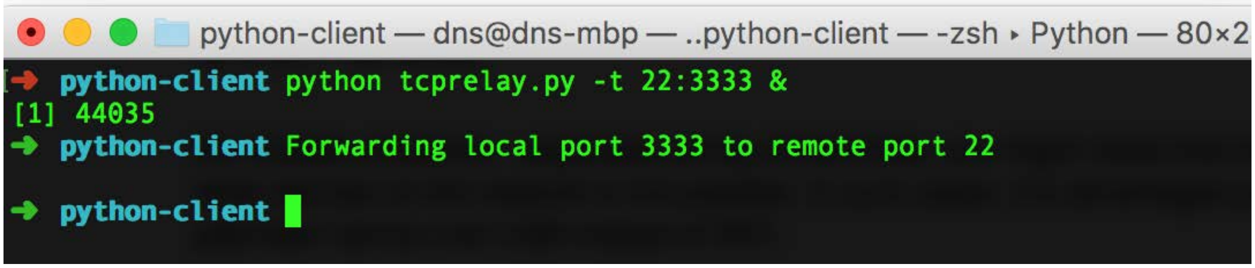
8.5 Reading Application Data Using SSH Over USB

Sometimes, especially during conferences- accessing multiple devices directly on a network is not possible. In these cases, the only way to access a jailbroken device is with USB instead of WiFi.

A tool that works well (even on iOS viz 10.2.1) is:
http://www.hackthatphone.com/5x/open_ssh_usb.shtml.

Install OpenSSH on the device. With the jailbroken iOS device connected to a laptop run the command below to create an SSH tunnel to access the iOS file system:

```
python tcprelay.py -t 22:3333 &
```


python-client — dns@dns-mbp — ..python-client — -zsh ▸ Python — 80×2

```
➜  python-client python tcprelay.py -t 22:3333 &
[1] 44035
➜  python-client Forwarding local port 3333 to remote port 22

➜  python-client 
```

Install SSH onto the device over USB, port 3333 (*ssh root@localhost -p 3333*) with credentials as *root/alpine*.

8.6 Reading Application Data on the iOS Device

On the first day of testing, it's typical to go through the application to become familiar with it. It may not be necessary to connect the iOS device to a laptop. Instead, it may be helpful to have simple tools to access the files in the application sandbox directly on the jailbroken device.

This section shows some of the tools that can be used.

8.6.1 FileExplorer/iFile

iFile can be installed on a jailbroken or a non-jailbroken device to access local data directly on a device. Application sandbox cannot be accessed without on a non-jailbroken device.

iFile is normally installed via Cydia, however it can be installed on a non-jailbroken device using Steps mentioned in "4.2 Method II - Using Cydia Impactor" and the IPA file here:
https://drive.google.com/open?id=0B0b4lUTjHfRKX3VrdW9GV2NUb2c

BillyE has a FileExplorer here: https://github.com/Billy-Ellis/iOS-File-Explorer This is a good alternative for non-jailbroken phones, but note that the accessible directories include only files that are publicly accessible.

8.6.2 Using Mobile Terminals

Use terminal programs on iOS devices to read data or access the iOS shell. The two terminals that work best are NewTerm and MobileTerminal.

9. Application Data Encryption

Developers can store sensitive data on an iOS device in a variety of ways. Data can be stored in the application sandbox or in the iOS Keychain. This section will cover the Apple Data Protection API and the ways in which data can be stored on an Apple device.

9.1 Understanding Apple Data Protection API

If sensitive data is stored in the application sandbox, data can be secured using Apple's Data Protection API. Apple's Data Protection API (DPAPI) specifies when the decryption key should be available. The DPAPI uses a combination of the user's device passcode and the hardware UID for encrypting each file.

Data protection is managed on a file-by-file basis. Every time a file is created on an iOS device, Apple uses a 256-bit unique file specific key and gives it to the built-in Apple AES hardware engine. The hardware engine encrypts that file using AES-CBC mode by default. On the A8 devices, the encryption is performed using AES-XTS.

This file specific key is encrypted with the "class key" depending upon how and when the file should be accessible and stored in the file's metadata which, in turn, is encrypted with the file system key. The class key is a simply random key, and is applied to the files based on the DPAPI level. Refer to page 15 of http://esec-lab.sogeti.com/static/publications/11-hitbamsterdam-iphonedataprotection.pdf for a detailed understanding of class key ID.

The data protection classes determine when the class is accessible. Below are currently available data protection classes:

a) *Complete Protection (NSFileProtectionComplete):*
This is the safest protection level that can be used unless a continuous read/write access to the file in the background is needed or if the device is locked. The class key is protected with a key derived from the user passcode and the device UID. If the device is locked, depending upon the ""require password"" setting on the device, the decrypted class key is soon discarded. The data secured by this attribute is not accessible until the user enters the passcode again or unlocks the device using Touch ID.

b) *Protected Unless Open (NSFileProtectionCompleteUnlessOpen)*
This protection level ensures that files that are open can still be accessed, even if the user locks the device. Other files with the same protection level cannot be accessed unless they were already opened when the device locked. Files can be created while the device is locked, but once closed, cannot be opened again until the device is unlocked.

c) *Protected Until First User Authentication (NSFileProtectionCompleteUntilFirstUserAuthentication):*
The default protection level for third party applications. This setting is automatically applied if another protection attribute is not specified. This protection class is similar to Complete Protection, but the file is available to the users after they first unlock the device. The file is stored in an encrypted format on a disk and cannot be accessed until after the device has booted and until the first device unlocks (similar to full-volume encryption on laptops).

d) *No Protection (NSFileProtectionNone):*
Only the class key with the UID is protected. The file can be read and written to at any time.

For a detailed source code, along with an explanation of these Data Protection classes refer to “Protection Levels” and “Checking for Protected Data Availability” sections in “iOS Application Security” by David Thiel (https://www.nostarch.com/iossecurity).

9.2 Validate the Data Protection Classes Being Used

FileDp is used to find the Data protection class of the file (http://www.securitylearn.net/wp-content/uploads/tools/iOS/FileDP.zip).

Below are the steps to use of FileDp:

1. Push FileDp to the iOS device using SSH
2. Make FileDp executable using chmod +x
3. Use the below command to view the Data protection accessibility constant for the file or directory in the application sandbox:
   ./FileDP -<f/d> <filepath/directorypath>

9.3 Insecure Local Data Storage

Below are some of the ways data can be stored on a device:

- PropertyList files
- NSUserDefaults class
- Keychain
- CoreData and SQLite databases

9.3.1 PropertyList files

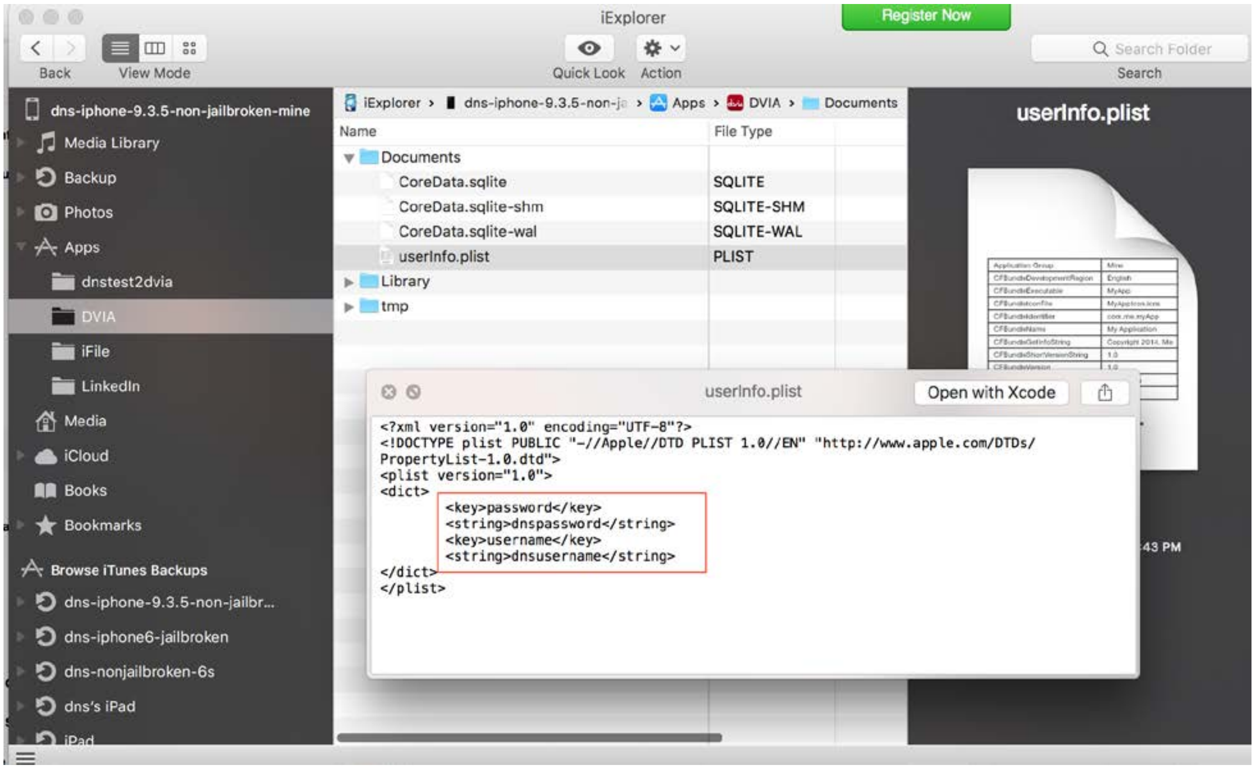
Plist files are one of the more standard ways of storing data on an iOS device in the form of key-value pair. Plist files are basically just XML files that can be read by Xcode. It is very common during penetration tests to notice that developers store sensitive data in plist files. Often, the sensitive data includes credentials, credit card information, API keys, financial information, PII etc. Plist files are not encrypted by default and should not be used to store sensitive information in clear text.

Application used for Example: Damn Vulnerable IOS Application

Black Box Testing Approach:

1. Launch the application and navigate to the Insecure Data Storage section.
2. Click on Plist.
3. Enter Username and Password.
4. Click on Save in Plist file.
5. Connect the device to the laptop.
6. It is possible to read the content of the iOS sandbox using any of the tools and methods mentioned in “8 Exploring iOS File System.” This example uses iExplorer.

7. Navigate to DVIA in iExplorer. In the Documents folder, right click on the userInfo.plist file and select “Quick Look.”



Name	File Type
Documents	
CoreData.sqlite	SQLITE
CoreData.sqlite-shm	SQLITE-SHM
CoreData.sqlite-wal	SQLITE-WAL
userInfo.plist	PLIST
Library	
tmp	

```
<?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>password</key>
        <string>dnspassword</string>
        <key>username</key>
        <string>dnsusername</string>
</dict>
</plist>
```

Note that credentials are stored in plaintext on the device. During a pentest, make sure that no sensitive information in stored in plist files without proper encryption.

9.3.2 NSUserDefaults Class

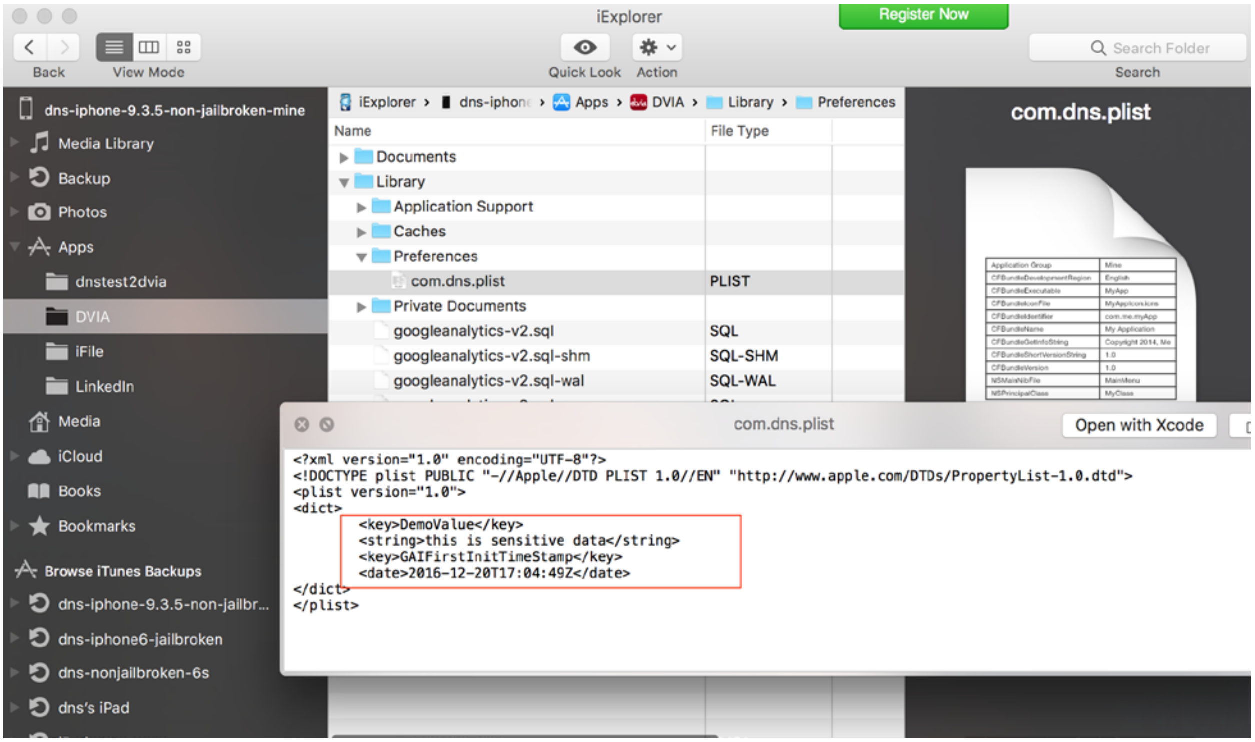
NSUserDefaults class is one more way data on the iOS device persists even after restart. The information stored NSUserDefaults class is stored in plaintext plist file at: <Application Directory>/Library/Preferences/<Bundle Identifier>.plist. During pentests, developers may assume that data will be encrypted and choose to store sensitive data here.

Application used for Example: Damn Vulnerable IOS Application

Black Box Testing Approach:

1. Launch the application and navigate to the Insecure Data Storage section.
2. Click on NSUserDefaults.

3. Enter text to be stored in the text field.
4. Click on Save in NSUserDefaults.
5. Connect the device to a laptop.
6. It is possible to read the content of the iOS sandbox using any of the tools and methods mentioned in “8 exploring iOS File System.” This example uses iExplorer.
7. Navigate to DVIA in iExplorer. In the Library > Preferences folder, right click on the plist file and select “Quick Look”.



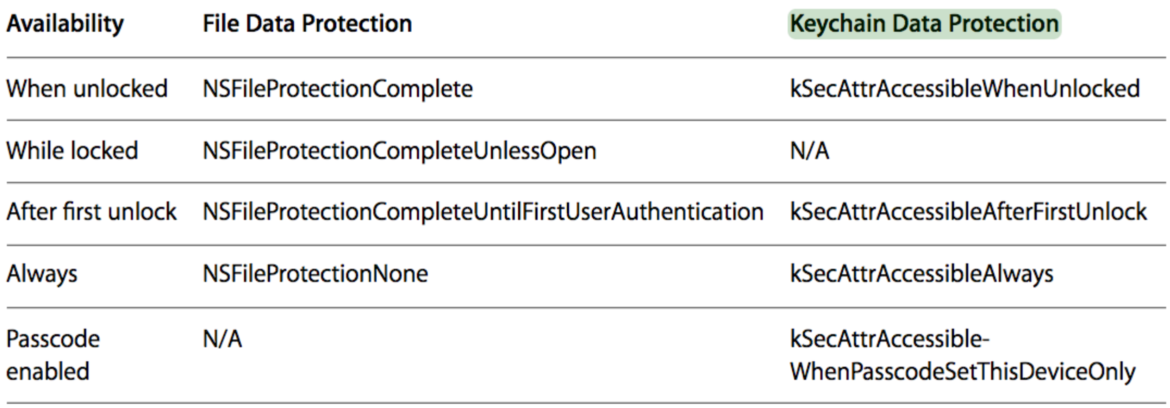
Name	File Type
Documents	
Library	
Application Support	
Caches	
Preferences	
com.dns.plist	PLIST
Private Documents	
googleanalytics-v2.sql	SQL
googleanalytics-v2.sql-shm	SQL-SHM
googleanalytics-v2.sql-wal	SQL-WAL

```
<?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>DemoValue</key>
        <string>this is sensitive data</string>
        <key>GAIFirstInitTimeStamp</key>
        <date>2016-12-20T17:04:49Z</date>
</dict>
</plist>
```

Note that the sensitive data entered is stored in plaintext on the device. During a pentest, make sure that no sensitive information is stored in plist files without proper encryption

9.3.3 Keychain

iOS Keychain is one of the best locations to store sensitive data like keys and login tokens. Keychain items can be shared only between applications from the same developer. The keychain items are encrypted using device hardware identifiers using AES-GCM. Keychain data is protected using a class structure similar to the one used in the Data Protection API. The classes also have similar behavior to that of the Data Protection API but use distinct keys and different names. See screenshot below from the Apple Security Guide: (https://www.apple.com/business/docs/iOS_Security_Guide.pdf)



Availability	File Data Protection	Keychain Data Protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionCompleteUnlessOpen	N/A
After first unlock	NSFileProtectionCompleteUntilFirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled	N/A	kSecAttrAccessible-WhenPasscodeSetThisDeviceOnly

Keychain protection classes determine when the class is accessible. Below are current data protection classes:

a. *Complete Protection (kSecAttrAccessibleWhenUnlocked):* The default value for keychain items added without explicitly setting an accessibility constant. Developers use this protection level when the application needs access to the keychain data only when the application is in the foreground. When used, the keychain item data can be accessible only when the device is unlocked. Keychain data items with this attribute migrate to a new device when using encrypted backup.

b. *Protected Until First User Authentication (kSecAttrAccessibleAfterFirstUnlock):* Similar to Complete Protection but keychain items are available to the users after they first unlock the device. The keychain items are stored in an encrypted format on a disk and cannot be accessed until after the device has booted and until the first device unlocks.

c. *Protected when passcode enabled (kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly):* Developers use this protection level when the application needs access to the keychain data only when the application is in the foreground and needs additional security. When used, the keychain item data can be accessible only when the device is unlocked and a passcode is enabled on the device. Data cannot be stored on the device keychain when the pin code is not set on the device. The keychain data items with this attribute never migrate to a new device. If the pin code is disabled, the keychain item data gets deleted.

d. *No Protection (kSecAttrAccessibleAlways)*: When this protection level is used, the data in the keychain item is always accessible even when the device is locked.

Even though keychain is a secure way of storing data, on a jailbroken device, an attacker can still gain access to this information. During a pentest, look for sensitive information like passwords stored in the keychain. Plaintext passwords should never be stored in keychains. As mitigation, set up a session handling mechanism by means of cookies to avoid the need to store credentials in iOS keychains.

Application used for Example: Damn Vulnerable IOS Application

Black Box Testing Approach:

1. Launch the application and navigate to the Insecure Data Storage section.
2. Click on Keychains.
3. Enter text to be stored in the text field.
4. Click on Save in Keychain.
5. SSH into the iOS device with the credentials root/alpine.
6. Download keychain dumper tool using the command below:
   - *wget http://github.com/ptoomey3/Keychain-Dumper/archive/master.zip --no-check-certificate*
7. Unzip the zip folder and navigate to the Keychain-dumper-master folder.
8. Make the keychain_dumper file executable using the below command:
   - *chmod +x keychain_dumper*
9. Run the command below and note that the information that was supposed to be critical and sensitive was found stored in plaintext.
   - *./keychain_dumper*

If not secured, the keychain data can also be found in the iOS device backup.

For detailed source code along with an explanation on the usage of these Keychain protection schemes, refer to Chapter 13 : “Using the Keychain” section in “iOS Application Security” by David Thiel (https://www.nostarch.com/iossecurity).

9.3.4 CoreData and SQLite Databases

CoreData is the framework that manages the layer between user interface and the data stored in a database. The main advantage of CoreData over SQLite databases is the speed and ease of use. Using CoreData creates sqlite files on the iOS device.

The main difference between using SQLite and CoreData is that the tables are prefixed with Z in CoreData. The SQLite files are stored in the Documents folder in the application sandbox.

Application used for Example: Damn Vulnerable IOS Application

Black Box Testing Approach:

1. Launch the application and navigate to the Insecure Data Storage section.
2. Click on Core Data.
3. Enter the data in all fields.
4. Click on Save in Core Data.
5. Connect the device to a laptop.
6. It is possible to read the content of the iOS sandbox using any of the tools and methods mentioned in "8 Exploring iOS File System." This example uses iExplorer.
7. Navigate to DVIA in iExplorer. In the Documents folder, right click on the CoreData.sqlite file and export it to laptop.
8. The sqlite file can be read using SQLite Browser (http://sqlitebrowser.org/) or SQLite Manager (https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/), or sqlite3 CLI. When using the sqlite3 on the iOS device use the below commands to view the contents of the sqlite3 database:
    - *sqlite3 CoreData.sqlite*
    - *.tables*
    - *select * from ZUSER*

   Note the credentials are being stored in plaintext on the iOS device.

Similar steps are to be used for testing SQLite storage vulnerabilities except the tables won't have 'Z' prefixed.

9.4 Broken Cryptography

CommonCrypto is the framework that iOS uses for Cryptographic operations. CCCrypt is the main encryption and decryption function of the framework for symmetric cryptography.

The method signature for CCCrypt:

```
CCCrypt(CCOperation op, CCAlgorithm alg, CCOptions options,    const void *key, size_t keyLength, const
void *iv, const void *dataIn,       size_t dataInLength, void *dataOut, size_t dataOutAvailable,       size_t
*dataOutMoved);
```

The types of arguments passed:

```
CCCryptorStatus CCCrypt(
 CCOperation op,         // operation: kCCEncrypt or kCCDecrypt
 CCAlgorithm alg,        // algorithm: kCCAlgorithmAES128...
 CCOptions options,      // operation: kCCOptionPKCS7Padding...
 const void *key,        // key
 size_t keyLength,       // key length
 const void *iv,         // initialization vector (optional)
 const void *dataIn,     // input data
 size_t dataInLength,    // input data length
 void *dataOut,          // output data buffer
 size_t dataOutAvailable, // output data length available
 size_t *dataOutMoved)    // real output data length generated
```

Some of the common test cases to test for when looking at the cryptographic security used in the application are outlined in the table below.

Test ID	Test Title	Test Description
1	Secure Random Number Generation	Confirm that secure random generation is performed by reading the bytes from /dev/random device file. This can be done by using the SecRandomCopyBytes

		function. Refer to https://developer.apple.com/reference/security/1658565-randomization_services for the randomization service details on iOS.
2	Random Initialization Vector	Confirm that the IV used in the encryption and hashing process is generated using a secure pseudorandom generator (using /dev/random as mentioned in 1) and is sufficiently random and unique every time.
3	Hardcoded Initialization Vector	Confirm that the IV in the encryption and hashing process is not hardcoded in the application source.
4	Key Size	Confirm that the Key size used in the encryption and hashing process for the cryptosystem is sufficiently large.
5	Random Salt	Confirm that the salt in the encryption and hashing process is generated using a secure pseudorandom (using /dev/random as mentioned in 1) generator and is sufficiently random and unique every time.
6	Hardcoded Salt	Confirm that the salt in the encryption and hashing process is not hardcoded in the application source.
7	Strong Cryptographic Seed for Random Number Generator	Ensure that the seed value has sufficient entropy and does not rely on weak sources of entropy. Also, ensure periodic reseed of the seed value.
9	Cryptographic Strength	Confirm that a cryptographically strong algorithm is used to encrypt sensitive data.
10	Integrity Checks - Encryption	Confirm that the integrity and authenticity of the encryption function is maintained by means of ENCRYPT-then-MAC or authenticated encryption mode.
11	Password Hashing	Confirm that the passwords are stored using a secure password hashing function.
12	Number of Iterations	Confirm that the number of iterations for the hashing/encryption function in use is sufficiently large.
13	Integrity Checks - Hashing	Confirm that the integrity/authenticity of hashing function is maintained by means of ENCRYPT-then-

		MAC.
14	Weak Cipher Mode	Confirm that the weak ECB cipher mode is not in use.
15	Error Messages	Confirm that error messages do not reveal sensitive information regarding the encryption system or the machine that hosts the application.
16	Password-Based Encryption	Confirm that weak and old password based encryptions algorithms are not currently in use.
17	Custom Cryptography	Confirm that the application does not use a custom crypto algorithm.

Ensure that the key used in CCCrypt or any of the related functions is never hardcoded. It has to be generated dynamically on the device and can then be stored in the keychain.

If adding encryption functionality is suggested, RNCryptor (https://github.com/rnapier/RNCryptor) is your best bet. Essentially, it serves as a wrapper over CommonCrypto and allows encryption using AES and a user supplied key. In most cases, the key is the one that is generated dynamically on the device and can then be stored in the keychain.
Use the latest version of RNCryptor as the older versions have known vulnerabilities: (http://robnapier.net/rncryptor-hmac-vulnerability).

10. Binary Analysis

Binary Analysis – Check for Exploit Mitigations – Making Buffer Overflows Difficult to Exploit

Buffer overflows occur when the attacker's cleverly constructed input overwrites memory resulting in the execution of the attacker's code.

There are 3 technologies that are used to prevent buffer overflows

- Address Space Layout Randomization (ASLR)
- Automatic Reference Counting (ARC)
- Stack Protectors

If one or more of these options are not used or if there are improperly handled strings or dangerous string functions, the application may be vulnerable to a buffer overflow exploit.
Although these don't directly affect security testing, it is much harder to write buffer overflows for code compiled with these options.

Test engineers need to know if these technologies are implemented in apps that are being tested. This section covers the required details to better understand these technologies.

10.1 Binary Analysis – Check for Exploit Mitigations – Position Independent Executable (PIE & ASLR)

- Memory corruption vulnerabilities typically rely on knowing where in the process address space to overwrite code or data
    - Code does not care where it is located in memory, so it is moved to a random spot. This makes Return Oriented Programming (ROP) attacks much more difficult to execute reliably. (https://access.redhat.com/blogs/766093/posts/1975793)
    - Address Space Layout Randomization (ASLR) randomizes where code and data are mapped to in the processes address space
- Executable binaries are made entirely from position-independent code
    - ASLR allows for the creation of Position Independent Executables
    - All built-in applications are compiled with PIE by default after iOS 5.0.

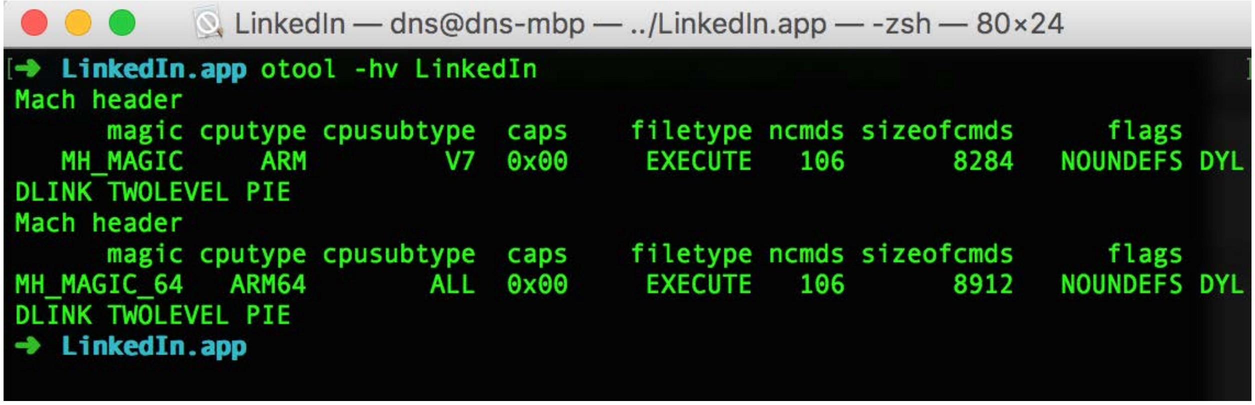
Application used for Example: LinkedIn from AppStore

White Box Testing Approach:

1. Check if “Generate Position-Dependent Code” is set to “YES” within the XCode project Build Setting
2. You can also look for *-fPIC* and *-pie* flags set for the compiler

Black Box Testing Approach:

1. Use otool to check whether PIE is enabled on the application binary. Run the command below and look for the PIE flag in the mach header
    - otool -hv <appname>



```
[➜  LinkedIn.app otool -hv LinkedIn
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
   MH_MAGIC     ARM         V7  0x00     EXECUTE   106       8284   NOUNDEFS DYL
DLINK TWOLEVEL PIE
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
MH_MAGIC_64   ARM64        ALL  0x00     EXECUTE   106       8912   NOUNDEFS DYL
DLINK TWOLEVEL PIE
➜  LinkedIn.app
```

10.2 Binary Analysis – Check for Exploit Mitigations – Automatic Reference Counting (ARC)

- Automatic Reference Counting (ARC) removes the responsibility of memory management
    - The compiler manages memory, reducing the likelihood of introducing memory corruption vulnerabilities into the application
- ARC evaluates the lifetime requirements of objects and automatically inserts the appropriate memory management calls during pre-compilation
    - Developers no longer have to remember when to use retain, release, and auto-release memory objects

  - The compiler determines when an object's lifetime has expired and will automatically dealloc objects for the developer
- Safeguards against many memory corruption vulnerabilities and specially object use-after-free and double-free flaws

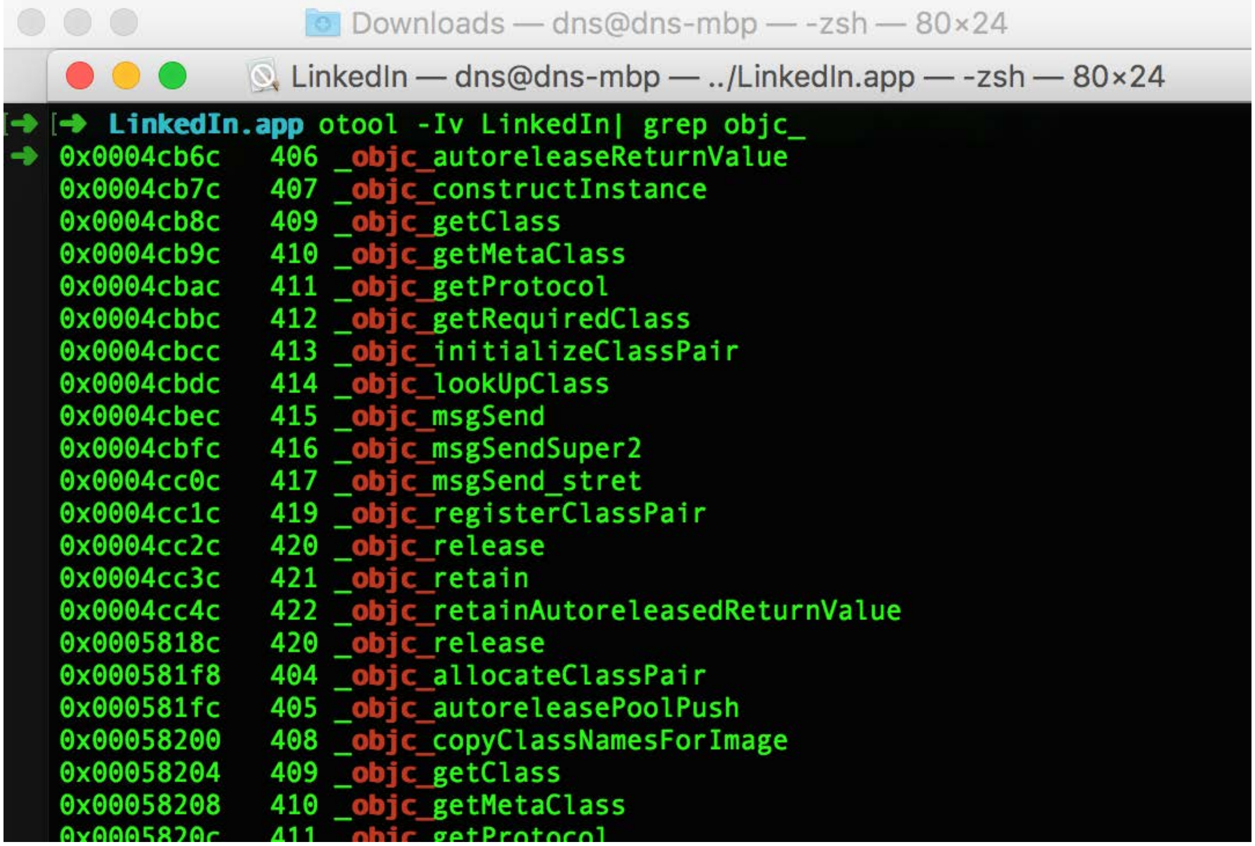
Application used for Example: LinkedIn from AppStore

White Box Testing Approach:

1. Check if "Objective-C Automatic Reference Counting" is set to "YES" within the XCode project Build Setting
2. You can also look for *-fobjc-arc* or *-fno-objc-arc* compiler flags being used

Black Box Testing Approach:

1. Use otool to check whether ARC is enabled on the application binary. Run the command below to look at ARC references:
   - otool -Iv <appbinary>| grep objc_


Downloads — dns@dns-mbp — -zsh — 80×24
LinkedIn — dns@dns-mbp — ../LinkedIn.app — -zsh — 80×24

```
➜ LinkedIn.app otool -Iv LinkedIn| grep objc_
0x0004cb6c   406 _objc_autoreleaseReturnValue
0x0004cb7c   407 _objc_constructInstance
0x0004cb8c   409 _objc_getClass
0x0004cb9c   410 _objc_getMetaClass
0x0004cbac   411 _objc_getProtocol
0x0004cbbc   412 _objc_getRequiredClass
0x0004cbcc   413 _objc_initializeClassPair
0x0004cbdc   414 _objc_lookUpClass
0x0004cbec   415 _objc_msgSend
0x0004cbfc   416 _objc_msgSendSuper2
0x0004cc0c   417 _objc_msgSend_stret
0x0004cc1c   419 _objc_registerClassPair
0x0004cc2c   420 _objc_release
0x0004cc3c   421 _objc_retain
0x0004cc4c   422 _objc_retainAutoreleasedReturnValue
0x0005818c   420 _objc_release
0x000581f8   404 _objc_allocateClassPair
0x000581fc   405 _objc_autoreleasePoolPush
0x00058200   408 _objc_copyClassNamesForImage
0x00058204   409 _objc_getClass
0x00058208   410 _objc_getMetaClass
0x0005820c   411 _objc_getProtocol
```

Otool won't indicate if ARC is in use, but the following symbols will indicate ARC usage:

_objc_retain
_objc_release
_objc_storeStrong
_objc_releaseReturnValue
_objc_autoreleaseReturnValue
_objc_retainAutoreleasedReturnValue

10.3 Binary Analysis – Check for Exploit Mitigations – Stack Protectors

- Protection against memory corruption vulnerabilities that attempt to overwrite the stack such as stack-based buffer overflows
- Achieved by placing a known value or "stack canary" before local variables on the stack to protect the saved base point, instruction pointer, and saved function arguments
- When the function returns, the canary value is checked to verify that the stack hasn't been overwritten
- Stack protection is enabled by default in the latest iOS

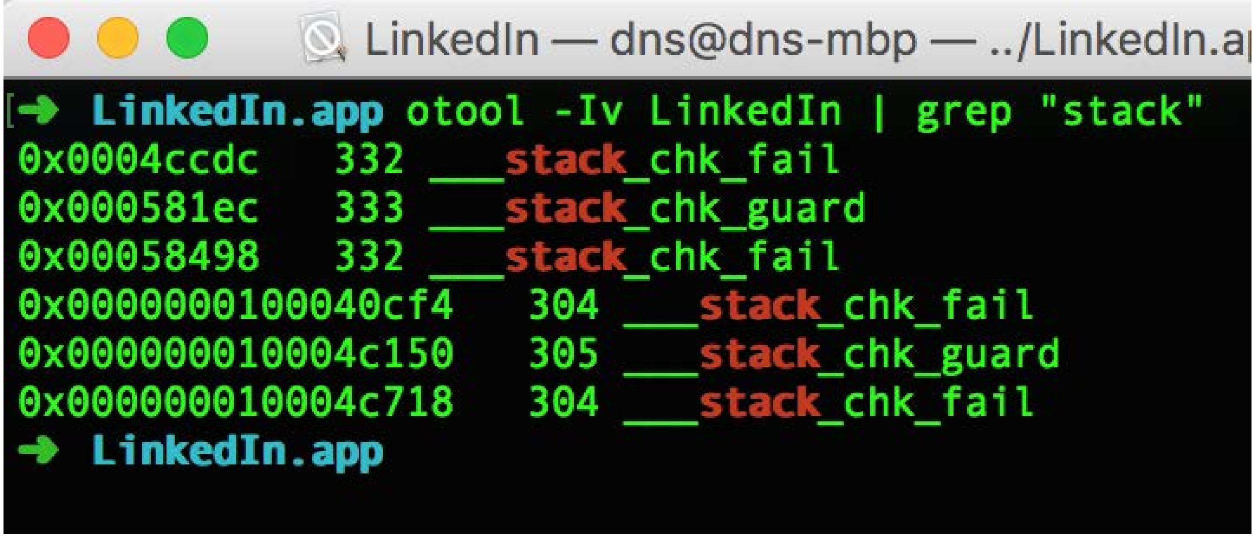
Application used for Example: LinkedIn from AppStore

White Box Testing Approach:

- Look for –fstack-protector-all compiler flags being used

Black Box Testing Approach:

- Use otool to check whether stack protectors are enabled on the application binary. Run the command below and to look at stack references:
    - otool -Iv <appbinary> | grep "stack"



```
[➜  LinkedIn.app otool -Iv LinkedIn | grep "stack"
 0x0004ccdc   332 ___stack_chk_fail
 0x000581ec   333 ___stack_chk_guard
 0x00058498   332 ___stack_chk_fail
 0x0000000100040cf4   304 ___stack_chk_fail
 0x000000010004c150   305 ___stack_chk_guard
 0x000000010004c718   304 ___stack_chk_fail
 ➜  LinkedIn.app
```

The presence of these symbols indicates stack protection

- ___stack_ chk_fail
- ___stack_ chk_guard

10.4 Binary Analysis – List All Libraries Used in the iOS Binary

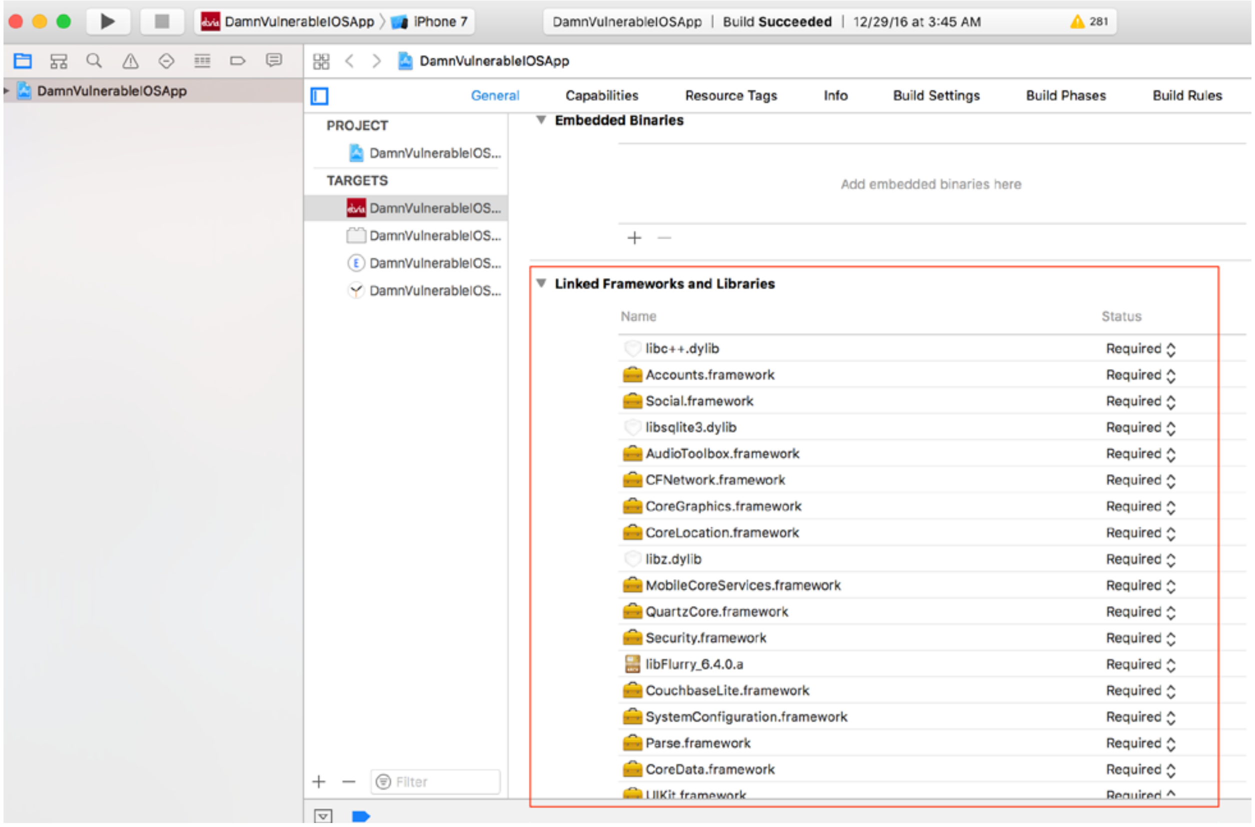
When performing a penetration test on an iOS application, check the security posture of the application as well as the libraries used in the application using otool or a more advanced tool named Jtool. Vulnerabilities libraries in the application could allow for leaks of application information that should not be available to attackers.

Download the latest version of Jtool from http://newosxbook.com/tools/jtool.html.

Application used for Example: Damn Vulnerable iOS Application

White Box Testing Approach:

1. Open the Xcode project and View the "General" project properties.
2. Scroll down to view the "Linked Framework and Libraries" section. This lists the frameworks used in the application.



Name	Status
libc++.dylib	Required
Accounts.framework	Required
Social.framework	Required
libsqlite3.dylib	Required
AudioToolbox.framework	Required
CFNetwork.framework	Required
CoreGraphics.framework	Required
CoreLocation.framework	Required
libz.dylib	Required
MobileCoreServices.framework	Required
QuartzCore.framework	Required
Security.framework	Required
libFlurry_6.4.0.a	Required
CouchbaseLite.framework	Required
SystemConfiguration.framework	Required
Parse.framework	Required
CoreData.framework	Required
UIKit.framework	Required

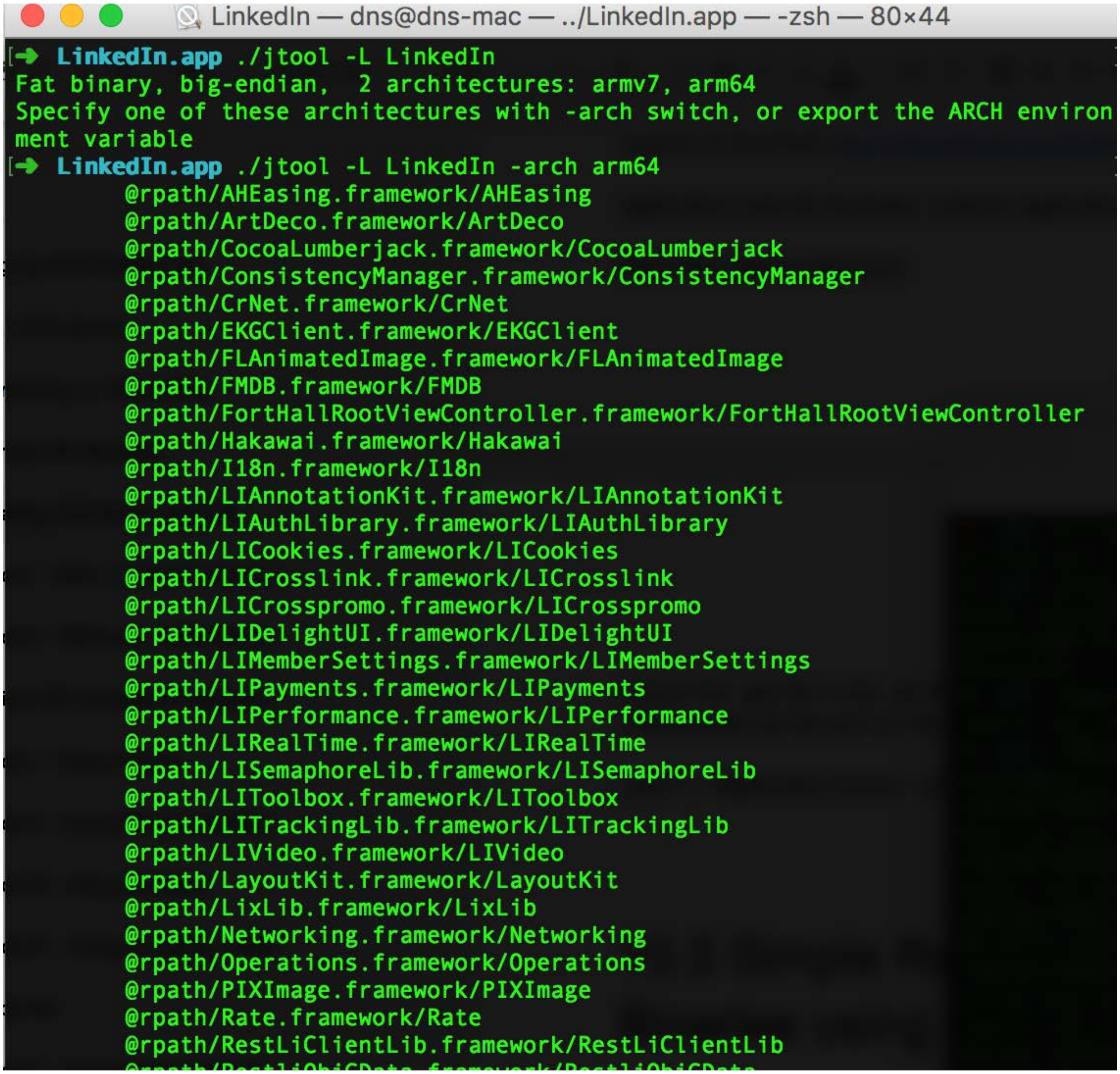
Make sure that all the libraries are up to the date with the latest versions. Any publicly known open vulnerabilities in these libraries leave the application vulnerable.

Application used for Example: LinkedIn Application

Black Box Testing Approach:

Extract the .app file or the .IPA file using the steps mentioned in "2. Acquiring iOS Binaries." Un-compress the file and run the command below on the binary file:

- *./jtool -L <application-binary> -arch arm64*



```
[➜  LinkedIn.app ./jtool -L LinkedIn
 Fat binary, big-endian,  2 architectures: armv7, arm64
 Specify one of these architectures with -arch switch, or export the ARCH environ
 ment variable
[➜  LinkedIn.app ./jtool -L LinkedIn -arch arm64
        @rpath/AHEasing.framework/AHEasing
        @rpath/ArtDeco.framework/ArtDeco
        @rpath/CocoaLumberjack.framework/CocoaLumberjack
        @rpath/ConsistencyManager.framework/ConsistencyManager
        @rpath/CrNet.framework/CrNet
        @rpath/EKGClient.framework/EKGClient
        @rpath/FLAnimatedImage.framework/FLAnimatedImage
        @rpath/FMDB.framework/FMDB
        @rpath/ForthHallRootViewController.framework/ForthHallRootViewController
        @rpath/Hakawai.framework/Hakawai
        @rpath/I18n.framework/I18n
        @rpath/LIAnnotationKit.framework/LIAnnotationKit
        @rpath/LIAuthLibrary.framework/LIAuthLibrary
        @rpath/LICookies.framework/LICookies
        @rpath/LICrosslink.framework/LICrosslink
        @rpath/LICrosspromo.framework/LICrosspromo
        @rpath/LIDelightUI.framework/LIDelightUI
        @rpath/LIMemberSettings.framework/LIMemberSettings
        @rpath/LIPayments.framework/LIPayments
        @rpath/LIPerformance.framework/LIPerformance
        @rpath/LIRealTime.framework/LIRealTime
        @rpath/LISemaphoreLib.framework/LISemaphoreLib
        @rpath/LIToolbox.framework/LIToolbox
        @rpath/LITrackingLib.framework/LITrackingLib
        @rpath/LIVideo.framework/LIVideo
        @rpath/LayoutKit.framework/LayoutKit
        @rpath/LixLib.framework/LixLib
        @rpath/Networking.framework/Networking
        @rpath/Operations.framework/Operations
        @rpath/PIXImage.framework/PIXImage
        @rpath/Rate.framework/Rate
        @rpath/RestLiClientLib.framework/RestLiClientLib
        @rpath/RestliObjCData.framework/RestliObjCData
```

Make sure that the latest versions of these libraries are being currently used. Any publicly known open vulnerabilities in theses libraries leave the application vulnerable.

10.5 Simple Reverse Engineering iOS Binaries Using class-dump-z

When performing a penetration test on an iOS application, it is important to read the code of the provided application and understand the backend classes and hidden information. This allows for exploitation of the application to gain access to sensitive information or to redirect the flow of the application in a malicious manner. Reverse Engineering an iOS application is completely different compared to an Android apk.

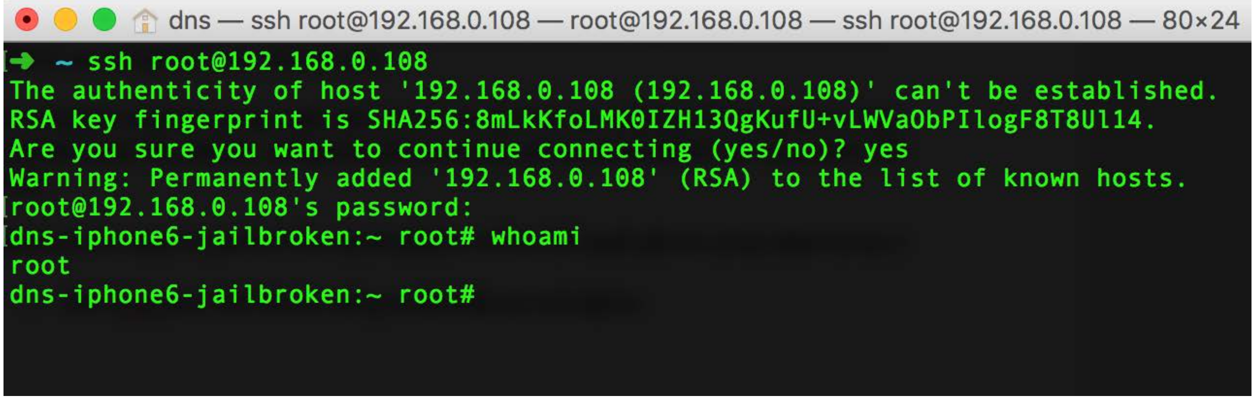
The complete original source code cannot be revived from an existing iOS application. Only declarations for the classes, categories, and protocols can be decompiled from any given iOS binary. Advance tools like IDA Pro or Hopper can be used to look at the pseudo code.

Using *class-dump-z* from *cydia.radare.org* repository on Cydia as an example, note that the default class dump utility that is bundled with Cydia does not support 64-bit Mach-O files. The Mac OSX version of class dump for reverse engineering the application may be used as well.

Application used for Example: Default Stocks application

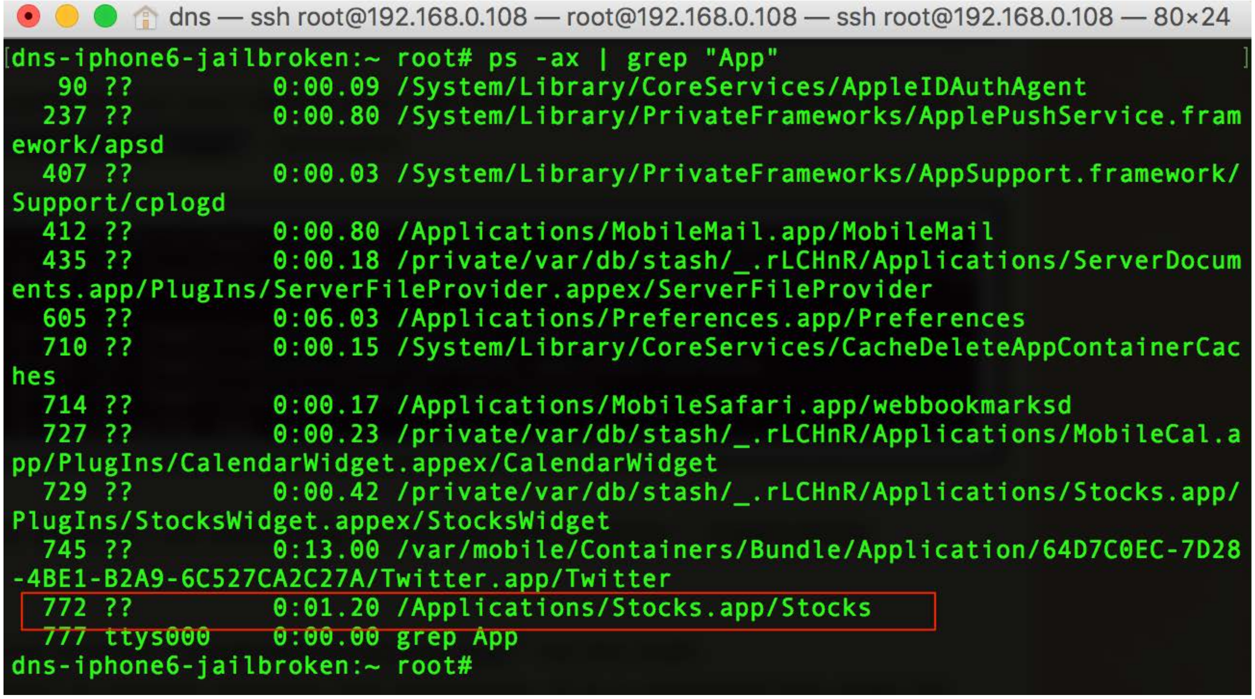
Below are the steps to perform the de-compilation of the iOS applications using class-dump-z:

1. SSH into iOS device using credentials as root:alpine.


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80×24

```
➜  ~ ssh root@192.168.0.108
The authenticity of host '192.168.0.108 (192.168.0.108)' can't be established.
RSA key fingerprint is SHA256:8mLkKfoLMK0IZH13QgKufU+vLWVaObPIlogF8T8Ul14.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.108' (RSA) to the list of known hosts.
root@192.168.0.108's password:
dns-iphone6-jailbroken:~ root# whoami
root
dns-iphone6-jailbroken:~ root#
```

2. Launch the Stocks application on the device and note the application location using the *ps -ax | grep "App"* command.


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80×24

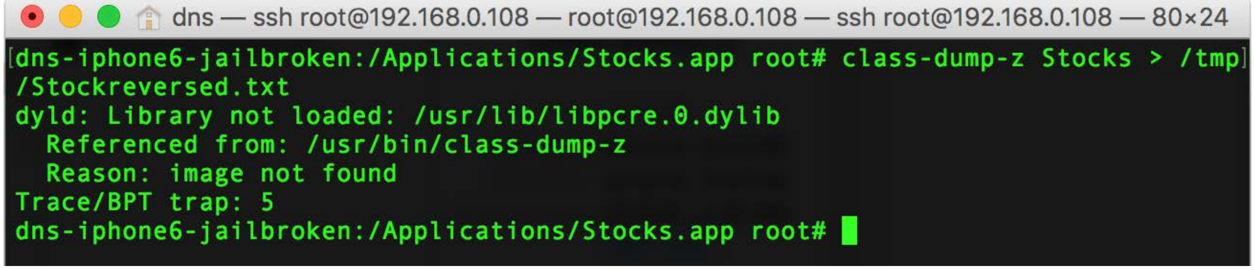
```
dns-iphone6-jailbroken:~ root# ps -ax | grep "App"
   90 ??         0:00.09 /System/Library/CoreServices/AppleIDAuthAgent
  237 ??         0:00.80 /System/Library/PrivateFrameworks/ApplePushService.fram
ework/apsd
  407 ??         0:00.03 /System/Library/PrivateFrameworks/AppSupport.framework/
Support/cplogd
  412 ??         0:00.80 /Applications/MobileMail.app/MobileMail
  435 ??         0:00.18 /private/var/db/stash/_.rLCHnR/Applications/ServerDocum
ents.app/PlugIns/ServerFileProvider.appex/ServerFileProvider
  605 ??         0:06.03 /Applications/Preferences.app/Preferences
  710 ??         0:00.15 /System/Library/CoreServices/CacheDeleteAppContainerCac
hes
  714 ??         0:00.17 /Applications/MobileSafari.app/webbookmarksd
  727 ??         0:00.23 /private/var/db/stash/_.rLCHnR/Applications/MobileCal.a
pp/PlugIns/CalendarWidget.appex/CalendarWidget
  729 ??         0:00.42 /private/var/db/stash/_.rLCHnR/Applications/Stocks.app/
PlugIns/StocksWidget.appex/StocksWidget
  745 ??         0:13.00 /var/mobile/Containers/Bundle/Application/64D7C0EC-7D28
-4BE1-B2A9-6C527CA2C27A/Twitter.app/Twitter
  772 ??         0:01.20 /Applications/Stocks.app/Stocks
  777 ttys000    0:00.00 grep App
dns-iphone6-jailbroken:~ root#
```

As shown in the above diagram, the application is running from location “ /Applications/Stocks.app/Stocks”.

3. Navigate to “/Applications/Stocks.app/” via the shell. Use class-dump-z to reverse engineer this application. It is a command-line utility for examining the Objective-C runtime information stored in Mach-O files. It generates declarations for the classes, categories and protocols. Do this using the command below:
    - *class-dump-z Stocks > /tmp/Stockreversed.txt.*

The class-dump-z -H /var/mobile/<app-binary-to-be-reversed> -o /var/mobile/<outputdirectory>/ may also be used to get the headers in separate files.

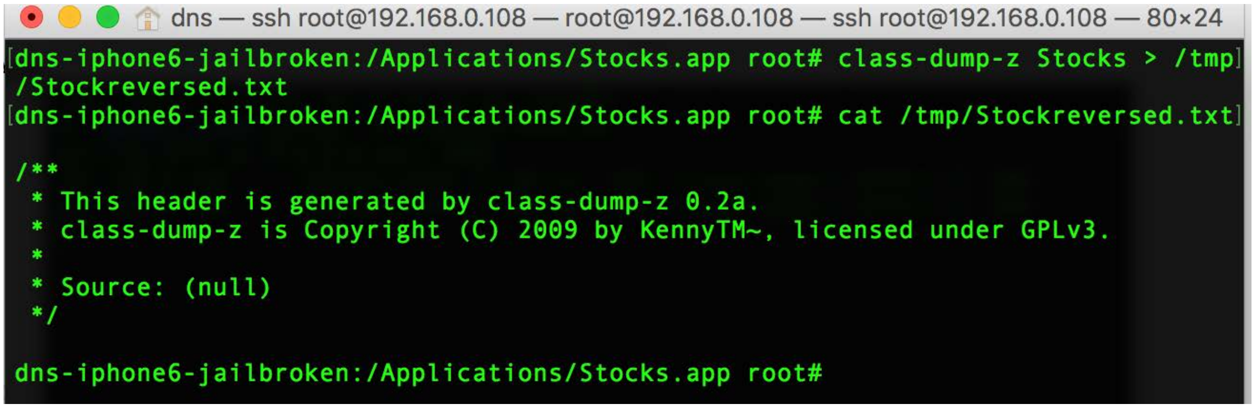
On arm64 devices the following error may occur:


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80×24

```
dns-iphone6-jailbroken:/Applications/Stocks.app root# class-dump-z Stocks > /tmp
/Stockreversed.txt
dyld: Library not loaded: /usr/lib/libpcre.0.dylib
  Referenced from: /usr/bin/class-dump-z
  Reason: image not found
Trace/BPT trap: 5
dns-iphone6-jailbroken:/Applications/Stocks.app root# 
```

If so, install “pcre” via Cydia to fix.

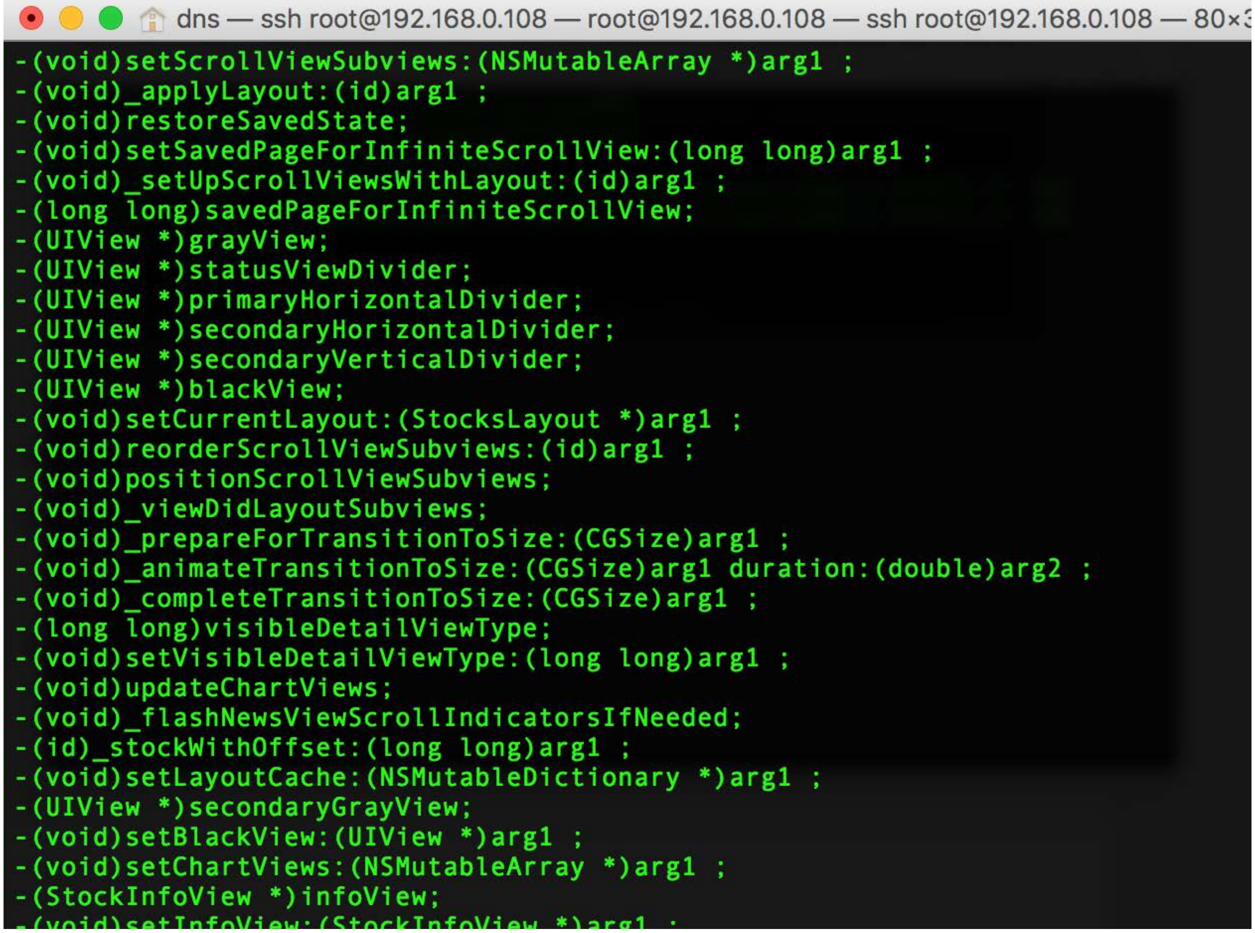
Note: Even after running the class-dump-z properly, a “null” error as shown in the following screenshot may occur.


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80×24

```
[dns-iphone6-jailbroken:/Applications/Stocks.app root# class-dump-z Stocks > /tmp]
/Stockreversed.txt
[dns-iphone6-jailbroken:/Applications/Stocks.app root# cat /tmp/Stockreversed.txt]

/**
 * This header is generated by class-dump-z 0.2a.
 * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 *
 * Source: (null)
 */

dns-iphone6-jailbroken:/Applications/Stocks.app root#
```

If so, install “classdump-dyld” instead of the default class-dump-z *(classdump-dyld Stocks > /tmp/Stockreversed.tx*t). Another option is to try *classdump-dyld -o /tmp/dump Stocks.*

4. The screenshot below shows the contents of the file “Stockreversed.txt.” It is readable and contains valuable information.


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 80×3

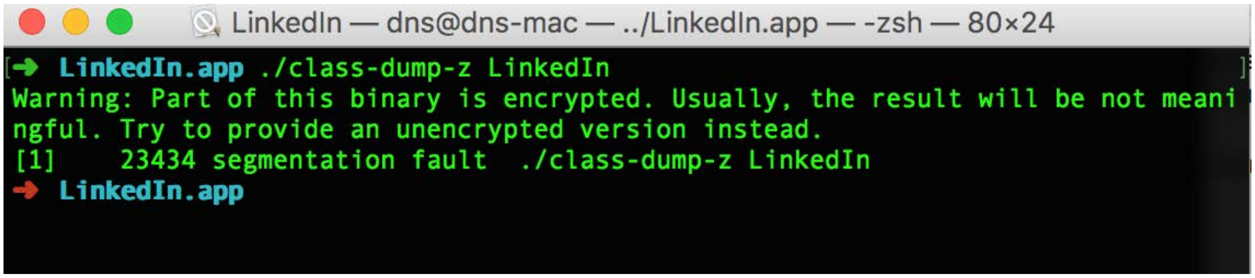
```
-(void)setScrollViewSubviews:(NSMutableArray *)arg1 ;
-(void)_applyLayout:(id)arg1 ;
-(void)restoreSavedState;
-(void)setSavedPageForInfiniteScrollView:(long long)arg1 ;
-(void)_setUpScrollViewsWithLayout:(id)arg1 ;
-(long long)savedPageForInfiniteScrollView;
-(UIView *)grayView;
-(UIView *)statusViewDivider;
-(UIView *)primaryHorizontalDivider;
-(UIView *)secondaryHorizontalDivider;
-(UIView *)secondaryVerticalDivider;
-(UIView *)blackView;
-(void)setCurrentLayout:(StocksLayout *)arg1 ;
-(void)reorderScrollViewSubviews:(id)arg1 ;
-(void)positionScrollViewSubviews;
-(void)_viewDidLayoutSubviews;
-(void)_prepareForTransitionToSize:(CGSize)arg1 ;
-(void)_animateTransitionToSize:(CGSize)arg1 duration:(double)arg2 ;
-(void)_completeTransitionToSize:(CGSize)arg1 ;
-(long long)visibleDetailViewType;
-(void)setVisibleDetailViewType:(long long)arg1 ;
-(void)updateChartViews;
-(void)_flashNewsViewScrollIndicatorsIfNeeded;
-(id)_stockWithOffset:(long long)arg1 ;
-(void)setLayoutCache:(NSMutableDictionary *)arg1 ;
-(UIView *)secondaryGrayView;
-(void)setBlackView:(UIView *)arg1 ;
-(void)setChartViews:(NSMutableArray *)arg1 ;
-(StockInfoView *)infoView;
-(void)setInfoView:(StockInfoView *)arg1 ;
```

The declarations of the classes and the protocols allow for debugging the application using GDB. Alternatively, it is possible to hook on to the functions present in the application via Cycript and try to change its behavior. This topic will be explained in greater detail later in the guide.

Note: The Mac OSX version of class dump for reverse engineering the application can also be used.

11. Decrypting iOS Applications (AppStore Binaries)

Sometimes it is necessary to test the applications that are live in the App Stores. If extracting the .IPA file from the App Store (using the methods mentioned in section 2, Acquiring iOS Binaries), and trying to decompile the application by means of tools like class-dump-z fails it’s likely due to Apple’s FairPlay DRM scheme to protect against piracy.



```
➜  LinkedIn.app ./class-dump-z LinkedIn
Warning: Part of this binary is encrypted. Usually, the result will be not meani
ngful. Try to provide an unencrypted version instead.
[1]    23434 segmentation fault  ./class-dump-z LinkedIn
➜  LinkedIn.app
```

Apps that are normally not encrypted include:

- Apps installed by default on the iOS device (located in /Applications/)
- Self-distributed apps
- Side-loaded apps

For these apps, there is no need to do anything to decrypt them. Binary analysis can be conducted on them “as-is.”

11.1 Manual Method

This is the most complicated and the most time-consuming decryption method.

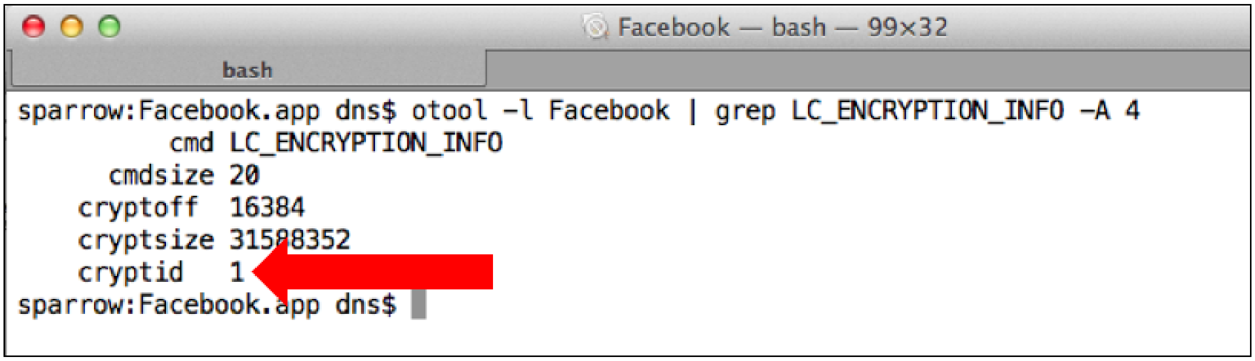
11.1.1 Using GDB

The easiest option is to use a jailbroken device that runs GDB properly without errors. If this is not an option, consider the suggestions in the next section and view LLDB usage. Use the GDB from *cydia.radare.org* repo. If there are issues using GDB, try using lipo.

Application used for Example: Facebook application from AppStore

Below are the steps to perform the decryption of the iOS binaries manually:

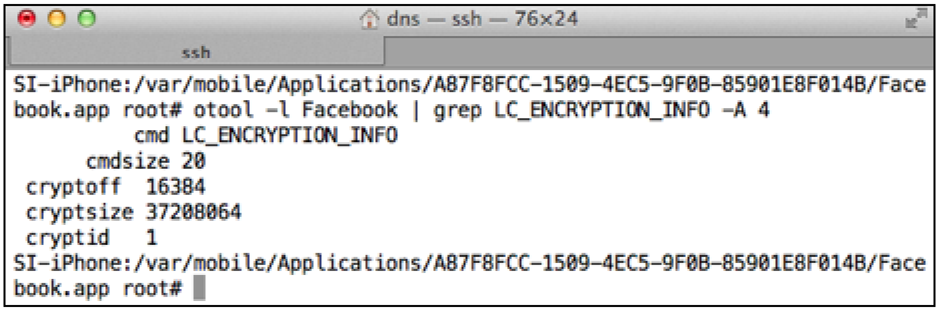
1. Launch the iOS application on a device and locate the encrypted segment by means of otool using the following syntax:
    - ○ *otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4*



```
sparrow:Facebook.app dns$ otool -l Facebook | grep LC_ENCRYPTION_INFO -A 4
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
    cryptoff  16384
    cryptsize 31588352
    cryptid   1
sparrow:Facebook.app dns$
```

The *cryptid*=1 indicates that the application is encrypted.

2. Locate the encrypted segment using the below command:
    - ○ *otool -l <app_name> | grep LC_ENCRYPTION_INFO -A 4*

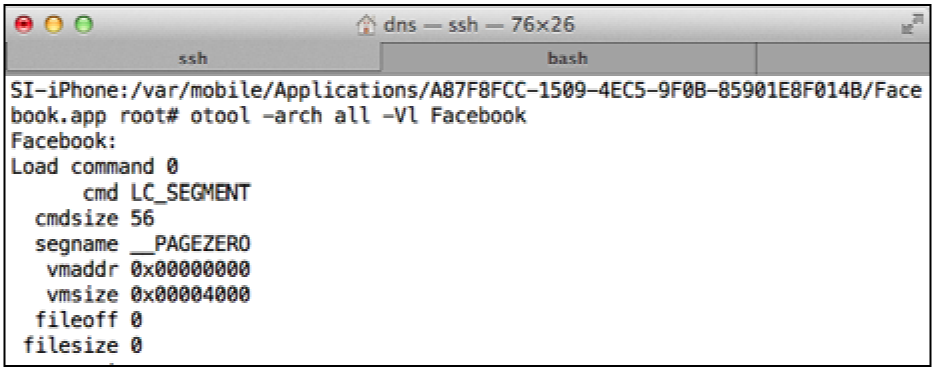


```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# otool -l Facebook | grep LC_ENCRYPTION_INFO -A 4
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
 cryptoff  16384
 cryptsize 37208064
 cryptid   1
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root#
```

The cryptoff field gives the start of the encrypted data (16384 bytes [0x4000] into the file)

The cryptsize field is the size of the encrypted segment (37208064, [0x237C000])

3. The command below gives the vmsize (the complete size of the segment)
    - ○ *otool –arch all -Vl <app_name>*



```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# otool -arch all -Vl Facebook
Facebook:
Load command 0
      cmd LC_SEGMENT
  cmdsize 56
  segname __PAGEZERO
   vmaddr 0x00000000
   vmsize 0x00004000
  fileoff 0
 filesize 0
```

Calculate the start and end addresses:

Start address = hex(cryptoff) + base address = 0x4000 + 0x4000 = 0x8000

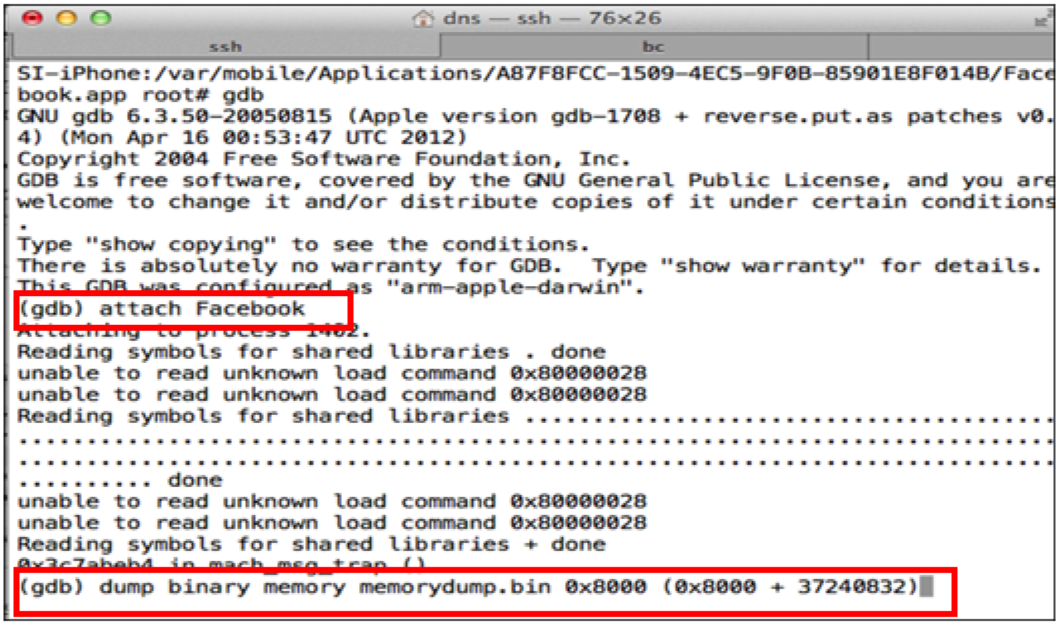
End address = Start address + cryptsize = 0x8000 + 0x237C000 = 0x2384000 (37240832)

Note: Base address is the same as vmsize or it can be found using “info sharedlibrary.”

4. Set a breakpoint using GDB
   - *gdb attach <app_name>*
5. Dump decrypted segment from memory and save to a file which will be used to patch the encrypted binary
   - *dump binary memory memorydump.bin <start_addr> <end_addr>*

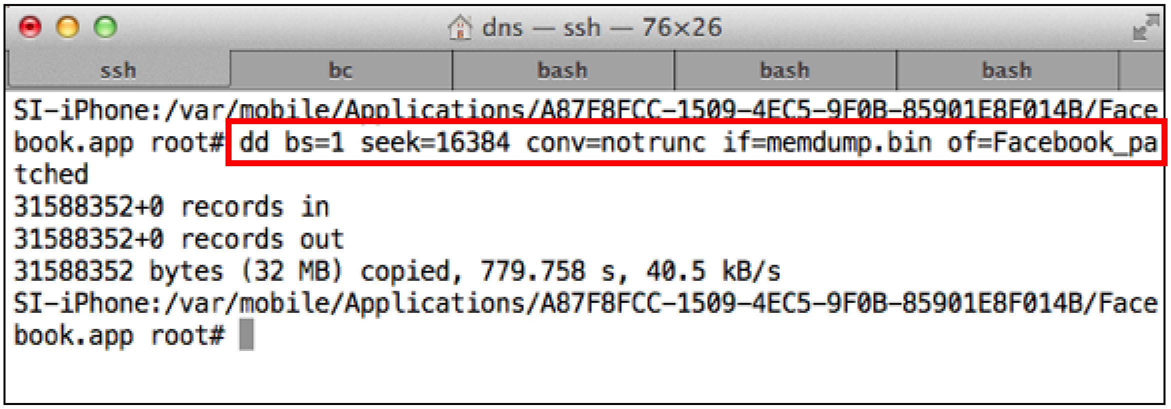
Note: START = hex(base)+hex(cryptoff)

END = DEC(START+hex(crypt size))



```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# gdb
GNU gdb 6.3.50-20050815 (Apple version gdb-1708 + reverse.put.as patches v0.
4) (Mon Apr 16 00:53:47 UTC 2012)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions
.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "arm-apple-darwin".
(gdb) attach Facebook
Attaching to process 1402.
Reading symbols for shared libraries . done
unable to read unknown load command 0x80000028
unable to read unknown load command 0x80000028
Reading symbols for shared libraries ................................
............................................................................
............................................................................
.......... done
unable to read unknown load command 0x80000028
unable to read unknown load command 0x80000028
Reading symbols for shared libraries + done
0x3c7abeb4 in mach_msg_trap ()
(gdb) dump binary memory memorydump.bin 0x8000 (0x8000 + 37240832)
```

6. Replace the encrypted data with decrypted data from memory. Copy the decrypted data into the binary using the below command:
   - *dd bs=1 seek=16384 conv=notrunc if=memdump.bin of=Facebook_patched*

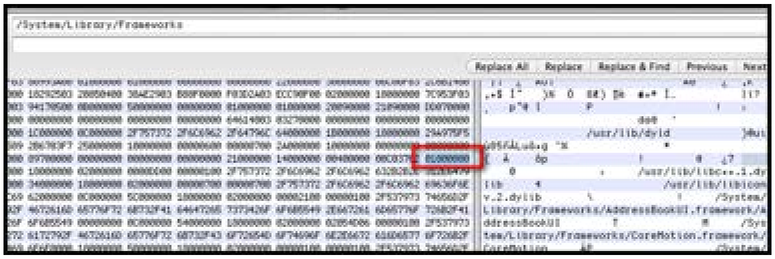


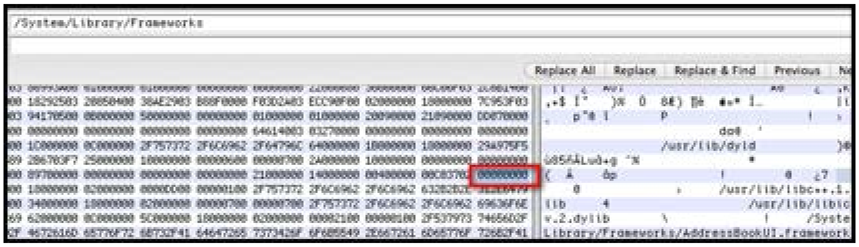
```
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# dd bs=1 seek=16384 conv=notrunc if=memdump.bin of=Facebook_pa
tched
31588352+0 records in
31588352+0 records out
31588352 bytes (32 MB) copied, 779.758 s, 40.5 kB/s
SI-iPhone:/var/mobile/Applications/A87F8FCC-1509-4EC5-9F0B-85901E8F014B/Face
book.app root# 
```

16384 is the cryptoff value found from the otool query.

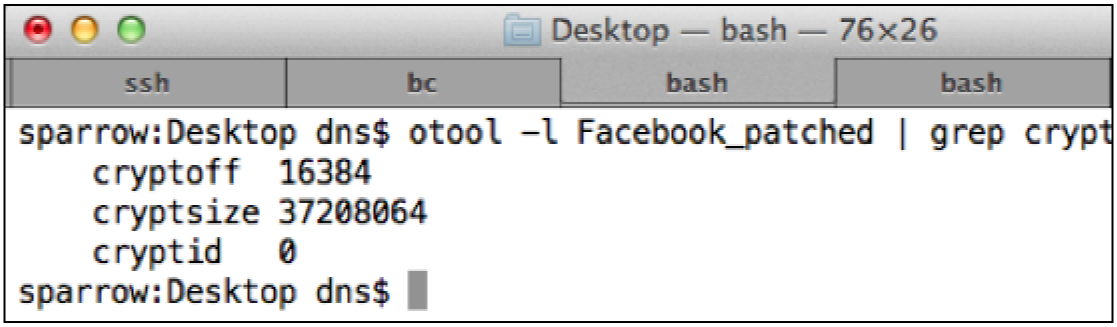
The binary will no longer be encrypted on the device.

7. To convert the binary into a decrypted binary is to patch cryptid to disable the encryption load command. Find the cryptid offset using MachOView. Use a hex editor to set cryptid field to 0x0.





8. Check Cryptid to show that encryption load is disabled (=0)



```
sparrow:Desktop dns$ otool -l Facebook_patched | grep crypt
    cryptoff  16384
    cryptsize 37208064
    cryptid   0
sparrow:Desktop dns$ 
```

NOTE: The base address can also be found using “info shared library” in GDB (Details mentioned in Mobile Application Hacker’s Handbook).

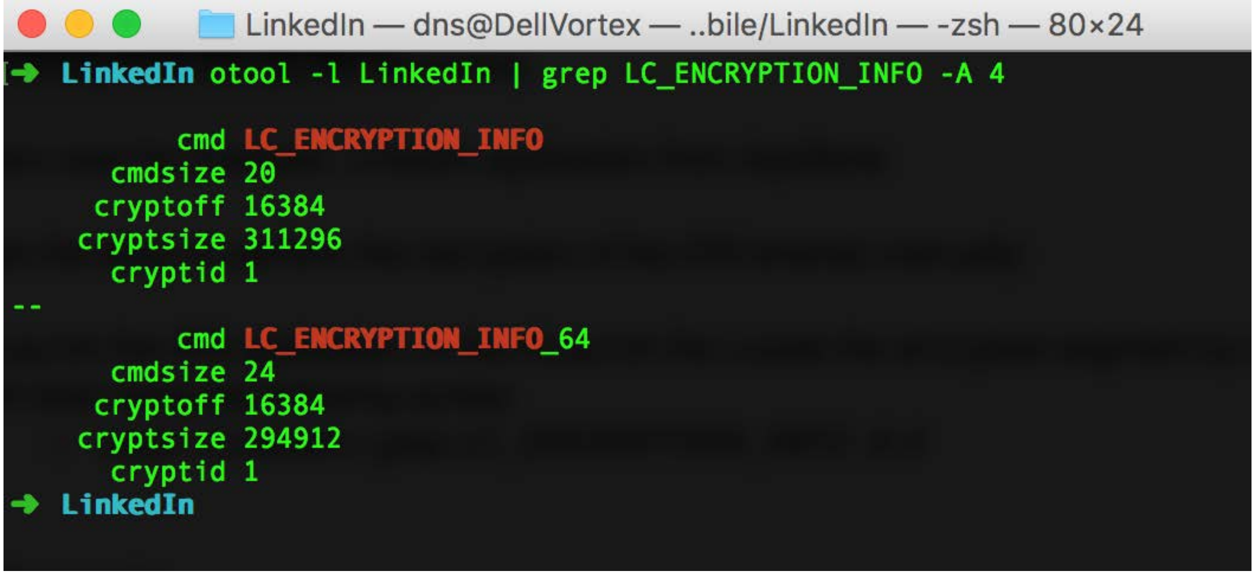
11.1.2 Using LLDB

Since Xcode 5, LLDB has been the standard for iOS debugging. It was created in close coordination with the LLVM compilers to replace GDB.

Application used for Example: LinkedIn application from AppStore

Below are the steps to perform the decryption of the iOS binaries manually:

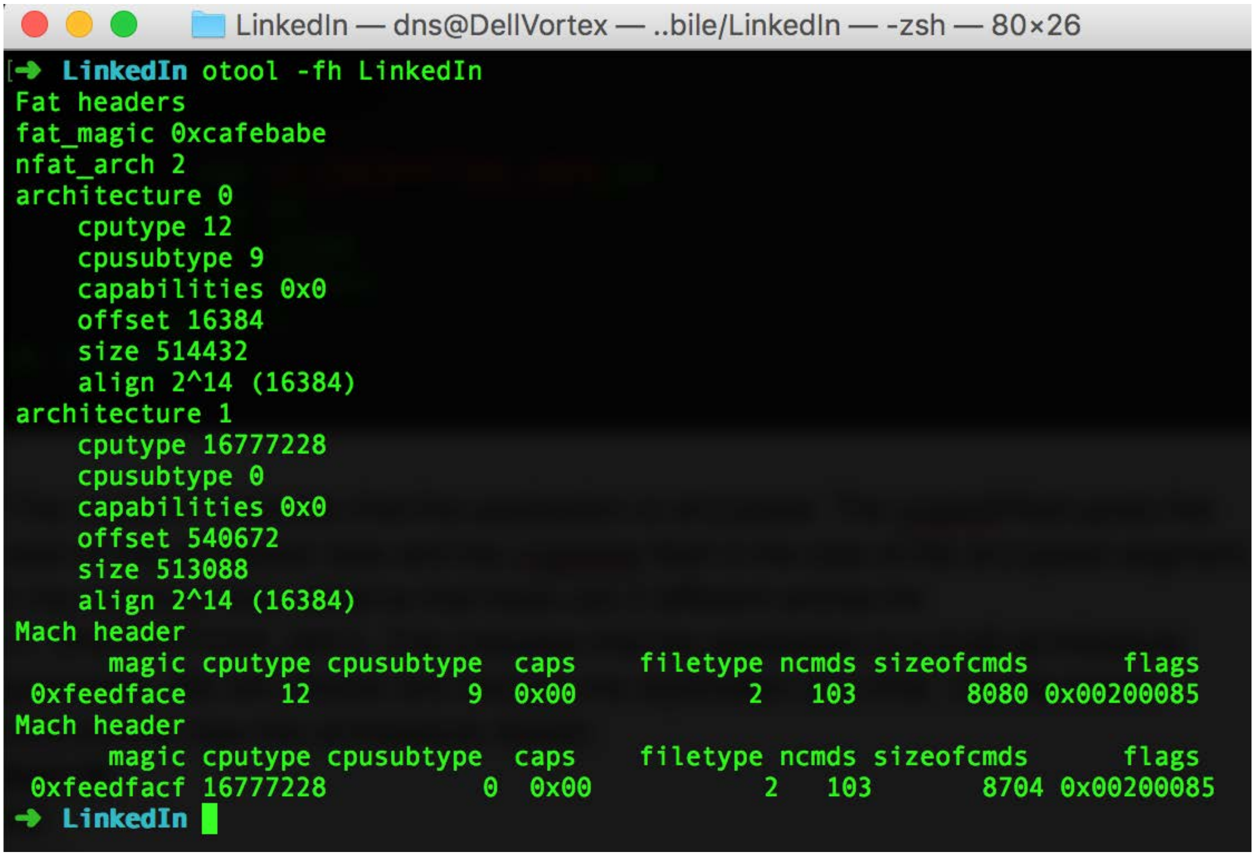
1. Launch the iOS application on the device and locate the encrypted segment by means of otool using the following syntax:
    - *otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4*



```
→ LinkedIn otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4

          cmd LC_ENCRYPTION_INFO
      cmdsize 20
     cryptoff 16384
    cryptsize 311296
      cryptid 1
--
          cmd LC_ENCRYPTION_INFO_64
      cmdsize 24
     cryptoff 16384
    cryptsize 294912
      cryptid 1
→ LinkedIn
```

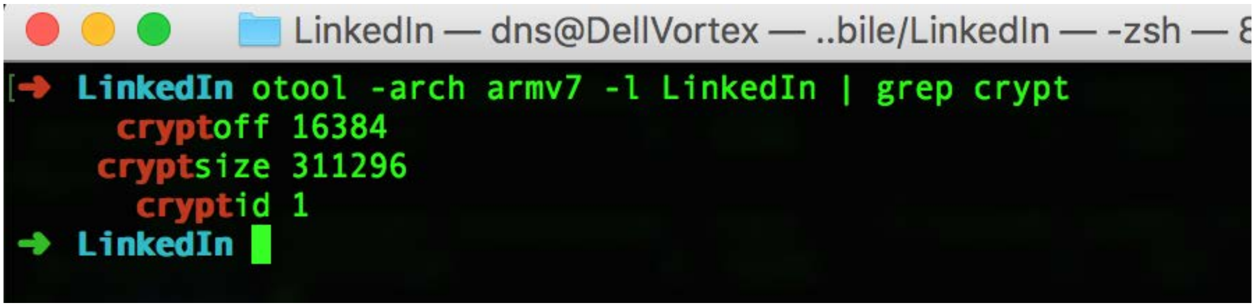
The *cryptid*=1 indicates that the application is encrypted. The *cryptoff* field gives the start of the encrypted data and the *cryptsize* field is the size of the encrypted segment.

2. In the previous step, note that there are 2 different entries for LC_ENCRYPTION_INFO. This indicates that the application is a multi-architecture application. Choose and decrypt one application at a time. Use the command below to view the architecture details:
    - *otool -fh LinkedIn*



```
➜ LinkedIn otool -fh LinkedIn
Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
    cputype 12
    cpusubtype 9
    capabilities 0x0
    offset 16384
    size 514432
    align 2^14 (16384)
architecture 1
    cputype 16777228
    cpusubtype 0
    capabilities 0x0
    offset 540672
    size 513088
    align 2^14 (16384)
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
 0xfeedface      12          9  0x00           2   103       8080 0x00200085
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
 0xfeedfacf 16777228          0  0x00           2   103       8704 0x00200085
➜ LinkedIn
```

3. The specific architecture can be chosen by using the *-arch* attribute.
   - *otool -arch armv7 -l LinkedIn | grep crypt*

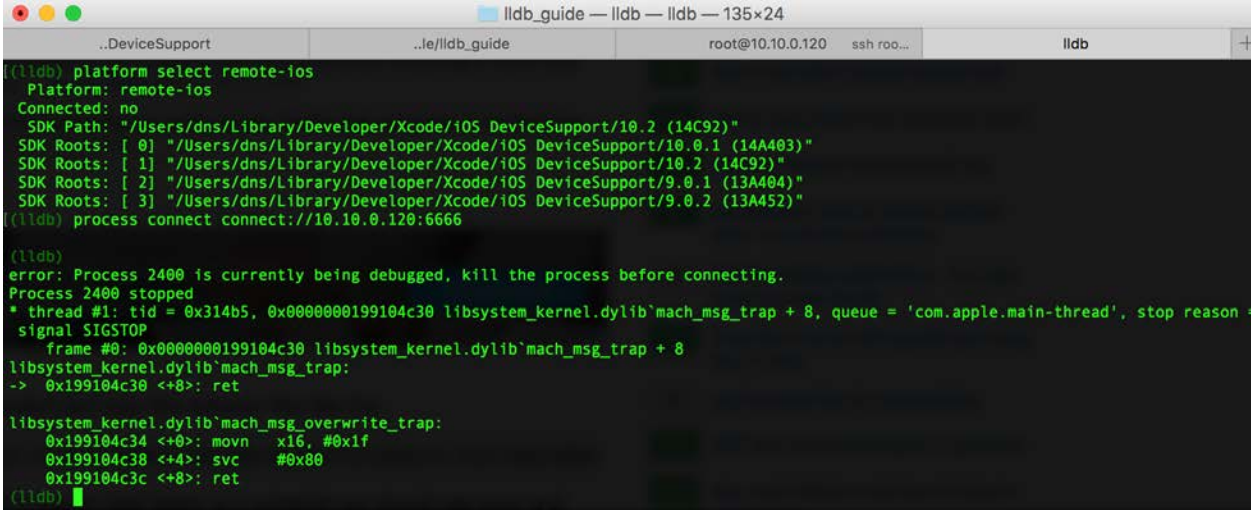


LinkedIn — dns@DellVortex — ..bile/LinkedIn — -zsh — 8

```
[➜  LinkedIn otool -arch armv7 -l LinkedIn | grep crypt
     cryptoff 16384
    cryptsize 311296
      cryptid 1
➜  LinkedIn █
```

The *cryptid*=1 indicates that the application is encrypted. The *cryptoff* field gives the start of the encrypted data (16384) and the *cryptsize* field is the size of the encrypted segment (311296).

4. On an iOS device, start a debug server and hook the LinkedIn application following the steps mentioned in the section titled “Debugging iOS application using LLDB.”



```
[(lldb) platform select remote-ios
  Platform: remote-ios
 Connected: no
  SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
 SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
 SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
 SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
 SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
[(lldb) process connect connect://10.10.0.120:6666

(lldb)
error: Process 2400 is currently being debugged, kill the process before connecting.
Process 2400 stopped
* thread #1: tid = 0x314b5, 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-thread', stop reason =
 signal SIGSTOP
    frame #0: 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
->  0x199104c30 <+8>: ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
    0x199104c34 <+0>: movn   x16, #0x1f
    0x199104c38 <+4>: svc    #0x80
    0x199104c3c <+8>: ret
(lldb) █
```

5. In the lldb interpreter enter the command “image list LinkedIn” to find the offset of the executable image in the memory.

   Note - If the application is compiled with ASLR(PIE) enabled, this image offset will be different each time the application is launched.

6. Dump decrypted segment from memory and save to a file which will be used to patch the encrypted binary using the below command:
   - *(lldb) memory read --force --outfile LinkedIn_memdump.bin --binary --count <cryptsize> <image offset>+<cryptoff>*

7. Replace the encrypted data with decrypted data from memory. Copy the decrypted data into the binary using the command below:
   - *dd bs=1 seek=<cryptoff> conv=notrunc if=LinkedIn_memdump.bin of=LinkedIn_patched*

     Where seekvalue= <offset from “otool -fh” + cryptoff from “otool -arch armv7 -l”>

   This output binary is no longer encrypted on the device.

8. Patch the cryptid to disable the encryption load command. This can be easily modified by means of MachOView. Download from: https://sourceforge.net/projects/machoview/. Open the LinkedIn patched binary in MachOView. Find "cryptid". In the UI, double click on "Data" in the "Crypt ID" for cryptid=1 and set it to zero. Save the binary.
9. To verify, if the cryptid change is reflected, view the value using the below command:
   - *otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4*
10. Using the decrypted binary, it is possible reverse it via tools like class-dump-z

NOTE: Recommended guides on manually decrypting apps from the AppStore can be found here:

- *http://codedigging.com/blog/2016-03-01-decrypting-apps-from-appstore/*
- *http://codedigging.com/blog/2016-04-27-debugging-ios-binaries-with-lldb/*

Or refer to Chapter 6 of "iOS Application Security" book by David Thiel (https://www.nostarch.com/iossecurity)

11.2 Automated Method

Because the manual method is time consuming, consider one of the following automated tools to help decrypt iOS binaries for binary analysis.

11.2.1 Using dump decrypted

Dump decrypted works by injecting a constructor via a dynamic linker into the application. This constructor extracts the decrypted segment in very much the same manner as the manual method.

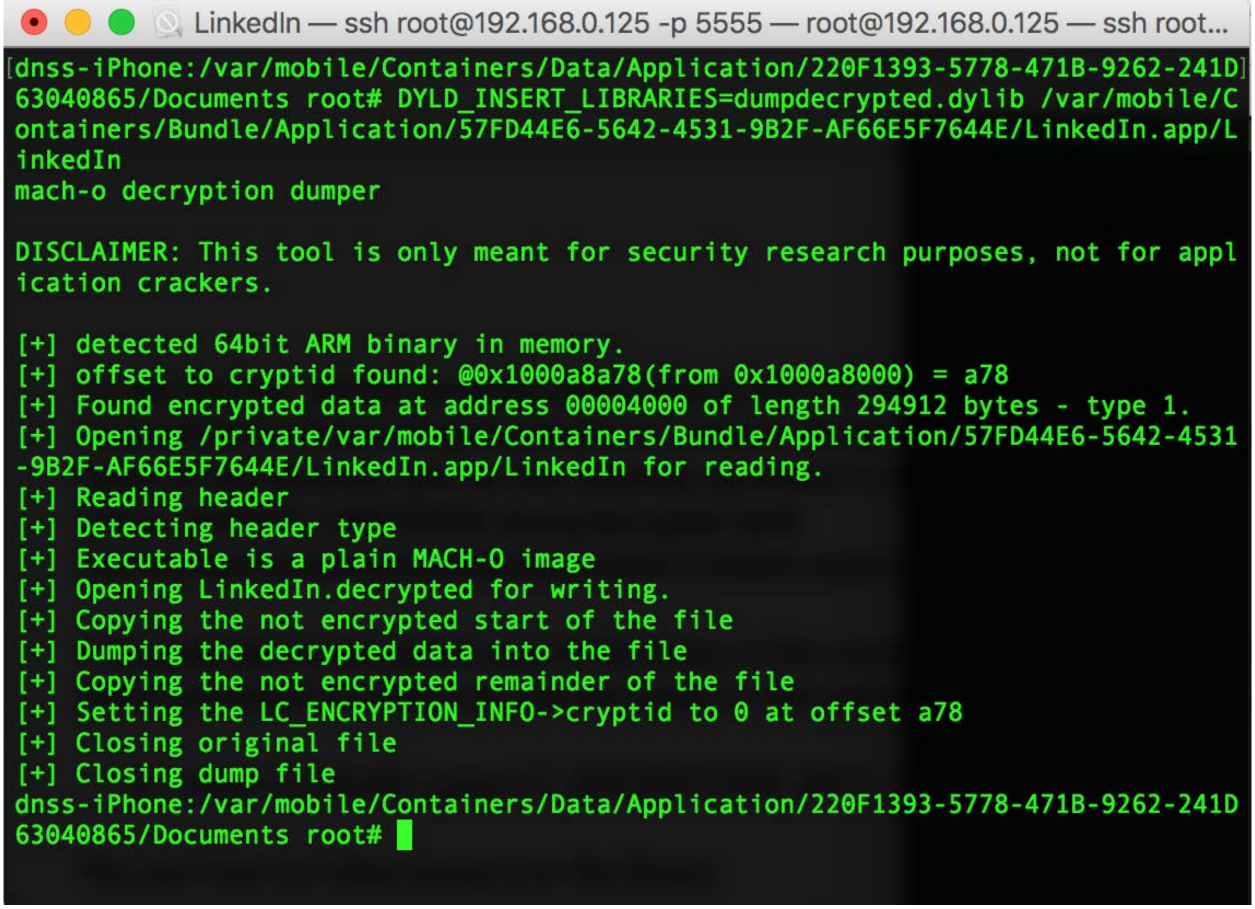
The link to the tool is here: https://github.com/stefanesser/dumpdecrypted. Use "make" to build the required .dylib.

Application used for Example: LinkedIn application from AppStore

To use the tool, upload dumpdecrypted.dylib to the iOS Documents folder on /var/mobile/Containers/Data/Application/<Linkedin-GUID>/Documents device. The Linkedin-GUID can be found by running 'ps aux|grep -i <appname>' and looking at the path in the last column of the display.

Change working folder to that directory and run the following command in the application sandbox:

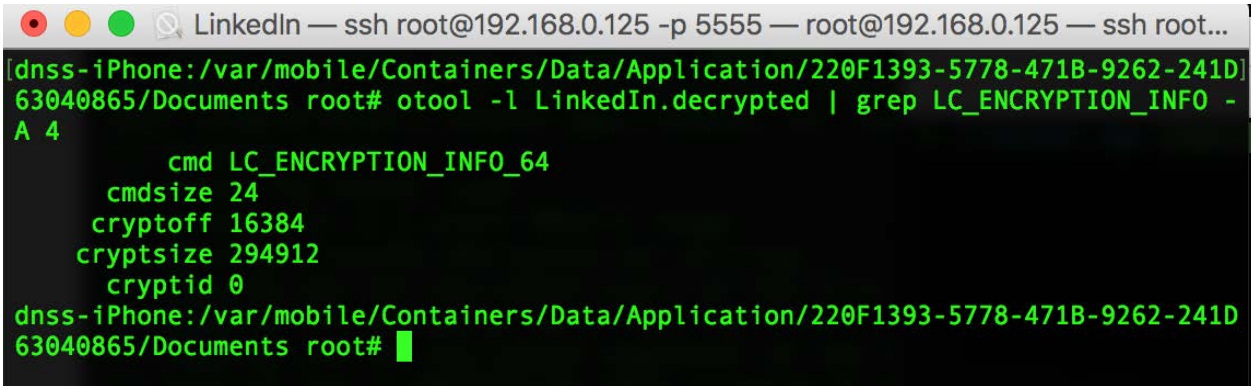
DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib
/var/mobile/Containers/Bundle/Application/LinkedIn.app/LinkedIn


LinkedIn — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root...

```
[dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D]
63040865/Documents root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/C
ontainers/Bundle/Application/57FD44E6-5642-4531-9B2F-AF66E5F7644E/LinkedIn.app/L
inkedIn
mach-o decryption dumper

DISCLAIMER: This tool is only meant for security research purposes, not for appl
ication crackers.

[+] detected 64bit ARM binary in memory.
[+] offset to cryptid found: @0x1000a8a78(from 0x1000a8000) = a78
[+] Found encrypted data at address 00004000 of length 294912 bytes - type 1.
[+] Opening /private/var/mobile/Containers/Bundle/Application/57FD44E6-5642-4531
-9B2F-AF66E5F7644E/LinkedIn.app/LinkedIn for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a plain MACH-O image
[+] Opening LinkedIn.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Setting the LC_ENCRYPTION_INFO->cryptid to 0 at offset a78
[+] Closing original file
[+] Closing dump file
dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D
63040865/Documents root# 
```

This generates a decrypted copy of the binary “LinkedIn.decrypted” in the current working directory. Now, run the command below to make sure that the binary is decrypted by looking at the value of cryptid.

- *otool -l LinkedIn.decrypted | grep LC_ENCRYPTION_INFO -A 4*



LinkedIn — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root...

```
[dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D]
63040865/Documents root# otool -l LinkedIn.decrypted | grep LC_ENCRYPTION_INFO -
A 4
          cmd LC_ENCRYPTION_INFO_64
      cmdsize 24
     cryptoff 16384
    cryptsize 294912
      cryptid 0
dnss-iPhone:/var/mobile/Containers/Data/Application/220F1393-5778-471B-9262-241D
63040865/Documents root# 
```

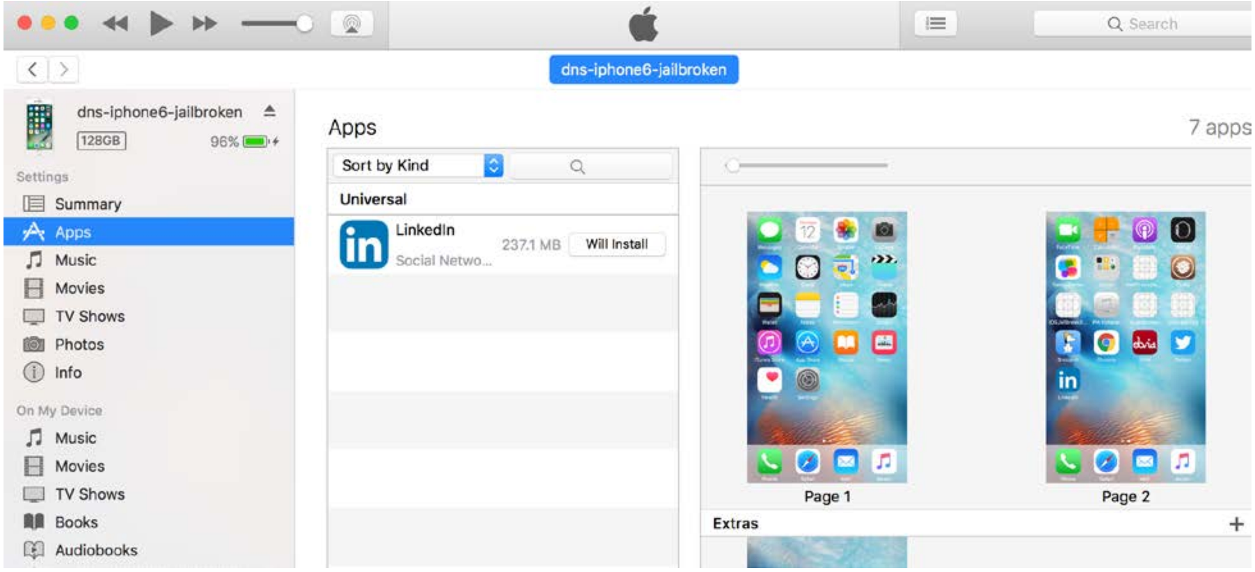
Cryptid of 0 indicates that the application has been decrypted.

11.2.2 Using Clutch

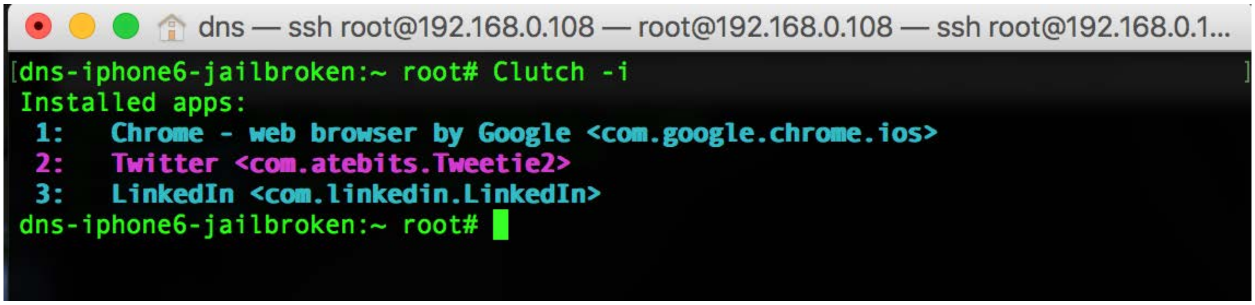
This is the easiest way to decrypt encrypted iOS binaries. Either download the latest version of Clutch from https://github.com/KJCracks/Clutch/releases and move it the /bin/ folder on the iOS device, or install Clutch from the cydia repo *http://cydia.iphonecake.com*.

Application used for Example: LinkedIn application from AppStore

1. Download the LinkedIn application using AppStore on the MacBook. Install and Sync the LinkedIn application to the iOS device.



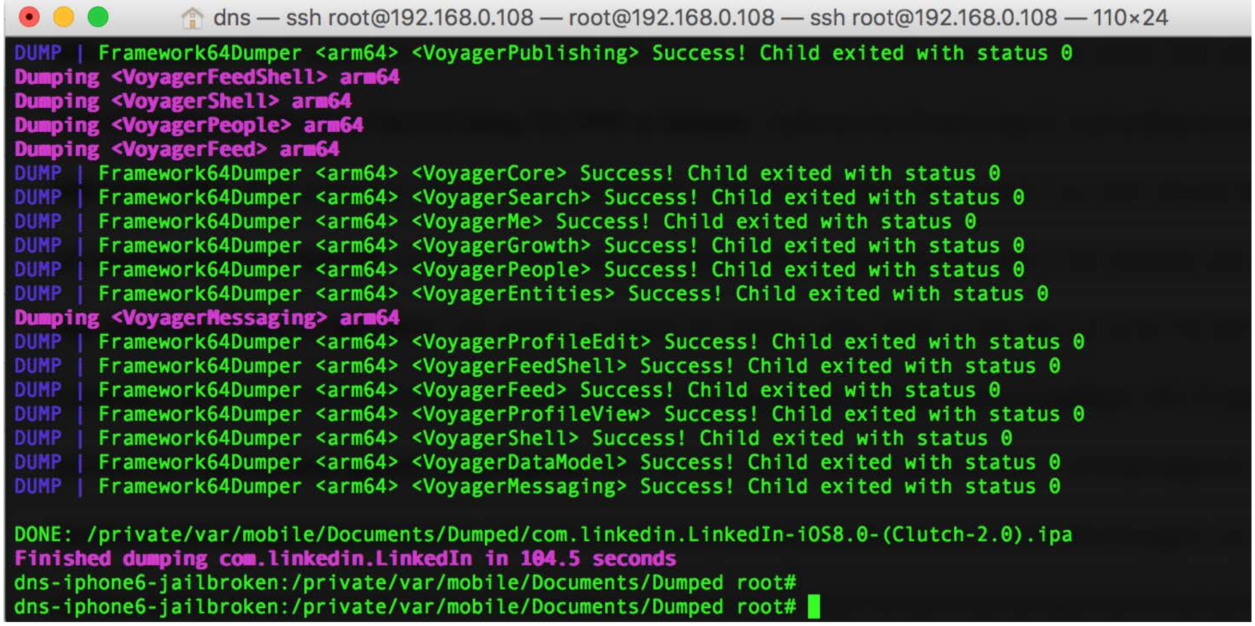
2. SSH into the iOS device.
3. Use the command below to list all the installed applications on the iOS device
    - *Clutch -i*


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.1...

```
dns-iphone6-jailbroken:~ root# Clutch -i
Installed apps:
  1:    Chrome - web browser by Google <com.google.chrome.ios>
  2:    Twitter <com.atebits.Tweetie2>
  3:    LinkedIn <com.linkedin.LinkedIn>
dns-iphone6-jailbroken:~ root#
```

4. Use the command below to decrypt the application.
    - *Clutch -d <app-id from previous command>*

   To decrypt LinkedIn type
    - *Clutch -d 3*


dns — ssh root@192.168.0.108 — root@192.168.0.108 — ssh root@192.168.0.108 — 110×24

```
DUMP | Framework64Dumper <arm64> <VoyagerPublishing> Success! Child exited with status 0
Dumping <VoyagerFeedShell> arm64
Dumping <VoyagerShell> arm64
Dumping <VoyagerPeople> arm64
Dumping <VoyagerFeed> arm64
DUMP | Framework64Dumper <arm64> <VoyagerCore> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerSearch> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerMe> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerGrowth> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerPeople> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerEntities> Success! Child exited with status 0
Dumping <VoyagerMessaging> arm64
DUMP | Framework64Dumper <arm64> <VoyagerProfileEdit> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerFeedShell> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerFeed> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerProfileView> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerShell> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerDataModel> Success! Child exited with status 0
DUMP | Framework64Dumper <arm64> <VoyagerMessaging> Success! Child exited with status 0

DONE: /private/var/mobile/Documents/Dumped/com.linkedin.LinkedIn-iOS8.0-(Clutch-2.0).ipa
Finished dumping com.linkedin.LinkedIn in 104.5 seconds
dns-iphone6-jailbroken:/private/var/mobile/Documents/Dumped root#
dns-iphone6-jailbroken:/private/var/mobile/Documents/Dumped root#
```

The decrypted application can be found in the form of an IPA file on the same device at /private/var/mobile/Documents/Dumped/

5. Unzip the decrypted application and run the command below to make sure that the binary is decrypted by looking at the value of cryptid.
   - *otool -l LinkedIn | grep LC_ENCRYPTION_INFO -A 4*
6. It is now possible to run class-dump-z on the binary.

Note:

- The message, “Segmentation fault: 11” issues with Clutch make use of “ulimit -n 2048”, indicates that the number of allowed open file handles per process is increased to solve the issue.

- Use “Clutch -f” to clear Clutch cache.
- Sometimes, the class-dump-z gives nil output on the device. If this happens, run Clutch, pull the ipa file off the device, and run Mac version of class-dump on it.
- The error message, “Killed: 9”, means there is a signing error. ldid -S <binary> should fix it.

12. iOS Application Debugging - Runtime Manipulation

Runtime (Dynamic) Analysis is the ability to manipulate apps while they are running. This is done by enabling debugging and runtime tracing functionality. With the debugging and tracing functionality enabled, an attacker can manipulate how the application behaves during runtime.

Runtime Manipulation allows the attacker to:

- Execute hidden functionality which should not be accessible
- Discover weak/missing encryption
- Bypass client-side restrictions
- Unlock additional features and premium content
- Dump copyright-protected content

12.1 Cycript on Jailbroken Device

Cycript is the most commonly used tool for performing debugging or runtime manipulation on iOS applications. A detailed guide on how to use Cycript can be found here: http://iphonedevwiki.net/index.php/Cycript_Tricks.

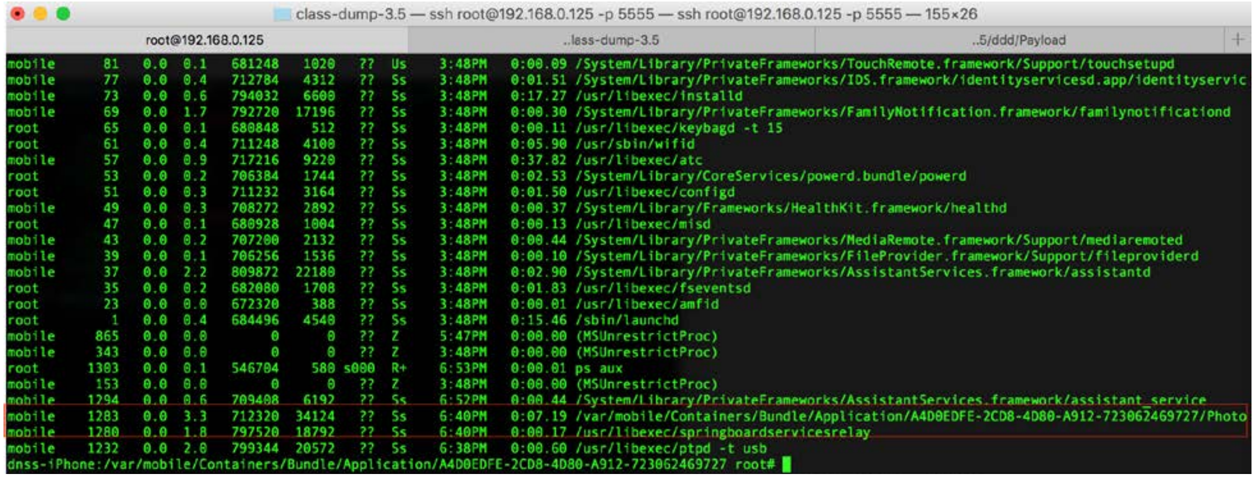
12.1.1 Using Cycript to Invoke Internal Methods

Application used for Example: Photo Vault application version 2.5/3.1 from:

- https://drive.google.com/open?id=0B0b4lUTjHfRKWTRlMW1WUy14bkE
- https://drive.google.com/open?id=0B0b4lUTjHfRKN1l3Mk1hSDNBU0k

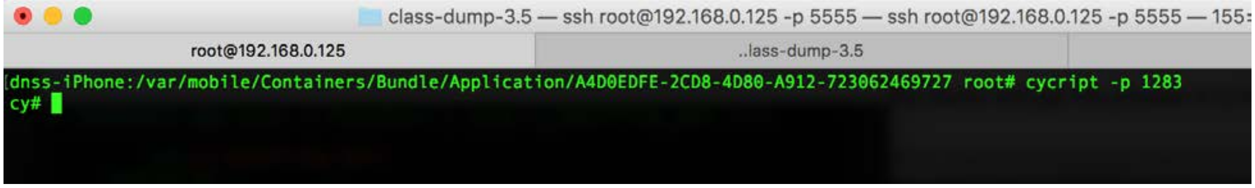
The steps below are for version 2.5.

1. Launch the Photo Vault application on the device. When prompted for the PIN, set it as “9876.” SSH into the iOS device, and get the process id of the application using the command “ps aux”.


class-dump-3.5 — ssh root@192.168.0.125 -p 5555 — ssh root@192.168.0.125 -p 5555 — 155×26
root@192.168.0.125
..lass-dump-3.5
..5/ddd/Payload

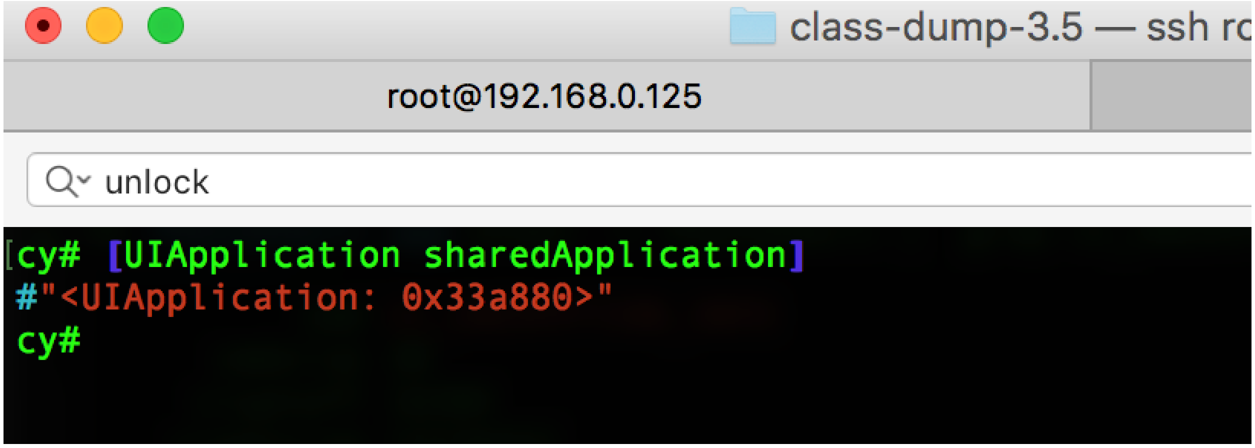
```
mobile      81   0.0  0.1   681248   1020   ??  Us    3:48PM   0:00.09 /System/Library/PrivateFrameworks/TouchRemote.framework/Support/touchsetupd
mobile      77   0.0  0.4   712784   4312   ??  Ss    3:48PM   0:01.51 /System/Library/PrivateFrameworks/IDS.framework/identityservicesd.app/identityservic
mobile      73   0.0  0.6   794032   6600   ??  Ss    3:48PM   0:17.27 /usr/libexec/installd
mobile      69   0.0  1.7   792720  17196   ??  Ss    3:48PM   0:00.30 /System/Library/PrivateFrameworks/FamilyNotification.framework/familynotificationd
root        65   0.0  0.1   680848    512   ??  Ss    3:48PM   0:00.11 /usr/libexec/keybagd -t 15
root        61   0.0  0.4   711248   4100   ??  Ss    3:48PM   0:05.90 /usr/sbin/wifid
mobile      57   0.0  0.9   717216   9220   ??  Ss    3:48PM   0:37.82 /usr/libexec/atc
root        53   0.0  0.2   706384   1744   ??  Ss    3:48PM   0:02.53 /System/Library/CoreServices/powerd.bundle/powerd
root        51   0.0  0.3   711232   3164   ??  Ss    3:48PM   0:01.50 /usr/libexec/configd
mobile      49   0.0  0.3   708272   2892   ??  Ss    3:48PM   0:00.37 /System/Library/Frameworks/HealthKit.framework/healthd
root        47   0.0  0.1   680928   1004   ??  Ss    3:48PM   0:00.13 /usr/libexec/misd
mobile      43   0.0  0.2   707200   2132   ??  Ss    3:48PM   0:00.44 /System/Library/PrivateFrameworks/MediaRemote.framework/Support/mediaremoted
mobile      39   0.0  0.1   706256   1536   ??  Ss    3:48PM   0:00.10 /System/Library/PrivateFrameworks/FileProvider.framework/Support/fileproviderd
mobile      37   0.0  2.2   809872  22180   ??  Ss    3:48PM   0:02.90 /System/Library/PrivateFrameworks/AssistantServices.framework/assistantd
root        35   0.0  0.2   682080   1708   ??  Ss    3:48PM   0:01.83 /usr/libexec/fseventsd
root        23   0.0  0.0   672320    388   ??  Ss    3:48PM   0:00.01 /usr/libexec/amfid
root         1   0.0  0.4   684496   4540   ??  Ss    3:48PM   0:15.46 /sbin/launchd
mobile     865   0.0  0.0        0      0   ??  Z     5:47PM   0:00.00 (MSUnrestrictProc)
mobile     343   0.0  0.0        0      0   ??  Z     3:48PM   0:00.00 (MSUnrestrictProc)
root      1303   0.0  0.1   546704    580 s000  R+    6:53PM   0:00.01 ps aux
mobile     153   0.0  0.0        0      0   ??  Z     3:48PM   0:00.00 (MSUnrestrictProc)
mobile    1294   0.0  0.6   709408   6192   ??  Ss    6:52PM   0:00.44 /System/Library/PrivateFrameworks/AssistantServices.framework/assistant_service
mobile    1283   0.0  3.3   712320  34124   ??  Ss    6:40PM   0:07.19 /var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727/Photo
mobile    1280   0.0  1.8   797520  18792   ??  Ss    6:40PM   0:00.17 /usr/libexec/springboardservicesrelay
mobile    1232   0.0  2.0   799344  20572   ??  Ss    6:38PM   0:00.60 /usr/libexec/ptpd -t usb
dnss-iPhone:/var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727 root# 
```

2. When prompted with the Cycript interpreter, use “cycript -p <process-id>” to hook on to the application.


class-dump-3.5 — ssh root@192.168.0.125 -p 5555 — ssh root@192.168.0.125 -p 5555 — 155:
root@192.168.0.125
..lass-dump-3.5

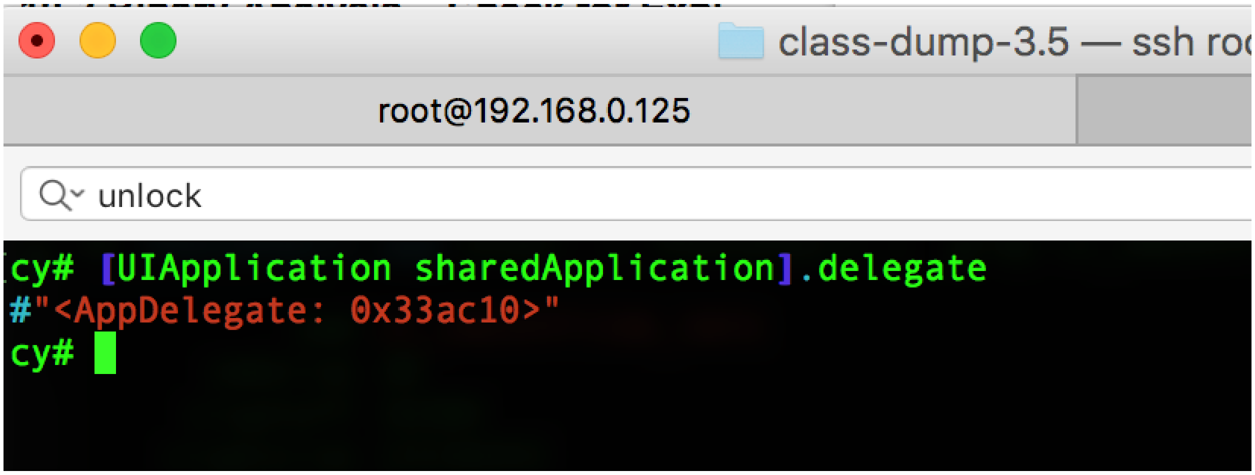
```
[dnss-iPhone:/var/mobile/Containers/Bundle/Application/A4D0EDFE-2CD8-4D80-A912-723062469727 root# cycript -p 1283
 cy# 
```

3. On the lock screen where the application requests the passcode, get the instance of the application using the command below:
   - *[UIApplication sharedApplication]*



```
cy# [UIApplication sharedApplication]
#"<UIApplication: 0x33a880>"
cy#
```

4. Get the delegate class for the application using the command below:
   - *[UIApplication sharedApplication].delegate*



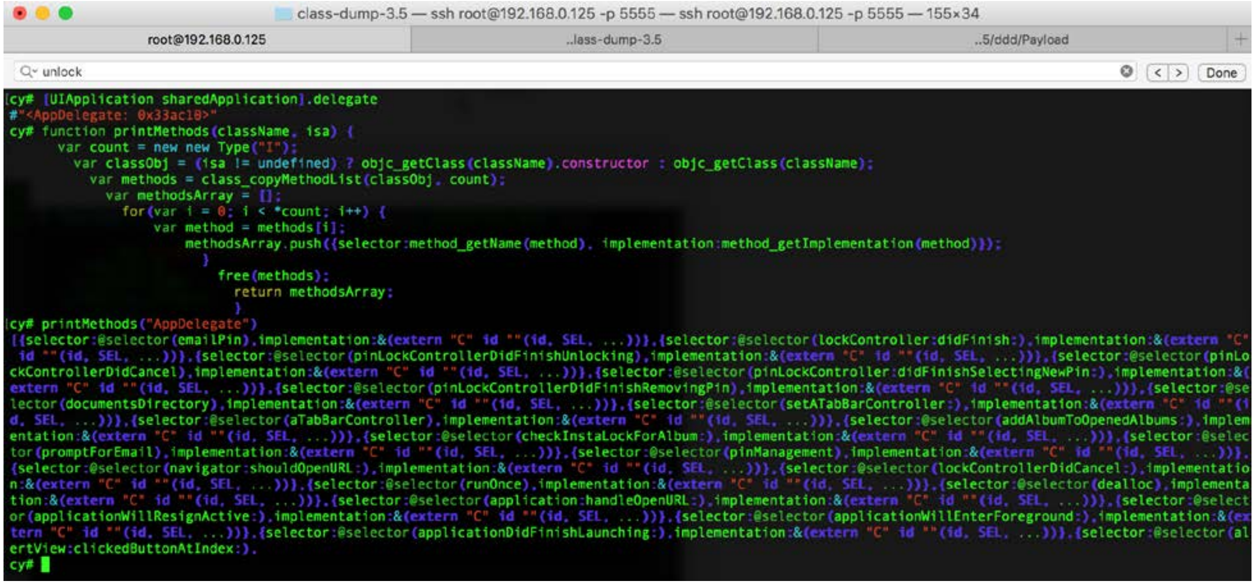
```
cy# [UIApplication sharedApplication].delegate
#"<AppDelegate: 0x33ac10>"
cy#
```

5. Use the function below to get the methods of the delegate class found in the previous step.

```
function printMethods(className, isa) {
  var count = new new Type("I");
  var classObj = (isa != undefined) ? objc_getClass(className).constructor :
objc_getClass(className);
  var methods = class_copyMethodList(classObj, count);
```

```
 var methodsArray = [];
 for(var i = 0; i < *count; i++) {
   var method = methods[i];
   methodsArray.push({selector:method_getName(method),
implementation:method_getImplementation(method)});
 }
 free(methods);
 return methodsArray;
}
```

Then, enter printMethods("AppDelegate") to print the complete list of possible methods for that screen.


class-dump-3.5 — ssh root@192.168.0.125 -p 5555 — ssh root@192.168.0.125 -p 5555 — 155×34
root@192.168.0.125
..lass-dump-3.5
..5/ddd/Payload
unlock
Done

```
cy# [UIApplication sharedApplication].delegate
#"<AppDelegate: 0x33ac10>"
cy# function printMethods(className, isa) {
      var count = new new Type("I");
        var classObj = (isa != undefined) ? objc_getClass(className).constructor : objc_getClass(className);
          var methods = class_copyMethodList(classObj, count);
            var methodsArray = [];
              for(var i = 0; i < *count; i++) {
                  var method = methods[i];
                    methodsArray.push({selector:method_getName(method), implementation:method_getImplementation(method)});
                      }
                        free(methods);
                          return methodsArray;
                          }
cy# printMethods("AppDelegate")
[{selector:@selector(emailPin),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(lockController:didFinish:),implementation:&(extern "C"
 id ""(id, SEL, ...))},{selector:@selector(pinLockControllerDidFinishUnlocking),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(pinLo
ckControllerDidCancel),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(pinLockController:didFinishSelectingNewPin:),implementation:&(
extern "C" id ""(id, SEL, ...))},{selector:@selector(pinLockControllerDidFinishRemovingPin),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@se
lector(documentsDirectory),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(setATabBarController:),implementation:&(extern "C" id ""(i
d, SEL, ...))},{selector:@selector(aTabBarController),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(addAlbumToOpenedAlbums:),implem
entation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(checkInstaLockForAlbum:),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selec
tor(promptForEmail),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(pinManagement),implementation:&(extern "C" id ""(id, SEL, ...))},
{selector:@selector(navigator:shouldOpenURL:),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(lockControllerDidCancel:),implementatio
n:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(runOnce),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(dealloc),implementa
tion:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(application:handleOpenURL:),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@select
or(applicationWillResignActive:),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(applicationWillEnterForeground:),implementation:&(ex
tern "C" id ""(id, SEL, ...))},{selector:@selector(applicationDidFinishLaunching:),implementation:&(extern "C" id ""(id, SEL, ...))},{selector:@selector(al
ertView:clickedButtonAtIndex:)]
cy# 
```

NOTE: This function list can also be found using class-dump-z. Search the class-dump output for AppDelegate and look for methods below the *AppDelegate @interface* in the output.

6. From the output of the previous step, note a method called "pinLockControllerDidFinishUnlocking." Class-dump-z output will reveal that this function does not take any arguments and can be called directly.
   Use the command below to call the function directly:
   - *[UIApp.delegate pinLockControllerDidFinishUnlocking]*

   Note that that the lock screen is bypassed.

A similar approach can be used for bypassing Jailbreak Detection in iOS applications.

It may be difficult to find the details of the current ViewController (current screen) that the user is on. Use the instructions below to locate the name of the current ViewController on the iOS device.

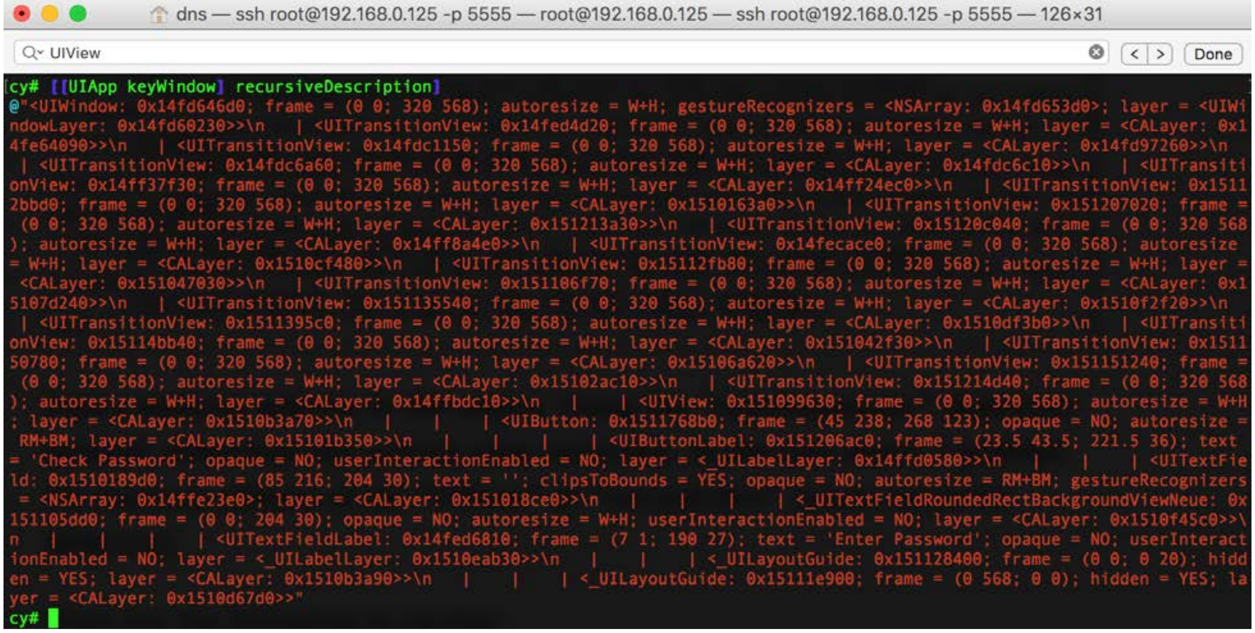
Application used for Example: CycriptDemoDNS application from the below mentioned link:

- https://drive.google.com/open?id=0B0b4lUTjHfRKXy1pU29oZmdSUjg

a. Launch the application and navigate to the appropriate ViewController.

b. Enter the command below for keyWindow (the current window accepting user touch events) details:

   - *[[UIApp keyWindow] recursiveDescription]*

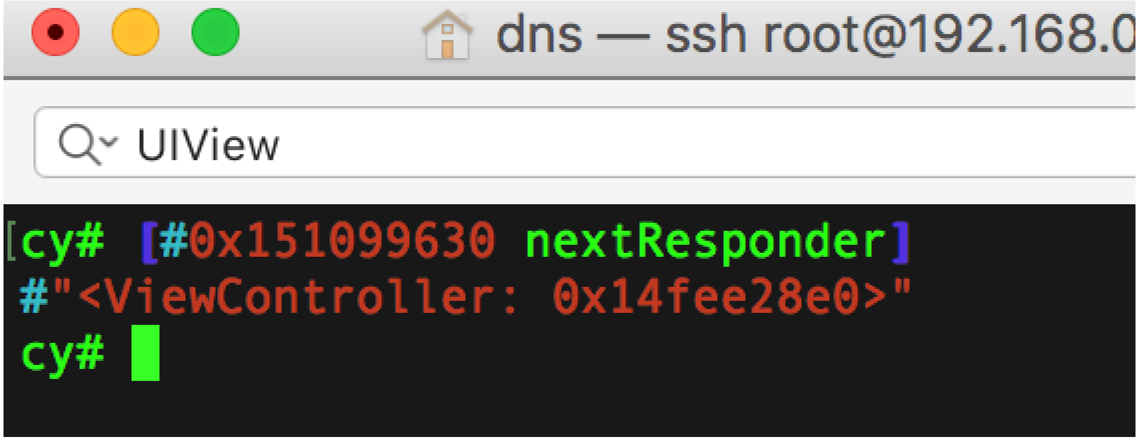


dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126×31
UIView
< >
Done

```
[cy# [[UIApp keyWindow] recursiveDescription]
@"<UIWindow: 0x14fd646d0; frame = (0 0; 320 568); autoresize = W+H; gestureRecognizers = <NSArray: 0x14fd653d0>; layer = <UIWi
ndowLayer: 0x14fd60230>>\n   | <UITransitionView: 0x14fed4d20; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1
4fe64090>>\n   | <UITransitionView: 0x14fdc1150; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14fd97260>>\n
 | <UITransitionView: 0x14fdc6a60; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14fdc6c10>>\n   | <UITransiti
onView: 0x14ff37f30; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x14ff24ec0>>\n   | <UITransitionView: 0x1511
2bbd0; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510163a0>>\n   | <UITransitionView: 0x151207020; frame =
 (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x151213a30>>\n   | <UITransitionView: 0x15120c040; frame = (0 0; 320 568
); autoresize = W+H; layer = <CALayer: 0x14ff8a4e0>>\n   | <UITransitionView: 0x14fecace0; frame = (0 0; 320 568); autoresize
= W+H; layer = <CALayer: 0x1510cf480>>\n   | <UITransitionView: 0x15112fb80; frame = (0 0; 320 568); autoresize = W+H; layer =
 <CALayer: 0x151047030>>\n   | <UITransitionView: 0x151106f70; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1
5107d240>>\n   | <UITransitionView: 0x151135540; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510f2f20>>\n
 | <UITransitionView: 0x1511395c0; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x1510df3b0>>\n   | <UITransiti
onView: 0x15114bb40; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x151042f30>>\n   | <UITransitionView: 0x1511
50780; frame = (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x15106a620>>\n   | <UITransitionView: 0x151151240; frame =
 (0 0; 320 568); autoresize = W+H; layer = <CALayer: 0x15102ac10>>\n   | <UITransitionView: 0x151214d40; frame = (0 0; 320 568
); autoresize = W+H; layer = <CALayer: 0x14ffbdc10>>\n   |    | <UIView: 0x151099630; frame = (0 0; 320 568); autoresize = W+H
; layer = <CALayer: 0x1510b3a70>>\n   |    |    | <UIButton: 0x1511768b0; frame = (45 238; 268 123); opaque = NO; autoresize =
 RM+BM; layer = <CALayer: 0x15101b350>>\n   |    |    |    | <UIButtonLabel: 0x151206ac0; frame = (23.5 43.5; 221.5 36); text
= 'Check Password'; opaque = NO; userInteractionEnabled = NO; layer = <_UILabelLayer: 0x14ffd0580>>\n   |    |    | <UITextFie
ld: 0x1510189d0; frame = (85 216; 204 30); text = ''; clipsToBounds = YES; opaque = NO; autoresize = RM+BM; gestureRecognizers
 = <NSArray: 0x14ffe23e0>; layer = <CALayer: 0x151018ce0>>\n   |    |    |    | <_UITextFieldRoundedRectBackgroundViewNeue: 0x
151105dd0; frame = (0 0; 204 30); opaque = NO; autoresize = W+H; userInteractionEnabled = NO; layer = <CALayer: 0x1510f45c0>>\
n   |    |    |    | <UITextFieldLabel: 0x14fed6810; frame = (7 1; 190 27); text = 'Enter Password'; opaque = NO; userInteract
ionEnabled = NO; layer = <_UILabelLayer: 0x1510eab30>>\n   |    |    | <_UILayoutGuide: 0x151128400; frame = (0 0; 0 20); hidd
en = YES; layer = <CALayer: 0x1510b3a90>>\n   |    |    | <_UILayoutGuide: 0x15111e900; frame = (0 568; 0 0); hidden = YES; la
yer = <CALayer: 0x1510d67d0>>"
cy#
```

c. Search for the value of "UIView" in the output. In this case, it is 0x151099630. Get the value of the nextResponder using the command below:

   - [#0x151099630 nextResponder]

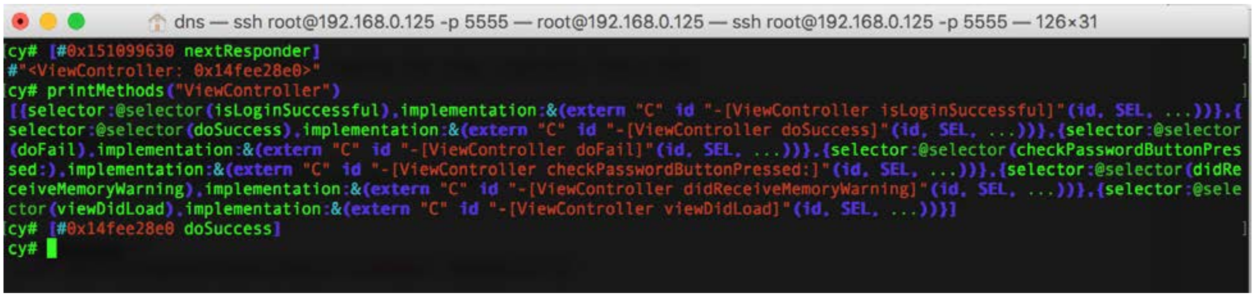


```
[cy# [#0x151099630 nextResponder]
#"<ViewController: 0x14fee28e0>"
cy#
```

The name of the current ViewController is indicated.

d. To use this detail to bypass the screen, get the function names using printMethod (detailed in the previous module) and run the command below to call the login function directly bypassing any available checks.

- ○ [#0x14fee28e0 doSuccess]
  - ■ Where 0x14fee28e0 is the ViewController address and doSuccess is the function to be called.



```
dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126×31
cy# [#0x151099630 nextResponder]
#"<ViewController: 0x14fee28e0>"
cy# printMethods("ViewController")
[{selector:@selector(isLoginSuccessful),implementation:&(extern "C" id "-[ViewController isLoginSuccessful]"(id, SEL, ...))},{
selector:@selector(doSuccess),implementation:&(extern "C" id "-[ViewController doSuccess]"(id, SEL, ...))},{selector:@selector
(doFail),implementation:&(extern "C" id "-[ViewController doFail]"(id, SEL, ...))},{selector:@selector(checkPasswordButtonPres
sed:),implementation:&(extern "C" id "-[ViewController checkPasswordButtonPressed:]"(id, SEL, ...))},{selector:@selector(didRe
ceiveMemoryWarning),implementation:&(extern "C" id "-[ViewController didReceiveMemoryWarning]"(id, SEL, ...))},{selector:@sele
ctor(viewDidLoad),implementation:&(extern "C" id "-[ViewController viewDidLoad]"(id, SEL, ...))}]
cy# [#0x14fee28e0 doSuccess]
cy# 
```

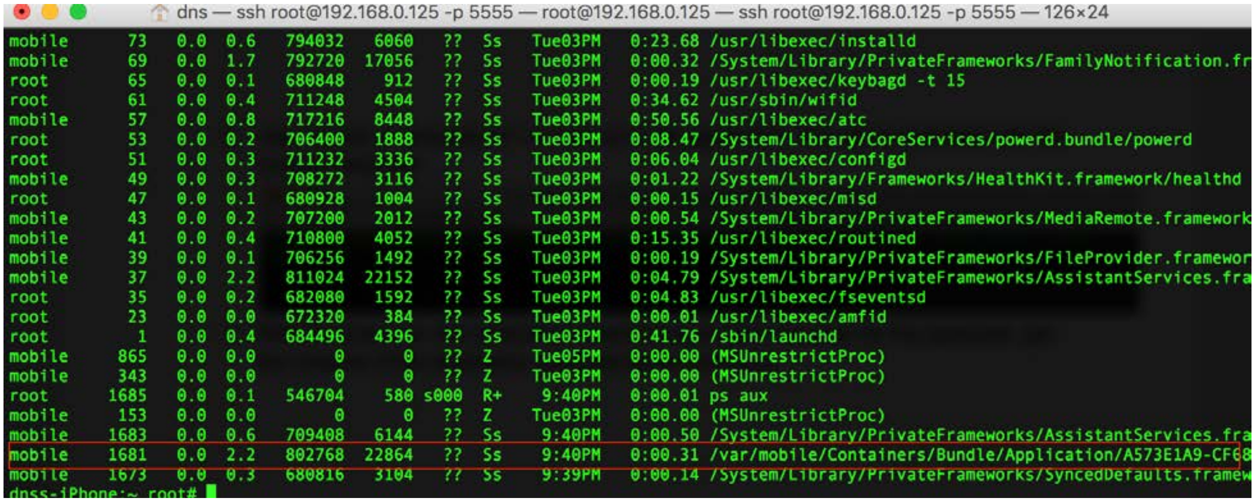
Note that the post login screen is called on the iOS device.

12.1.2 Using Cycript to Override Internal Methods

Application used for Example: CycriptDemoDNS application from the below mentioned link:

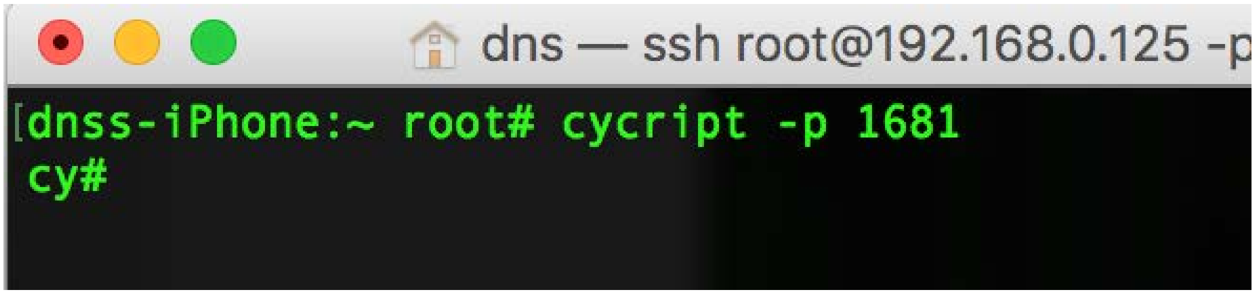
- https://drive.google.com/open?id=0B0b4lUTjHfRKXy1pU29oZmdSUjg

1. Launch the CycriptDemoDNS application on the device. SSH into the iOS device, and get the process id of the application using the command "ps aux".



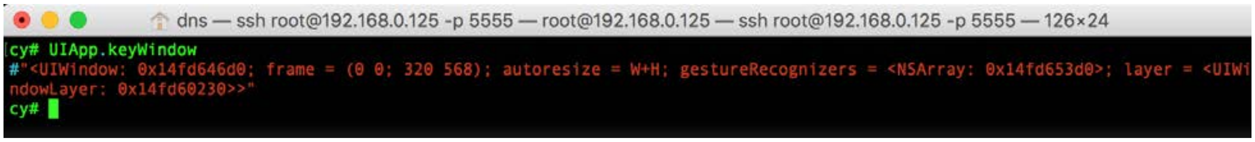
```
dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126×24
mobile      73  0.0  0.6  794032   6060   ??  Ss   Tue03PM   0:23.68 /usr/libexec/installd
mobile      69  0.0  1.7  792720  17056   ??  Ss   Tue03PM   0:00.32 /System/Library/PrivateFrameworks/FamilyNotification.fr
root        65  0.0  0.1  680848    912   ??  Ss   Tue03PM   0:00.19 /usr/libexec/keybagd -t 15
root        61  0.0  0.4  711248   4504   ??  Ss   Tue03PM   0:34.62 /usr/sbin/wifid
mobile      57  0.0  0.8  717216   8448   ??  Ss   Tue03PM   0:50.56 /usr/libexec/atc
root        53  0.0  0.2  706400   1888   ??  Ss   Tue03PM   0:08.47 /System/Library/CoreServices/powerd.bundle/powerd
root        51  0.0  0.3  711232   3336   ??  Ss   Tue03PM   0:06.04 /usr/libexec/configd
mobile      49  0.0  0.3  708272   3116   ??  Ss   Tue03PM   0:01.22 /System/Library/Frameworks/HealthKit.framework/healthd
root        47  0.0  0.1  680928   1004   ??  Ss   Tue03PM   0:00.15 /usr/libexec/misd
mobile      43  0.0  0.2  707200   2012   ??  Ss   Tue03PM   0:00.54 /System/Library/PrivateFrameworks/MediaRemote.framework
mobile      41  0.0  0.4  710800   4052   ??  Ss   Tue03PM   0:15.35 /usr/libexec/routined
mobile      39  0.0  0.1  706256   1492   ??  Ss   Tue03PM   0:00.19 /System/Library/PrivateFrameworks/FileProvider.framewor
mobile      37  0.0  2.2  811024  22152   ??  Ss   Tue03PM   0:04.79 /System/Library/PrivateFrameworks/AssistantServices.fra
root        35  0.0  0.2  682080   1592   ??  Ss   Tue03PM   0:04.83 /usr/libexec/fseventsd
root        23  0.0  0.0  672320    384   ??  Ss   Tue03PM   0:00.01 /usr/libexec/amfid
root         1  0.0  0.4  684496   4396   ??  Ss   Tue03PM   0:41.76 /sbin/launchd
mobile     865  0.0  0.0       0      0   ??  Z    Tue05PM   0:00.00 (MSUnrestrictProc)
mobile     343  0.0  0.0       0      0   ??  Z    Tue03PM   0:00.00 (MSUnrestrictProc)
root      1685  0.0  0.1  546704    580 s000  R+     9:40PM   0:00.01 ps aux
mobile     153  0.0  0.0       0      0   ??  Z    Tue03PM   0:00.00 (MSUnrestrictProc)
mobile    1683  0.0  0.6  709408   6144   ??  Ss     9:40PM   0:00.50 /System/Library/PrivateFrameworks/AssistantServices.fra
mobile    1681  0.0  2.2  802768  22864   ??  Ss     9:40PM   0:00.31 /var/mobile/Containers/Bundle/Application/A573E1A9-CF68
mobile    1673  0.0  0.3  680816   3104   ??  Ss     9:39PM   0:00.14 /System/Library/PrivateFrameworks/SyncedDefaults.framew
dnss-iPhone:~ root# 
```

2. When prompted with the Cycript interpreter, use "cycript -p <process-id>" to hook on to the application.



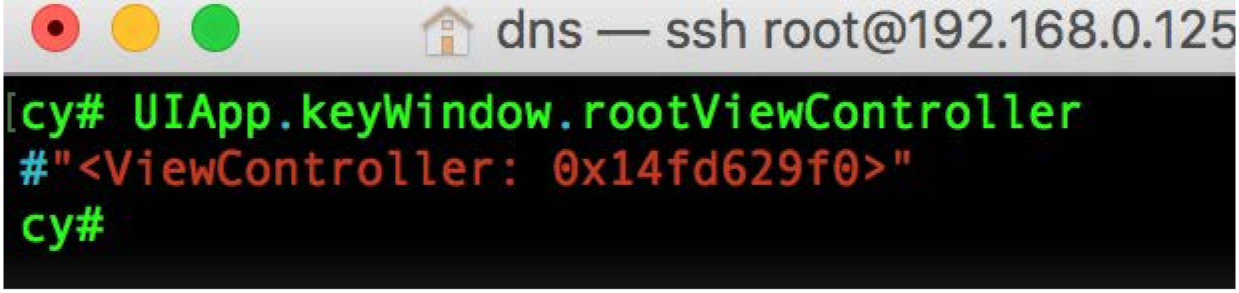
```
[dnss-iPhone:~ root# cycript -p 1681
 cy#
```

3. Find the keyWindow (the current window accepting user touch events) using the command below:
    - *UIApp.keyWindow*


dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126×24

```
[cy# UIApp.keyWindow
#"<UIWindow: 0x14fd646d0; frame = (0 0; 320 568); autoresize = W+H; gestureRecognizers = <NSArray: 0x14fd653d0>; layer = <UIWi
ndowLayer: 0x14fd60230>>"
cy#
```

4. The below command provides the rootViewController for the keyWindow.



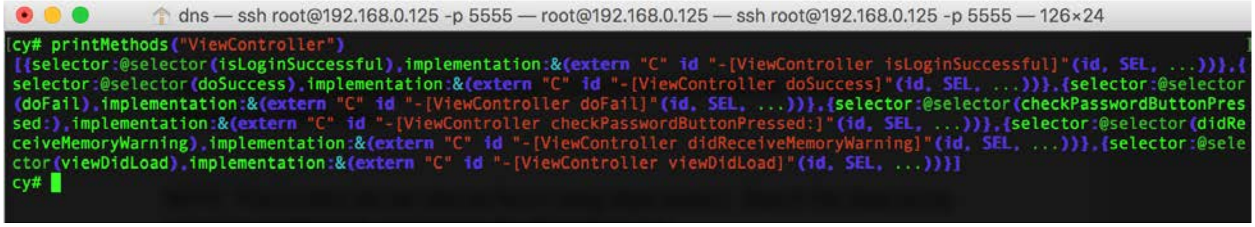
```
[cy# UIApp.keyWindow.rootViewController
 #"<ViewController: 0x14fd629f0>"
 cy#
```

5. Use the function below toobtain the methods of the delegate class found in the previous step.

```
function printMethods(className, isa) {
  var count = new new Type("I");
  var classObj = (isa != undefined) ? objc_getClass(className).constructor :
objc_getClass(className);
  var methods = class_copyMethodList(classObj, count);
  var methodsArray = [];
  for(var i = 0; i < *count; i++) {
    var method = methods[i];
    methodsArray.push({selector:method_getName(method),
implementation:method_getImplementation(method)});
  }
```

```
 free(methods);
 return methodsArray;
}
```

Enter printMethods("ViewController") to print the complete list of possible methods for that screen.



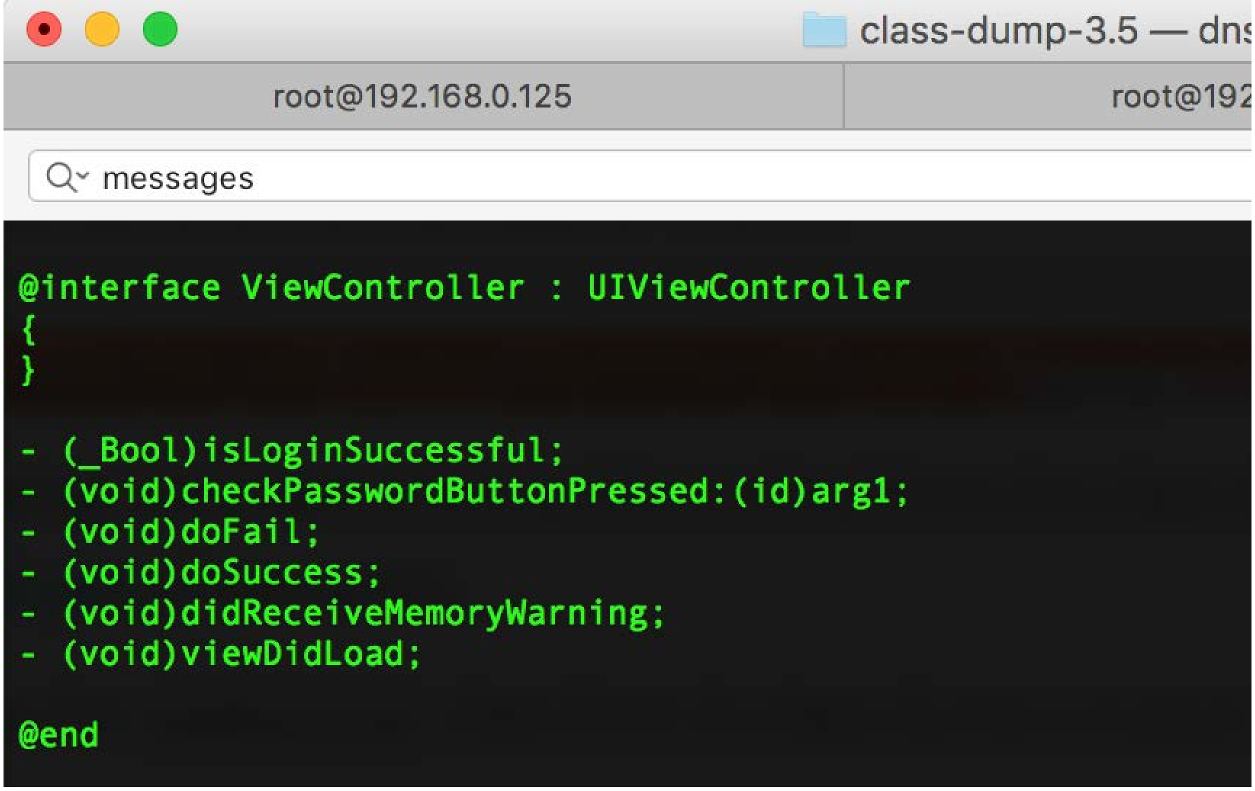
dns — ssh root@192.168.0.125 -p 5555 — root@192.168.0.125 — ssh root@192.168.0.125 -p 5555 — 126×24

```
cy# printMethods("ViewController")
[{selector:@selector(isLoginSuccessful),implementation:&(extern "C" id "-[ViewController isLoginSuccessful]"(id, SEL, ...))},{
selector:@selector(doSuccess),implementation:&(extern "C" id "-[ViewController doSuccess]"(id, SEL, ...))},{selector:@selector
(doFail),implementation:&(extern "C" id "-[ViewController doFail]"(id, SEL, ...))},{selector:@selector(checkPasswordButtonPres
sed:),implementation:&(extern "C" id "-[ViewController checkPasswordButtonPressed:]"(id, SEL, ...))},{selector:@selector(didRe
ceiveMemoryWarning),implementation:&(extern "C" id "-[ViewController didReceiveMemoryWarning]"(id, SEL, ...))},{selector:@sele
ctor(viewDidLoad),implementation:&(extern "C" id "-[ViewController viewDidLoad]"(id, SEL, ...))}]
cy# █
```

NOTE: This function list can also be found using class-dump-z. Search the class-dump output for AppDelegate and look for the @interface in it.

NOTE: "UIApp.keyWindow.rootViewController.visibleViewController" can often be used to view the current view controller.

6. Class-dump reveals that the isLoginSuccessful returns a BOOL value determining whether the login is supposed to be successful or not.

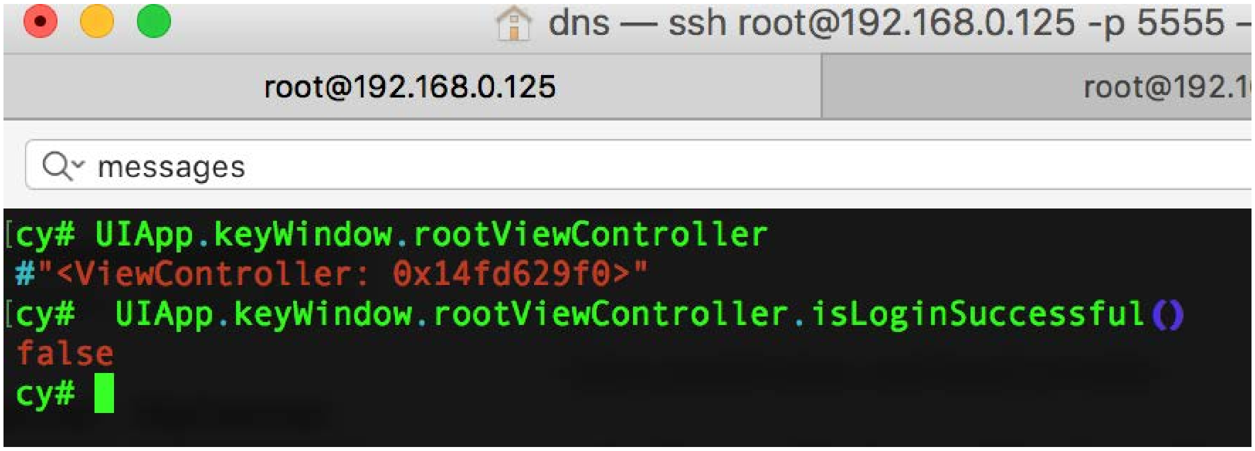


```
@interface ViewController : UIViewController
{
}

- (_Bool)isLoginSuccessful;
- (void)checkPasswordButtonPressed:(id)arg1;
- (void)doFail;
- (void)doSuccess;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;

@end
```

7. Look at the current value of the isLoginSuccessful function using the following command:
   - *UIApp.keyWindow.rootViewController.isLoginSuccessful()*

OR

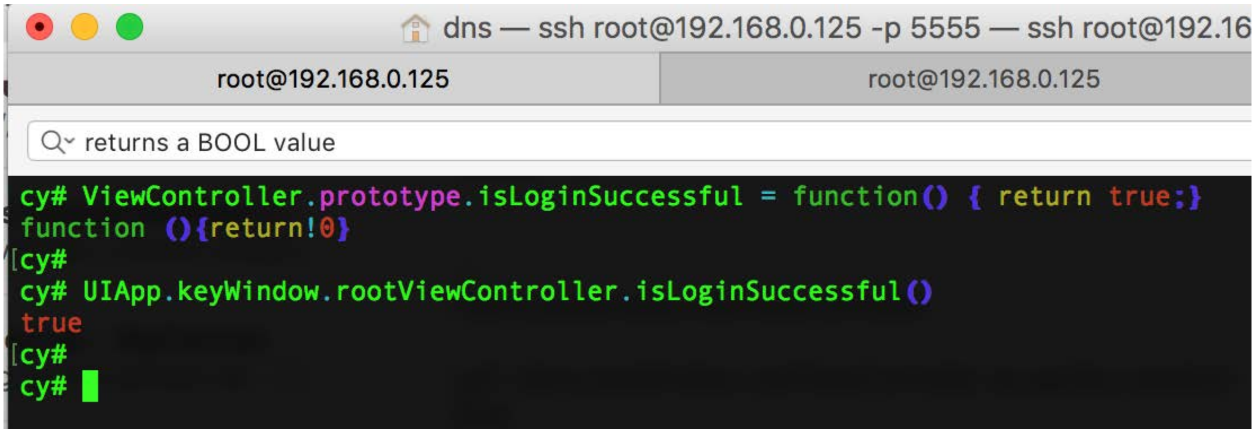
- *[UIApp.keyWindow.rootViewController isLoginSuccessful]*



```
cy# UIApp.keyWindow.rootViewController
#"<ViewController: 0x14fd629f0>"
cy#  UIApp.keyWindow.rootViewController.isLoginSuccessful()
false
cy# 
```

Note that the current value is a boolean false.

8. Use the command below to modify the function to always return TRUE irrespective of the values or the operation performed.
   - *ViewController.prototype.isLoginSuccessful = function() { return true;}*



```
cy# ViewController.prototype.isLoginSuccessful = function() { return true;}
function (){return!0}
cy#
cy# UIApp.keyWindow.rootViewController.isLoginSuccessful()
true
cy#
cy# 
```

9. Click on the “Check Password” button on the device screen and note that the login is successful even though no password was provided.

A similar approach can be used for bypassing Jailbreak Detection in iOS applications.

12.2 Debugging iOS Applications Using LLDB

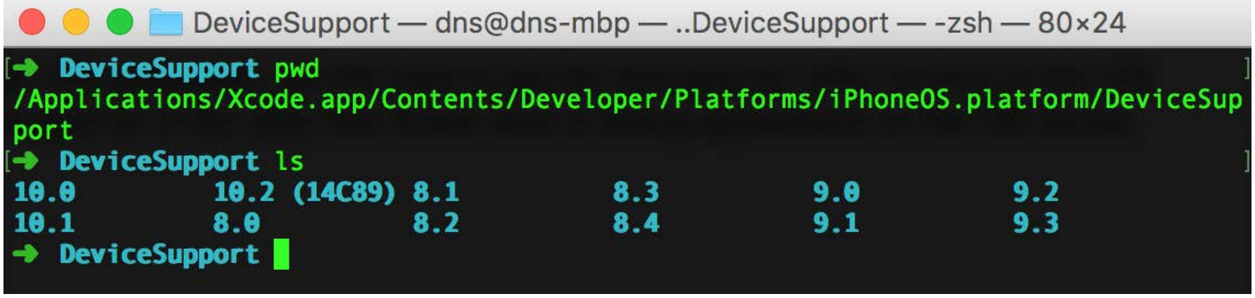
Since the introduction of iOS 8, GDB may not be a viable solution for debugging as GDB support is only available to arm7. In these cases, consider Apple's GDB replacement - LLDB (learn more about LLDB from Apple WWDC videos). For more information refer to the links below:

- http://asciiwwdc.com/2016/sessions/417
- https://developer.apple.com/videos/play/wwdc2015/402/
- https://developer.apple.com/videos/play/wwdc2013/413/

A similar command set to GDB makes LLDB user friendly for those familiar with GDB. Additionally, LLDB is reputed to be faster than GDB.

To debug an iOS application, start the debug server utility running on the iOS device. Debug server is the utility that Xcode uses to debug applications on the iOS device.

By default, *debug server* can be found on the Mac in the Xcode's developer disk image. Navigate to the following location:/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/ to view all the versions of iOS available to you.

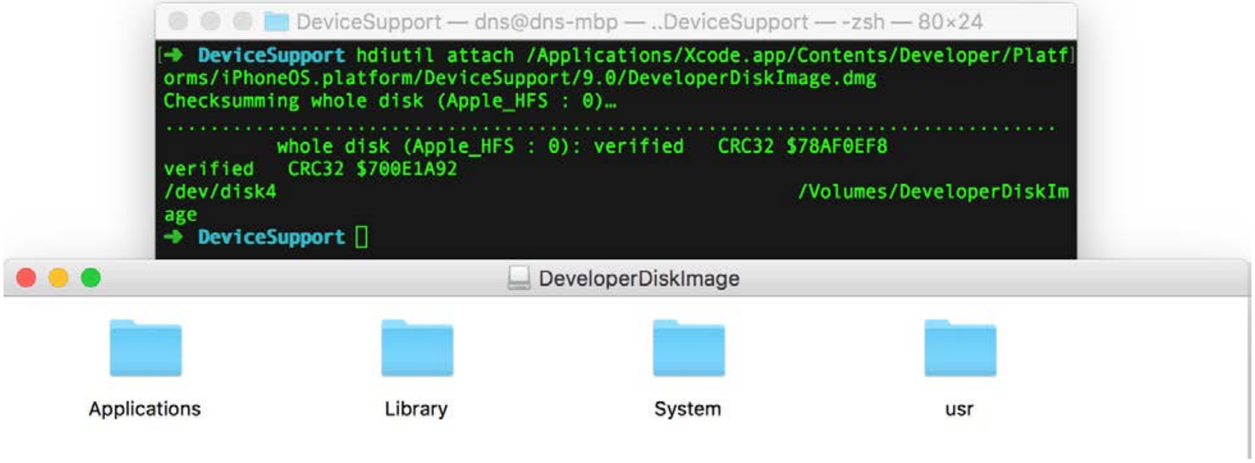

DeviceSupport — dns@dns-mbp — ..DeviceSupport — -zsh — 80×24

```
➜ DeviceSupport pwd
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSup
port
➜ DeviceSupport ls
10.0         10.2 (14C89) 8.1          8.3          9.0          9.2
10.1         8.0          8.2          8.4          9.1          9.3
➜ DeviceSupport
```

Choose the version that is running on the iOS device and mount the related Xcode's developer disk image on your Mac to extract the debug server. In this case, the iOS version running is iOS 9.0.*.

Use the command below to mount the developer disk image:

- *hdiutil attach /Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/9.0/DeveloperDiskImage.dmg*



DeviceSupport — dns@dns-mbp — ..DeviceSupport — -zsh — 80×24

```
➜ DeviceSupport hdiutil attach /Applications/Xcode.app/Contents/Developer/Platf
orms/iPhoneOS.platform/DeviceSupport/9.0/DeveloperDiskImage.dmg
Checksumming whole disk (Apple_HFS : 0)…
................................................................................
          whole disk (Apple_HFS : 0): verified   CRC32 $78AF0EF8
verified   CRC32 $700E1A92
/dev/disk4                                                  /Volumes/DeveloperDiskIm
age
➜ DeviceSupport
```

DeveloperDiskImage
Applications
Library
System
usr

Copy the debugserver binary to your device at a known location using the command below:

- *cp /Volumes/DeveloperDiskImage/usr/bin/debugserver /Users/dns/Desktop/mobile/lldb_guide*

By default, this debug server binary can only debug applications that are signed by a specific provisioning profile. This is due to the lack of entitlement to allow task_for_pid(). To counter this, create an entitlement file with the task_for_pid flag set to true and use it to sign the debug server binary.

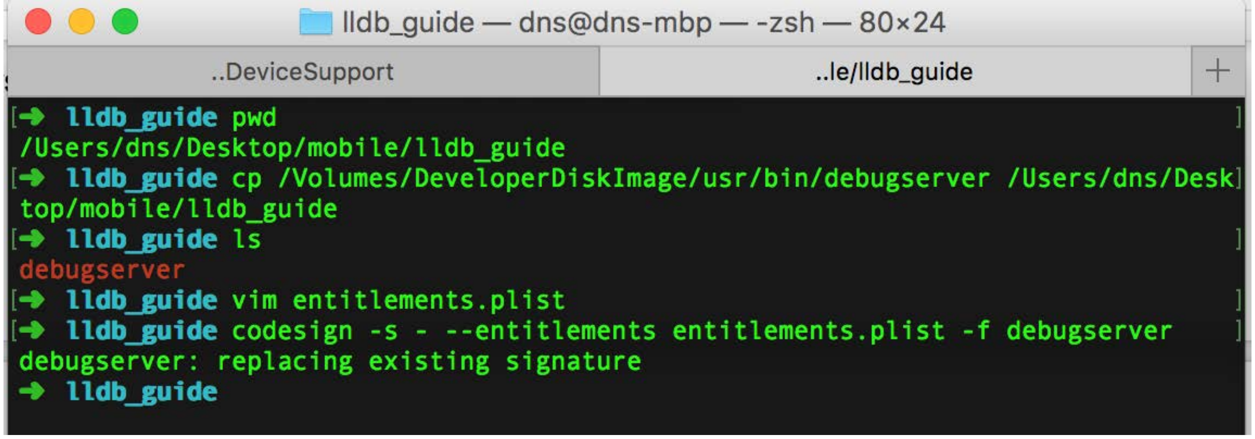
See the content of the entitlement file entitlements.plist below:

```
<?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>com.apple.springboard.debugapplications</key> <true/>
   <key>run-unsigned-code</key> <true/>
   <key>get-task-allow</key> <true/>
   <key>task_for_pid-allow</key> <true/>
```

</dict>
</plist>

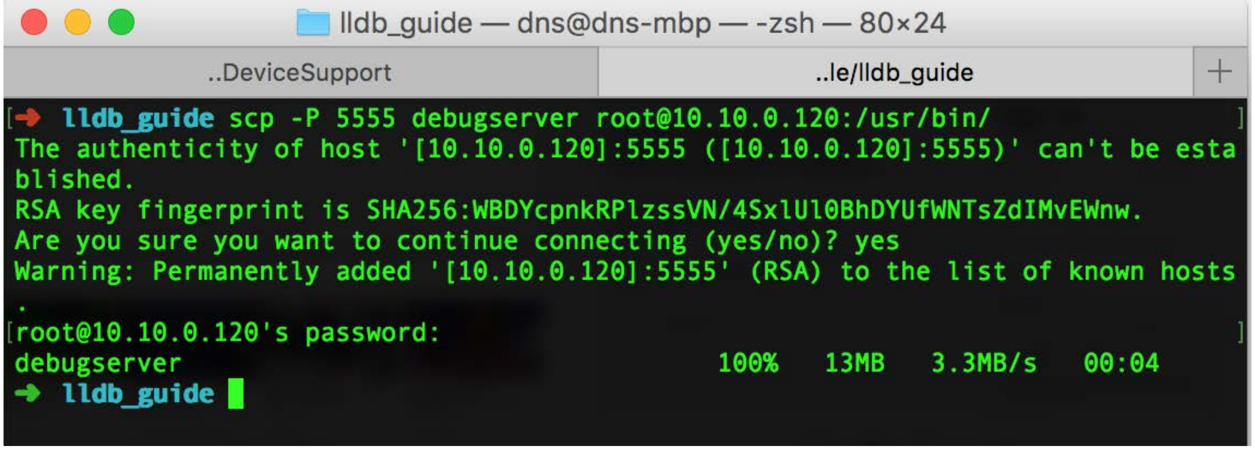
Resign the debug server binary using the command below:

- *codesign -s - --entitlements entitlements.plist -f debugserver*



```
➜ lldb_guide pwd
/Users/dns/Desktop/mobile/lldb_guide
➜ lldb_guide cp /Volumes/DeveloperDiskImage/usr/bin/debugserver /Users/dns/Desk
top/mobile/lldb_guide
➜ lldb_guide ls
debugserver
➜ lldb_guide vim entitlements.plist
➜ lldb_guide codesign -s - --entitlements entitlements.plist -f debugserver
debugserver: replacing existing signature
➜ lldb_guide
```

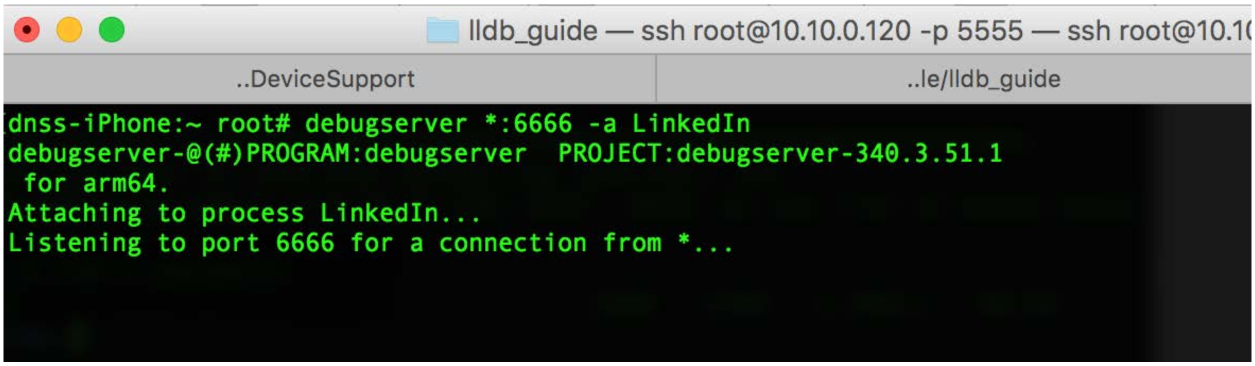
Copy the signed debug server to the iOS device.



```
➜ lldb_guide scp -P 5555 debugserver root@10.10.0.120:/usr/bin/
The authenticity of host '[10.10.0.120]:5555 ([10.10.0.120]:5555)' can't be esta
blished.
RSA key fingerprint is SHA256:WBDYcpnkRPlzssVN/4SxlUl0BhDYUfWNTsZdIMvEWnw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[10.10.0.120]:5555' (RSA) to the list of known hosts
.
root@10.10.0.120's password:
debugserver                                   100%   13MB   3.3MB/s   00:04
➜ lldb_guide
```

To start debugging, SSH into the iOS device and attach the debug server to the running iOS application to be debugged.

- *debugserver *:6666 -a LinkedIn*



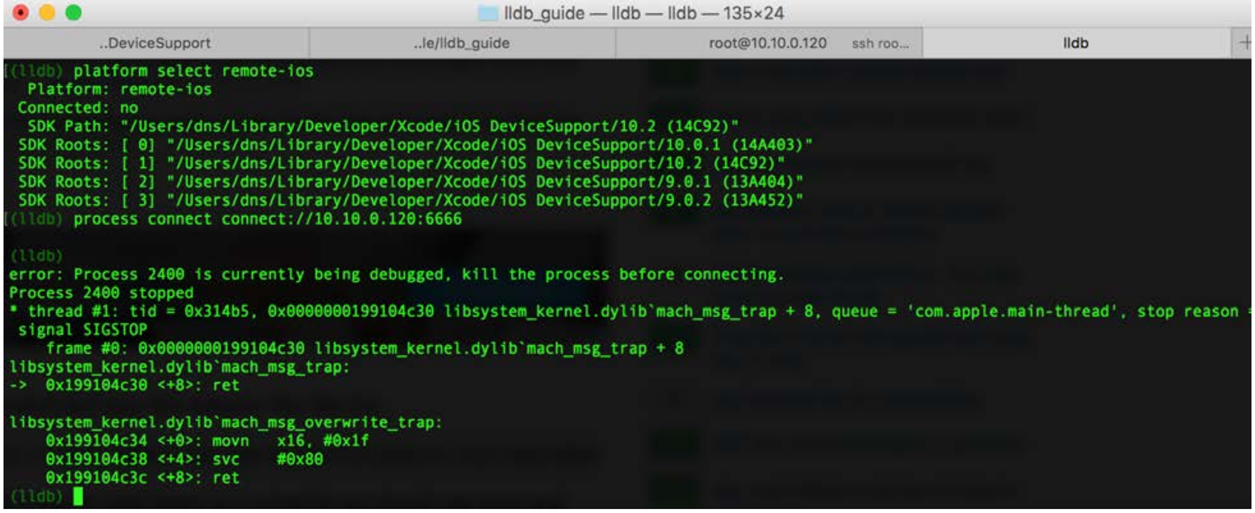
```
dnss-iPhone:~ root# debugserver *:6666 -a LinkedIn
debugserver-@(#)PROGRAM:debugserver  PROJECT:debugserver-340.3.51.1
 for arm64.
Attaching to process LinkedIn...
Listening to port 6666 for a connection from *...
```

NOTE: If the tool does not work, strip the fat binary to anything other than arm64 since many of the tools do not work on arm64.

On the Mac, while connected to the same network as your iOS device, enter the command below to connect to the debug server instance:

lldb
(lldb) platform select remote-ios
(lldb) process connect connect://<iphone-ip>:6666

The connection may take some time. When connected, the output will look like the screenshot below:



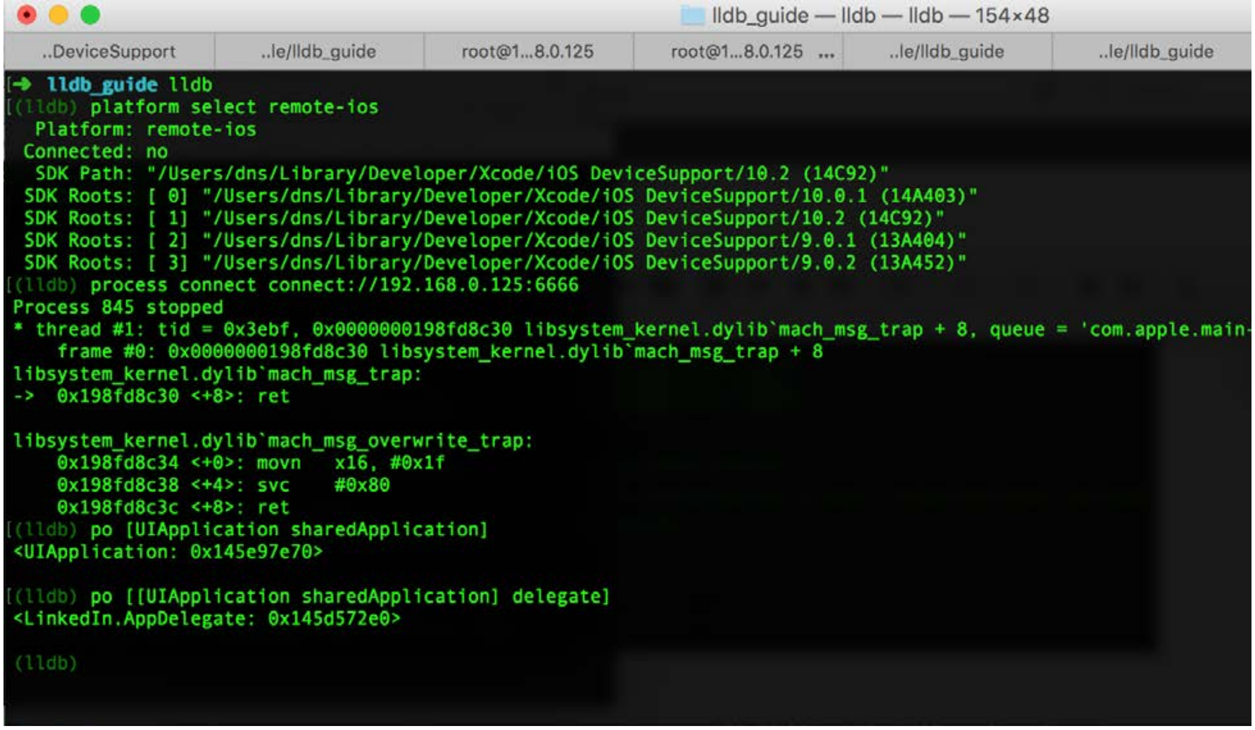
```
(lldb) platform select remote-ios
  Platform: remote-ios
 Connected: no
  SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
 SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
 SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
 SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
 SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
(lldb) process connect connect://10.10.0.120:6666

(lldb)
error: Process 2400 is currently being debugged, kill the process before connecting.
Process 2400 stopped
* thread #1: tid = 0x314b5, 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-thread', stop reason
 signal SIGSTOP
    frame #0: 0x0000000199104c30 libsystem_kernel.dylib`mach_msg_trap + 8
libsystem_kernel.dylib`mach_msg_trap:
->  0x199104c30 <+8>: ret

libsystem_kernel.dylib`mach_msg_overwrite_trap:
    0x199104c34 <+0>: movn   x16, #0x1f
    0x199104c38 <+4>: svc    #0x80
    0x199104c3c <+8>: ret
(lldb) █
```

You can now debug the application.

Following are some of the basic LLDB commands.

“po” can be used to print the object instances and the delegates via the commands below:

- *po [UIApplication sharedApplication]*
- *po [[UIApplication sharedApplication] delegate]*



```
[→ lldb_guide lldb
[(lldb) platform select remote-ios
   Platform: remote-ios
  Connected: no
   SDK Path: "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
  SDK Roots: [ 0] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.0.1 (14A403)"
  SDK Roots: [ 1] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/10.2 (14C92)"
  SDK Roots: [ 2] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.1 (13A404)"
  SDK Roots: [ 3] "/Users/dns/Library/Developer/Xcode/iOS DeviceSupport/9.0.2 (13A452)"
[(lldb) process connect connect://192.168.0.125:6666
 Process 845 stopped
 * thread #1: tid = 0x3ebf, 0x0000000198fd8c30 libsystem_kernel.dylib`mach_msg_trap + 8, queue = 'com.apple.main-
     frame #0: 0x0000000198fd8c30 libsystem_kernel.dylib`mach_msg_trap + 8
 libsystem_kernel.dylib`mach_msg_trap:
 ->  0x198fd8c30 <+8>: ret

 libsystem_kernel.dylib`mach_msg_overwrite_trap:
     0x198fd8c34 <+0>: movn   x16, #0x1f
     0x198fd8c38 <+4>: svc    #0x80
     0x198fd8c3c <+8>: ret
[(lldb) po [UIApplication sharedApplication]
 <UIApplication: 0x145e97e70>

[(lldb) po [[UIApplication sharedApplication] delegate]
 <LinkedIn.AppDelegate: 0x145d572e0>

 (lldb)
```

To dump all symbols from the main executable and any shared libraries use the command “image dump symtab.”

To dump all sections from the main executable and any shared libraries use the command “image dump sections.”

To list the main executable and all dependent shared libraries along with their location in memory use “image list.”.

To dump the information stored for a raw address in the executable or any shared libraries use “image lookup --address 0x00011e4.”

To look up functions matching a regular expression in a binary use the commands below:

image lookup -r -n <FUNC_REGEX> → finds debug symbols
image lookup -r -s <FUNC_REGEX> → finds non-debug symbols
NOTE: Provide a list of binaries as arguments to limit the search.

It is possible to set breakpoints on any of the methods in the iOS application. To do this, locate the method names. This can be gathered via a class-dump-z of Hopper.

Decompile the binary using Clutch, pull it from the iOS device, and run it on Hopper to view some of the method names. To pull the binary from the device use:

- *scp -P 5555 root@192.168.0.125:"/tmp/LinkedIn_Patches" ./*

The breakpoint can be set using a syntax of the form shown below:

```
breakpoint set --name "-[NSString stringWithFormat:]"
b -[NSString stringWithFormat:]
b -[AppDelegate pinLockControllerDidFinishUnlocking:]
```

Use “target stop-hook add” to add any command or script to be executed every time a breakpoint is reached.

Use “process continue” to continue once breakpoint is set.

Refer to http://lldb.llvm.org/lldb-gdb.html for a GDB to LLDB Command Map along with a detailed command set for LLDB.

NOTE: To use LLDB for a local iOS binary from a Mac, most of the steps remain the same. The only changes will be in the lldb initialization:

```
(lldb) platform select remote-ios
(lldb) target create --arch arm64 /Users/dns/Desktop/mobile/lldb_guide/newiOSBinaryWithoutPIE
```

13. Reverse Engineering Using Hopper

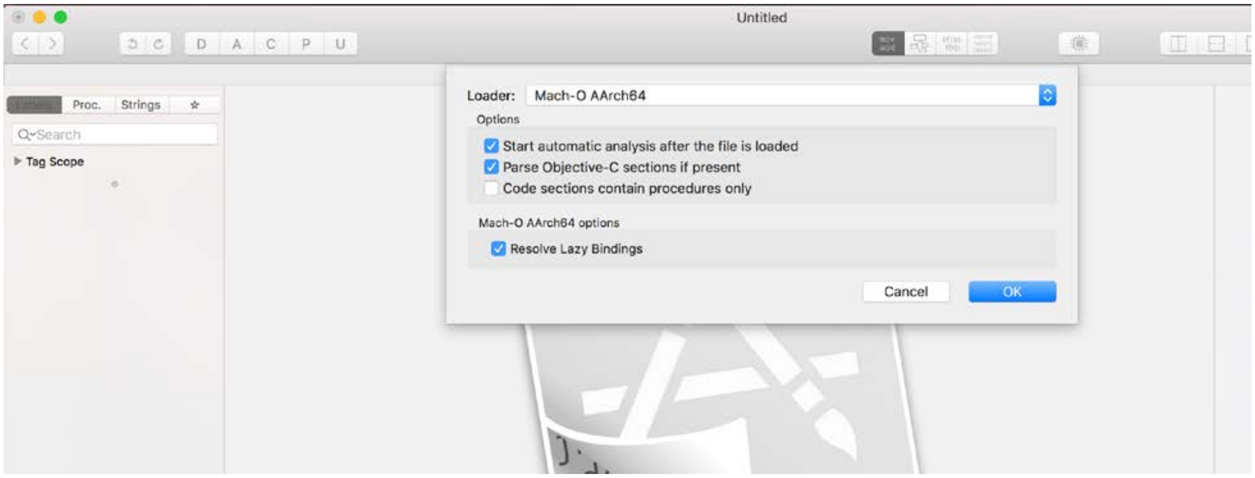
Hopper is the best (and often the cheapest) disassembler for reverse engineering iOS applications. The demo version does allow reverse engineering, but it is time limited and does not allow dumping the newly modified executables. Hopper can be used for reversing 32bit and 64bit iOS applications.

Application used for Example: JailbreakDetectionDemoDNS application from the below mentioned link:

- https://drive.google.com/open?id=0B0b4lUTjHfRKNTdCVzlvN2ZReVk

Install the JailbreakDetectionDemoDNS application using Xcode on to the iOS device. (Steps will be different if the .IPA file is used instead of installation via Xcode).

1. Extract the JailbreakDetectionDemoDNS ipa file and open the JailbreakDetectionDemoDNS binary in Hopper.



Loader: Mach-O AArch64

Options

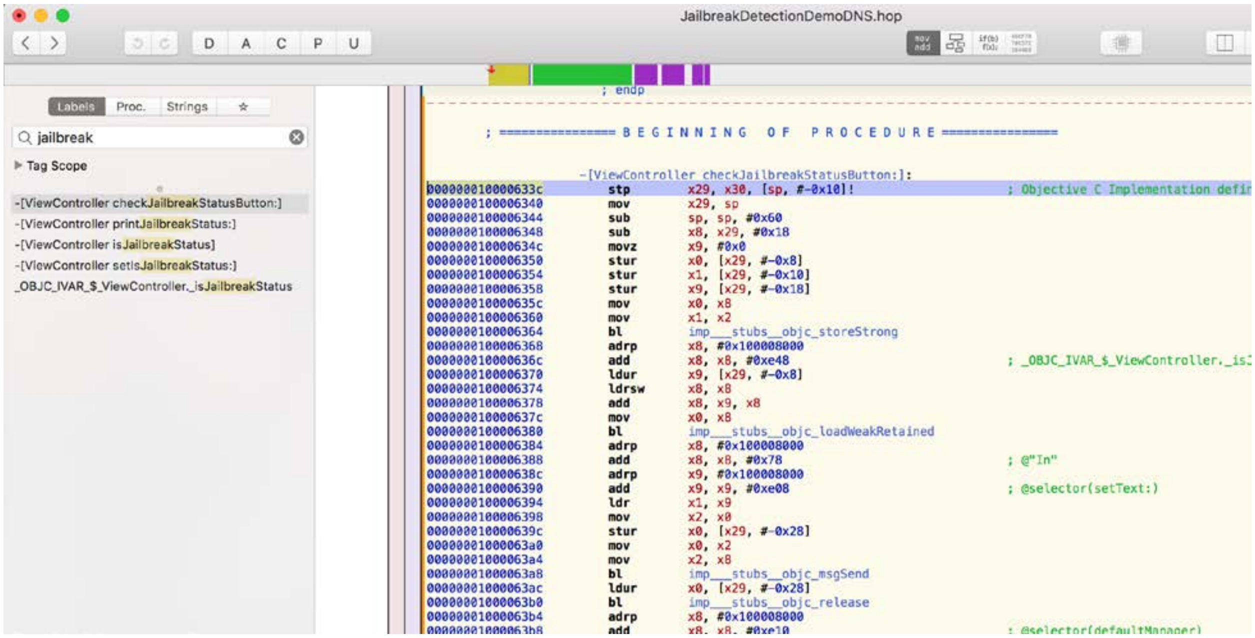
☑ Start automatic analysis after the file is loaded
☑ Parse Objective-C sections if present
☐ Code sections contain procedures only

Mach-O AArch64 options

☑ Resolve Lazy Bindings

When prompted for the loader, select AArch64.

2. Once the automatic analysis is complete, look at the Labels present in the binary. To bypass Jailbreak detection, search for the keyword “jailbreak.”



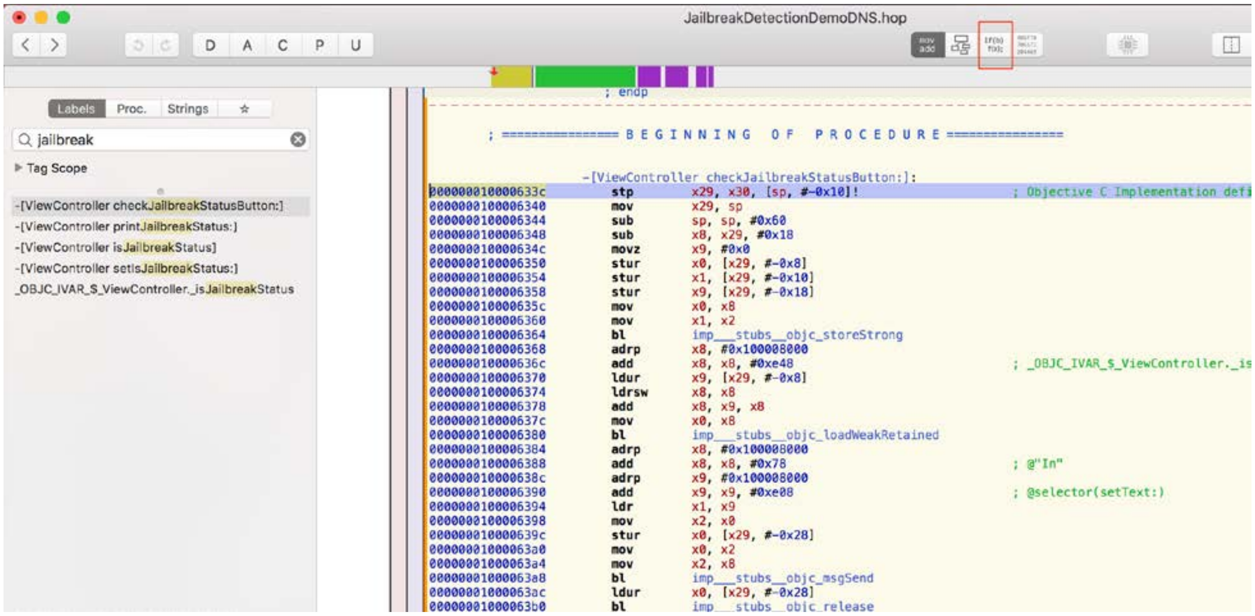
jailbreak

- -[ViewController checkJailbreakStatusButton:]
- -[ViewController printJailbreakStatus:]
- -[ViewController isJailbreakStatus]
- -[ViewController setIsJailbreakStatus:]
- _OBJC_IVAR_$_ViewController._isJailbreakStatus

```
                        ; endp
 ; =============== B E G I N N I N G   O F   P R O C E D U R E ===============

              -[ViewController checkJailbreakStatusButton:]:
000000010000633c        stp        x29, x30, [sp, #-0x10]!                          ; Objective C Implementation defir
0000000100006340        mov        x29, sp
0000000100006344        sub        sp, sp, #0x60
0000000100006348        sub        x8, x29, #0x18
000000010000634c        movz       x9, #0x0
0000000100006350        stur       x0, [x29, #-0x8]
0000000100006354        stur       x1, [x29, #-0x10]
0000000100006358        stur       x9, [x29, #-0x18]
000000010000635c        mov        x0, x8
0000000100006360        mov        x1, x2
0000000100006364        bl         imp___stubs__objc_storeStrong
0000000100006368        adrp       x8, #0x100008000
000000010000636c        add        x8, x8, #0xe48                                   ; _OBJC_IVAR_$_ViewController._is
0000000100006370        ldur       x9, [x29, #-0x8]
0000000100006374        ldrsw      x8, x8
0000000100006378        add        x8, x9, x8
000000010000637c        mov        x0, x8
0000000100006380        bl         imp___stubs__objc_loadWeakRetained
0000000100006384        adrp       x8, #0x100008000
0000000100006388        add        x8, x8, #0x78                                    ; @"In"
000000010000638c        adrp       x9, #0x100008000
0000000100006390        add        x9, x9, #0xe08                                   ; @selector(setText:)
0000000100006394        ldr        x1, x9
0000000100006398        mov        x2, x0
000000010000639c        stur       x0, [x29, #-0x28]
00000001000063a0        mov        x0, x2
00000001000063a4        mov        x2, x8
00000001000063a8        bl         imp___stubs__objc_msgSend
00000001000063ac        ldur       x0, [x29, #-0x28]
00000001000063b0        bl         imp___stubs__objc_release
00000001000063b4        adrp       x8, #0x100008000
00000001000063b8        add        x8, x8, #0xe10                                   ; @selector(defaultManager)
```

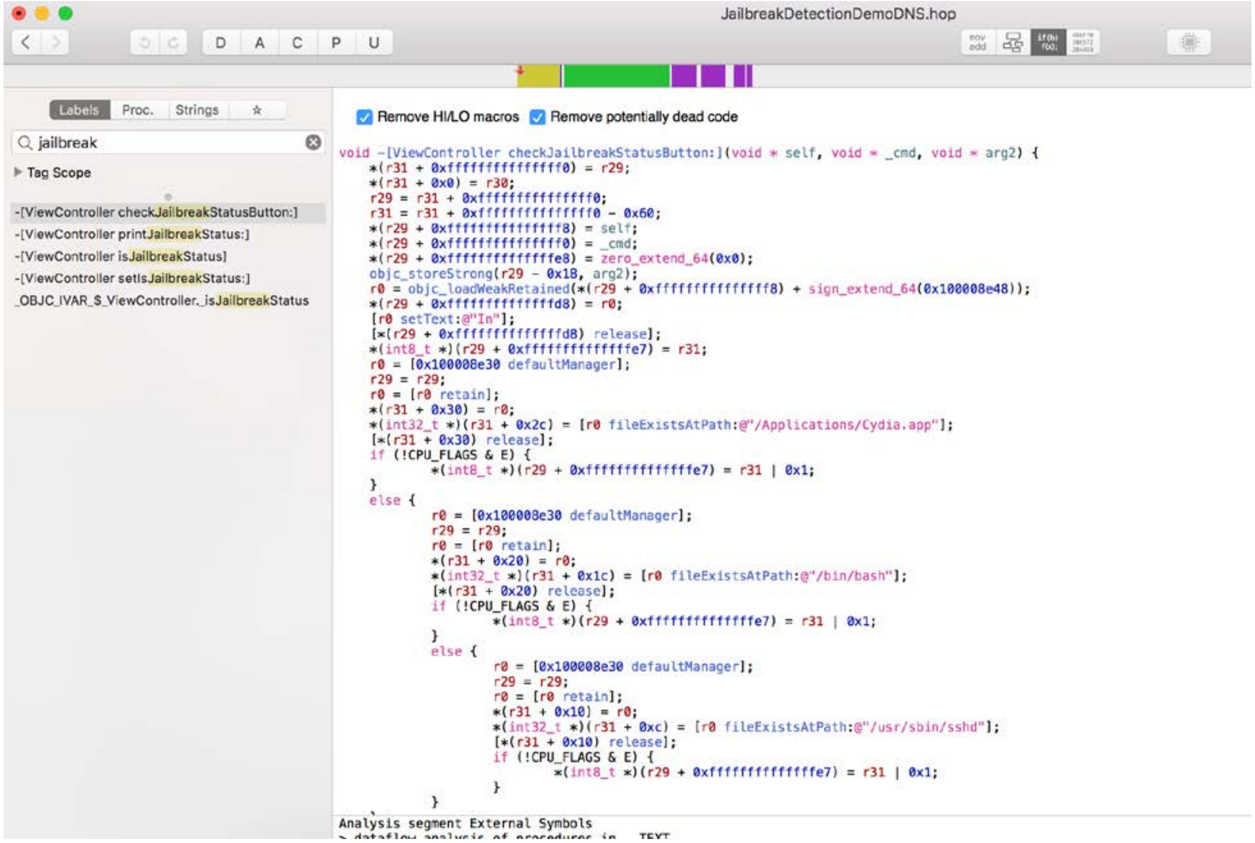
Lookat “checkJailbreakStatusButton.” See the disassembled code on the right.

3. An interesting feature of Hopper is the option to look at the pseudocode of a function. To look at the Pseudocode for the application click the button shown in the following screenshot.



-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus:]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus:]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

```
                       ; endp

     ; =============== B E G I N N I N G   O F   P R O C E D U R E ===============

                       -[ViewController checkJailbreakStatusButton:]:
000000010000633c         stp        x29, x30, [sp, #-0x10]!                   ; Objective C Implementation defi
0000000100006340         mov        x29, sp
0000000100006344         sub        sp, sp, #0x60
0000000100006348         sub        x8, x29, #0x18
000000010000634c         movz       x9, #0x0
0000000100006350         stur       x0, [x29, #-0x8]
0000000100006354         stur       x1, [x29, #-0x10]
0000000100006358         stur       x9, [x29, #-0x18]
000000010000635c         mov        x0, x8
0000000100006360         mov        x1, x2
0000000100006364         bl         imp___stubs__objc_storeStrong
0000000100006368         adrp       x8, #0x100008000
000000010000636c         add        x8, x8, #0xe48                            ; _OBJC_IVAR_$_ViewController._is
0000000100006370         ldur       x9, [x29, #-0x8]
0000000100006374         ldrsw      x8, x8
0000000100006378         add        x8, x9, x8
000000010000637c         mov        x0, x8
0000000100006380         bl         imp___stubs__objc_loadWeakRetained
0000000100006384         adrp       x8, #0x100008000
0000000100006388         add        x8, x8, #0x78                             ; @"In"
000000010000638c         adrp       x9, #0x100008000
0000000100006390         add        x9, x9, #0xe08                            ; @selector(setText:)
0000000100006394         ldr        x1, x9
0000000100006398         mov        x2, x0
000000010000639c         stur       x0, [x29, #-0x28]
00000001000063a0         mov        x0, x2
00000001000063a4         mov        x2, x8
00000001000063a8         bl         imp___stubs__objc_msgSend
00000001000063ac         ldur       x0, [x29, #-0x28]
00000001000063b0         bl         imp___stubs__objc_release
```



jailbreak

-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus:]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus:]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

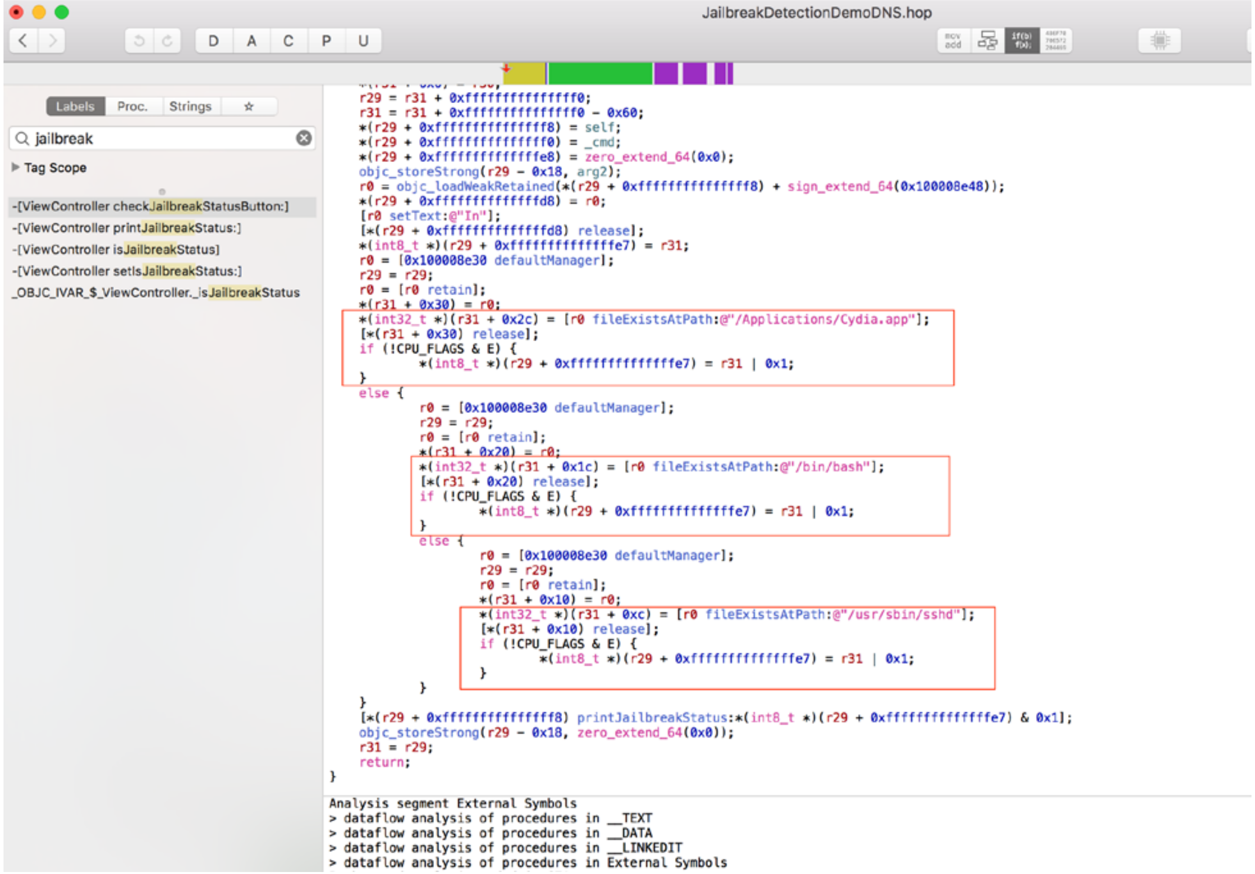
☑ Remove HI/LO macros ☑ Remove potentially dead code

```
void -[ViewController checkJailbreakStatusButton:](void * self, void * _cmd, void * arg2) {
    *(r31 + 0xfffffffffffffff0) = r29;
    *(r31 + 0x0) = r30;
    r29 = r31 + 0xfffffffffffffff0;
    r31 = r31 + 0xfffffffffffffff0 - 0x60;
    *(r29 + 0xfffffffffffffff8) = self;
    *(r29 + 0xfffffffffffffff0) = _cmd;
    *(r29 + 0xffffffffffffffe8) = zero_extend_64(0x0);
    objc_storeStrong(r29 - 0x18, arg2);
    r0 = objc_loadWeakRetained(*(r29 + 0xfffffffffffffff8) + sign_extend_64(0x100008e48));
    *(r29 + 0xffffffffffffffd8) = r0;
    [r0 setText:@"In"];
    [*(r29 + 0xffffffffffffffd8) release];
    *(int8_t *)(r29 + 0xffffffffffffffe7) = r31;
    r0 = [0x100008e30 defaultManager];
    r29 = r29;
    r0 = [r0 retain];
    *(r31 + 0x30) = r0;
    *(int32_t *)(r31 + 0x2c) = [r0 fileExistsAtPath:@"/Applications/Cydia.app"];
    [*(r31 + 0x30) release];
    if (!CPU_FLAGS & E) {
            *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
    }
    else {
            r0 = [0x100008e30 defaultManager];
            r29 = r29;
            r0 = [r0 retain];
            *(r31 + 0x20) = r0;
            *(int32_t *)(r31 + 0x1c) = [r0 fileExistsAtPath:@"/bin/bash"];
            [*(r31 + 0x20) release];
            if (!CPU_FLAGS & E) {
                    *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
            }
            else {
                    r0 = [0x100008e30 defaultManager];
                    r29 = r29;
                    r0 = [r0 retain];
                    *(r31 + 0x10) = r0;
                    *(int32_t *)(r31 + 0xc) = [r0 fileExistsAtPath:@"/usr/sbin/sshd"];
                    [*(r31 + 0x10) release];
                    if (!CPU_FLAGS & E) {
                            *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
                    }
            }
```

```
Analysis segment External Symbols
> dataflow analysis of procedures in __TEXT
```

Pseudo Code makes it easy to understand what the code does.

4. The screenshot below shows how the application looks for the presence of the binaries like Cydia, bash and sshd on the file system.

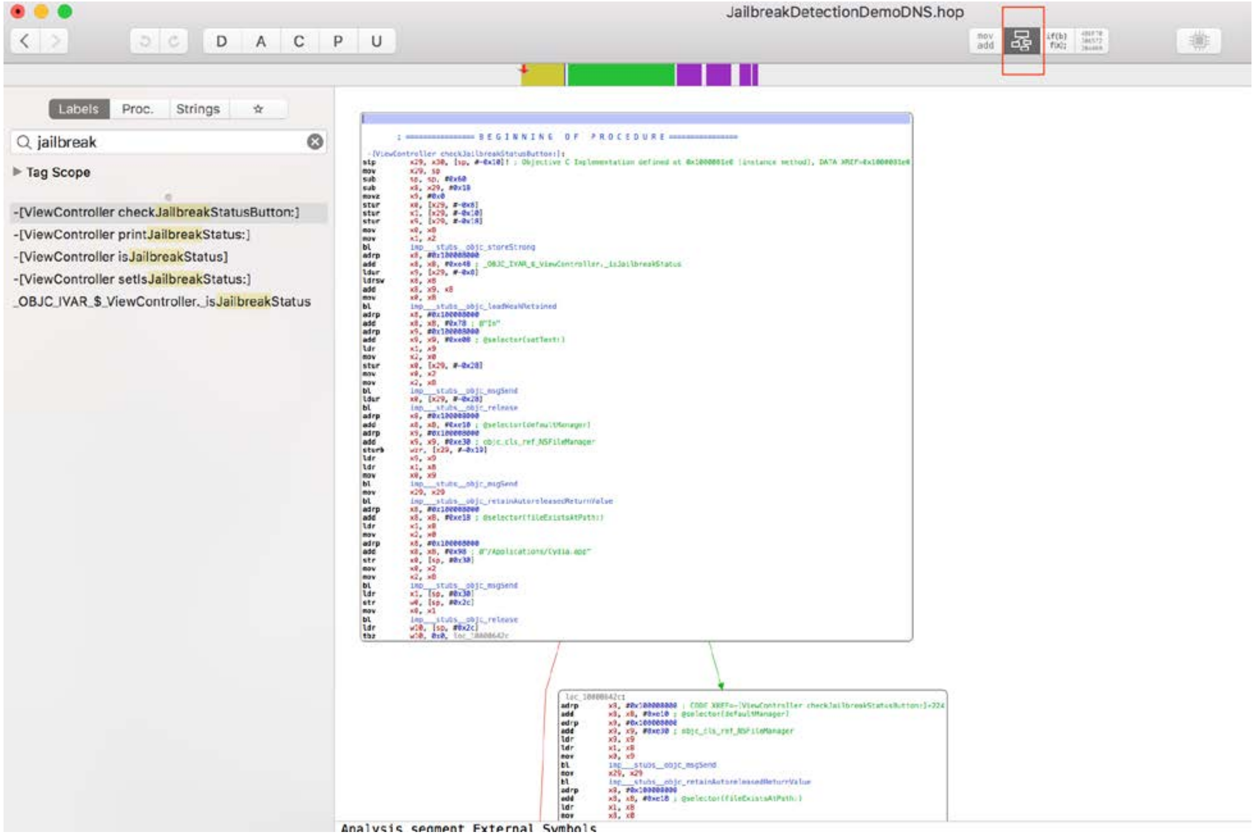

JailbreakDetectionDemoDNS.hop
D
A
C
P
U
Labels
Proc.
Strings
☆
jailbreak
Tag Scope
-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus:]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus:]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

```
    r29 = r31 + 0xfffffffffffffff0;
    r31 = r31 + 0xfffffffffffffff0 - 0x60;
    *(r29 + 0xfffffffffffffff8) = self;
    *(r29 + 0xfffffffffffffff0) = _cmd;
    *(r29 + 0xffffffffffffffe8) = zero_extend_64(0x0);
    objc_storeStrong(r29 - 0x18, arg2);
    r0 = objc_loadWeakRetained(*(r29 + 0xfffffffffffffff8) + sign_extend_64(0x100008e48));
    *(r29 + 0xffffffffffffffd8) = r0;
    [r0 setText:@"In"];
    [*(r29 + 0xffffffffffffffd8) release];
    *(int8_t *)(r29 + 0xffffffffffffffe7) = r31;
    r0 = [0x100008e30 defaultManager];
    r29 = r29;
    r0 = [r0 retain];
    *(r31 + 0x30) = r0;
    *(int32_t *)(r31 + 0x2c) = [r0 fileExistsAtPath:@"/Applications/Cydia.app"];
    [*(r31 + 0x30) release];
    if (!CPU_FLAGS & E) {
            *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
    }
    else {
            r0 = [0x100008e30 defaultManager];
            r29 = r29;
            r0 = [r0 retain];
            *(r31 + 0x20) = r0;
            *(int32_t *)(r31 + 0x1c) = [r0 fileExistsAtPath:@"/bin/bash"];
            [*(r31 + 0x20) release];
            if (!CPU_FLAGS & E) {
                    *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
            }
            else {
                    r0 = [0x100008e30 defaultManager];
                    r29 = r29;
                    r0 = [r0 retain];
                    *(r31 + 0x10) = r0;
                    *(int32_t *)(r31 + 0xc) = [r0 fileExistsAtPath:@"/usr/sbin/sshd"];
                    [*(r31 + 0x10) release];
                    if (!CPU_FLAGS & E) {
                            *(int8_t *)(r29 + 0xffffffffffffffe7) = r31 | 0x1;
                    }
            }
    }
    [*(r29 + 0xfffffffffffffff8) printJailbreakStatus:*(int8_t *)(r29 + 0xffffffffffffffe7) & 0x1];
    objc_storeStrong(r29 - 0x18, zero_extend_64(0x0));
    r31 = r29;
    return;
}
```

```
Analysis segment External Symbols
> dataflow analysis of procedures in __TEXT
> dataflow analysis of procedures in __DATA
> dataflow analysis of procedures in __LINKEDIT
> dataflow analysis of procedures in External Symbols
```

If those binaries are present, the application infers that the device is jailbroken and sets the required flags.

5. In conjunction with Pseudo code, it is always important to look at the Control Flow Graph (CFG) that Hopper provides, to understand the code and branching. To get the CFG, click on the button shown in the following screenshot.



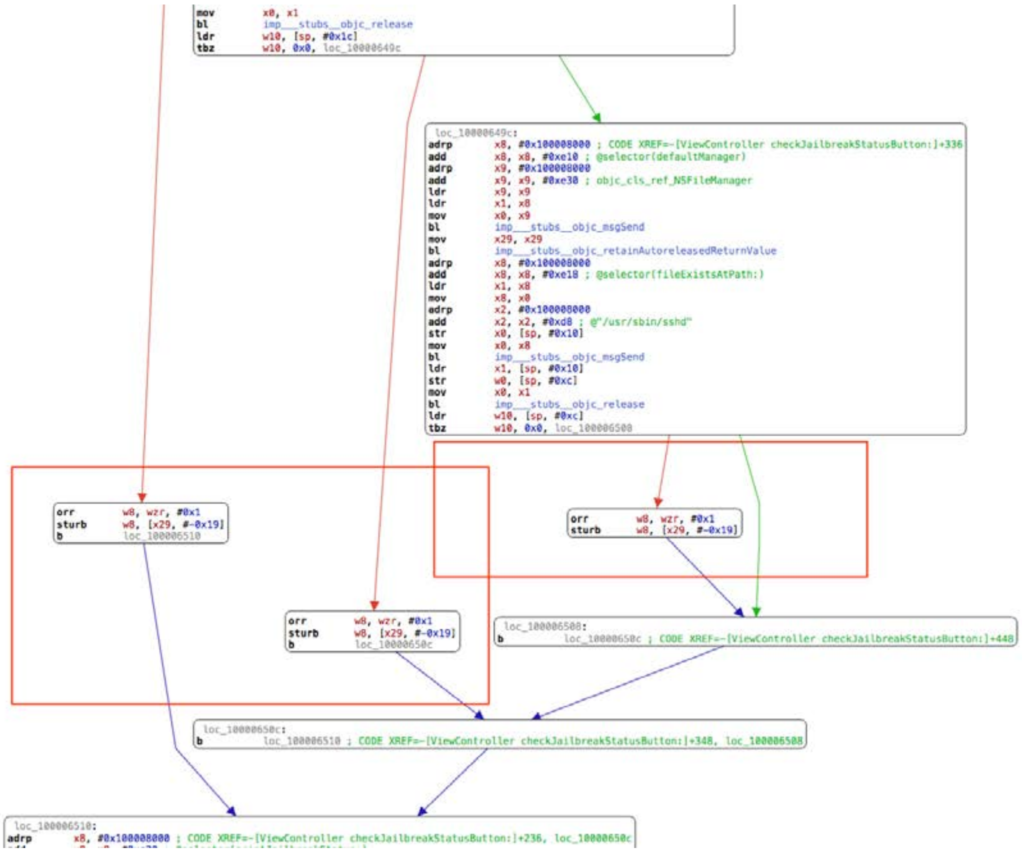
-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus:]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus:]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

```
        ; ================ B E G I N N I N G   O F   P R O C E D U R E ================

        -[ViewController checkJailbreakStatusButton:]:
stp         x29, x30, [sp, #-0x10]! ; Objective C Implementation defined at 0x1000081e0 (instance method), DATA XREF=0x1000081e0
mov         x29, sp
sub         sp, sp, #0x60
sub         x8, x29, #0x18
movz        x9, #0x0
stur        x0, [x29, #-0x8]
stur        x1, [x29, #-0x10]
stur        x9, [x29, #-0x18]
mov         x0, x8
mov         x1, x2
bl          imp___stubs__objc_storeStrong
adrp        x8, #0x100008000
add         x8, x8, #0xe48 ; _OBJC_IVAR_$_ViewController._isJailbreakStatus
ldur        x9, [x29, #-0x8]
ldrsw       x8, x8
add         x8, x9, x8
mov         x0, x8
bl          imp___stubs__objc_loadWeakRetained
adrp        x8, #0x100008000
add         x8, x8, #0x78 ; @"In"
adrp        x9, #0x100008000
add         x9, x9, #0xe00 ; @selector(setText:)
ldr         x1, x9
mov         x2, x8
stur        x0, [x29, #-0x28]
mov         x0, x2
mov         x2, x8
bl          imp___stubs__objc_msgSend
ldur        x0, [x29, #-0x28]
bl          imp___stubs__objc_release
adrp        x8, #0x100008000
add         x8, x8, #0xe10 ; @selector(defaultManager)
adrp        x9, #0x100008000
add         x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
sturb       wzr, [x29, #-0x19]
ldr         x9, x9
ldr         x1, x8
mov         x0, x9
bl          imp___stubs__objc_msgSend
mov         x29, x29
bl          imp___stubs__objc_retainAutoreleasedReturnValue
adrp        x8, #0x100008000
add         x8, x8, #0xe18 ; @selector(fileExistsAtPath:)
ldr         x1, x8
mov         x2, x8
adrp        x8, #0x100008000
add         x8, x8, #0x98 ; @"/Applications/Cydia.app"
str         x0, [sp, #0x30]
mov         x0, x2
mov         x2, x8
bl          imp___stubs__objc_msgSend
ldr         x1, [sp, #0x30]
str         w0, [sp, #0x2c]
mov         x0, x1
bl          imp___stubs__objc_release
ldr         w10, [sp, #0x2c]
tbz         w10, 0x0, loc_10000642c
```

```
        loc_10000642c:
adrp        x8, #0x100008000 ; CODE XREF=-[ViewController checkJailbreakStatusButton:]+224
add         x8, x8, #0xe10 ; @selector(defaultManager)
adrp        x9, #0x100008000
add         x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
ldr         x9, x9
ldr         x1, x8
mov         x0, x9
bl          imp___stubs__objc_msgSend
mov         x29, x29
bl          imp___stubs__objc_retainAutoreleasedReturnValue
adrp        x8, #0x100008000
add         x8, x8, #0xe18 ; @selector(fileExistsAtPath:)
ldr         x1, x8
mov         x8, x8
```

Analysis segment External Symbols

6. Use this CFG and map it to the appropriate Pseudo code.



```
mov         x0, x1
bl          imp___stubs__objc_release
ldr         w10, [sp, #0x1c]
tbz         w10, 0x0, loc_10000649c
```

```
loc_10000649c:
adrp        x8, #0x100008000 ; CODE XREF=-[ViewController checkJailbreakStatusButton:]+336
add         x8, x8, #0xe10 ; @selector(defaultManager)
adrp        x9, #0x100008000
add         x9, x9, #0xe30 ; objc_cls_ref_NSFileManager
ldr         x9, x9
ldr         x1, x8
mov         x0, x9
bl          imp___stubs__objc_msgSend
mov         x29, x29
bl          imp___stubs__objc_retainAutoreleasedReturnValue
adrp        x8, #0x100008000
add         x8, x8, #0xe18 ; @selector(fileExistsAtPath:)
ldr         x1, x8
mov         x8, x0
adrp        x2, #0x100008000
add         x2, x2, #0xd8 ; @"/usr/sbin/sshd"
str         x0, [sp, #0x10]
mov         x0, x8
bl          imp___stubs__objc_msgSend
ldr         x1, [sp, #0x10]
str         w0, [sp, #0xc]
mov         x0, x1
bl          imp___stubs__objc_release
ldr         w10, [sp, #0xc]
tbz         w10, 0x0, loc_100006508
```

```
orr         w8, wzr, #0x1
sturb       w8, [x29, #-0x19]
b           loc_100006510
```

```
orr         w8, wzr, #0x1
sturb       w8, [x29, #-0x19]
```

```
orr         w8, wzr, #0x1
sturb       w8, [x29, #-0x19]
b           loc_10000650c
```

```
loc_100006508:
b           loc_10000650c ; CODE XREF=-[ViewController checkJailbreakStatusButton:]+448
```

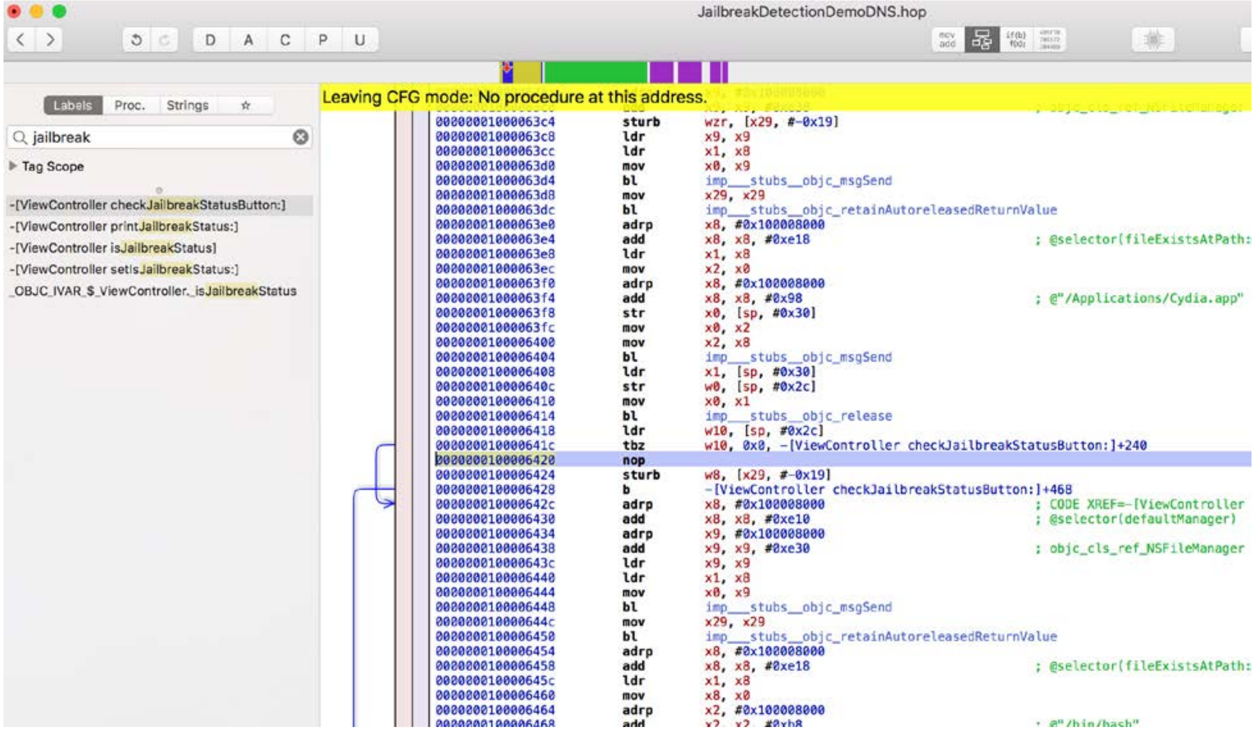
```
loc_10000650c:
b           loc_100006510 ; CODE XREF=-[ViewController checkJailbreakStatusButton:]+348, loc_100006508
```

```
loc_100006510:
adrp        x8, #0x100008000 ; CODE XREF=-[ViewController checkJailbreakStatusButton:]+236, loc_10000650c
```

Note the CFG and the 3 nodes that could affect the branching conditions. To bypass the jailbreak detection, use any of the methods below:

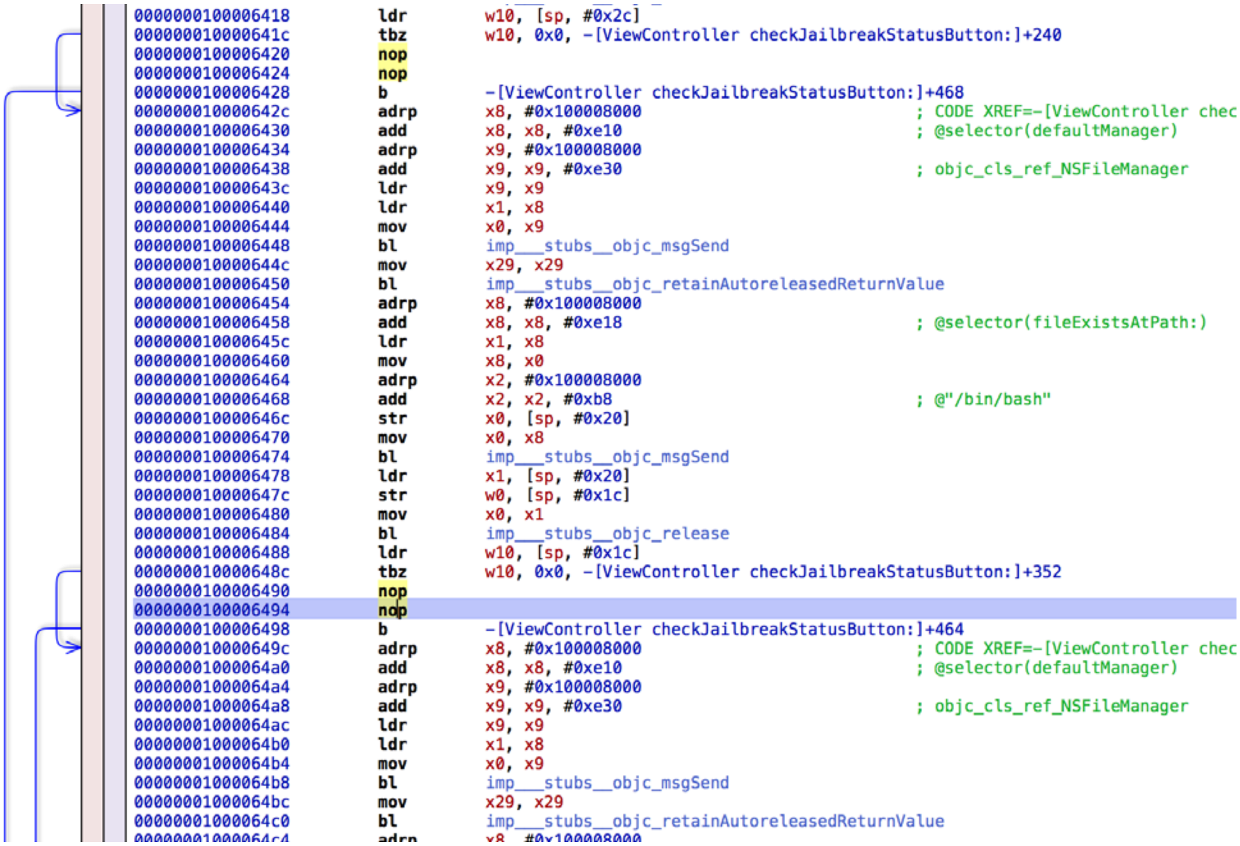
a) Convert the “tbz” in the CFG to “tbnz”
b) Convert “tbz” to “jmp address”
c) Convert “tbz” to “mov w10, 0x0”
d) NOP out all the operations that set the jailbreak status boolean to true.

7. NOP out all the operations that set the jailbreak status boolean to true. Do this by selecting the instruction and choosing NOP from Modify -> NOP region.

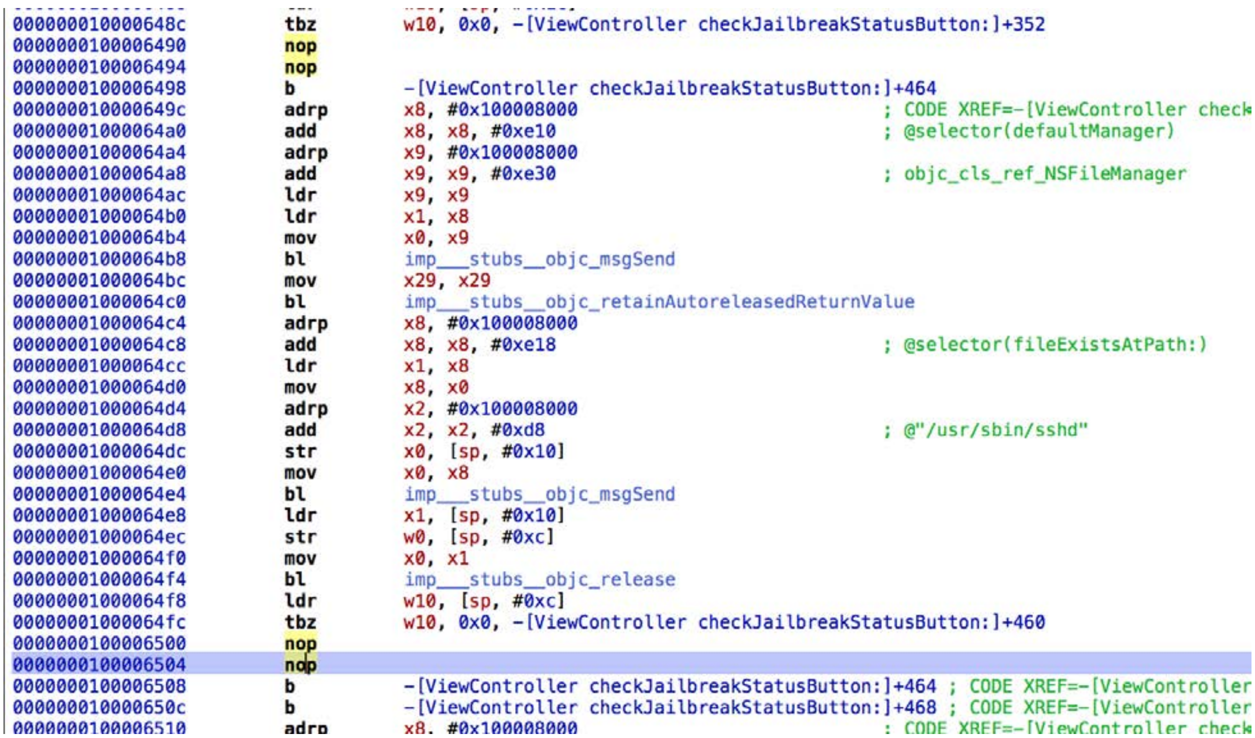

JailbreakDetectionDemoDNS.hop
D
A
C
P
U
Labels
Proc.
Strings
☆
jailbreak
Tag Scope
-[ViewController checkJailbreakStatusButton:]
-[ViewController printJailbreakStatus:]
-[ViewController isJailbreakStatus]
-[ViewController setIsJailbreakStatus:]
_OBJC_IVAR_$_ViewController._isJailbreakStatus

Leaving CFG mode: No procedure at this address.

```
00000001000063c4         sturb      wzr, [x29, #-0x19]
00000001000063c8         ldr        x9, x9
00000001000063cc         ldr        x1, x8
00000001000063d0         mov        x0, x9
00000001000063d4         bl         imp___stubs__objc_msgSend
00000001000063d8         mov        x29, x29
00000001000063dc         bl         imp___stubs__objc_retainAutoreleasedReturnValue
00000001000063e0         adrp       x8, #0x100008000
00000001000063e4         add        x8, x8, #0xe18                            ; @selector(fileExistsAtPath:
00000001000063e8         ldr        x1, x8
00000001000063ec         mov        x2, x0
00000001000063f0         adrp       x8, #0x100008000
00000001000063f4         add        x8, x8, #0x98                             ; @"/Applications/Cydia.app"
00000001000063f8         str        x0, [sp, #0x30]
00000001000063fc         mov        x0, x2
0000000100006400         mov        x2, x8
0000000100006404         bl         imp___stubs__objc_msgSend
0000000100006408         ldr        x1, [sp, #0x30]
000000010000640c         str        w0, [sp, #0x2c]
0000000100006410         mov        x0, x1
0000000100006414         bl         imp___stubs__objc_release
0000000100006418         ldr        w10, [sp, #0x2c]
000000010000641c         tbz        w10, 0x0, -[ViewController checkJailbreakStatusButton:]+240
0000000100006420         nop
0000000100006424         sturb      w8, [x29, #-0x19]
0000000100006428         b          -[ViewController checkJailbreakStatusButton:]+468
000000010000642c         adrp       x8, #0x100008000                          ; CODE XREF=-[ViewController
0000000100006430         add        x8, x8, #0xe10                            ; @selector(defaultManager)
0000000100006434         adrp       x9, #0x100008000
0000000100006438         add        x9, x9, #0xe30                            ; objc_cls_ref_NSFileManager
000000010000643c         ldr        x9, x9
0000000100006440         ldr        x1, x8
0000000100006444         mov        x0, x9
0000000100006448         bl         imp___stubs__objc_msgSend
000000010000644c         mov        x29, x29
0000000100006450         bl         imp___stubs__objc_retainAutoreleasedReturnValue
0000000100006454         adrp       x8, #0x100008000
0000000100006458         add        x8, x8, #0xe18                            ; @selector(fileExistsAtPath:
000000010000645c         ldr        x1, x8
0000000100006460         mov        x8, x0
0000000100006464         adrp       x2, #0x100008000
0000000100006468         add        x2, x2, #0xb8                             ; @"/bin/bash"
```

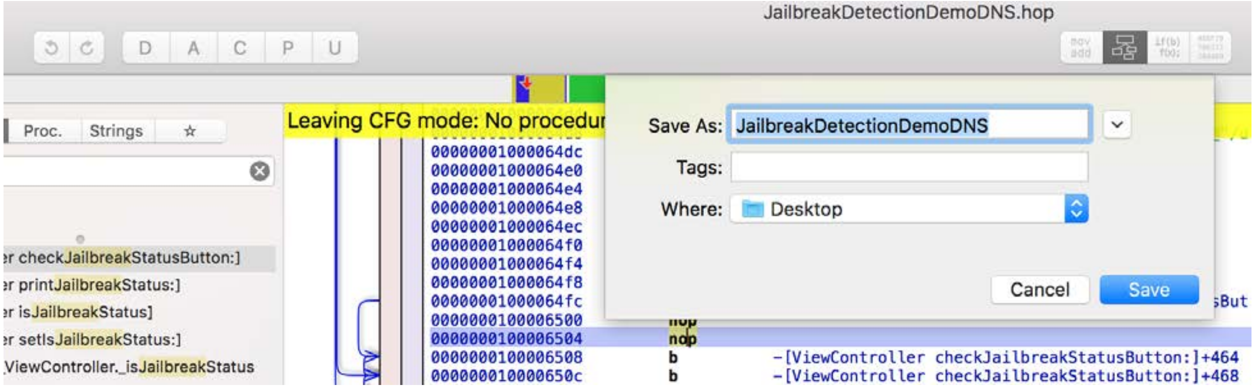


```
0000000100006418         ldr      w10, [sp, #0x2c]
000000010000641c         tbz      w10, 0x0, -[ViewController checkJailbreakStatusButton:]+240
0000000100006420         nop
0000000100006424         nop
0000000100006428         b        -[ViewController checkJailbreakStatusButton:]+468
000000010000642c         adrp     x8, #0x100008000                           ; CODE XREF=-[ViewController chec
0000000100006430         add      x8, x8, #0xe10                             ; @selector(defaultManager)
0000000100006434         adrp     x9, #0x100008000
0000000100006438         add      x9, x9, #0xe30                             ; objc_cls_ref_NSFileManager
000000010000643c         ldr      x9, x9
0000000100006440         ldr      x1, x8
0000000100006444         mov      x0, x9
0000000100006448         bl       imp___stubs__objc_msgSend
000000010000644c         mov      x29, x29
0000000100006450         bl       imp___stubs__objc_retainAutoreleasedReturnValue
0000000100006454         adrp     x8, #0x100008000
0000000100006458         add      x8, x8, #0xe18                             ; @selector(fileExistsAtPath:)
000000010000645c         ldr      x1, x8
0000000100006460         mov      x8, x0
0000000100006464         adrp     x2, #0x100008000
0000000100006468         add      x2, x2, #0xb8                              ; @"/bin/bash"
000000010000646c         str      x0, [sp, #0x20]
0000000100006470         mov      x0, x8
0000000100006474         bl       imp___stubs__objc_msgSend
0000000100006478         ldr      x1, [sp, #0x20]
000000010000647c         str      w0, [sp, #0x1c]
0000000100006480         mov      x0, x1
0000000100006484         bl       imp___stubs__objc_release
0000000100006488         ldr      w10, [sp, #0x1c]
000000010000648c         tbz      w10, 0x0, -[ViewController checkJailbreakStatusButton:]+352
0000000100006490         nop
0000000100006494         nop
0000000100006498         b        -[ViewController checkJailbreakStatusButton:]+464
000000010000649c         adrp     x8, #0x100008000                           ; CODE XREF=-[ViewController chec
00000001000064a0         add      x8, x8, #0xe10                             ; @selector(defaultManager)
00000001000064a4         adrp     x9, #0x100008000
00000001000064a8         add      x9, x9, #0xe30                             ; objc_cls_ref_NSFileManager
00000001000064ac         ldr      x9, x9
00000001000064b0         ldr      x1, x8
00000001000064b4         mov      x0, x9
00000001000064b8         bl       imp___stubs__objc_msgSend
00000001000064bc         mov      x29, x29
00000001000064c0         bl       imp___stubs__objc_retainAutoreleasedReturnValue
00000001000064c4         adrp     x8, #0x100008000
```



```
000000010000648c         tbz        w10, 0x0, -[ViewController checkJailbreakStatusButton:]+352
0000000100006490         nop
0000000100006494         nop
0000000100006498         b          -[ViewController checkJailbreakStatusButton:]+464
000000010000649c         adrp       x8, #0x100008000                              ; CODE XREF=-[ViewController check
00000001000064a0         add        x8, x8, #0xe10                                ; @selector(defaultManager)
00000001000064a4         adrp       x9, #0x100008000
00000001000064a8         add        x9, x9, #0xe30                                ; objc_cls_ref_NSFileManager
00000001000064ac         ldr        x9, x9
00000001000064b0         ldr        x1, x8
00000001000064b4         mov        x0, x9
00000001000064b8         bl         imp___stubs__objc_msgSend
00000001000064bc         mov        x29, x29
00000001000064c0         bl         imp___stubs__objc_retainAutoreleasedReturnValue
00000001000064c4         adrp       x8, #0x100008000
00000001000064c8         add        x8, x8, #0xe18                                ; @selector(fileExistsAtPath:)
00000001000064cc         ldr        x1, x8
00000001000064d0         mov        x8, x0
00000001000064d4         adrp       x2, #0x100008000
00000001000064d8         add        x2, x2, #0xd8                                 ; @"/usr/sbin/sshd"
00000001000064dc         str        x0, [sp, #0x10]
00000001000064e0         mov        x0, x8
00000001000064e4         bl         imp___stubs__objc_msgSend
00000001000064e8         ldr        x1, [sp, #0x10]
00000001000064ec         str        w0, [sp, #0xc]
00000001000064f0         mov        x0, x1
00000001000064f4         bl         imp___stubs__objc_release
00000001000064f8         ldr        w10, [sp, #0xc]
00000001000064fc         tbz        w10, 0x0, -[ViewController checkJailbreakStatusButton:]+460
0000000100006500         nop
0000000100006504         nop
0000000100006508         b          -[ViewController checkJailbreakStatusButton:]+464 ; CODE XREF=-[ViewController
000000010000650c         b          -[ViewController checkJailbreakStatusButton:]+468 ; CODE XREF=-[ViewController
0000000100006510         adrp       x8, #0x100008000                              ; CODE XREF=-[ViewController check
```

8. Select File -> Produce New Executable to save the modified binary.



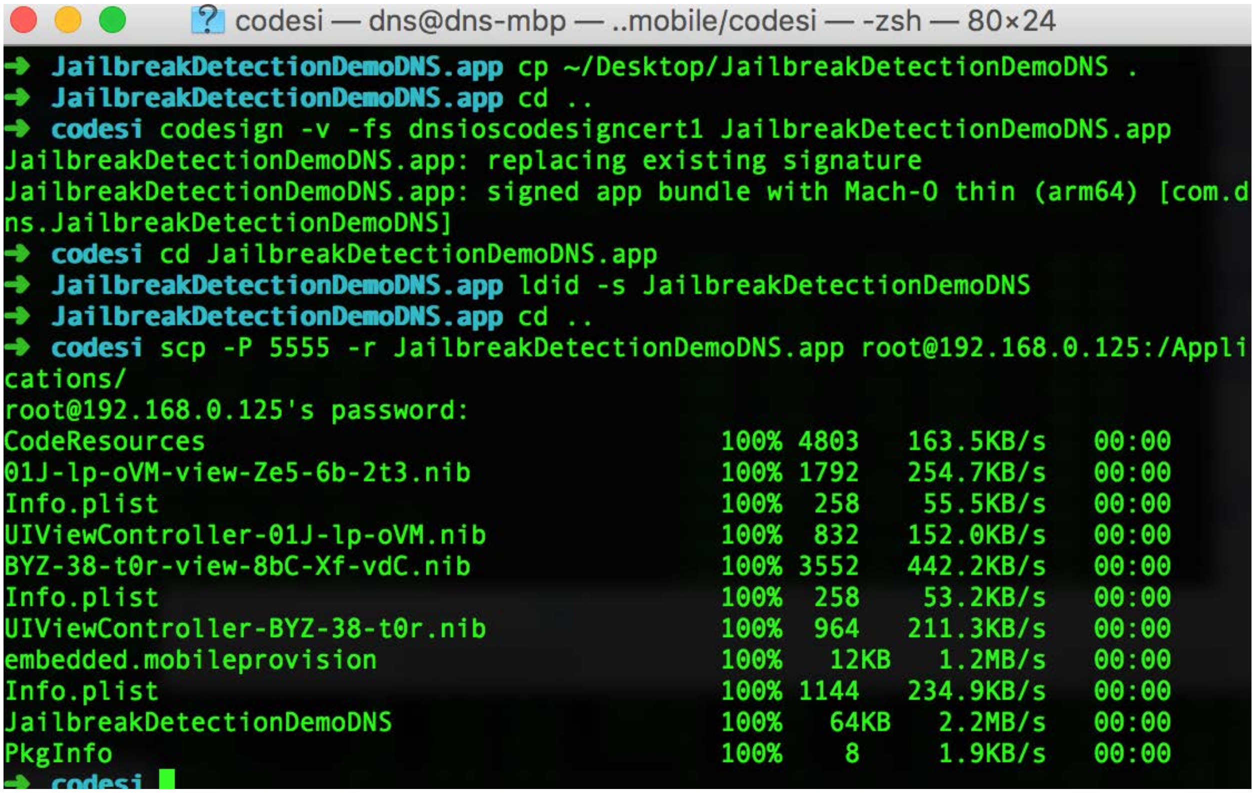
Save As: JailbreakDetectionDemoDNS

Tags:

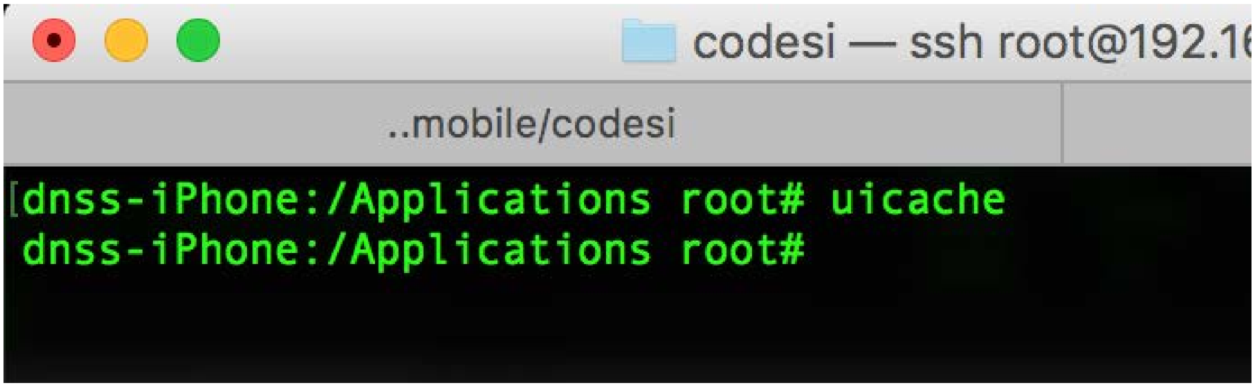
Where: Desktop

```
r checkJailbreakStatusButton:]
r printJailbreakStatus:]
r isJailbreakStatus]
r setIsJailbreakStatus:]
ViewController._isJailbreakStatus

00000001000064dc
00000001000064e0
00000001000064e4
00000001000064e8
00000001000064ec
00000001000064f0
00000001000064f4
00000001000064f8
00000001000064fc
0000000100006500        nop
0000000100006504        nop
0000000100006508        b        -[ViewController checkJailbreakStatusButton:]+464
000000010000650c        b        -[ViewController checkJailbreakStatusButton:]+468
```

9. Install the modified binary back on the iOS device using the steps mentioned in “3.2 Method II - Without a valid developer account” and “Module 4” (modified .app method).



```
➜ JailbreakDetectionDemoDNS.app cp ~/Desktop/JailbreakDetectionDemoDNS .
➜ JailbreakDetectionDemoDNS.app cd ..
➜ codesi codesign -v -fs dnsioscodesigncert1 JailbreakDetectionDemoDNS.app
JailbreakDetectionDemoDNS.app: replacing existing signature
JailbreakDetectionDemoDNS.app: signed app bundle with Mach-O thin (arm64) [com.d
ns.JailbreakDetectionDemoDNS]
➜ codesi cd JailbreakDetectionDemoDNS.app
➜ JailbreakDetectionDemoDNS.app ldid -s JailbreakDetectionDemoDNS
➜ JailbreakDetectionDemoDNS.app cd ..
➜ codesi scp -P 5555 -r JailbreakDetectionDemoDNS.app root@192.168.0.125:/Appli
cations/
root@192.168.0.125's password:
CodeResources                                 100% 4803   163.5KB/s   00:00
01J-lp-oVM-view-Ze5-6b-2t3.nib                100% 1792   254.7KB/s   00:00
Info.plist                                    100%  258    55.5KB/s   00:00
UIViewController-01J-lp-oVM.nib               100%  832   152.0KB/s   00:00
BYZ-38-t0r-view-8bC-Xf-vdC.nib                100% 3552   442.2KB/s   00:00
Info.plist                                    100%  258    53.2KB/s   00:00
UIViewController-BYZ-38-t0r.nib               100%  964   211.3KB/s   00:00
embedded.mobileprovision                      100%   12KB   1.2MB/s   00:00
Info.plist                                    100% 1144   234.9KB/s   00:00
JailbreakDetectionDemoDNS                     100%   64KB   2.2MB/s   00:00
PkgInfo                                       100%    8     1.9KB/s   00:00
➜ codesi
```



```
[dnss-iPhone:/Applications root# uicache
 dnss-iPhone:/Applications root#
```

10. After the application is installed launch the application and click the button to check the jailbreak status. Note that the application now states that the device is not jailbroken - indicating that the patching was successful.

Note: If you notice that the application crashes when opened, instead of using the .app approach of installing the application - use the .IPA method and install the IPA on the device using Cydia Impactor

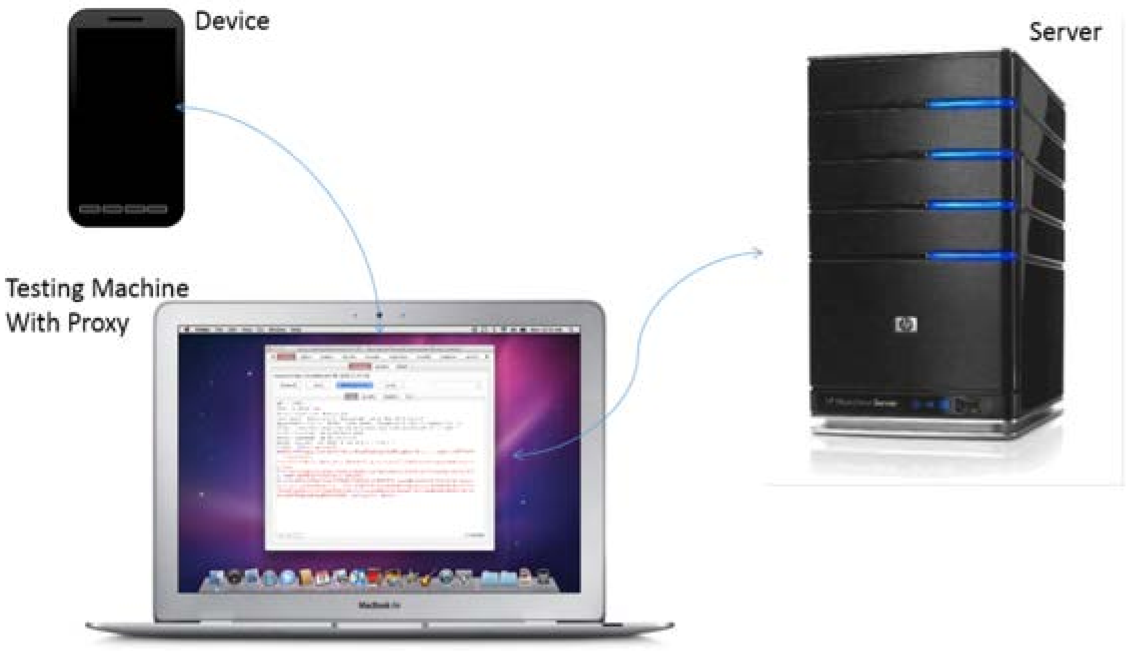
14. Reverse Engineering Using IDA PRO

Hex-rays has an excellent document on performing reverse engineering using IDA PRO. Find it at the link below:
https://www.hex-rays.com/products/ida/support/tutorials/ios_debugger_tutorial.pdf

15. MITM on iOS

The basic intention of an interception proxy is to determine whether or not a transmission channel is secure. Unlike traditional web applications, the protocol used by a mobile application can vary. Additionally, there are additional channels to consider like 4g, GPRS, SMS, USSD, Bluetooth, NFC etc. during testing.

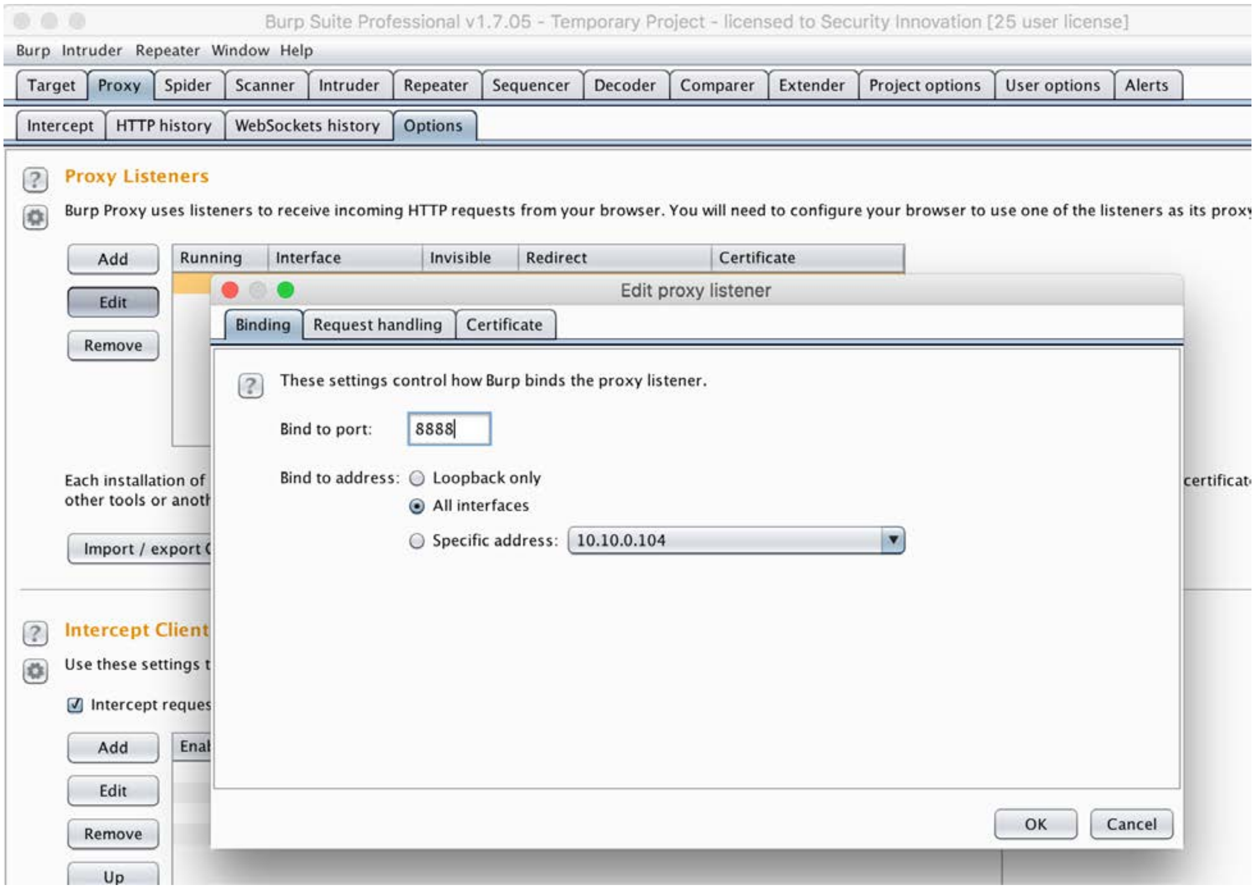
The simplest setup for intercepting iOS traffic is shown below:



The aim is to make sure that any data that leaves the iOS device should go through the laptop. This laptop can be used for sniffing the traffic or for data manipulation.

15.1 MITM HTTP Traffic

1. Launch Burp Suite and configure it to listen for traffic on all interfaces. Set the port number to a random number.



Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Running
Interface
Invisible
Redirect
Certificate
Import / export C
Add
Edit
Remove
Up
Enal
Edit proxy listener
Binding Request handling Certificate
OK
Cancel

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy

Each installation of ... other tools or anoth ... certificat

Intercept Client

Use these settings t

☑ Intercept reques

These settings control how Burp binds the proxy listener.

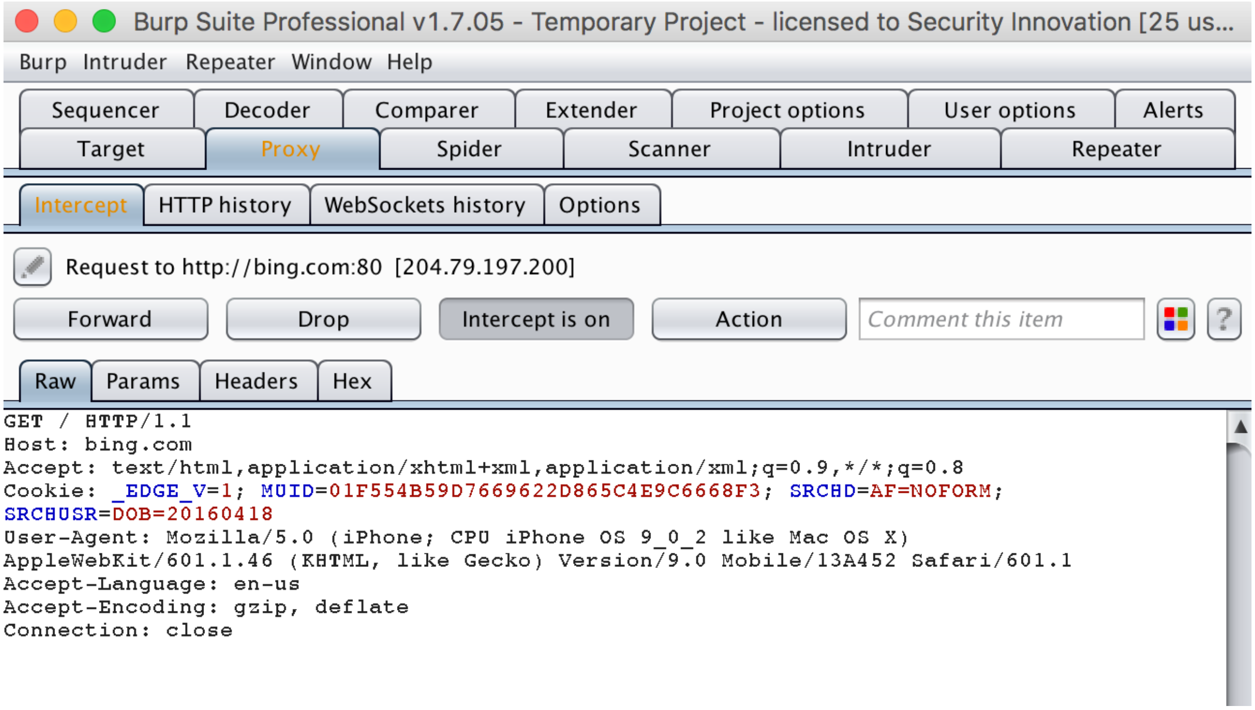
Bind to port: 8888

Bind to address: ○ Loopback only

● All interfaces

○ Specific address: 10.10.0.104

2. On the iOS device, go to Settings -> WiFi and connect to the same WiFi network as the laptop.
3. Click on the exclamation mark beside the network connection.
4. Select “Manual” in the HTTP PROXY section.
5. Set the Server to use the laptop as the proxy and the port on which Burp Suite is running.
6. Note that the traffic is intercepted in Burp Suite without any issues.

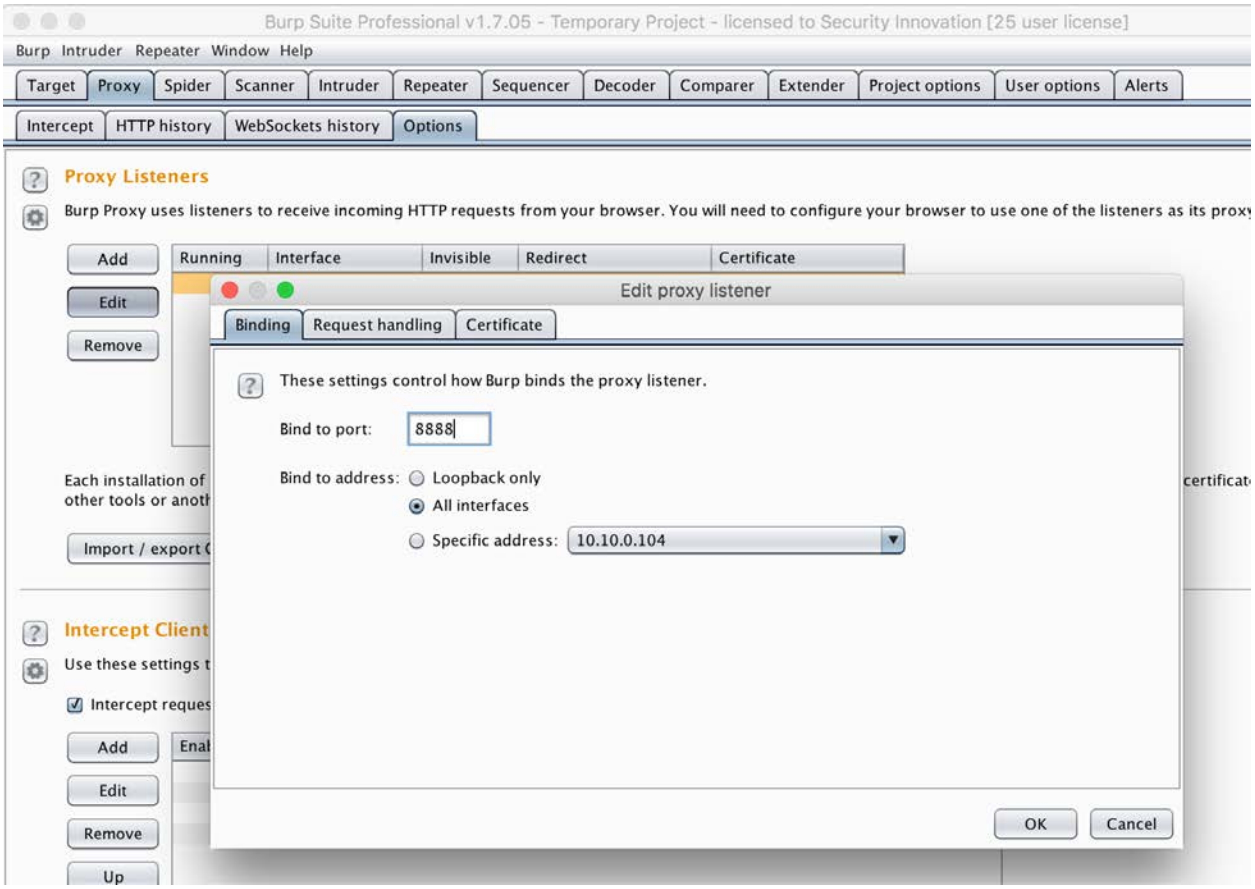


Request to http://bing.com:80 [204.79.197.200]

```
GET / HTTP/1.1
Host: bing.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Cookie: _EDGE_V=1; MUID=01F554B59D7669622D865C4E9C6668F3; SRCHD=AF=NOFORM;
SRCHUSR=DOB=20160418
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_0_2 like Mac OS X)
AppleWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13A452 Safari/601.1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: close
```

15.2 MITM SSL/TLS Traffic

1. Launch Burp Suite and configure it to listen for traffic on all interfaces. Set the port number to a random number.


Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 user license]
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Add
Edit
Remove
Running Interface Invisible Redirect Certificate
Import / export C
Add
Edit
Remove
Up
Enal
Edit proxy listener
Binding Request handling Certificate
OK
Cancel

Proxy Listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy

Each installation of … other tools or anoth … certificat

Intercept Client

Use these settings t

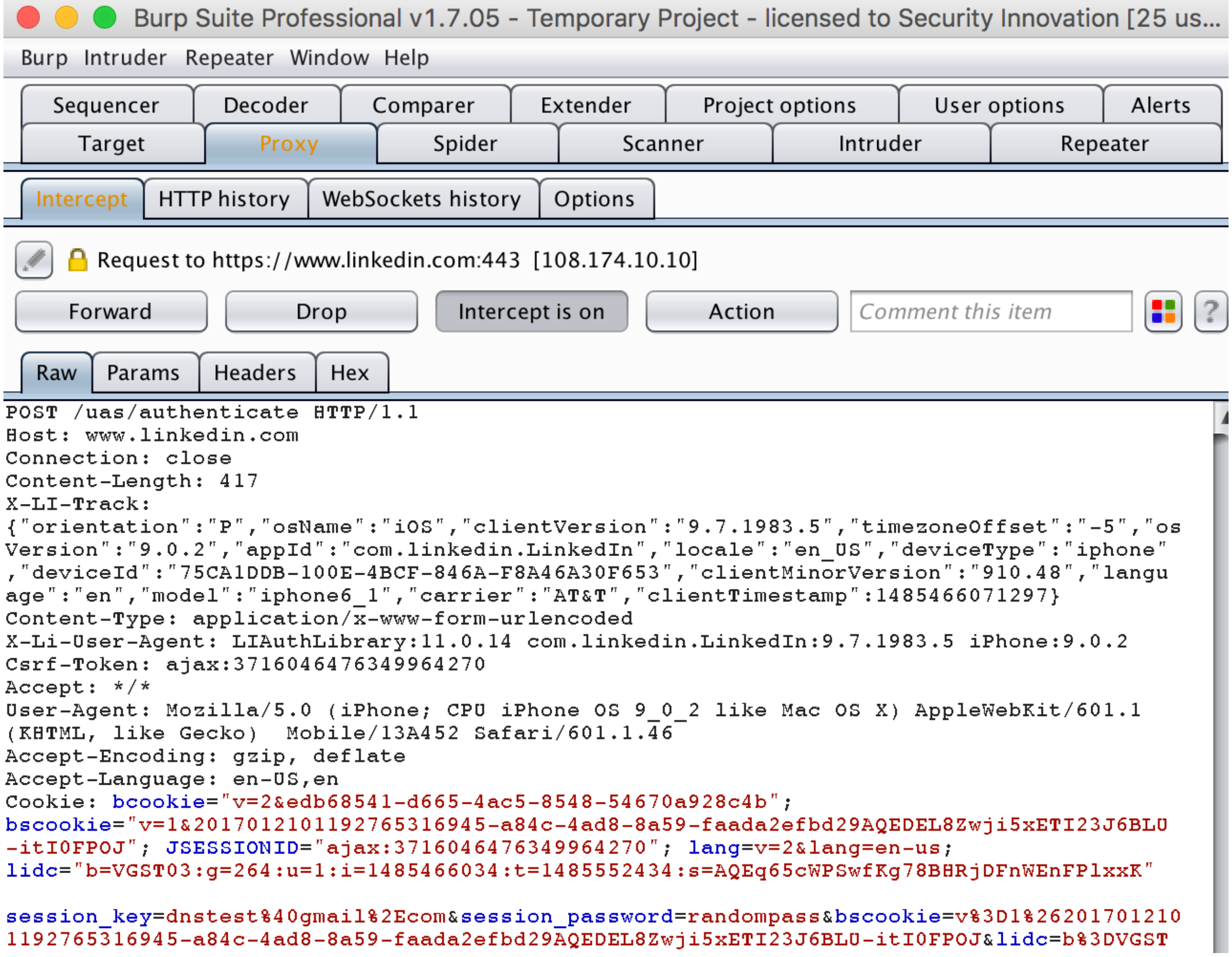
☑ Intercept reques

These settings control how Burp binds the proxy listener.

Bind to port: 8888

Bind to address: ○ Loopback only
● All interfaces
○ Specific address: 10.10.0.104

2. On the iOS device, go to Settings -> WiFi and connect to the same WiFi network as the laptop.
3. Click on the exclamation mark beside the network connection.
4. Select "Manual" in the HTTP PROXY section.
5. Set the Server to use your laptop as the proxy and the port on which Burp Suite is running.
6. In the mobile browser navigate to http://burp
7. Click on CA Certificate
8. When prompted, install the PortSwigger CA certificate.
9. Launch the LinkedIn application and note that TLS traffic is intercepted in Burp Suite without any issues



Request to https://www.linkedin.com:443 [108.174.10.10]

```
POST /uas/authenticate HTTP/1.1
Host: www.linkedin.com
Connection: close
Content-Length: 417
X-LI-Track:
{"orientation":"P","osName":"iOS","clientVersion":"9.7.1983.5","timezoneOffset":"-5","os
Version":"9.0.2","appId":"com.linkedin.LinkedIn","locale":"en_US","deviceType":"iphone"
,"deviceId":"75CA1DDB-100E-4BCF-846A-F8A46A30F653","clientMinorVersion":"910.48","langu
age":"en","model":"iphone6_1","carrier":"AT&T","clientTimestamp":1485466071297}
Content-Type: application/x-www-form-urlencoded
X-Li-User-Agent: LIAuthLibrary:11.0.14 com.linkedin.LinkedIn:9.7.1983.5 iPhone:9.0.2
Csrf-Token: ajax:3716046476349964270
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_0_2 like Mac OS X) AppleWebKit/601.1
(KHTML, like Gecko)  Mobile/13A452 Safari/601.1.46
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en
Cookie: bcookie="v=2&edb68541-d665-4ac5-8548-54670a928c4b";
bscookie="v=1&2017012101192765316945-a84c-4ad8-8a59-faada2efbd29AQEDEL8Zwji5xETI23J6BLU
-itI0FPOJ"; JSESSIONID="ajax:3716046476349964270"; lang=v=2&lang=en-us;
lidc="b=VGST03:g=264:u=1:i=1485466034:t=1485552434:s=AQEq65cWPSwfKg78BHRjDFnWEnFPlxxK"

session_key=dnstest%40gmail%2Ecom&session_password=randompass&bscookie=v%3D1%26201701210
1192765316945-a84c-4ad8-8a59-faada2efbd29AQEDEL8Zwji5xETI23J6BLU-itI0FPOJ&lidc=b%3DVGST
```

15.3 MITM non HTTP/SSL/TLS Traffic

For non HTTP/(s) traffic use Mallory. You can refer to the guide below on installing and using Mallory: https://www.pentestpartners.com/blog/advanced-traffic-interception-for-mobile-apps-using-mallory-and-burp/

15.4 MITM using VPN

in a Mobile Device Management (MDM) environment there is often the challenge of showing a customer how to intercept traffic without connecting to the WiFi on the iOS device. This can be done by means of iOS VPN.

1. On the linux laptop that will act as the VPN server, install the pptpd package using the command below:
    - sudo apt-get install pptpd
2. Edit the /etc/pptfpd.conf file to limit the ip range of addresses allocated to VPN clients (here, lan range is 10.10.1.0/24). Set the VPN client to use 10.10.10.1 as gateway
    - localip 10.10.10.1
    - remoteip 10.10.10.100-254
3. Edit the /etc/ppp/pptpd-options and configure the DNS server
    - ms-dns 8.8.8.8
    - ms-dns 8.8.4.4
4. Edit the /etc/ppp/chap-secrets configuration file and set up the credentials
    - #   client server secret IP addresses
    - dinesh pptpd admin   *
        - Here admin/admin is the credential and pptpd is the type of service
5. Set up IP forwarding using the command below
    - sysctl -w net.ipv4.ip_forward=1
6. Set up IP masquerading using the command below
    - iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
7. Restart the VPN service using the command below:
    - /etc/init.d/pptpd restart or service pptpd restart
8. On iOS device locate the VPN settings option. Edit the PPTP settings to reflect the information below:

- Description - SomeVPNName
- Server - 10.10.1.106 (VPN server IP)(Remote IP)
- Account - dinesh
- Password - admin
- Encryption Level - Auto
- Send all Traffic - ON

9. Set PROXY to Manual and enter the proxy settings to the machine and port for Burp. Note that the traffic is now captured in Burp Suite.

15.5 MITM When iOS Application Accessible Only Via VPN

Andreas Kurtz wrote an excellent article ~~nice write-up~~ explaining how to pentest iOS applications when the application is accessible only via a VPN connection. This can happen when the backend is only accessible from an internal corporate environment and local LAN access is restricted when a VPN connection is established. Below are the steps as suggested by Kurtz.

1. Launch Burp Suite on the laptop
2. In Burp Suite, Options -> Connections -> Upstream Proxy Servers, set up Server as 127.0.0.1 and Port as 7777
3. On the iOS device, install “3proxy” from Cydia
4. SSH into the iOS device and set up the 3proxy config file at /var/root/3proxy.cfg with the below configuration:
    - log /var/root/3proxy.log D
    - Logformat “%d-%m-%Y %H:%M:%S %U %C:%c %R:%r %0 %I %T”
    - proxy -p7777 -n
5. Launch 3proxy on the iOS device using the below command:
    - 3proxy /var/root/3proxy.cfg &
6. On the iOS device in the VPN settings set up Proxy settings to Manual and set the configuration as shown below:
    - Server: 127.0.0.1
    - Port: 8080 (Burp port number)

NOTE: If due to MDM solution, it is not possible to enter the IP as 127.0.0.1 then edit /etc/hosts on the iOS device and set an alias for localhost.

7. On the laptop, since connecting to local addresses is blocked when VPN is launched, SSH into the device using SSH over USB as explained in Module 8 (reading application data using SSH over USB).
8. On the device, run the command below:
    - ssh -p 2222 -L 7777:127.0.0.1:7777 -R 8080:127.0.0.1:8080 root@127.0.0.1

Note: Refer to either of the links below for further details:

- https://andreas-kurtz.de/2013/07/ios-proxy-fight/
- http://techie-anand.blogspot.com/2015/02/how-to-intercept-traffic-from-devices.html

15.6 MITM Bypassing Certificate Pinning

Certificate pinning is the way to limit a Mobile application by allowing only the server's certificate to be trusted rather than relying on the Mobile's own Trusted Certificate Store.

Rather than relying on the device trusted store, some developers set the application to trust only the server's SSL certificate. This way, when connecting to a specific SSL/TLS server, there's no need for a third party to share the server's identity. Additionally, compromises to any of the CA in the device trusted store is not an issue as the connection no longer relies on it. However, certificate pinning can still be bypassed by means of tools like SSL Kill Switch or iOS TrustMe.

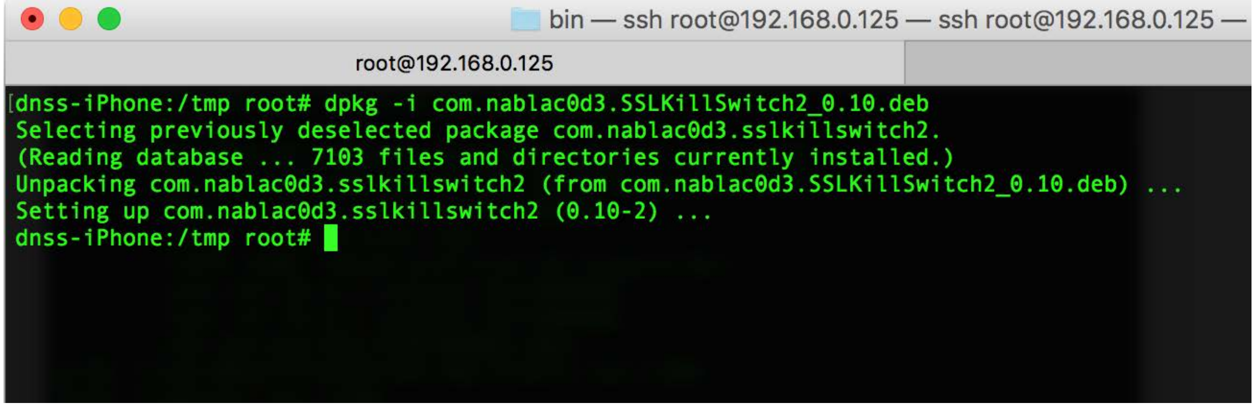
The latest version of SSL Kill Switch is v2 and can be found here: https://github.com/nabla-c0d3/ssl-kill-switch2.

Before using SSL Kill Switch 2, install the following dependencies using Cydia:

- Debian Packager
- Cydia Substrate
- PreferenceLoader

Pull the latest installation binary from https://github.com/nabla-c0d3/ssl-kill-switch2/releases into the device using the command below:

- *wget https://github.com/nabla-c0d3/ssl-kill-switch2/releases/download/0.10/com.nablac0d3.SSLKillSwitch2_0.10.deb --no-check-certificate*



```
[dnss-iPhone:/tmp root# dpkg -i com.nablac0d3.SSLKillSwitch2_0.10.deb
 Selecting previously deselected package com.nablac0d3.sslkillswitch2.
 (Reading database ... 7103 files and directories currently installed.)
 Unpacking com.nablac0d3.sslkillswitch2 (from com.nablac0d3.SSLKillSwitch2_0.10.deb) ...
 Setting up com.nablac0d3.sslkillswitch2 (0.10-2) ...
 dnss-iPhone:/tmp root#
```

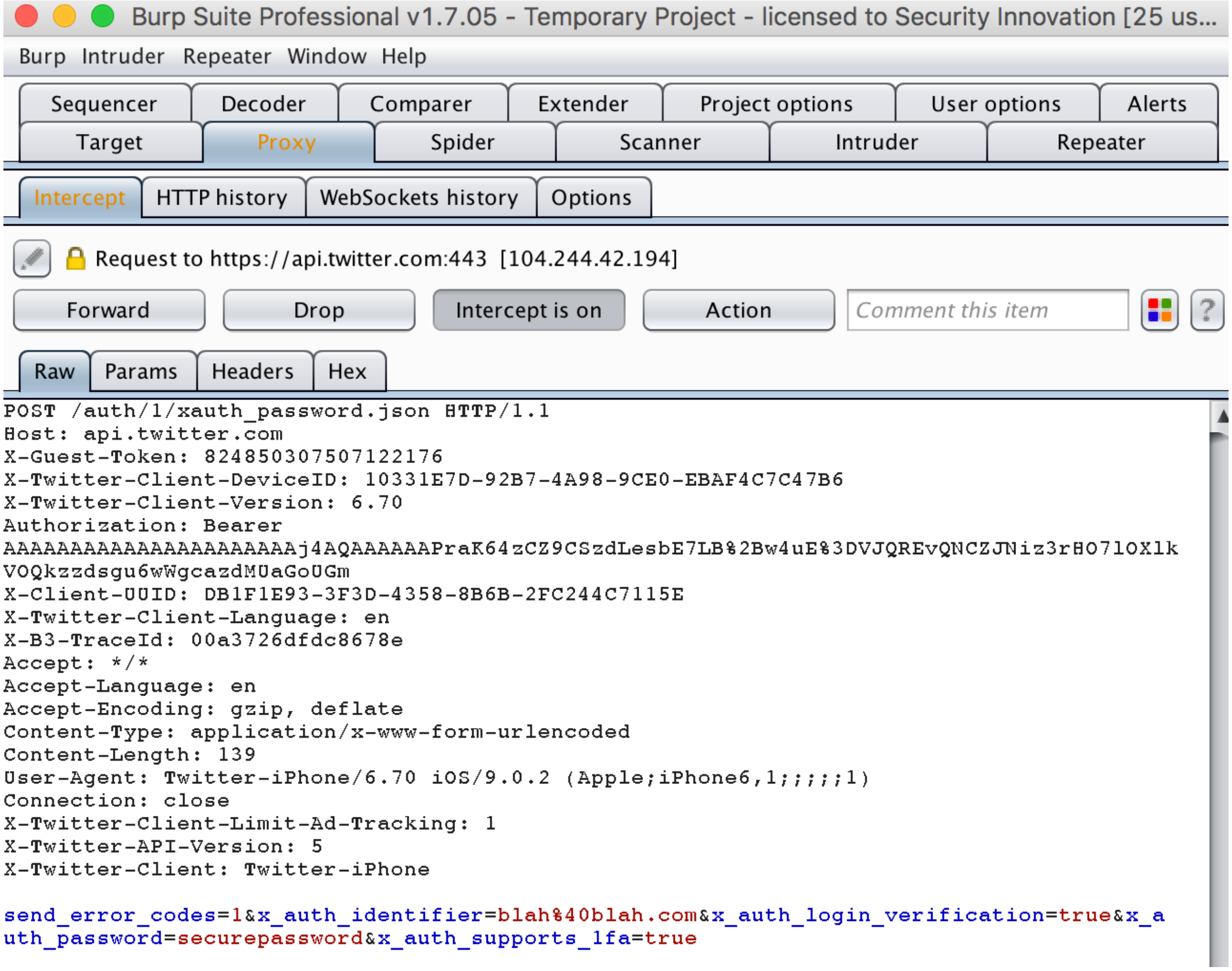
Respring the device using the below command:

- *killall -HUP SpringBoard*

In the Settings menu see an additional “SSL Kill Switch 2” option that allows disabling Certificate Validation and bypassing certificate pinning.

Application used for Example: Twitter application from AppStore

1. Set up the iOS device to proxy the traffic via Burp Suite using the steps mentioned in Module 15 (MITM SSL/TLS Traffic)
2. Launch the Twitter application on the iOS device and try to log in to the application. Note that you are restricted from logging in to the application and are prompted with a “TLS trust verification failed” error. This indicates that certificate pinning is enabled in the Twitter application.
   Also note that the requests do not reach Burp Suite.
3. Enable “Disable Certificate Validation” using SSL Kill Switch 2 from the Settings menu.
4. Relaunch the Twitter application and now try to log in to the application. Note that the certificate pinning was bypassed and the traffic now reaches Burp Suite.


Burp Suite Professional v1.7.05 - Temporary Project - licensed to Security Innovation [25 us...
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action Comment this item
Raw Params Headers Hex

Request to https://api.twitter.com:443 [104.244.42.194]

```
POST /auth/1/xauth_password.json HTTP/1.1
Host: api.twitter.com
X-Guest-Token: 824850307507122176
X-Twitter-Client-DeviceID: 10331E7D-92B7-4A98-9CE0-EBAF4C7C47B6
X-Twitter-Client-Version: 6.70
Authorization: Bearer
AAAAAAAAAAAAAAAAAAAAAAj4AQAAAAAAPraK64zCZ9CSzdLesbE7LB%2Bw4uE%3DVJQREvQNCZJNiz3rHO7lOXlk
VOQkzzdsgu6wWgcazdMUaGoUGm
X-Client-UUID: DB1F1E93-3F3D-4358-8B6B-2FC244C7115E
X-Twitter-Client-Language: en
X-B3-TraceId: 00a3726dfdc8678e
Accept: */*
Accept-Language: en
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 139
User-Agent: Twitter-iPhone/6.70 iOS/9.0.2 (Apple;iPhone6,1;;;;;1)
Connection: close
X-Twitter-Client-Limit-Ad-Tracking: 1
X-Twitter-API-Version: 5
X-Twitter-Client: Twitter-iPhone

send_error_codes=1&x_auth_identifier=blah%40blah.com&x_auth_login_verification=true&x_a
uth_password=securepassword&x_auth_supports_1fa=true
```

It is possible to log in to the application even with Burp Suite intercepting your requests.

A detailed guide for SSL Pinning can be found here:

- http://media.blackhat.com/bh-us-12/Turbo/Diquet/BH_US_12_Diqut_Osborne_Mobile_Certificate_Pinning_Slides.pdf

Another tool to use for bypassing certificate pinning on iOS is TrustMe, and it can be downloaded here:

- https://github.com/intrepidusgroup/trustme

Instructions for TrustMe are the same as for SSL Kill Switch 2.

Follow the guide below to learn to bypass open ssl pinning on iOS applications:

- https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2015/january/bypassing-openssl-certificate-pinning-in-ios-apps/

15.7 MITM by DNS Hijacking

Use DNSChef to try to MITM the iOS traffic using DNS Hijacking. Details can be found here:

- http://thesprawl.org/research/ios-data-interception/#dns-hijacking

15.7 MITM Using Network Gateway

The majority of MITM techniques have been covered in this guide. If you still find that the traffic interception fails, however, follow the guide at the link below to understand how to MITM using additional IP Table rules:

- http://thesprawl.org/research/ios-data-interception/#capturing-on-the-network-gateway

15.8 Monitoring iOS FileSystem Activities

There may be times when an application is writing to files on the file system without your knowledge. In this situation, a tool like Filemon can be helpful.

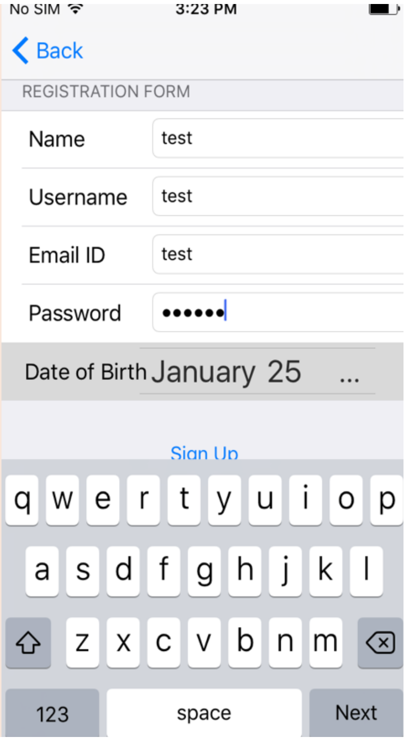
Filemon is a utility that tracks the file system activities in IOS. It shows changes made along with the operations performed in the application's file system during runtime.

Application used for Example: DemoLogin application from
https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

For Installing Filemon on a jailbroken device

- SSH to iOS device from a PC
- Download file mon from http://newosxbook.com/tools/filemon.tgz
  - wget http://newosxbook.com/tools/filemon.tgz
- Unzip the file filemon.tgz

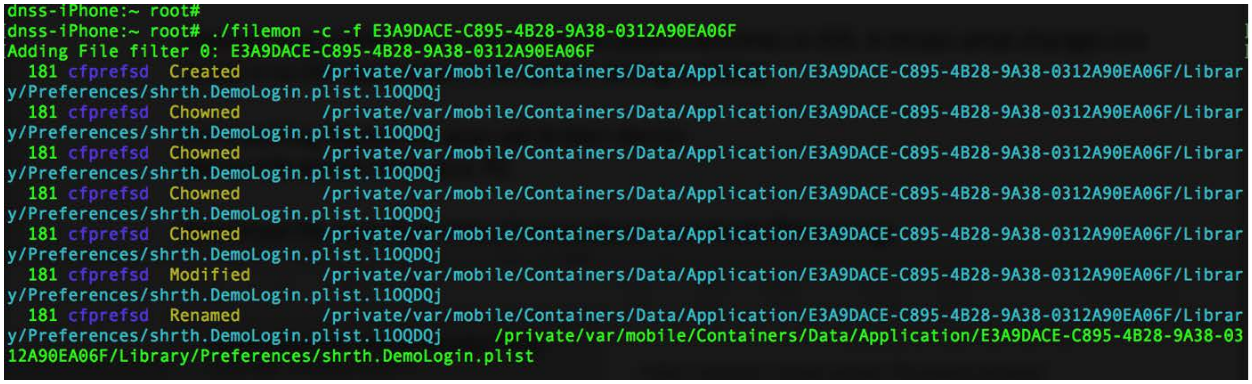
1. Run Filemon by using option –f [string] which will filter paths containing given string. In this case, the application GUID is E3A9DACE-C895-4B28-9A38-0312A90EA06F. Run the below command:
   a. ./filemon –c –f E3A9DACE-C895-4B28-9A38-0312A90EA06F"
2. Launch the "DemoLogin" application on the iOS device
3. Navigate to the "Register" screen and fill out the registration form. Then click on "Sign Up".



REGISTRATION FORM

Name	test
Username	test
Email ID	test
Password	••••••
Date of Birth	January 25 ...

4. Filemon logs and shows what files were accessed and what operations were performed during runtime:



```
dnss-iPhone:~ root#
dnss-iPhone:~ root# ./filemon -c -f E3A9DACE-C895-4B28-9A38-0312A90EA06F
Adding File filter 0: E3A9DACE-C895-4B28-9A38-0312A90EA06F
  181 cfprefsd  Created         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Chowned         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Chowned         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Chowned         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Chowned         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Modified        /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj
  181 cfprefsd  Renamed         /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-0312A90EA06F/Librar
y/Preferences/shrth.DemoLogin.plist.l1OQDQj     /private/var/mobile/Containers/Data/Application/E3A9DACE-C895-4B28-9A38-03
12A90EA06F/Library/Preferences/shrth.DemoLogin.plist
```

Opening the files indicated that the registration details were stored in plaintext on the iOS filesystem.

16. Side Channel Leakage

Side Channel Leakage happens when data is being cached on the device either by the application or by iOS itself.

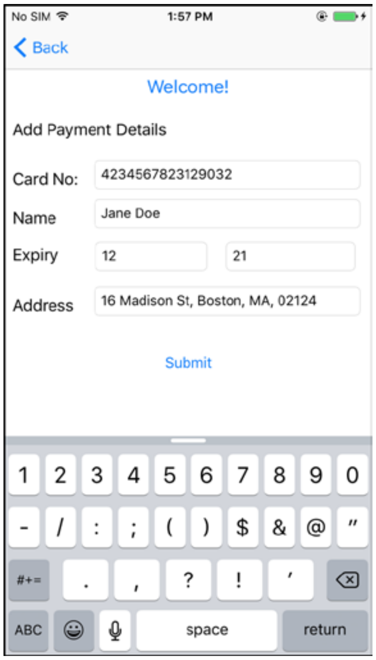
16.1 iOS Default Screen Shot Caching Mechanism

The iOS device takes a screenshot of the application's current page when the HOME key is pressed. This is used to generate an animation when the application shrinks into the background and expands back to the screen when the user opens the application again. The image is cached in local storage and can be easily accessed by an attacker who has physical access to the device. If sensitive information such as payment details, SSN or other PII are being displayed on the page during backgrounding, these can be stolen by an attacker by retrieving the screenshot file from the device.

Application used for Example: DemoLogin application from https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

Black Box Testing Approach:

1. Launch the “DemoLogin” application, register a new user, and log in to the application.
2. Enter the payment information and press the HOME button on the device.



No SIM 1:57 PM

< Back

Welcome!

Add Payment Details

Card No: 4234567823129032

Name: Jane Doe

Expiry: 12 21

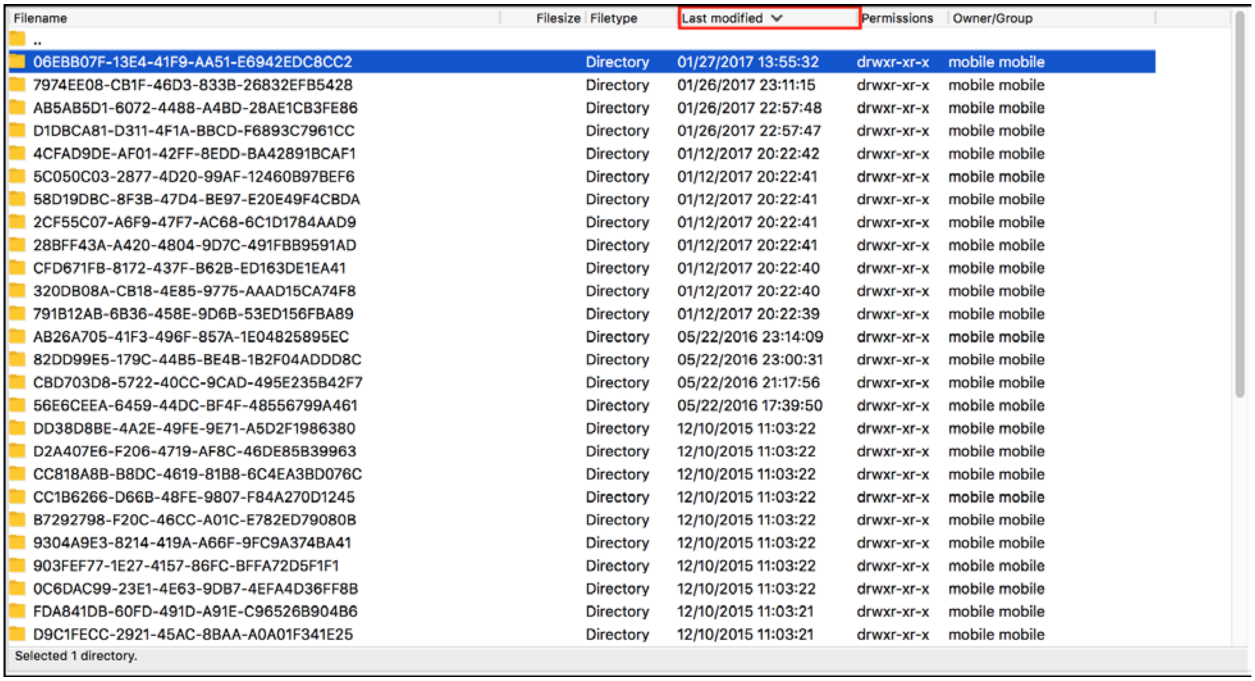
Address: 16 Madison St, Boston, MA, 02124

Submit

3. FTP to the iOS device using a client such as FileZilla to grab the screenshot from the Snapshots folder. This can be found here:
   - */private/var/mobile/Containers/Data/Application/<DemoLogin-GUID>/Library/Caches/Snapshots/*

Since the GUID in the folder name changes with each installation and/or during runtime, it is necessary to manually look into each folder to find the "DemoLogin" application folder. Some other techniques that can be employed are:

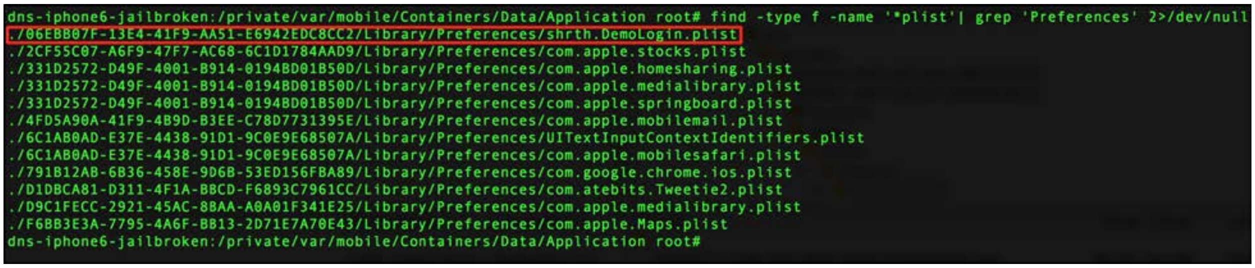
4. In FileZilla, navigate to "/private/var/mobile/Containers/Data/Application/" and sort by Last modified. Since DemoLogin was the last used application, the recently modified folder is where to find the application data such as the Screenshot file.



Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
06EBB07F-13E4-41F9-AA51-E6942EDC8CC2		Directory	01/27/2017 13:55:32	drwxr-xr-x	mobile mobile
7974EE08-CB1F-46D3-833B-26832EFB5428		Directory	01/26/2017 23:11:15	drwxr-xr-x	mobile mobile
AB5AB5D1-6072-4488-A4BD-28AE1CB3FE86		Directory	01/26/2017 22:57:48	drwxr-xr-x	mobile mobile
D1DBCA81-D311-4F1A-BBCD-F6893C7961CC		Directory	01/26/2017 22:57:47	drwxr-xr-x	mobile mobile
4CFAD9DE-AF01-42FF-8EDD-BA42891BCAF1		Directory	01/12/2017 20:22:42	drwxr-xr-x	mobile mobile
5C050C03-2877-4D20-99AF-12460B97BEF6		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
58D19DBC-8F3B-47D4-BE97-E20E49F4CBDA		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
28BFF43A-A420-4804-9D7C-491FBB9591AD		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
CFD671FB-8172-437F-B62B-ED163DE1EA41		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
320DB08A-CB18-4E85-9775-AAAD15CA74F8		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
791B12AB-6B36-458E-9D6B-53ED156FBA89		Directory	01/12/2017 20:22:39	drwxr-xr-x	mobile mobile
AB26A705-41F3-496F-857A-1E04825895EC		Directory	05/22/2016 23:14:09	drwxr-xr-x	mobile mobile
82DD99E5-179C-44B5-BE4B-1B2F04ADDD8C		Directory	05/22/2016 23:00:31	drwxr-xr-x	mobile mobile
CBD703D8-5722-40CC-9CAD-495E235B42F7		Directory	05/22/2016 21:17:56	drwxr-xr-x	mobile mobile
56E6CEEA-6459-44DC-BF4F-48556799A461		Directory	05/22/2016 17:39:50	drwxr-xr-x	mobile mobile
DD38D8BE-4A2E-49FE-9E71-A5D2F1986380		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
D2A407E6-F206-4719-AF8C-46DE85B39963		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC818A8B-B8DC-4619-81B8-6C4EA3BD076C		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC1B6266-D66B-48FE-9807-F84A270D1245		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
B7292798-F20C-46CC-A01C-E782ED79080B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
9304A9E3-8214-419A-A66F-9FC9A374BA41		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
903FEF77-1E27-4157-86FC-BFFA72D5F1F1		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
FDA841DB-60FD-491D-A91E-C96526B904B6		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile
D9C1FECC-2921-45AC-8BAA-A0A01F341E25		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile

Selected 1 directory.

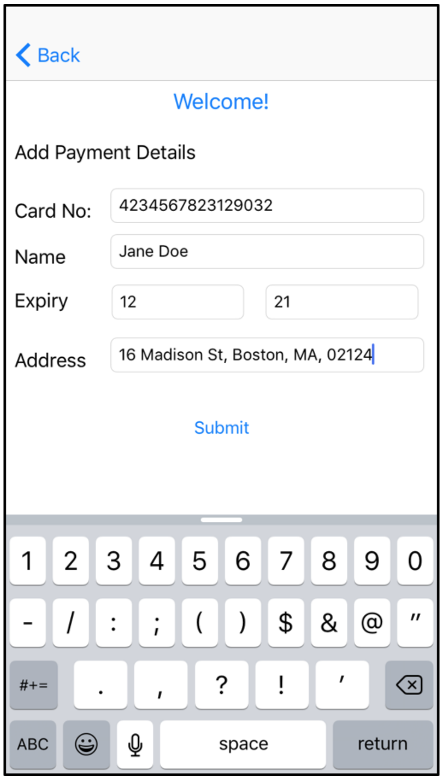
5. Alternatively, SSH into the iOS device, navigate to "/private/var/mobile/Containers/Data/Application/" and run the command below to list all the plist files. If plist files were created, we can use the plist filename to identify the application folder.
   - ○ *find -type f -name '*plist'| grep 'Preferences' 2>/dev/null*



```
dns-iphone6-jailbroken:/private/var/mobile/Containers/Data/Application root# find -type f -name '*plist'| grep 'Preferences' 2>/dev/null
./06EBB07F-13E4-41F9-AA51-E6942EDC8CC2/Library/Preferences/shrth.DemoLogin.plist
./2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9/Library/Preferences/com.apple.stocks.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.homesharing.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.medialibrary.plist
./331D2572-D49F-4001-B914-0194BD01B50D/Library/Preferences/com.apple.springboard.plist
./4FD5A90A-41F9-4B9D-B3EE-C78D7731395E/Library/Preferences/com.apple.mobilemail.plist
./6C1AB0AD-E37E-4438-91D1-9C0E9E68507A/Library/Preferences/UITextInputContextIdentifiers.plist
./6C1AB0AD-E37E-4438-91D1-9C0E9E68507A/Library/Preferences/com.apple.mobilesafari.plist
./791B12AB-6B36-458E-9D6B-53ED156FBA89/Library/Preferences/com.google.chrome.ios.plist
./D1DBCA81-D311-4F1A-BBCD-F6893C7961CC/Library/Preferences/com.atebits.Tweetie2.plist
./D9C1FECC-2921-45AC-8BAA-A0A01F341E25/Library/Preferences/com.apple.medialibrary.plist
./F6BB3E3A-7795-4A6F-BB13-2D71E7A70E43/Library/Preferences/com.apple.Maps.plist
dns-iphone6-jailbroken:/private/var/mobile/Containers/Data/Application root#
```

6. Navigate to the location below and download the screenshots.
   - */private/var/mobile/Containers/Data/Application/06EBB07F-13E4-41F9-AA51-E6942EDC8CC2/Library/Caches/Snapshots/shrth.DemoLogin*

7. The screenshot contains the payment details that were cached on the device.



Welcome!

Add Payment Details

Card No: 4234567823129032

Name Jane Doe

Expiry 12 21

Address 16 Madison St, Boston, MA, 02124

16.2 iOS UIPasteboard Caching

When text is copied in an iOS device the data goes into the buffer and this data can be used in different areas of the application as well as by other applications on the device. If any sensitive data is being copied, this data in pasteboard could be leaked to other third-party applications. A third-party application running on the device could log the contents of the pasteboard. An example code is shown below:

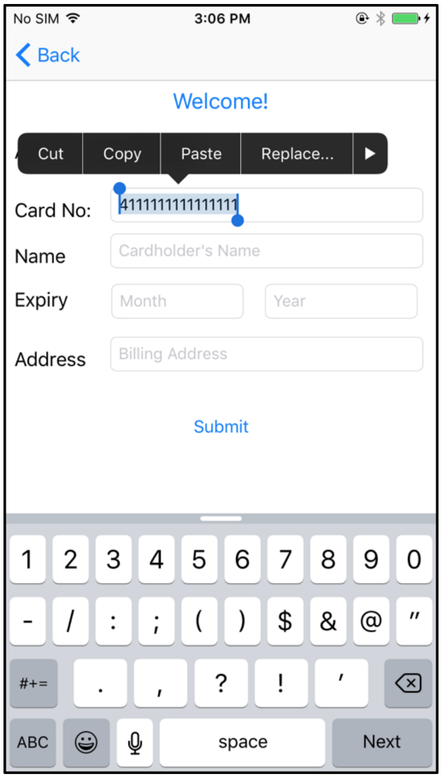
- *NSLog("@Print the contents of pasteboard: %@", [UIPasteboard generalPasteboard] string);*

During a black box pentest look for sensitive fields in the application that allow the copy-paste feature and see if they are being cached by the iOS.

Application used for Example: DemoLogin application from https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

Black Box Testing Approach:

1. Open the “DemoLogin” application, register a new user and log in to the application.
2. Enter any data in the text field and copy the content to clipboard.



No SIM 3:06 PM

Back

Welcome!

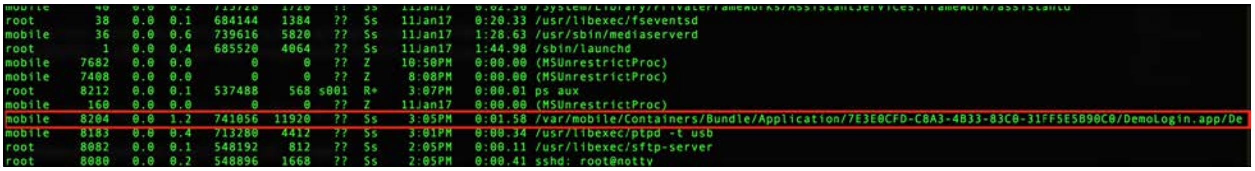
Card No: 4111111111111111

Name: Cardholder's Name

Expiry: Month Year

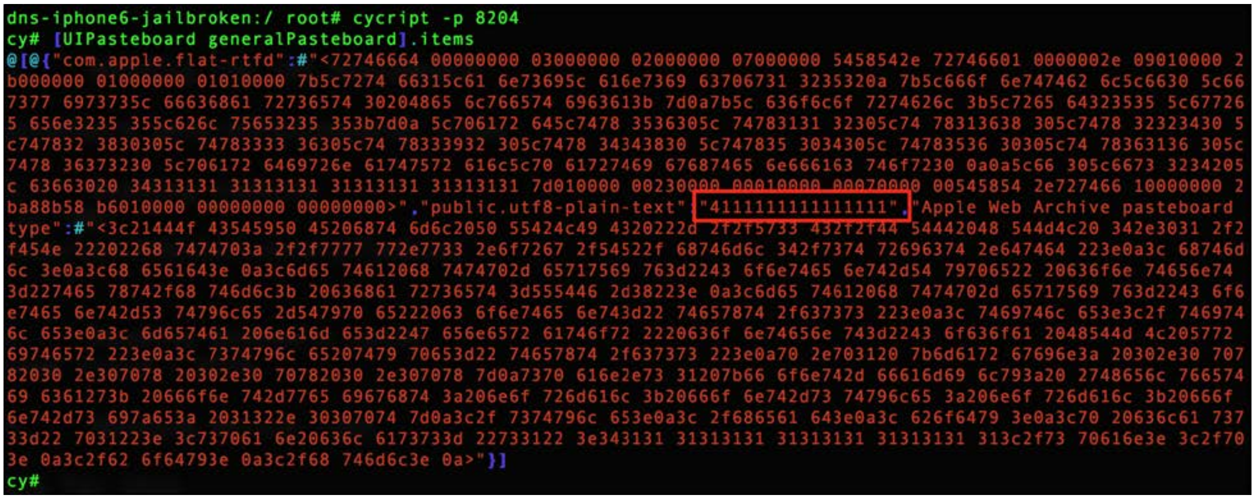
Address: Billing Address

3. SSH into the iOS device, and get the process id of the application using the command “ps aux”.
4. Use “cycript -p <process-id>” to hook on to the application. You will be prompted with the Cycript interpreter.



```
mobile      40   0.0  0.2  713720   1720   ??  Ss   11Jan17   0:02.50 /System/Library/PrivateFrameworks/AssistantServices.framework/assistantd
root        38   0.0  0.1  684144   1384   ??  Ss   11Jan17   0:20.33 /usr/libexec/fseventsd
mobile      36   0.0  0.6  739616   5820   ??  Ss   11Jan17   1:28.63 /usr/sbin/mediaserverd
root         1   0.0  0.4  685520   4064   ??  Ss   11Jan17   1:44.98 /sbin/launchd
mobile    7682   0.0  0.0       0      0   ??  Z    10:50PM   0:00.00 (MSUnrestrictProc)
mobile    7408   0.0  0.0       0      0   ??  Z     8:08PM   0:00.00 (MSUnrestrictProc)
root      8212   0.0  0.1  537488    568 s001  R+    3:07PM   0:00.01 ps aux
mobile     160   0.0  0.0       0      0   ??  Z    11Jan17   0:00.00 (MSUnrestrictProc)
mobile    8204   0.0  1.2  741056  11920   ??  Ss    3:05PM   0:01.58 /var/mobile/Containers/Bundle/Application/7E3E0CFD-C8A3-4B33-83C0-31FF5E5B90C0/DemoLogin.app/De
mobile    8183   0.0  0.4  713280   4412   ??  Ss    3:01PM   0:00.34 /usr/libexec/ptpd -t usb
root      8082   0.0  0.1  548192    812   ??  Ss    2:05PM   0:00.11 /usr/libexec/sftp-server
root      8080   0.0  0.2  548896   1668   ??  Ss    2:05PM   0:00.41 sshd: root@notty
```

5. Enter the command below to list the contents of the device clipboard. The screenshot below highlights the copied item.
   - [UIPasteboard generalPasteboard]. items



```
dns-iphone6-jailbroken:/ root# cycript -p 8204
cy# [UIPasteboard generalPasteboard].items
@[@{"com.apple.flat-rtfd":#"<72746664 00000000 03000000 02000000 07000000 5458542e 72746601 0000002e 09010000 2
b000000 01000000 01010000 7b5c7274 66315c61 6e73695c 616e7369 63706731 3235320a 7b5c666f 6e747462 6c5c6630 5c66
7377 6973735c 66636861 72736574 30204865 6c766574 6963613b 7d0a7b5c 636f6c6f 7274626c 3b5c7265 64323535 5c67726
5 656e3235 355c626c 75653235 353b7d0a 5c706172 645c7478 3536305c 74783131 32305c74 78313638 305c7478 32323430 5
c747832 3830305c 74783333 36305c74 78333932 305c7478 34343830 5c747835 3034305c 74783536 30305c74 78363136 305c
7478 36373230 5c706172 6469726e 61747572 616c5c70 61727469 67687465 6e666163 746f7230 0a0a5c66 305c6673 3234205
c 63663020 34313131 31313131 31313131 31313131 7d010000 00230000 00010000 00070000 00545854 2e727466 10000000 2
ba88b58 b6010000 00000000 00000000>","public.utf8-plain-text":"4111111111111111","Apple Web Archive pasteboard
type":#"<3c21444f 43545950 45206874 6d6c2050 55424c49 4320222d 2f2f5733 432f2f44 54442048 544d4c20 342e3031 2f2
f454e 22202268 7474703a 2f2f7777 772e7733 2e6f7267 2f54522f 68746d6c 342f7374 72696374 2e647464 223e0a3c 68746d
6c 3e0a3c68 6561643e 0a3c6d65 74612068 7474702d 65717569 763d2243 6f6e7465 6e742d54 79706522 20636f6e 74656e74
3d227465 78742f68 746d6c3b 20636861 72736574 3d555446 2d38223e 0a3c6d65 74612068 7474702d 65717569 763d2243 6f6
e7465 6e742d53 74796c65 2d547970 65222063 6f6e7465 6e743d22 74657874 2f637373 223e0a3c 7469746c 653e3c2f 746974
6c 653e0a3c 6d657461 206e616d 653d2247 656e6572 61746f72 2220636f 6e74656e 743d2243 6f636f61 2048544d 4c205772
69746572 223e0a3c 7374796c 65207479 70653d22 74657874 2f637373 223e0a70 2e703120 7b6d6172 67696e3a 20302e30 707
82030 2e307078 20302e30 70782030 2e307078 7d0a7370 616e2e73 31207b66 6f6e742d 66616d69 6c793a20 2748656c 766574
69 6361273b 20666f6e 742d7765 69676874 3a206e6f 726d616c 3b20666f 6e742d73 74796c65 3a206e6f 726d616c 3b20666f
6e742d73 697a653a 2031322e 30307074 7d0a3c2f 7374796c 653e0a3c 2f686561 643e0a3c 626f6479 3e0a3c70 20636c61 737
33d22 7031223e 3c737061 6e20636c 6173733d 22733122 3e343131 31313131 31313131 31313131 313c2f73 70616e3e 3c2f70
3e 0a3c2f62 6f64793e 0a3c2f68 746d6c3e 0a>"}]
cy#
```

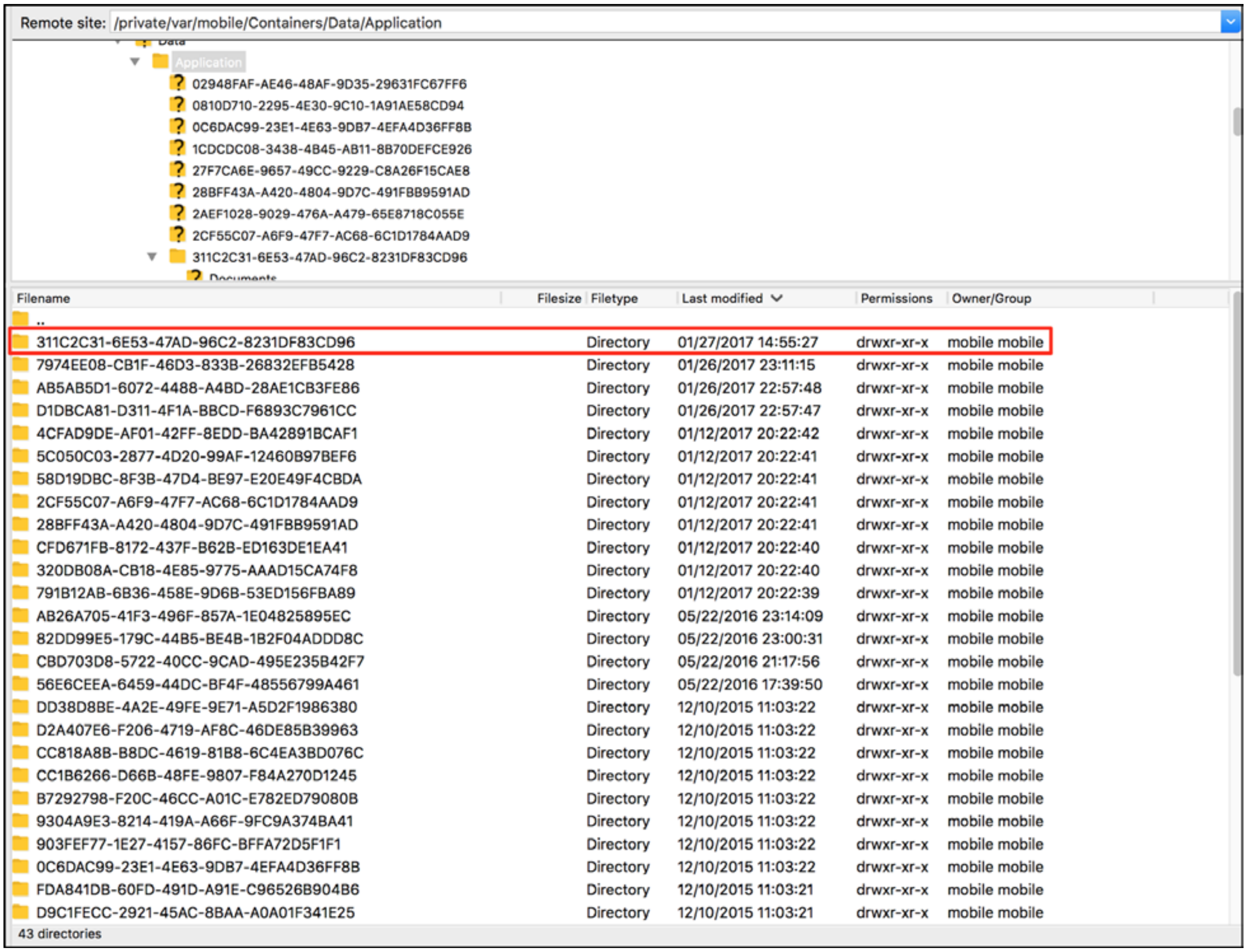
16.3 iOS Cookie Storage

The iOS device allows application to store cookies which could be created by the URL loading system or HTTP request by web view. The cookies are stored in the device storage and an attacker with physical access to the device can steal the cookies. The cookies are stored in binary format and can be parsed using the *BinaryCookieReader.py* script from http://securitylearn.net/wp-content/uploads/tools/iOS/BinaryCookieReader.py

Application used for Example: DemoLogin application from https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

Black Box Testing Approach:

1. Launch the “DemoLogin” application and login with valid credentials. Fill in the payment information and tap the submit button.
2. FTP to the iOS device using a client such as FileZilla to grab the cookie file. This can be found at: /private/var/mobile/Containers/Data/Application/<DemoLogin-GUID>/Library/Cookies Since the GUID in the folder name changes with each installation and/or during runtime, we would have to manually look into each folder to find the “DemoLogin” application folder. Some other techniques that can be employed are shown below.
3. In Filezilla, navigate to “/private/var/mobile/Containers/Data/Application/” and sort by Last modified.



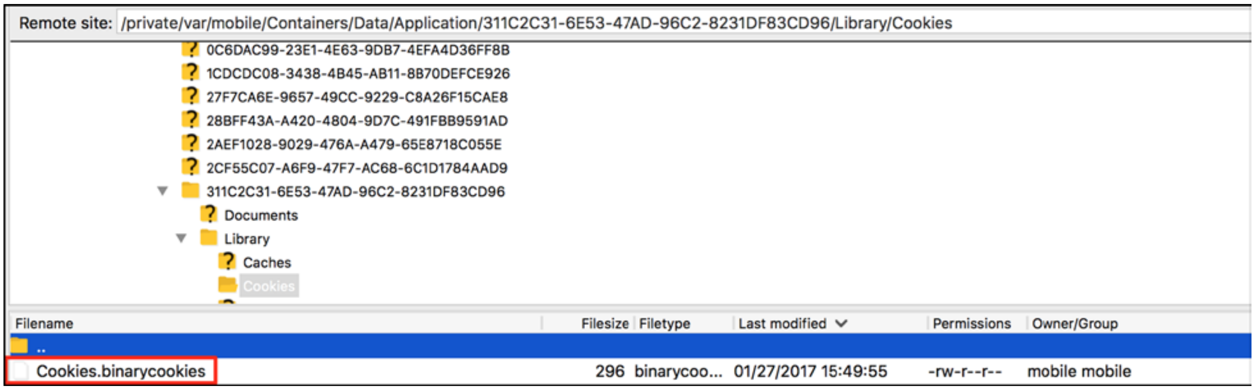
Remote site: /private/var/mobile/Containers/Data/Application

```
    Data
  ▼ Application
      02948FAF-AE46-48AF-9D35-29631FC67FF6
      0810D710-2295-4E30-9C10-1A91AE58CD94
      0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B
      1CDCDC08-3438-4B45-AB11-8B70DEFCE926
      27F7CA6E-9657-49CC-9229-C8A26F15CAE8
      28BFF43A-A420-4804-9D7C-491FBB9591AD
      2AEF1028-9029-476A-A479-65E8718C055E
      2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9
    ▼ 311C2C31-6E53-47AD-96C2-8231DF83CD96
        Documents
```

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
311C2C31-6E53-47AD-96C2-8231DF83CD96		Directory	01/27/2017 14:55:27	drwxr-xr-x	mobile mobile
7974EE08-CB1F-46D3-833B-26832EFB5428		Directory	01/26/2017 23:11:15	drwxr-xr-x	mobile mobile
AB5AB5D1-6072-4488-A4BD-28AE1CB3FE86		Directory	01/26/2017 22:57:48	drwxr-xr-x	mobile mobile
D1DBCA81-D311-4F1A-BBCD-F6893C7961CC		Directory	01/26/2017 22:57:47	drwxr-xr-x	mobile mobile
4CFAD9DE-AF01-42FF-8EDD-BA42891BCAF1		Directory	01/12/2017 20:22:42	drwxr-xr-x	mobile mobile
5C050C03-2877-4D20-99AF-12460B97BEF6		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
58D19DBC-8F3B-47D4-BE97-E20E49F4CBDA		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
28BFF43A-A420-4804-9D7C-491FBB9591AD		Directory	01/12/2017 20:22:41	drwxr-xr-x	mobile mobile
CFD671FB-8172-437F-B62B-ED163DE1EA41		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
320DB08A-CB18-4E85-9775-AAAD15CA74F8		Directory	01/12/2017 20:22:40	drwxr-xr-x	mobile mobile
791B12AB-6B36-458E-9D6B-53ED156FBA89		Directory	01/12/2017 20:22:39	drwxr-xr-x	mobile mobile
AB26A705-41F3-496F-857A-1E04825895EC		Directory	05/22/2016 23:14:09	drwxr-xr-x	mobile mobile
82DD99E5-179C-44B5-BE4B-1B2F04ADDD8C		Directory	05/22/2016 23:00:31	drwxr-xr-x	mobile mobile
CBD703D8-5722-40CC-9CAD-495E235B42F7		Directory	05/22/2016 21:17:56	drwxr-xr-x	mobile mobile
56E6CEEA-6459-44DC-BF4F-48556799A461		Directory	05/22/2016 17:39:50	drwxr-xr-x	mobile mobile
DD38D8BE-4A2E-49FE-9E71-A5D2F1986380		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
D2A407E6-F206-4719-AF8C-46DE85B39963		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC818A8B-B8DC-4619-81B8-6C4EA3BD076C		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
CC1B6266-D66B-48FE-9807-F84A270D1245		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
B7292798-F20C-46CC-A01C-E782ED79080B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
9304A9E3-8214-419A-A66F-9FC9A374BA41		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
903FEF77-1E27-4157-86FC-BFFA72D5F1F1		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B		Directory	12/10/2015 11:03:22	drwxr-xr-x	mobile mobile
FDA841DB-60FD-491D-A91E-C96526B904B6		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile
D9C1FECC-2921-45AC-8BAA-A0A01F341E25		Directory	12/10/2015 11:03:21	drwxr-xr-x	mobile mobile

43 directories

4. Navigate to the Cookies folder: /private/var/mobile/Containers/Data/Application/311C2C31-6E53-47AD-96C2-8231DF83CD96/Library/Cookies



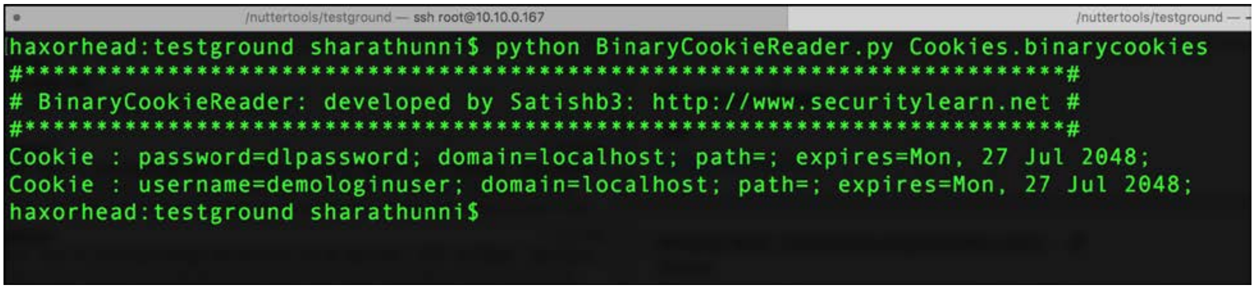
Remote site: /private/var/mobile/Containers/Data/Application/311C2C31-6E53-47AD-96C2-8231DF83CD96/Library/Cookies

- 0C6DAC99-23E1-4E63-9DB7-4EFA4D36FF8B
- 1CDCDC08-3438-4B45-AB11-8B70DEFCE926
- 27F7CA6E-9657-49CC-9229-C8A26F15CAE8
- 28BFF43A-A420-4804-9D7C-491FBB9591AD
- 2AEF1028-9029-476A-A479-65E8718C055E
- 2CF55C07-A6F9-47F7-AC68-6C1D1784AAD9
- 311C2C31-6E53-47AD-96C2-8231DF83CD96
  - Documents
  - Library
    - Caches
    - Cookies

Filename	Filesize	Filetype	Last modified	Permissions	Owner/Group
..					
Cookies.binarycookies	296	binarycoo...	01/27/2017 15:49:55	-rw-r--r--	mobile mobile

5. The Binary cookies can be parsed using the *BinaryCookieReader.py* script. Run the command as shown in the screenshot.

   Download *BinaryCookieReader.py* from the below mentioned link:
   https://gist.github.com/sh1n0b1/4bb8b737370bfe5f5ab8


/nuttertools/testground — ssh root@10.10.0.167
/nuttertools/testground — -

```
haxorhead:testground sharathunni$ python BinaryCookieReader.py Cookies.binarycookies
#*************************************************************************#
# BinaryCookieReader: developed by Satishb3: http://www.securitylearn.net #
#*************************************************************************#
Cookie : password=dlpassword; domain=localhost; path=; expires=Mon, 27 Jul 2048;
Cookie : username=demologinuser; domain=localhost; path=; expires=Mon, 27 Jul 2048;
haxorhead:testground sharathunni$
```

The cookies stored by the application contain sensitive information such as the username and password.

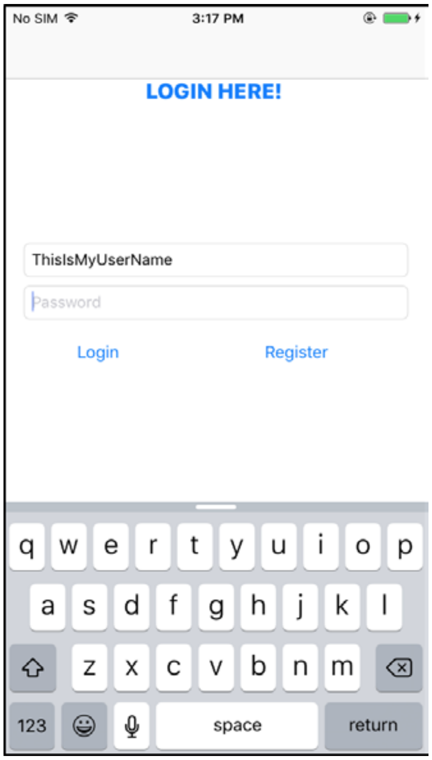
16.4 iOS Keyboard Cache Storage

The iOS caches data being typed which is utilized by the auto correction feature. The data is cached in clear text in the order it is typed by the iOS and can be found in the file "/var/mobile/Library/Keyboard/dynamic-text.dat".

Application used for Example: DemoLogin application from https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

Black Box Testing Approach:

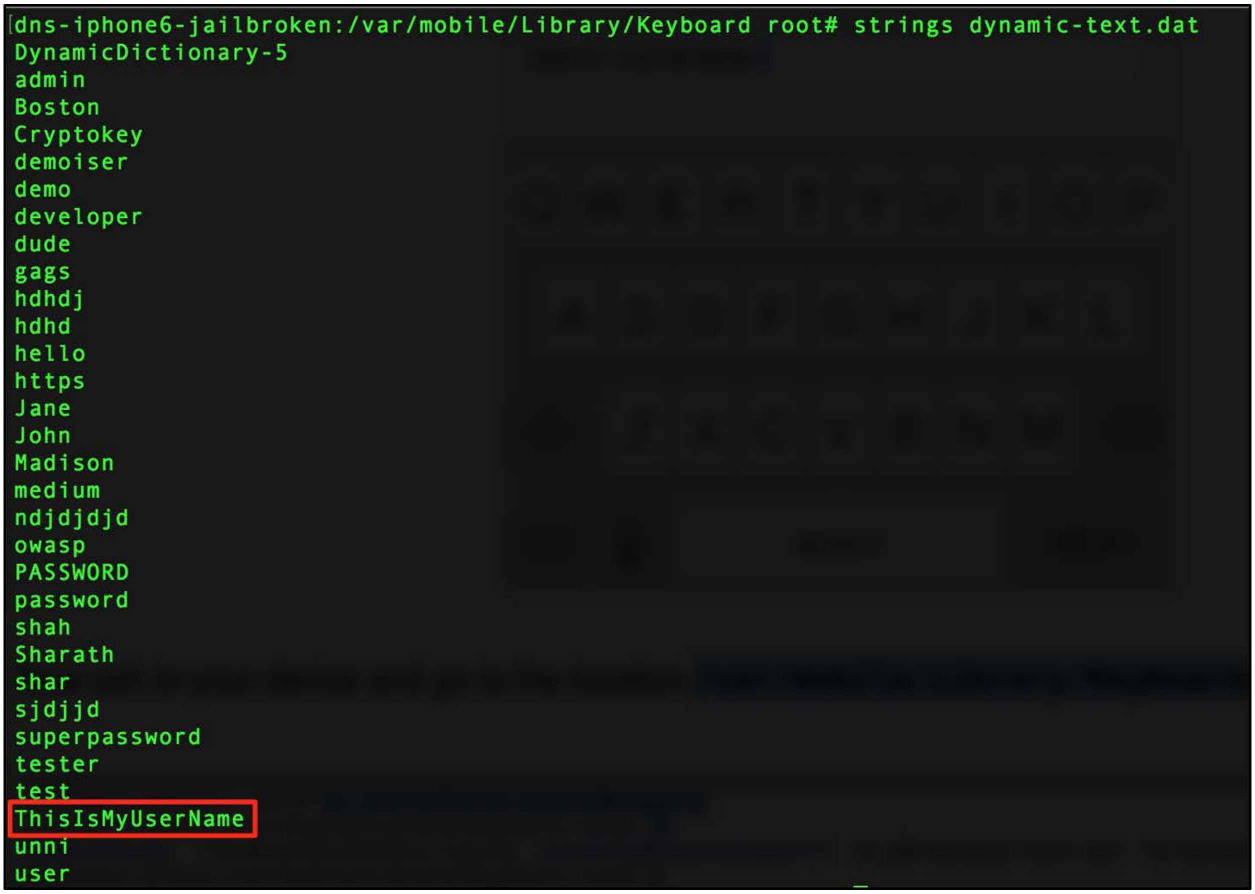
1. Launch the "DemoLogin" application and enter any value in the Username field.



ThisIsMyUserName

Password

2. SSH in to the iOS device and navigate to the below folder:
   - */var/mobile/Library/Keyboard*
3. Run strings command on the file "dynamic-text.dat" and note that the typed text is being cached by the device.



```
dns-iphone6-jailbroken:/var/mobile/Library/Keyboard root# strings dynamic-text.dat
DynamicDictionary-5
admin
Boston
Cryptokey
demoiser
demo
developer
dude
gags
hdhdj
hdhd
hello
https
Jane
John
Madison
medium
ndjdjdjd
owasp
PASSWORD
password
shah
Sharath
shar
sjdjjd
superpassword
tester
test
ThisIsMyUserName
unni
user
```

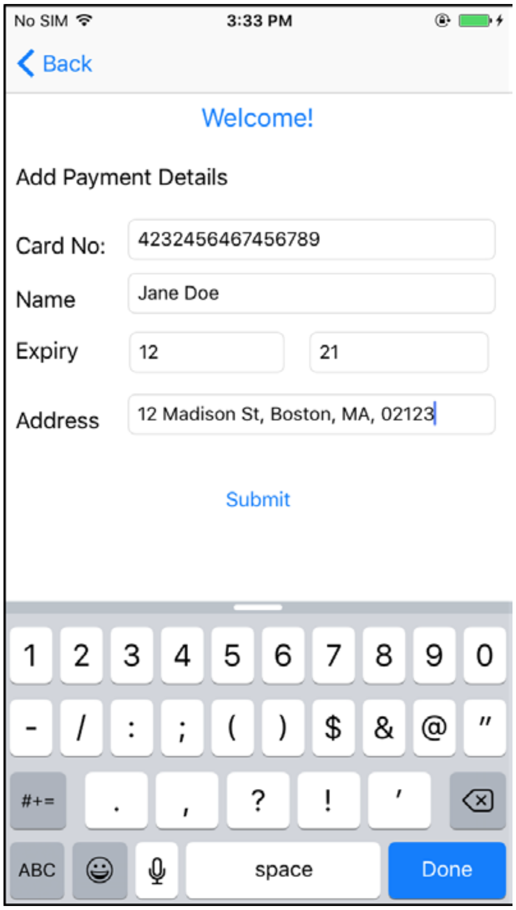
16.5 iOS Device Logging

Developers commonly use NSLog for debugging and diagnostic purposes. Sometimes sensitive data is logged and this data is captured in the device logs. Since error logs are not bound by the application sandbox they can be read by another application in the device. This could allow a malicious application running on the device to steal sensitive data logged by the application. It’s possible to read the error logs created by the application using the Console app or Xcode. During pentest, it is important to navigate through all areas of the application to see if any sensitive data is being logged.

Application used for Example: DemoLogin application from https://drive.google.com/open?id=0B0b4lUTjHfRKRW9uQ0hKVzM2dEU

BlackBox Testing approach:

1. Launch the “DemoLogin” application, register a new user and log in to the application.
2. Enter Payment details and tap the Submit button.



Welcome!

Add Payment Details

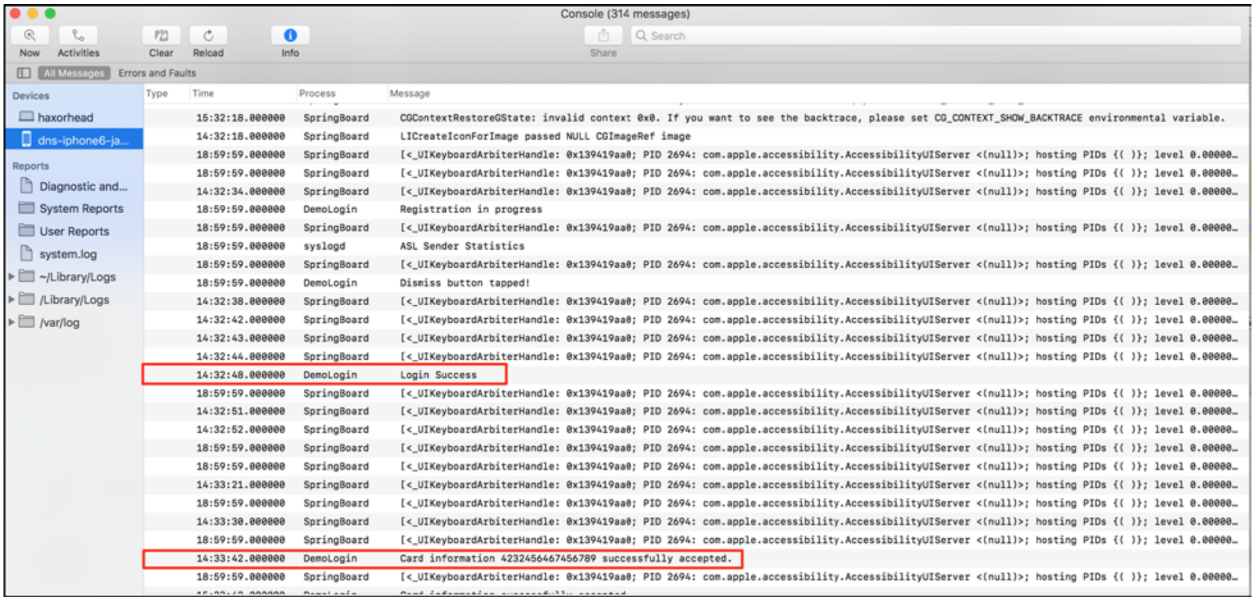
Card No: 4232456467456789

Name: Jane Doe

Expiry: 12 21

Address: 12 Madison St, Boston, MA, 02123

3. Launch the iOS "Console" application in Mac OS and select the appropriate device in the left-hand pane.
4. The screenshot below shows the user data being logged by the application.



```
Type  Time             Process      Message
      15:32:18.000000  SpringBoard  CGContextRestoreGState: invalid context 0x0. If you want to see the backtrace, please set CG_CONTEXT_SHOW_BACKTRACE environmental variable.
      14:32:18.000000  SpringBoard  LICreateIconForImage passed NULL CGImageRef image
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:34.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  DemoLogin    Registration in progress
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  syslogd      ASL Sender Statistics
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  DemoLogin    Dismiss button tapped!
      14:32:38.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:42.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:43.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:44.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:48.000000  DemoLogin    Login Success
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:51.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:32:52.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:33:21.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:33:30.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
      14:33:42.000000  DemoLogin    Card information 4232456467456789 successfully accepted.
      18:59:59.000000  SpringBoard  [<_UIKeyboardArbiterHandle: 0x139419aa0; PID 2694: com.apple.accessibility.AccessibilityUIServer <(null)>; hosting PIDs {( )}; level 0.00000…
```