Introduction to iOS Application Penetration Testing

Michael Gianarakis
OWASP Melbourne

27 February 2014

Introduction

About Me

- Michael Gianarakis
- Senior security consultant based in Melbourne
- No, I’m not a certified accountant but yes I do love romantic walks along the beach if anyone is interested

What We Will Cover

- The aim of the talk is to provide an introduction to iOS application penetration testing
- Focus on black box assessment
- **Penetration testers/security professionals:**
  - For those who haven't done much in the way of iOS application testing this talk should provide a basic approach that you can use.
  - For those that have done a bunch of iOS application testing - enjoy the free beer.

What We Will Cover

- **Developers:**
  - How many iOS developers do we have?
  - Hopefully this will give you an insight into some of the security issues facing iOS applications and some tips to allow you to assess the security of your own apps.
- Although I will touch on remediation at points, the focus is on penetration testing. If you have any questions about what can be done to remediate the issues we discuss yell out or come see me after.

Topics

- Overview of iOS and Objective-C
- Setting up a penetration testing environment
- Assessing data security
- Binary analysis
- Runtime manipulation
- Assessing transport security
- Data validation, authentication, session management (and other stuff that is tested essentially like a web app)

What We Won’t Cover

- Developing iOS jailbreaks or exploiting iOS itself
- ARM, ARM assembly and other low level stuff
- Pen testing web services
- Pen testing infrastructure

Overview of iOS and Objective-C

iOS Overview

- iOS is a mobile operating system developed and distributed by Apple Inc. Originally unveiled in 2007 for the iPhone, it has been extended to support other Apple devices such as the iPod Touch (September 2007), iPad (January 2010), iPad Mini (November 2012) and second-generation Apple TV (September 2010) [From Wikipedia]

- iOS is derived from OS X, with which it shares the Darwin foundation and various application frameworks.

- iOS is not open source......but opensource.apple.com. Various elements of iOS are open source and it’s built on the same XNU kernel as OS X which is open source.

iOS Security Features

- **Sandboxing**
  - All third party iOS applications run in a sandbox.
  - All applications and resources are contained and run in a unique directory.
  - Applications running in the sandbox cannot access other applications or their data, nor can they access system files and other resources.
  - Apple provides classes to interface with the camera, GPS, and other resources on the device, but prevents the application from accessing many components directly.
  - Dionysus Blazakis - The Apple Sandbox http://dl.packetstormsecurity.net/papers/general/apple-sandbox.pdf

iOS Security Features

- **Address Space Layout Randomisation**
  - iOS implemented ASLR in iOS 4.3
  - ASLR randomises code and data is mapped in a processes address space
  - iOS applications implement either full ASLR or partial ASLR depending on whether it has been compiled as a position independent executable.
  - With PIE all application memory regions are randomised while without will load the base binary at a fixed address and use a static location for the dynamic linker.
  - Stefan Esser - http://antid0te.com/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf

iOS Security Features

- **Code Signing**
  - All applications must be signed by a trusted certificate. In order for an application to be permitted to run under iOS, it must be signed by Apple or with a certificate issued by Apple.
  - Apple also performs runtime checks to test the integrity of an application to ensure that unsigned code hasn’t been injected into it.

iOS Security Features

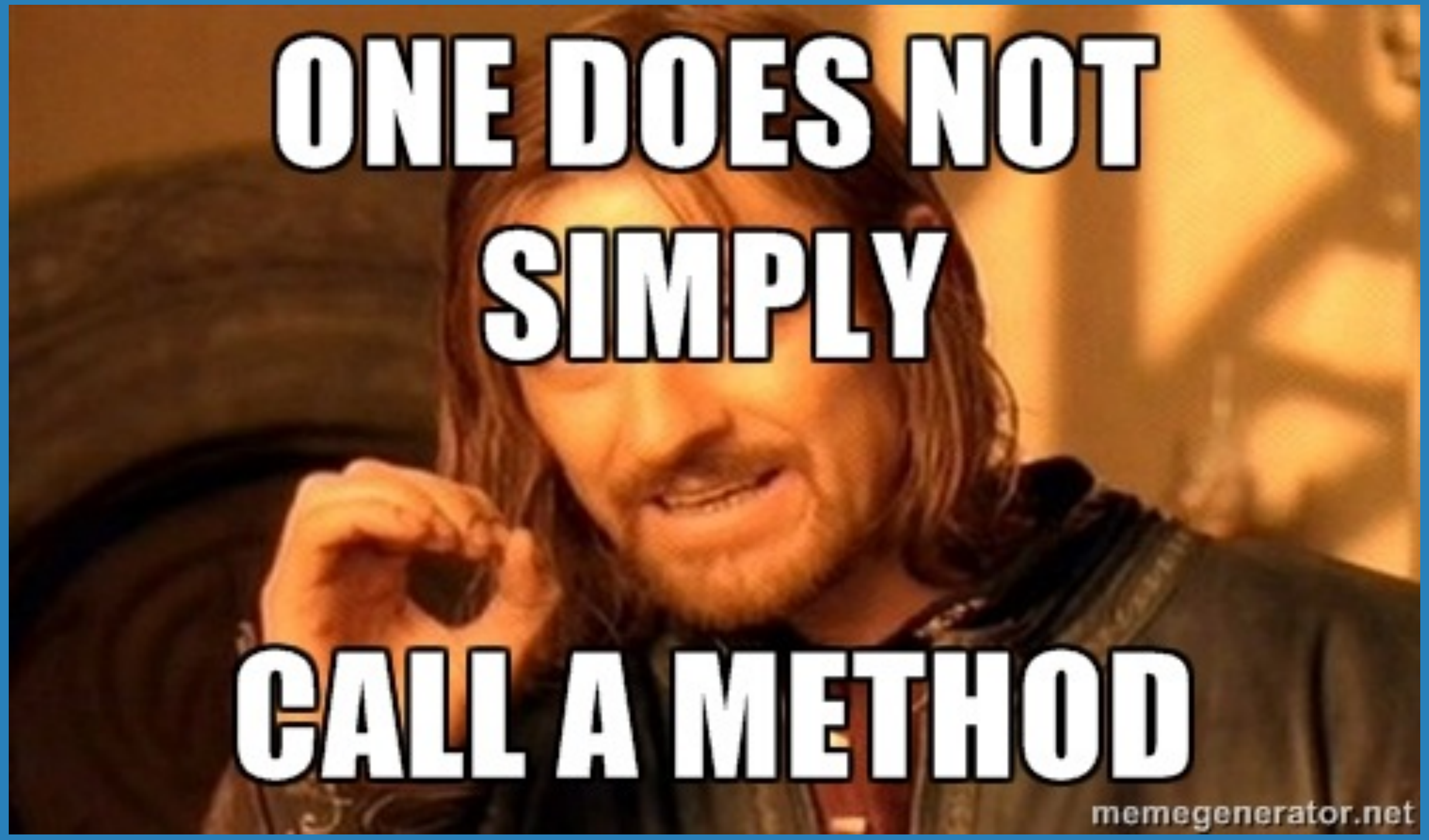
- **Data Encryption**
  - Data is encrypted at rest using hardware based keys. When the device is powered on the data is decrypted.
  - Individual files and keychain items can be encrypted using the data protection API that uses a key derived from the device passcode. When the device is locked, content encrypted using the data protection API will be inaccessible unless cached in memory.

Objective-C Basics

- Native iOS applications are written in Objective-C
- Objective-C is basically C with Smalltalk-style messaging and object syntax
- Objective-C is a strict superset of C.
    - Can compile C with an Objective-C compiler
    - C code can be included in Objective-C code

Objective-C Basics

- Objective-C uses message passing. You technically don't call methods in Objective-C like you do in say C++ you send messages.



Objective-C Basics

- **C++**
- `obj.method(argument); // Sending the message method to the object pointed to by the pointer obj`
- **Objective-C**
- `[obj method:argument]; // Sending the message method to the object pointed to by the pointer obj`

Objective-C Basics

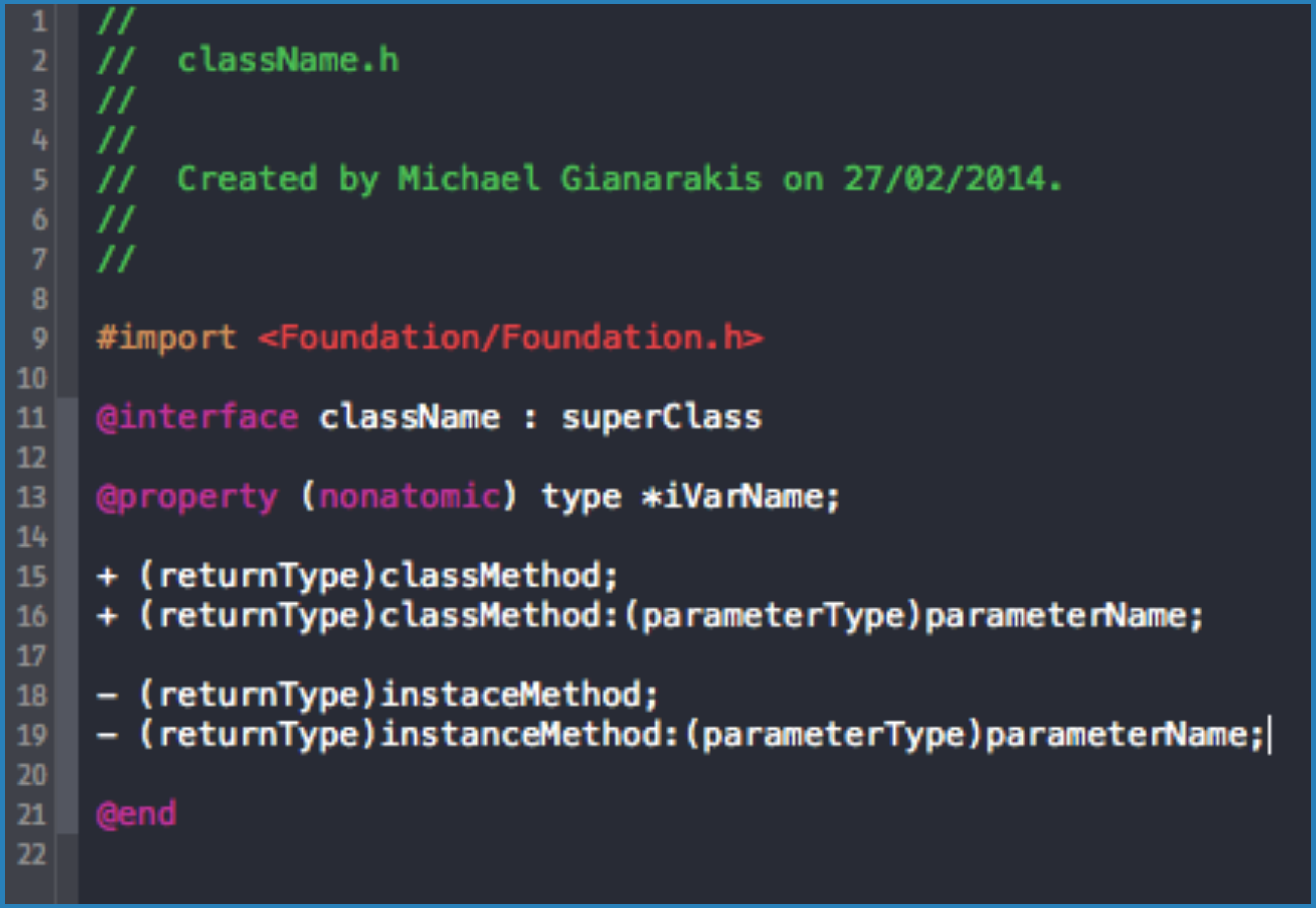
- **Instantiating an Object:**
  - `MyPoint* point; // Declare a pointer to the object`
  - `point = [MyPoint alloc]; // Allocate memory for object`
  - `point = [point init]; // Initialise the object`
- **One Liner**
  - `MyPoint* point = [[MyPoint alloc] init]; // Allocate and initialise the object`

Objective-C Basics

- Interface and Implementations
- Objective-C requires that the interface and implementation of a class be in separately declared code blocks.
- By convention, developers place the interface in a header file and the implementation in a code file. The header files, normally suffixed .h, are similar to C header files while the implementation (method) files, normally suffixed .m, can be very similar to C code files.

Overview of the iOS Platform

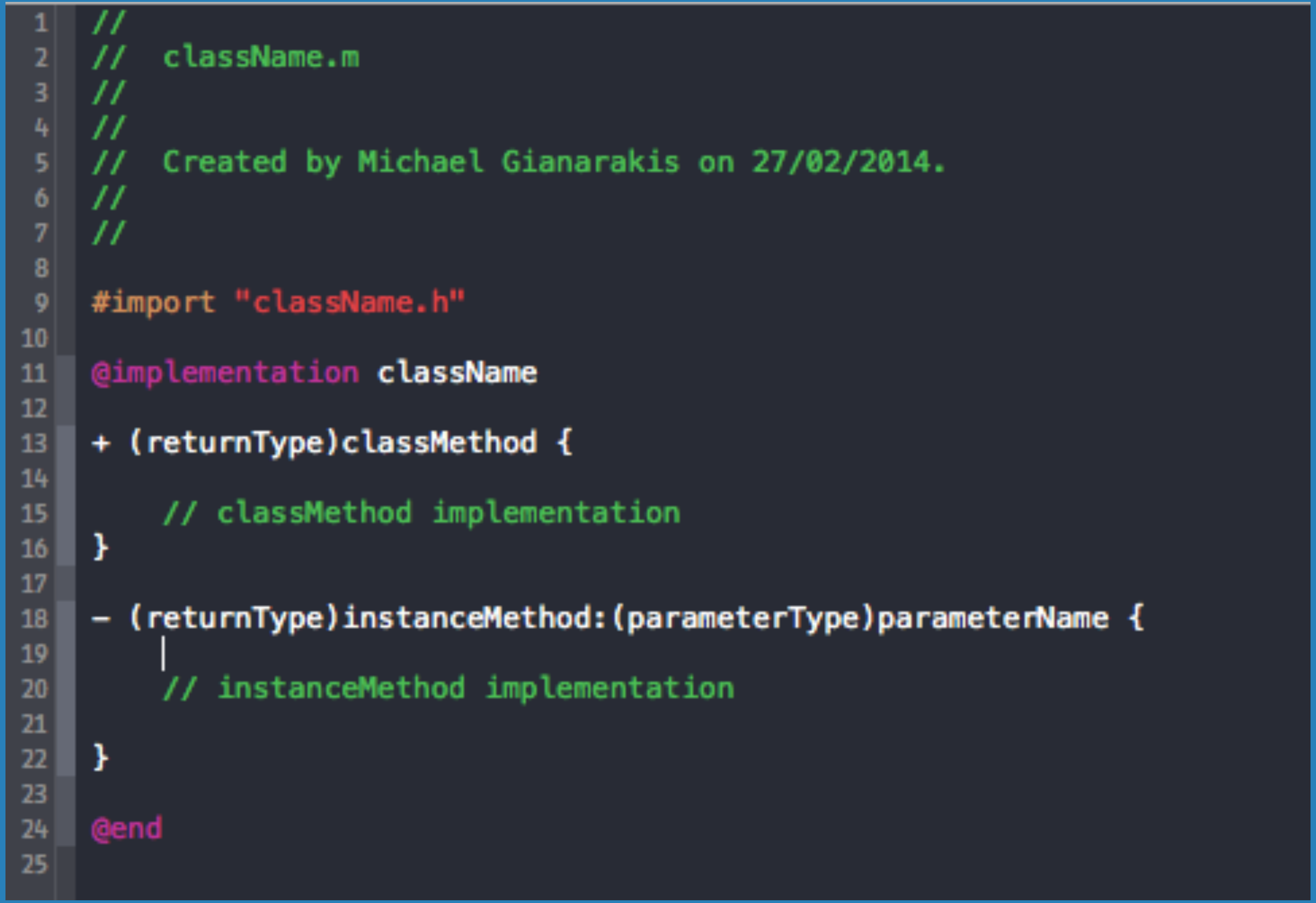
- **Interface**



```
//
//  className.h
//
//
//  Created by Michael Gianarakis on 27/02/2014.
//
//

#import <Foundation/Foundation.h>

@interface className : superClass

@property (nonatomic) type *iVarName;

+ (returnType)classMethod;
+ (returnType)classMethod:(parameterType)parameterName;

- (returnType)instaceMethod;
- (returnType)instanceMethod:(parameterType)parameterName;

@end

```

Overview of the iOS Platform

- **Implementation**



```
//
//  className.m
//
//
//  Created by Michael Gianarakis on 27/02/2014.
//
//

#import "className.h"

@implementation className

+ (returnType)classMethod {

    // classMethod implementation
}

- (returnType)instanceMethod:(parameterType)parameterName {

    // instanceMethod implementation

}

@end

```

Overview of the iOS Platform

- **Automatic Reference Counting (optional but default)**
  - With traditional Objective-C, the programmer would send retain and release messages to objects in order to mark objects for deallocation or to prevent deallocation. Under ARC, the compiler does this automatically by examining the source code and then adding the retain and release messages in the compiled code.

Overview of the iOS Platform

- **Properties**
  - Instance variables can be set as properties and the accessor methods will be generated automatically at compile-time (in a sense properties are public instance variables).
  - A property may be declared as "readonly", and may be provided with storage semantics such as "assign", "copy" or "retain". By default, properties are considered atomic, which results in a lock preventing multiple threads from accessing them at the same time. A property can be declared as "nonatomic", which removes this lock

Setting Up a Penetration Testing Environment

Hardware

- iOS Device (derp!)
- Lightning USB Cable (or 30 Pin depending on device)
- Wireless Network (ideally but not required)
- Computer
  - OS X is ideal but most of this will work with any *nix type system
  - Don’t ask about Windows

iOS Device

- Doesn’t matter too much which iOS device you use
  - iPhone 4 through to iPhone 5S
  - iPad 2 through to iPad Air
  - iPad mini and iPad mini with Retina Display
  - iPod Touch 5th Generation
- Needs to be jailbroken though

Jailbreaking

- iOS jailbreaking is the process of removing the limitations on Apple devices running iOS through the use of software and hardware exploits.

- Jailbreaking permits root access to the iOS operating system, allowing users to run unsigned coed on the device.

- Jailbreaking is a form of privilege escalation and the term has been used to describe privilege escalation on devices by other manufacturers as well.

Jailbreaking a Test Device

- Careful when jailbreaking as it will significantly compromise the security of your device.
- It’s recommended that you have a separate test device from your regular device.
- Current jailbreak is for iOS 7.0 - 7.0.6 using the evasi0n7 jailbreak
- Only update the device when there is a new jailbreak for the firmware version as Apple restricts which firmware can be loaded on the device

Testing on Non-Jailbroken Devices

- You can do some testing on a non-jailbroken device but the coverage is not as great.
- Some testing is much harder to do if you don't have a jailbroken device.
- Should always push back on any requests to use a non-jailbroken device for testing. Apart from being less efficient and a PITA it is doesn't realistically cover all the possible risks affecting an iOS app.
- Developers/organisations should assume an app will run on a compromised device even if you have MDM and totally ISO 27001 compliant security processes.
- DEFENCE IN DEPTH PEOPLE!

Software

- Xcode and Command Line Utilities available from the Mac App Store (requires an Apple ID)
- Intercepting Proxy such as Burp Proxy, ZAP Proxy or Abrupt Proxy. I will use Burp.
- openSSH and iPhone-tunnel
- SQLite or SQLite Database Browser
- keychain-dumper
- class-dump-z
- gdb
- cycript
- cydiasubstrate
- adv-cmds
- Darwin CC Tools
- evasi0n7 (latest public Jailbreak)
- ios-ssl-killswitch by iSEC Partners

Setting Up The Environment

- I will go through setting up some of these tools briefly but for a more detailed overview of setting up the tools I have a blog post on the matter:
  - http://eightbit.io/post/64319534191/how-to-set-up-an-ios-pen-testing-environment

Setting Up The Environment

- **Install Xcode and Command Line Tools**
  - Xcode is Apple's IDE and includes the latest iOS SDK and iOS Simulator. It's available for free on the Mac App Store (requires an Apple ID). Once Xcode is installed be sure to install the Command Line Tools.
    - `xcode-select —install`
  - Xcode is handy for testing things, easy access to documentation etc.

Setting Up The Environment

- **Install Burp Suite (or other Proxy)**
- If you work in info sec don't know how to do this then shame on you. I hear they still want Certified Ethical Hackers in some parts of the world.
- If you are a developer see:
  - http://portswigger.net/burp/help/suite_gettingstarted.html
  - https://code.google.com/p/zaproxy/wiki/HelpStartStart

Setting Up The Environment

- **Add Burp Cert**
- In order to intercept HTTPS traffic you will need to add the self signed Burp/Portswigger cert.
- The easiest way to do this is to attach the cert to an email and send it to an email address accessible on the phone.
- Open up the attachment and follow the prompts to add the cert.
- Don’t forget to set the proxy settings on the device (available in the network settings) appropriately.
- If the application employs certificate pinning or other application level SSL validation methods (which is increasingly common) it may impact the interception of HTTPS traffic. We will discuss ways to bypass this later.

Setting Up The Environment

- **Jailbreak the Device**
- The current public jailbreak is available at evasi0n.com. The jailbreak software is pretty slick and easy to use so I won’t detail the steps here however as a caution always back up your device first.

Setting Up The Environment

- **Install OpenSSH**
- OpenSSH can be installed via Cydia (which would have been installed when the device was jailbroken).
- Cydia is a package manager/app store typically installed with public jailbreaks.
- Uses apt and dpkg and all the standard Debian package management tools.
- The default password for the root account on iOS is alpine but you should probably change it, as well as the password for the user mobile, to something else.

Setting Up The Environment

- **Install OpenSSH**
- To change the root password first SSH into the device over the wireless network using the default credentials:
  - `ssh root@[IP ADDRESS]`
- If you don’t have a wireless network the easiest way to connect to the device is using iPhone Tunnel:

  -
    ```
    ./itnl -lport 2222
    ssh -p 2222 root@127.0.0.1
    ```
- Use the passwd command to change the root password:
  - `passwd`
- To change the password for the mobile user:
  - `passwd mobile`

Setting Up The Environment

- **Install Cycript**
- Cycript is a “programming language designed to blend the barrier between Objective-C and JavaScript” and is very useful when testing iOS apps (but more on this later)
- Easiest way to install is through Cydia (just search for cycript)
- **Manual install:**
    - Download cycript package (link is in the blog post)
    - Copy the package to your device and install it using dpkg:
        - dpkg -i cycript_xxxxxx_iphoneos-arm.deb

Assessing iOS Applications - Getting Started

Obtaining the Application

- Some clients will request that we do a completely black box test of the app and will only provide a link to the App Store. App Store executables are encrypted. The encryption applied to App Store executables is similar to the FairPlay DRM used on iTunes music and will require decrypting to perform some of the analysis listed in this document.

- It is preferable to obtain an unencrypted binary in the first place and only use the encrypted binary if the client specifically requests a completely black-box test or we are unable to obtain an unencrypted binary.

- Most clients that develop apps in house have a means of distributing builds internally and it's usually pretty easy to install these

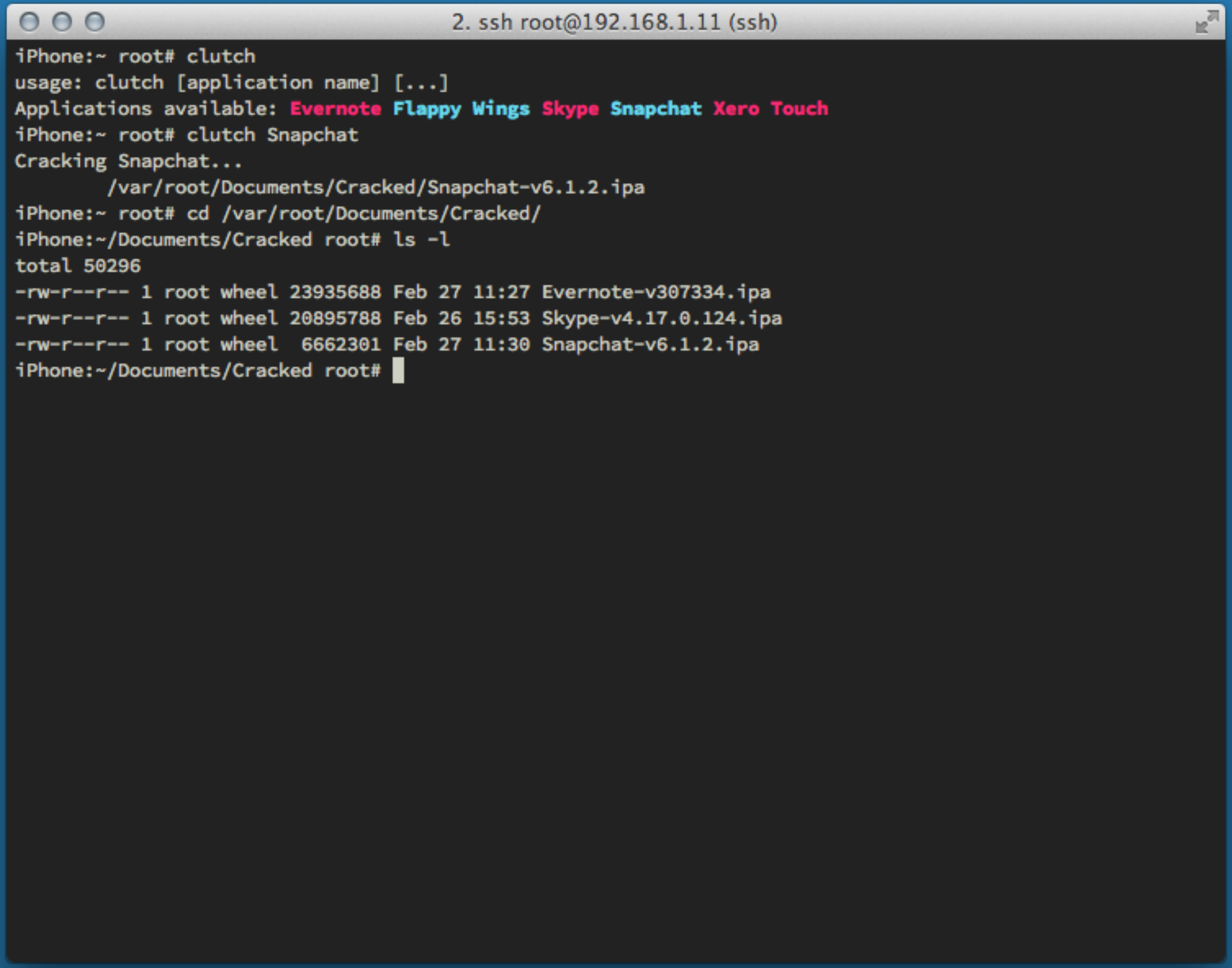
Breaking Encryption

- Easy way or hard way
    - Easy way is to use the Clutch
    - Alternative is dumpencrypted by Stefan Essar (https://github.com/stefanesser/dumpdecrypted)
    - Hard way is manual
        - All Clutch does is automate the process
        - Manual process is documented across the web but basically involves extracting the encrypted portion after iOS decrypts it to run and then patching that decrypted portion into the binary
            - http://www.sensepost.com/blog/6254.html
            - http://www.infointox.net/?p=114

Breaking Encryption

- I don’t condone or endorse app piracy. Decrypting binaries is only for security testing purposes and in reality (on client gigs) it isn’t required much.

Breaking Encryption



```
iPhone:~ root# clutch
usage: clutch [application name] [...]
Applications available: Evernote Flappy Wings Skype Snapchat Xero Touch
iPhone:~ root# clutch Snapchat
Cracking Snapchat...
        /var/root/Documents/Cracked/Snapchat-v6.1.2.ipa
iPhone:~ root# cd /var/root/Documents/Cracked/
iPhone:~/Documents/Cracked root# ls -l
total 50296
-rw-r--r-- 1 root wheel 23935688 Feb 27 11:27 Evernote-v307334.ipa
-rw-r--r-- 1 root wheel 20895788 Feb 26 15:53 Skype-v4.17.0.124.ipa
-rw-r--r-- 1 root wheel  6662301 Feb 27 11:30 Snapchat-v6.1.2.ipa
iPhone:~/Documents/Cracked root#
```

Anatomy of an App

- The application directory is located at **/var/mobile/Applications/xxxxxx**
- Where xxxxxx is the Application ID.
- This folder contains the following:
    - **/Documents** – Persistent store for application data; this data will be synched and backed up to iTunes.
    - **/Library/Caches** – Caches
    - **/Library/Caches/Snapshots** – Application screenshots taken when the app is backgrounded
    - **/Library/Cookies** – Cookie plists
    - **/Library/Preferences** – Preference plists
    - **/Library/WebKit** – Webkit local storage
    - **/xxxxxx.app** – App resources (binary, graphics, nibs, Info.plist, localisation resources etc.)
    - **/tmp** – tmp

Assessing iOS Applications - Data Security

Data Security

- Improperly secured data stored on the device is very common
- I have come across all kinds of sensitive information stored in clear text including:
    - Usernames and passwords
    - Encryption keys
    - Personal information
    - Location data

Data Security

- **Two main types of insecure data storage:**
    - Data stored by the operating system automatically
    - Sensitive data stored on the device by the application that was not secured appropriately by the developer

Data Security

- Sensitive data potentially stored by the operating system
  - Backgrounding screenshots
  - Caches
    - Autocorrect Cache
    - Pasteboard
    - Webkit Caches
    - etc.
- Oftentimes developers do not realise that the OS is storing this information

Data Security

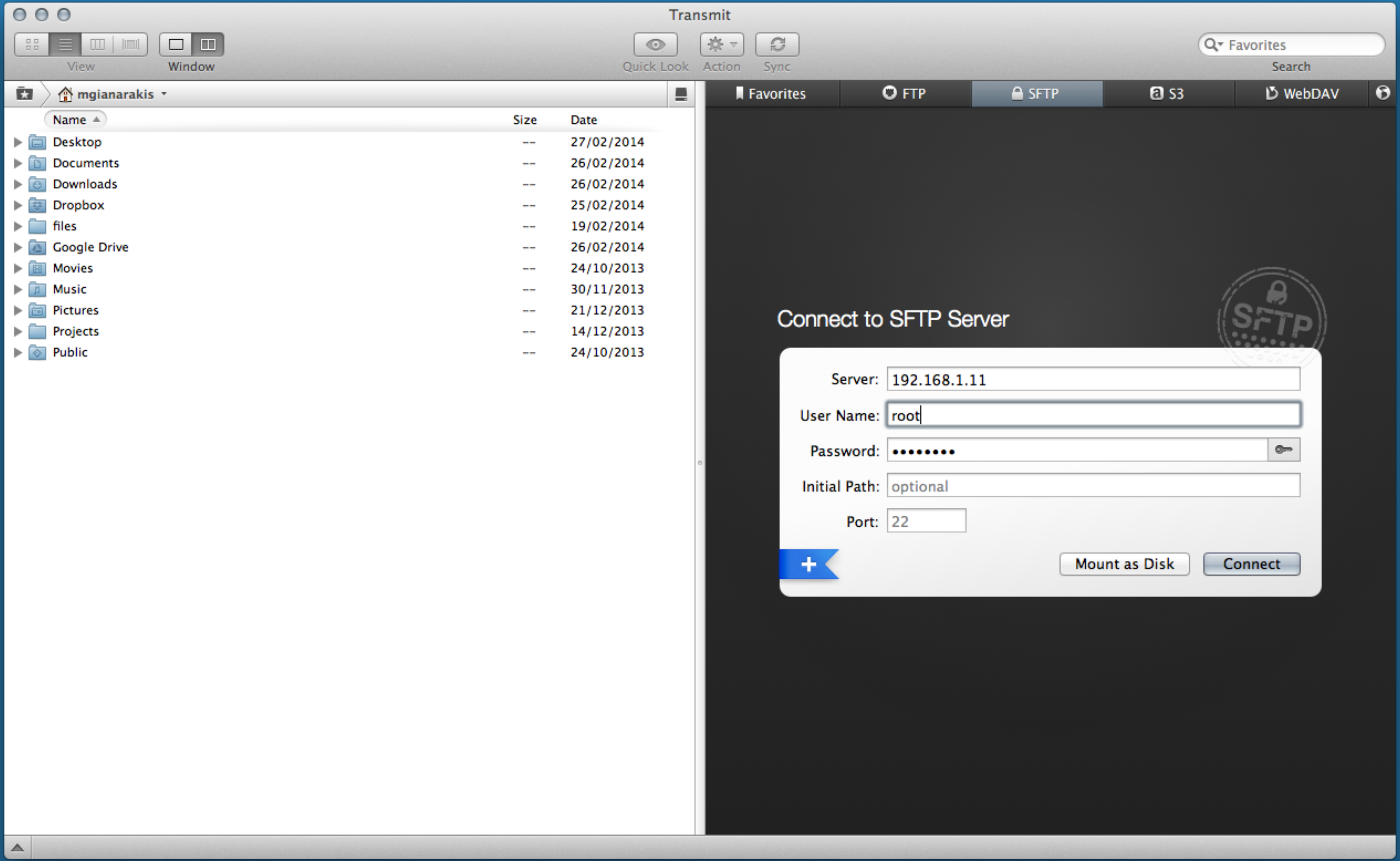
- **Secure Backgrounding**
- Every time an application suspends into the background (typically by pressing the Home button or receiving a phone call), a snapshot is taken to produce desired aesthetic effects (the "scaling" animation).
- This allows attackers to view the last thing a user was looking at, and if they can scrape deleted files off of a raw disk image, have access to a compromised device or access to an application backup, they may also be able to obtain multiple copies of the last thing a user was looking at.
- Third-party applications have their own snapshot cache inside their application folder (/Library/Caches/Snapshots). The /var/mobile/Library/Caches/Snapshots directory also contains screenshots of the most recent states of applications at the time they were suspended.

Data Security

- **Secure Backgrounding**
- To test secure backgrounding, “browse” to a part of the app that displays sensitive information (this may include an editable text field that has sensitive data in it such as a credit card number entry field) and press the home button.
- Then browse the the applications snapshots folder (e.g. **/var/mobile/Applications/xxxxxxxxxx/Library/Caches/Snapshots**) and view the snapshot taken.
- If the developer has securely implemented backgrounding, the image should either be a generic image (such as a splash screen) or have all the sensitive information stripped out or obfuscated.

Demo - Evernote

Data Security



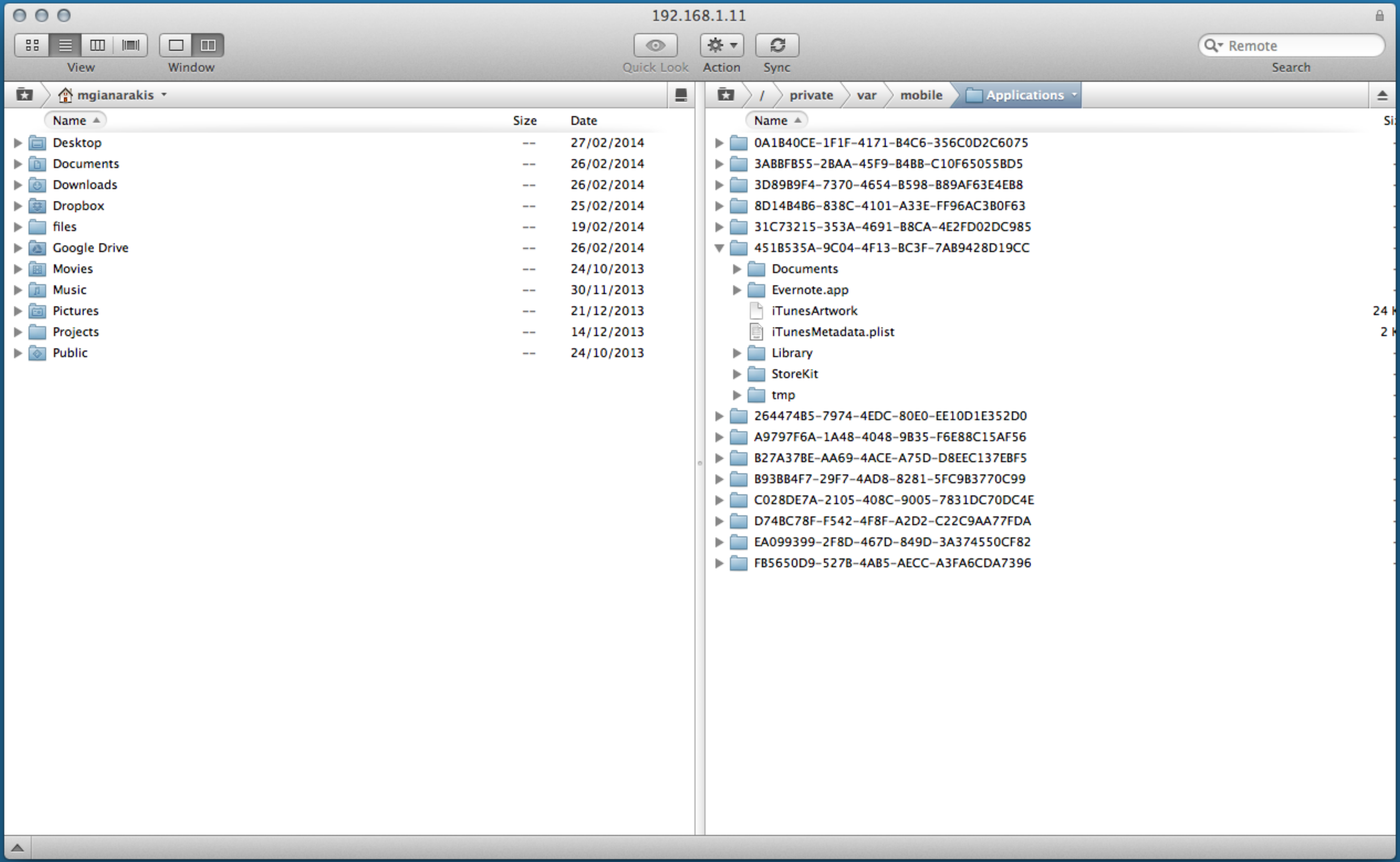
Transmit

Name	Size	Date
Desktop	--	27/02/2014
Documents	--	26/02/2014
Downloads	--	26/02/2014
Dropbox	--	25/02/2014
files	--	19/02/2014
Google Drive	--	26/02/2014
Movies	--	24/10/2013
Music	--	30/11/2013
Pictures	--	21/12/2013
Projects	--	14/12/2013
Public	--	24/10/2013

Connect to SFTP Server

Server: 192.168.1.11
User Name: root
Password: ••••••••
Initial Path: optional
Port: 22

Data Security



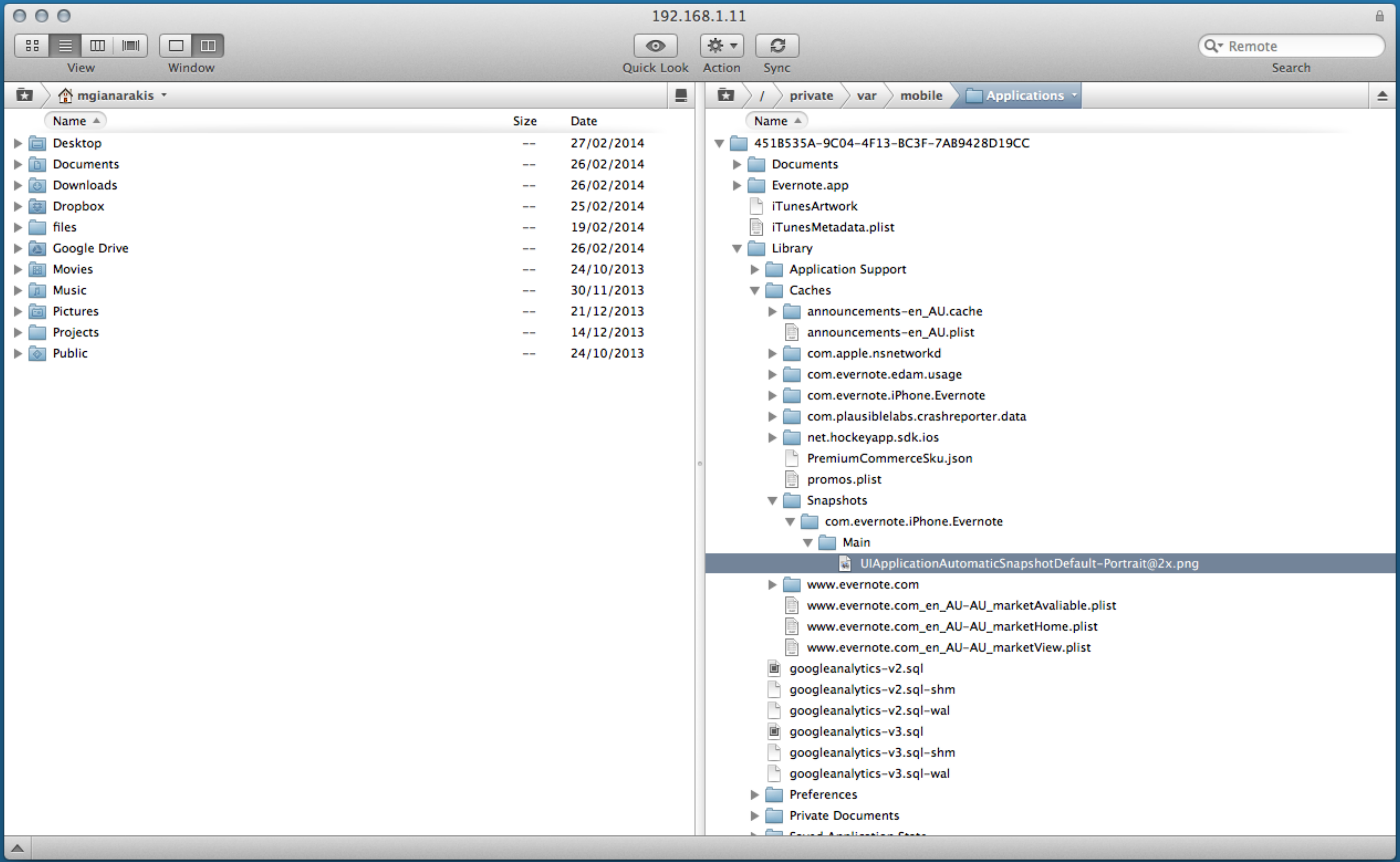
mgianarakis

Name	Size	Date
Desktop	--	27/02/2014
Documents	--	26/02/2014
Downloads	--	26/02/2014
Dropbox	--	25/02/2014
files	--	19/02/2014
Google Drive	--	26/02/2014
Movies	--	24/10/2013
Music	--	30/11/2013
Pictures	--	21/12/2013
Projects	--	14/12/2013
Public	--	24/10/2013

/ › private › var › mobile › Applications

Name	Si
0A1B40CE-1F1F-4171-B4C6-356C0D2C6075	
3ABBFB55-2BAA-45F9-B4BB-C10F65055BD5	
3D89B9F4-7370-4654-B598-B89AF63E4EB8	
8D14B4B6-838C-4101-A33E-FF96AC3B0F63	
31C73215-353A-4691-B8CA-4E2FD02DC985	
451B535A-9C04-4F13-BC3F-7AB9428D19CC	
Documents	
Evernote.app	
iTunesArtwork	24
iTunesMetadata.plist	2
Library	
StoreKit	
tmp	
264474B5-7974-4EDC-80E0-EE10D1E352D0	
A9797F6A-1A48-4048-9B35-F6E88C15AF56	
B27A37BE-AA69-4ACE-A75D-D8EEC137EBF5	
B93BB4F7-29F7-4AD8-8281-5FC9B3770C99	
C028DE7A-2105-408C-9005-7831DC70DC4E	
D74BC78F-F542-4F8F-A2D2-C22C9AA77FDA	
EA099399-2F8D-467D-849D-3A374550CF82	
FB5650D9-527B-4AB5-AECC-A3FA6CDA7396	

Data Security



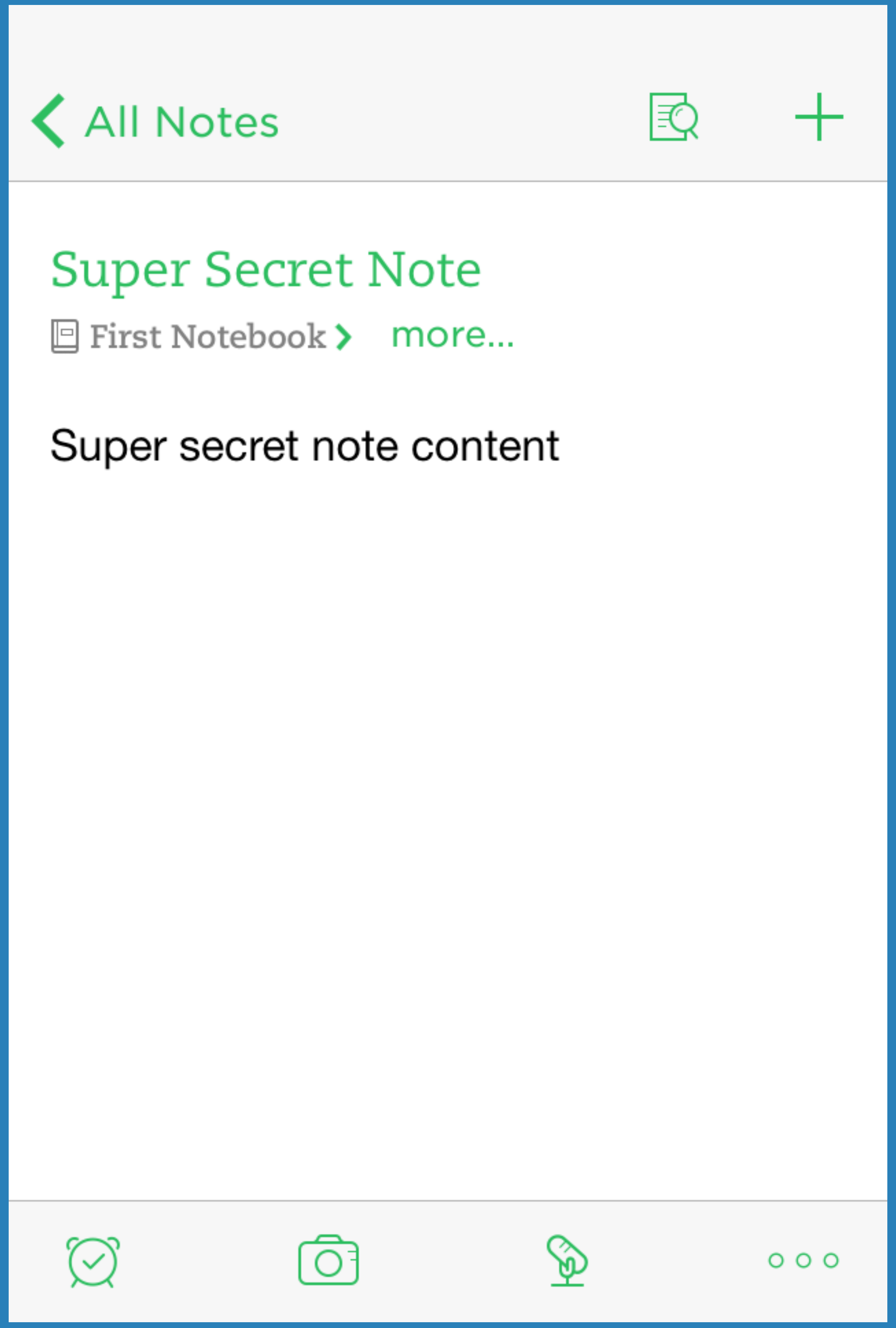
mgianarakis

Name	Size	Date
Desktop	--	27/02/2014
Documents	--	26/02/2014
Downloads	--	26/02/2014
Dropbox	--	25/02/2014
files	--	19/02/2014
Google Drive	--	26/02/2014
Movies	--	24/10/2013
Music	--	30/11/2013
Pictures	--	21/12/2013
Projects	--	14/12/2013
Public	--	24/10/2013

/ private var mobile Applications

```
Name
451B535A-9C04-4F13-BC3F-7AB9428D19CC
  Documents
  Evernote.app
  iTunesArtwork
  iTunesMetadata.plist
  Library
    Application Support
    Caches
      announcements-en_AU.cache
      announcements-en_AU.plist
      com.apple.nsnetworkd
      com.evernote.edam.usage
      com.evernote.iPhone.Evernote
      com.plausiblelabs.crashreporter.data
      net.hockeyapp.sdk.ios
      PremiumCommerceSku.json
      promos.plist
      Snapshots
        com.evernote.iPhone.Evernote
          Main
            UIApplicationAutomaticSnapshotDefault-Portrait@2x.png
      www.evernote.com
      www.evernote.com_en_AU-AU_marketAvaliable.plist
      www.evernote.com_en_AU-AU_marketHome.plist
      www.evernote.com_en_AU-AU_marketView.plist
    googleanalytics-v2.sql
    googleanalytics-v2.sql-shm
    googleanalytics-v2.sql-wal
    googleanalytics-v3.sql
    googleanalytics-v3.sql-shm
    googleanalytics-v3.sql-wal
    Preferences
    Private Documents
```

Data Security



Super Secret Note

Super secret note content

Data Security

- To securely implement backgrounding the screen's contents must be hidden before the screenshot is taken. The simplest way to clear the screen contents is to set the key window's hidden property to YES.

- When an application is about to be made inactive, the applicationWillResignActive delegate method is invoked by the runtime. To hide the window add the following code to the method to hide the window:

```
- (void)applicationWillResignActive:(UIApplication *)application
 {
     [ UIApplication sharedApplication ].keyWindow.hidden = YES;
 }
```

Data Security

- Another important place to include this content-clearing code is in the application delegate's applicationDidEnterBackground method. This method is called when the application is forced into the background, but before a screenshot is taken:

```
- (void)applicationDidEnterBackground:(UIApplication *)application
 {
     [ UIApplication sharedApplication ].keyWindow.hidden = YES;
 }
```

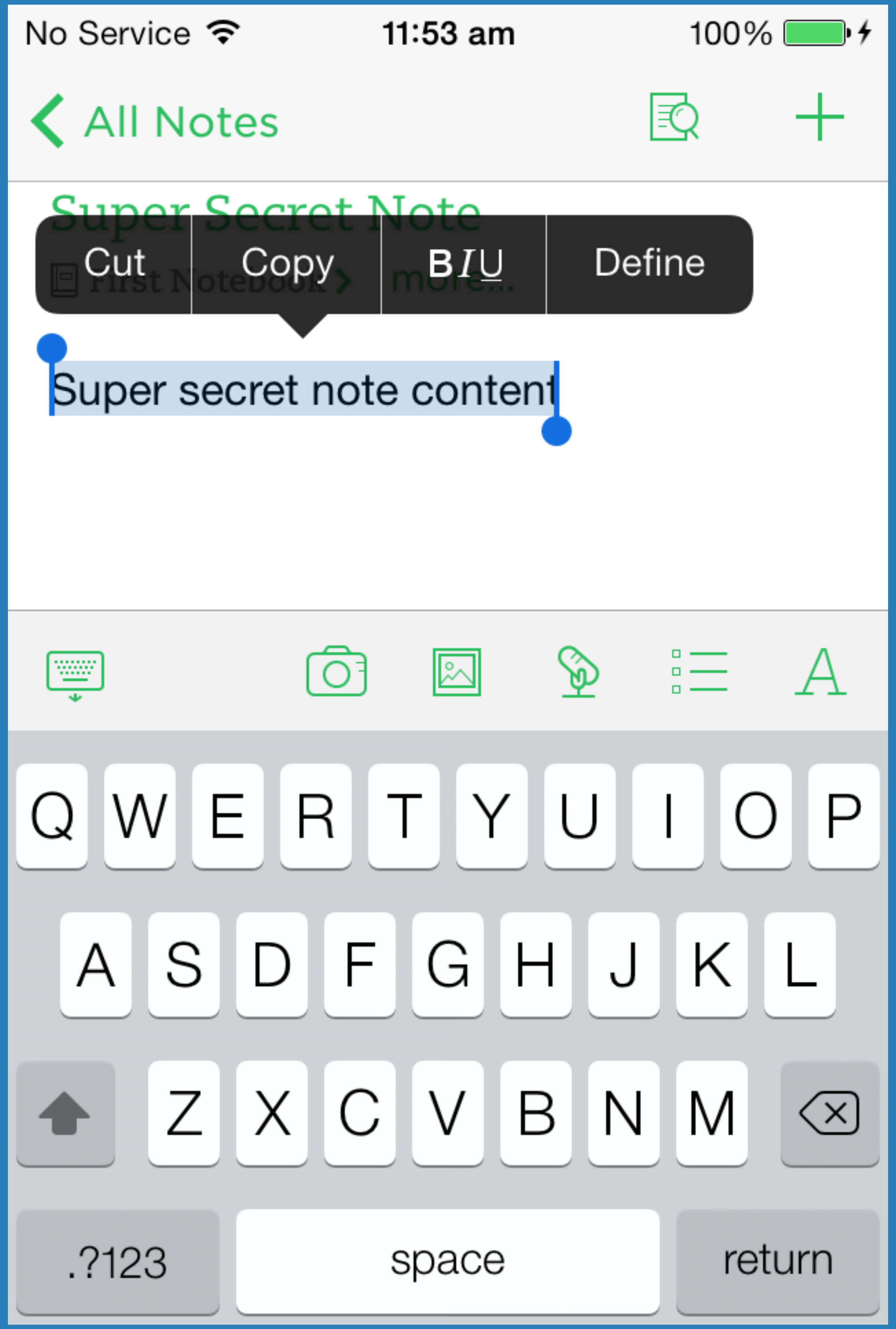
- If there are any other views behind the current view, these may become visible when the key window is hidden. The developer should ensure that they are adequately hiding any other windows when performing this action.

Data Security

- **Pasteboard**
- When text is selected and the Cut or Copy buttons are tapped, and this can happen from within any application that allows Copy/Paste functionality, a cached copy of the data stored on the device's clipboard – /private/var/mobile/Library/Caches/com.apple.UIKit.pboard/pasteboard.
- Some developers have used the pasteboard as an IPC hack to transfer information from once app to another, for example – to migrate information from a free version to a paid version - but that mostly died with in-app purchases
- If the application handles sensitive data that may be copied to the pasteboard it is recommended that the developer either disable Copy/Paste functionality for sensitive text fields (note this is done by default for password fields) or implement a private pasteboard for the application.
- To test if Copy/Paste has been disabled, attempt to Copy/Paste from a sensitive field (e.g. Credit Card Number). If the option to copy the data appears, select it and confirm that this has been copied to the pasteboard.

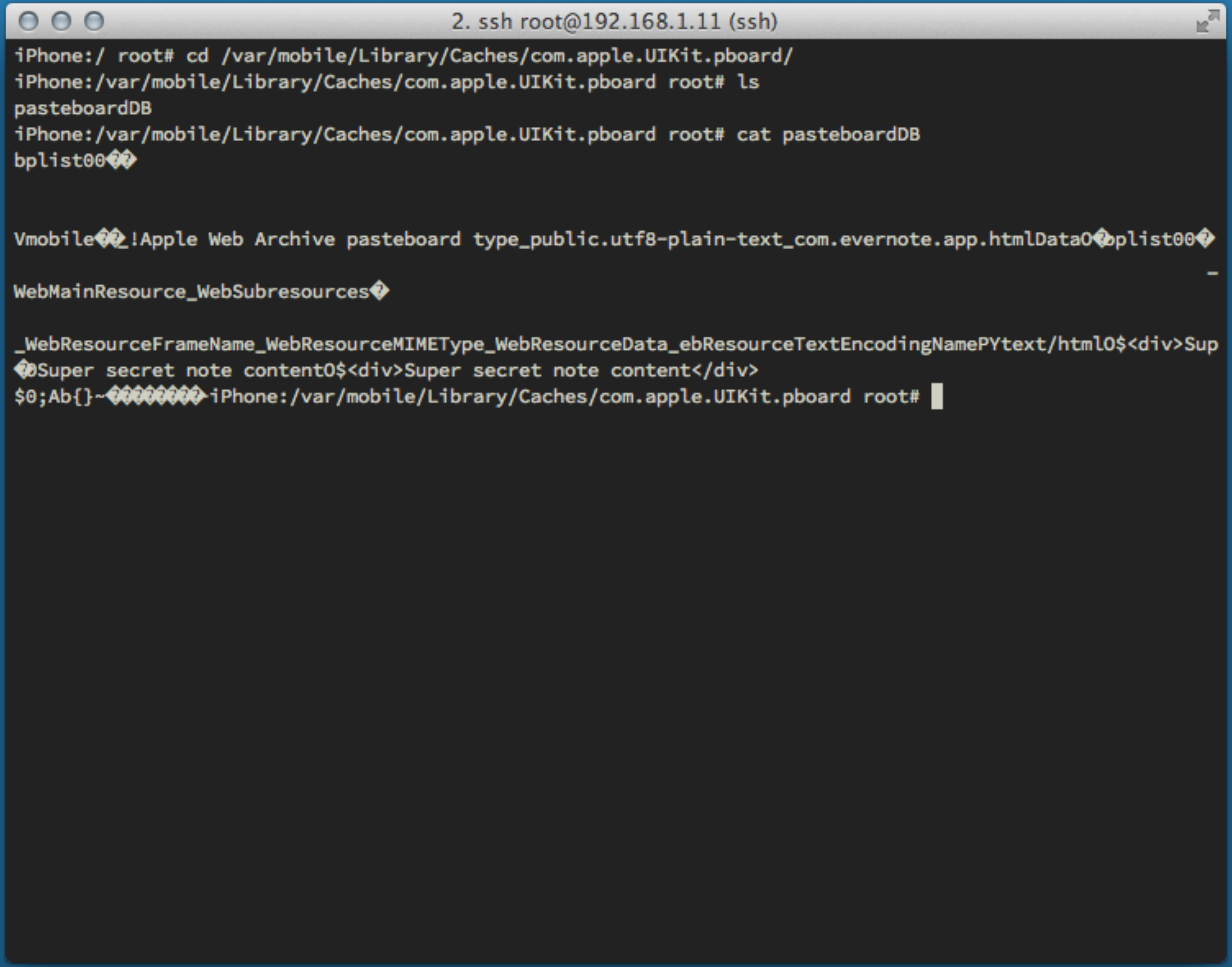
Demo - Evernote

Data Security



Super secret note content

Data Security



```
iPhone:/ root# cd /var/mobile/Library/Caches/com.apple.UIKit.pboard/
iPhone:/var/mobile/Library/Caches/com.apple.UIKit.pboard root# ls
pasteboardDB
iPhone:/var/mobile/Library/Caches/com.apple.UIKit.pboard root# cat pasteboardDB
bplist00��


Vmobile��_!Apple Web Archive pasteboard type_public.utf8-plain-text_com.evernote.app.htmlDataO�bplist00�
                                                                                                      -
WebMainResource_WebSubresources�

_WebResourceFrameName_WebResourceMIMEType_WebResourceData_ebResourceTextEncodingNamePYtext/htmlO$<div>Sup
�Super secret note contentO$<div>Super secret note content</div>
$0;Ab{}~��������iPhone:/var/mobile/Library/Caches/com.apple.UIKit.pboard root# █
```

Data Security

- The easiest way to disable pasteboard operations for sensitive fields is to create a subclass of UITextView that overrides the canPerformAction:withSender: method to return NO for actions that you don't want to allow or just disable the menu completely:

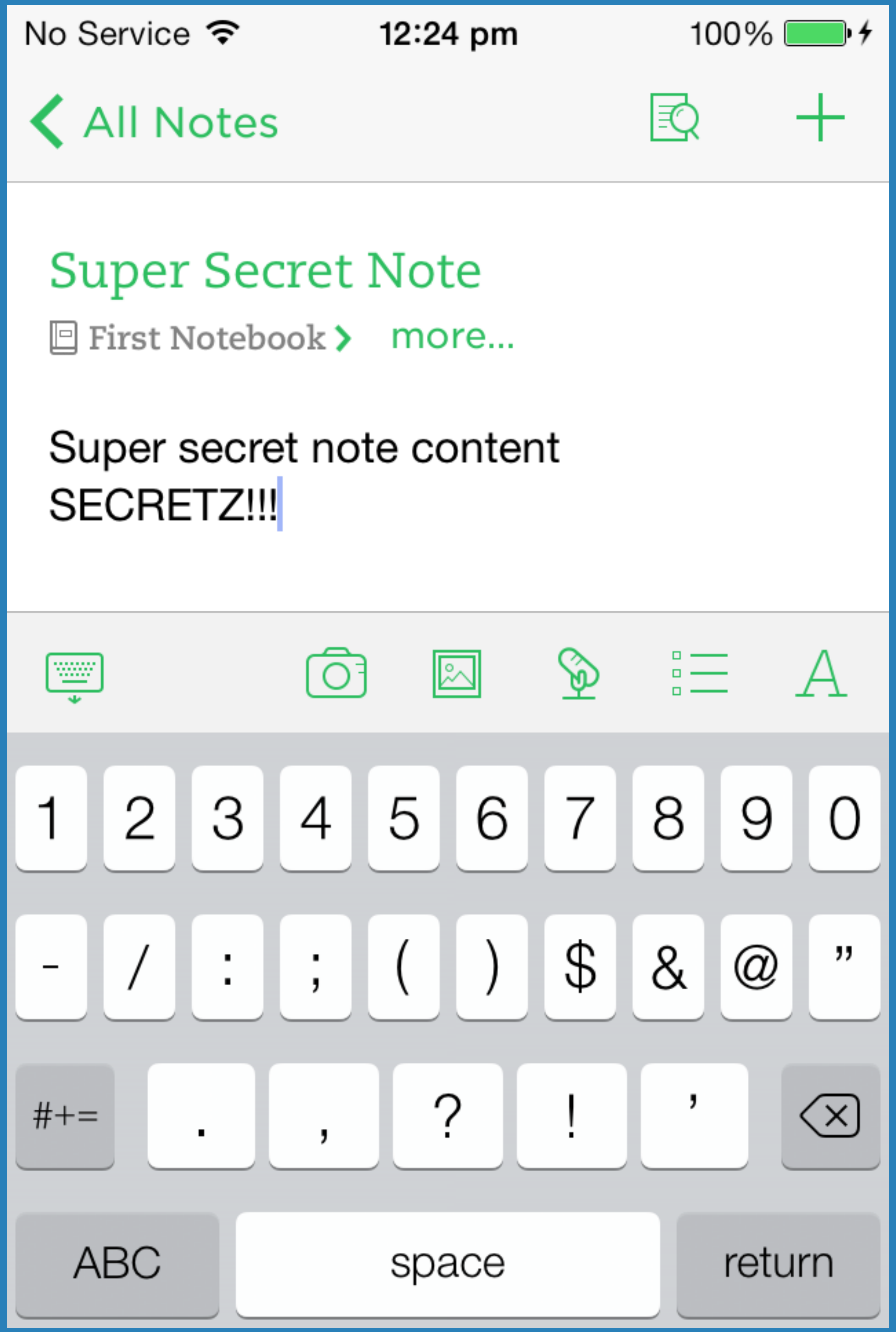
```
- (BOOL)canPerformAction:(SEL)action withSender:(ID)sender {
    UIMenuController *menuController = [UIMenuController sharedMenuController];
    if (menuController) {
        [UIMenuController sharedMenuController].menuVisible = NO;
    }
 return NO;
 }
```

Data Security

- **Autocorrect**
- iOS keeps a binary keyboard cache containing ordered phrases of text entered by the user – /private/var/mobile/Library/Keyboard/dynamic-text.dat
- This text is cached as part of the device's autocorrect feature.
- Often, text is entered in the order it is typed, enabling you to piece together phrases or sentences of typed communication. Think of it in terms of a keyboard logger. Although the text displayed may be out of order or consist of various "slices" of different threads assembled together.

Demo - Evernote

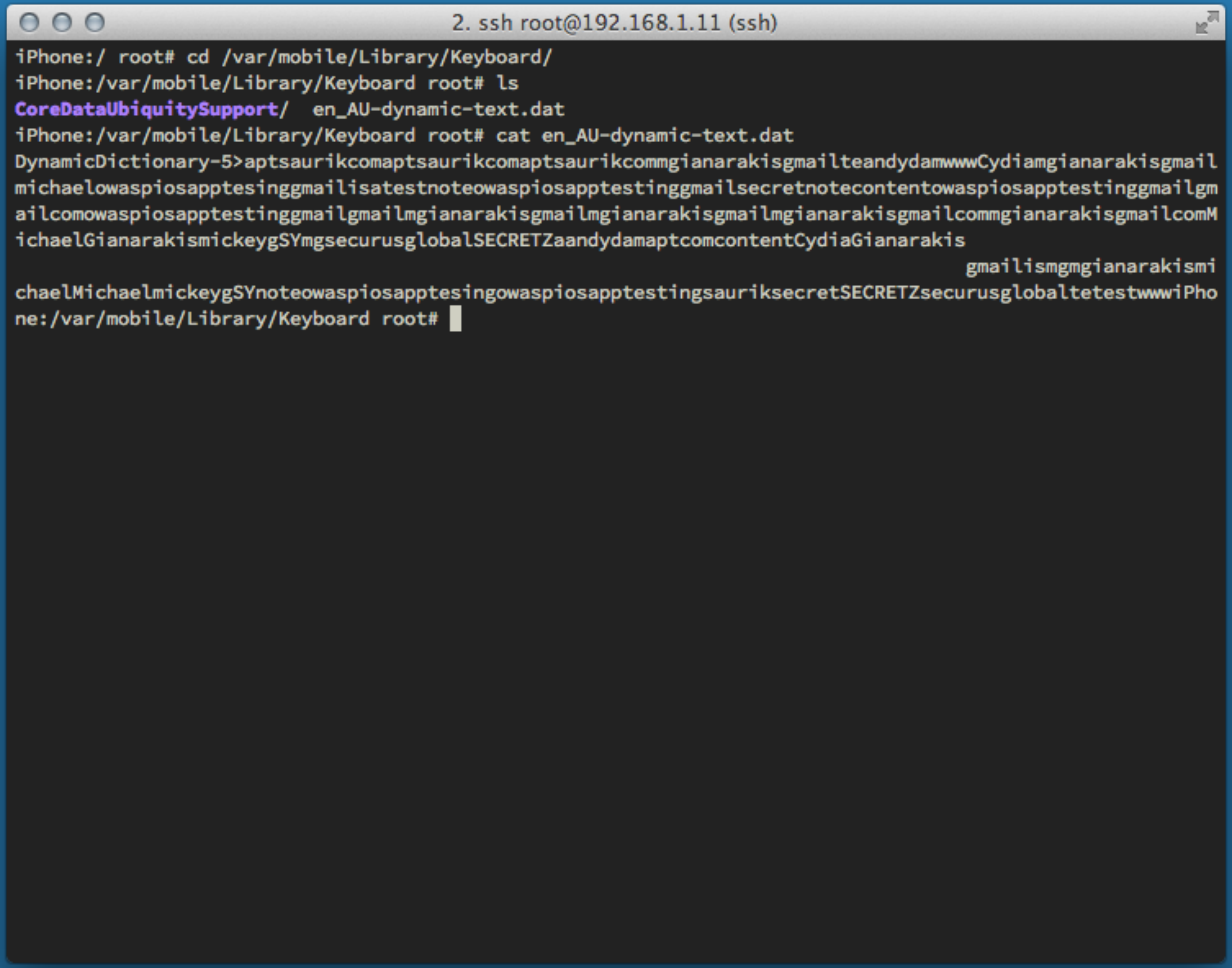
Data Security



Super Secret Note

Super secret note content
SECRETZ!!!

Data Security



```
iPhone:/ root# cd /var/mobile/Library/Keyboard/
iPhone:/var/mobile/Library/Keyboard root# ls
CoreDataUbiquitySupport/  en_AU-dynamic-text.dat
iPhone:/var/mobile/Library/Keyboard root# cat en_AU-dynamic-text.dat
DynamicDictionary-5>aptsaurikcomaptsaurikcomaptsaurikcommgianarakisgmailteandydamwwwCydiamgianarakisgmail
michaelowaspiosapptesinggmailisatestnoteowaspiosapptestinggmailsecretnotecontentowaspiosapptestinggmailgm
ailcomowaspiosapptestinggmailgmailmgianarakisgmailmgianarakisgmailmgianarakisgmailcommgianarakisgmailcomM
ichaelGianarakismickeygSYmgsecurusglobalSECRETZaandydamaptcomcontentCydiaGianarakis
                                                                                     gmailismgmgianarakismi
chaelMichaelmickeygSYnoteowaspiosapptesingowaspiosapptestingsauriksecretSECRETZsecurusglobaltetestwwwiPho
ne:/var/mobile/Library/Keyboard root# 
```

Data Security

- To avoid writing data to this cache developers should turn autocorrect off in text fields whose input should remain private, or consider writing their own keyboard class for your application.
- To turn autocorrect off, the developer should set `textField.autocorrectionType` to `UITextAutocorrectionTypeNo` for all sensitive text fields.

Data Security

- **Webkit Storage**
- Webkit Storage Some applications cache data in WebKit storage databases. Safari also stores information from various sites in WebKit databases. It's good to scan through WebKit caches to find any sensitive information or information that may be useful when attacking an application.
- The **/private/var/mobile/Library/WebKit** directory contains a LocalStorage directory with unique databases for each website. Often, these local storage databases can also be found within a third party application's Library folder, and contain some cached information downloaded or displayed in the application. The application or website can define its own local data, and so the types of artefacts found in these databases can vary.
- In addition to WebKit storage the **/private/var/mobile/Library/Caches/Safari/Thumbnails** directory contains screenshots of the last active browser pages viewed with WebKit. If a third-party application displays web pages, reduced versions of these pages may get cached here. Even though the sizes are reduced, however, much of the text can still be readable. This is a particular problem with secure email and banking clients using WebKit, as account information and confidential email can be cached here.

Data Security

- **Other Interesting Files**

- **/private/var/mobile/Library/Cookies/Cookies.binarycookies** – Contains a standard binary cookie file containing cookies saved when web pages are displayed on the device.

- **/private/var/mobile/Library/Caches/Safari/** – In this directory there is a Thumbnails directory containing screenshots of recently viewed web pages, along with a timestamp of when the thumbnail was made. There is a property list named RecentSearches.plist, containing the most recent searches entered into Safari's search bar.

- **/private/var/mobile/Library/SpringBoard/LockBackground.cpbitmap** – The current background wallpaper set for the device. This is complemented with a thumbnail named LockBackgroundThumbnail.jpg in the same directory. Although this is unlikely to contain any sensitive information it may be useful in black-box scenarios where we must use a decoy device or social engineering.

Data Security

- **Property List Files**

- Property lists are XML manifests used to describe various configurations, states, and other stored information. Applications typically have a number of property list files to manage configuration and state and are usually located in the Application's Library folder. You should scan these files to see if any sensitive information is stored in them. It is not uncommon for developers to store PINs and Passwords in these files.

- Property lists can be formatted in either ASCII or binary format. When formatted for ASCII, a file can be easily read using any standard text editor, because the data appears as XML. When formatted for binary, a property list file must be opened by an application capable of reading or converting the format to ASCII.

Demo

Data Security

- **Logging Files**
- Logging can prove a valuable resource for debugging during development, however in some cases it can leak sensitive or proprietary information, which is then cached on the device until the next reboot.
- Logging in Objective-C is typically performed using the NSLog method that causes a message to be sent to the Apple System Log. These console logs are not only accessible using the Xcode organiser application but by any app installed on the device, using the ASL library.
- In some cases jailbreaking a device will cause NSLog output to be redirected to syslog. In this scenario, it is possible that sensitive information may be stored on the file system in syslog.

Data Security

- It is unfortunately very common to find apps logging sensitive information and not even realising it. Particularly when it comes to authentication functions.
- Examples seen during tests:
    - Logging requests and responses of a SOAP web service including the authentication request with the username and password and the response with the authenticated session key.
    - Logging HTTP requests and responses including logging authentication credentials and credit card details
    - Logging password changes (old password and new password)
    - Logging GPS coordinates tied to certain actions
    - Logging frame coordinates of tap events in a PIN view

Demo - Evernote

Data Security

- Developers should avoid using NSLog to log sensitive or proprietary information and if they must log this information in development for debugging purposes special care should be taken to ensure that this code is not pushed to production. One way to do this is to redefine NSLog with a pre-processor macro such as “#define NSLog(...)”.

Data Security

- **Keychain**
  - Keychain is an encrypted data store.
  - Good place to store sensitive data on the device (only if you have to).
  - But store it securely (no cleartext!). Passwords should be hashed using a strong one-way algorithm with a random salt.
  - At a minimum the keychain accessibility constant should be set to **`kSecAttrAccessibleWhenUnlocked`** or **`kSecAttrAccessibleWhenUnlockedThisDeviceOnly`**.
  - Use the keychain_dumper tool to dump the keychain data and review for sensitive information.

Demo

Assessing iOS Applications - Binary Analysis

Binary Analysis

- **Position Independent Executable**
- Position Independent Executable (PIE) is an exploit mitigation security feature that allows an application to take full advantage of ASLR.
- Applications can have either partial ASLR or full ASLR depending on whether they have been compiled with support for PIE.
- With PIE, all the application memory regions are randomised and iOS will load a PIE enabled binary at a random address each time it is executed.
- Without PIE, load the base binary at a fixed address and use a static location for the dynamic linker.

Binary Analysis

- To determine if the application has been compiled with the PIE flag set inspect the Mach-O header using otool:
    - otool -hv xxxxxx

Binary Analysis

- To enable PIE the application must be compiled using the –fPIE –pie flag. Using XCode this can be enabled/disabled using the “Generate Position-Dependent Code” option from the compiler code generation build setting.

Binary Analysis

- Stack Smashing Protection

- iOS applications can look to add additional exploit mitigation at compile time through stack smashing protection.

- Stack canaries provide some protection against buffer overflows by placing a random, known value before the local variables. The stack canary is checked upon return of the function. If an overflow occurs and the canary is corrupted, the application is able to detect and protect against the overflow.

Binary Analysis

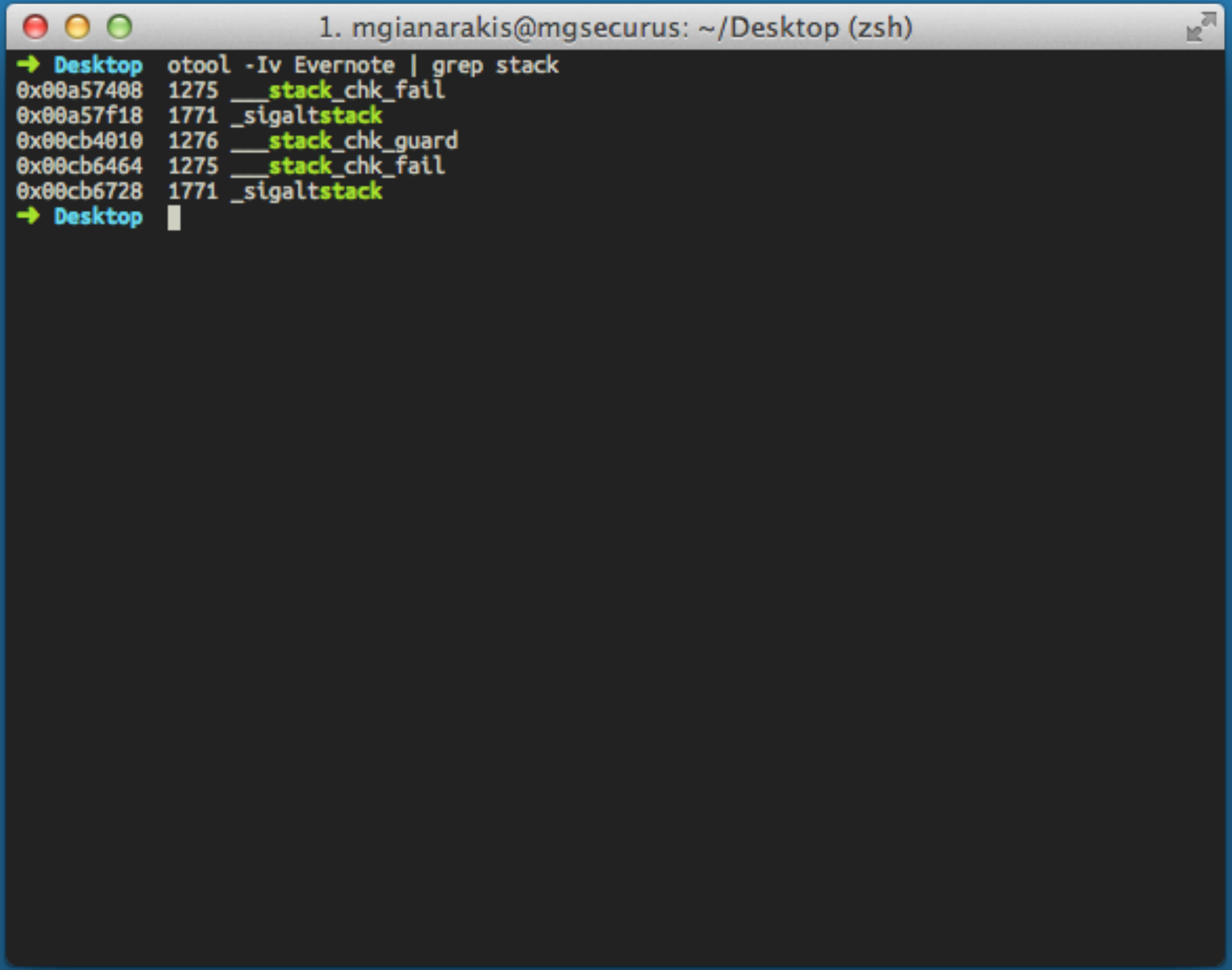
- Determining whether the application has been compiled with the stack smashing flag set can be achieved by examine the symbol table of the binary. If stack smashing protection is compiled into the application, two undefined symbols will be present:
    - ___stack_chk_fail
    - ___stack_chk_guard

Binary Analysis

- To dump the symbol table use otool:
  - otool -I -v xxxxxx | grep stack

Demo - Evernote

Binary Analysis



```
➜ Desktop  otool -Iv Evernote | grep stack
0x00a57408  1275 ___stack_chk_fail
0x00a57f18  1771 _sigaltstack
0x00cb4010  1276 ___stack_chk_guard
0x00cb6464  1275 ___stack_chk_fail
0x00cb6728  1771 _sigaltstack
➜ Desktop
```

Binary Analysis

- Stack smashing protection can be enabled by specifying the –fstack-protector-all compiler flag.

Binary Analysis

- **Automatic Reference Counting**
- Automatic Reference Counting (ARC) was introduced in iOS SDK 5.0 to move the responsibility of memory management from the developer to the compiler.
- ARC offers some security benefits as it reduces the likelihood of developers introducing memory corruption vulnerabilities in to apps, specifically object use after free and double free vulnerabilities.

Binary Analysis

- Determining whether the application has been compiled with the ARC enabled can be achieved by examine the symbol table of the binary. The following symbols highlight the presence of ARC:
    - _objc_retainAutoreleaseReturnValue
    - _objc_autoreleaseReturnValue
    - _objc_storeStrong
    - _objc_retain
    - _objc_release
    - _objc_retainAutoreleasedReturnValue

Binary Analysis

- To dump the symbol table use otool:
    - otool -I -v xxxxxx | grep _objc_release

Binary Analysis

- **Obtaining a Class Map**
- Given that it is a reflective language, Objective-C requires method names and types to be stored in the binary, class names and references etc.
- If the application is using protocols, categories, string objects, instance variables, or other Objective-C components, these will also be stored in their own data segments as well.

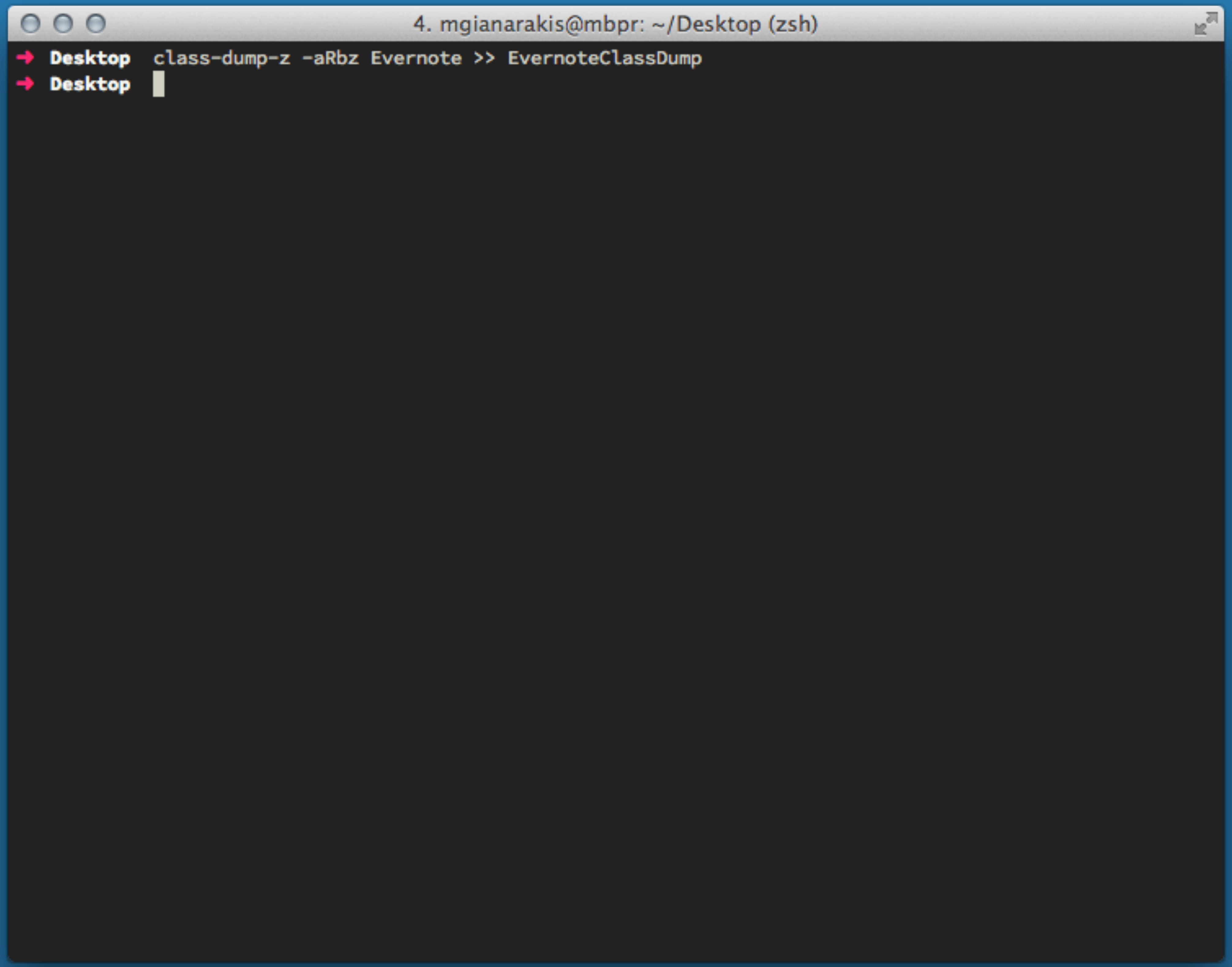
Binary Analysis

- class-dump-z will output the equivalent of an Objective-C header file (my text editor even recognises it as such and applies syntax highlighting) identifying each class compiled into the program and its associated methods, instance variables and properties.

- A full class dump can give you enormous insight into what's going on inside an application, and essentially provide a "map" for navigating around in it. This information will come in handy when performing attacks against the application through manipulation of the runtime

Demo - Evernote

Binary Analysis



```
➜ Desktop  class-dump-z -aRbz Evernote >> EvernoteClassDump
➜ Desktop  
```

Assessing iOS Applications - Runtime Manipulation

Runtime Analysis

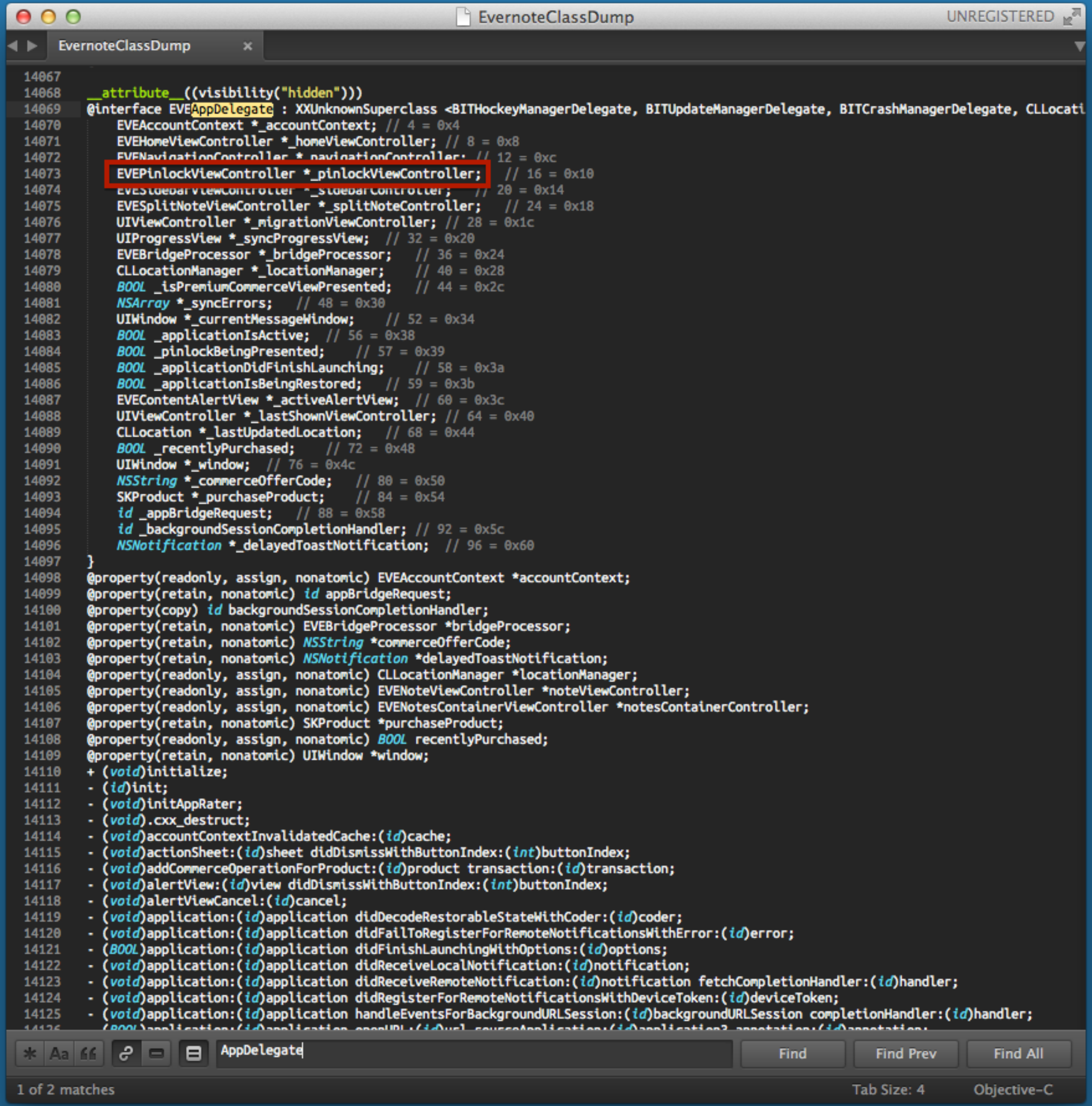
- As a reflective language, Objective-C can observe and modify it's own behaviour at runtime.
- The Objective-C runtime allows a program to create and call ad hoc methods as well as create ad hoc classes and methods on the fly.
- These properties allows us (and attackers) to manipulate and abuse the runtime of the application.
- Typically you can manipulate the runtime to bypass security locks, break logic checks, escalate privilege or steal information from memory.

Runtime Analysis

- For most runtime analysis tasks I use cycript
- One of many great tools by Jay Freeman (@saurik)
- Cycript is a “programming language designed to blend the barrier between Objective-C and JavaScript”.
- A lot better than it sounds.
- Can use gdb as well
- Although most of what I will talk about will be manipulating a running app, it’s possible that all these issues can be persistent through use of tools such as MobileSubstrate

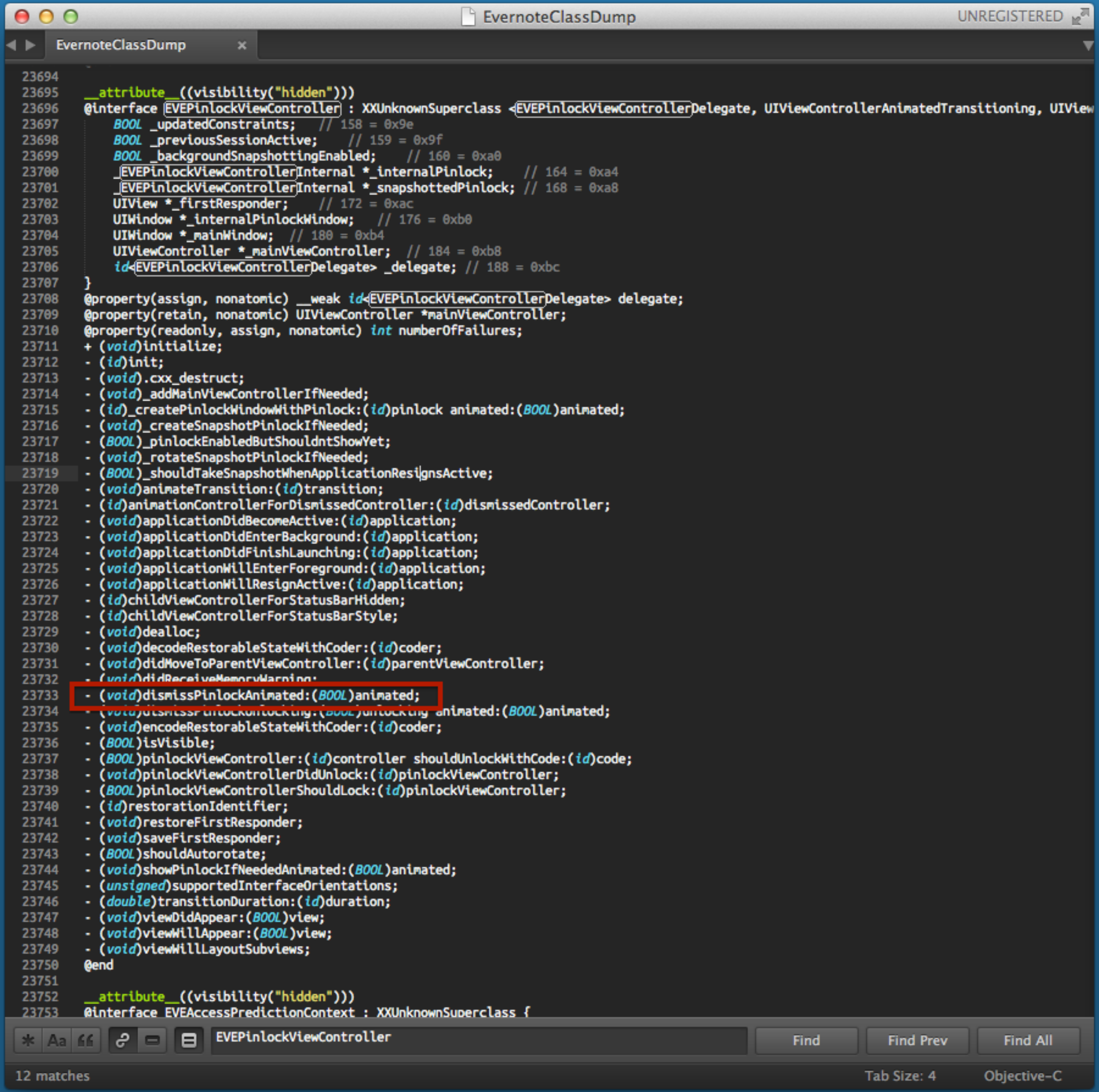
Demo - Evernote

Runtime Analysis



```
__attribute__((visibility("hidden")))
@interface EVEAppDelegate : XXUnknownSuperclass <BITHockeyManagerDelegate, BITUpdateManagerDelegate, BITCrashManagerDelegate, CLLocati
    EVEAccountContext *_accountContext; // 4 = 0x4
    EVEHomeViewController *_homeViewController; // 8 = 0x8
    EVENavigationController *_navigationController; // 12 = 0xc
    EVEPinlockViewController *_pinlockViewController;   // 16 = 0x10
    EVESidebarViewController *_sidebarController;   // 20 = 0x14
    EVESplitNoteViewController *_splitNoteController;   // 24 = 0x18
    UIViewController *_migrationViewController; // 28 = 0x1c
    UIProgressView *_syncProgressView;  // 32 = 0x20
    EVEBridgeProcessor *_bridgeProcessor;   // 36 = 0x24
    CLLocationManager *_locationManager;    // 40 = 0x28
    BOOL _isPremiumCommerceViewPresented;   // 44 = 0x2c
    NSArray *_syncErrors;   // 48 = 0x30
    UIWindow *_currentMessageWindow;    // 52 = 0x34
    BOOL _applicationIsActive;  // 56 = 0x38
    BOOL _pinlockBeingPresented;    // 57 = 0x39
    BOOL _applicationDidFinishLaunching;    // 58 = 0x3a
    BOOL _applicationIsBeingRestored;   // 59 = 0x3b
    EVEContentAlertView *_activeAlertView;  // 60 = 0x3c
    UIViewController *_lastShownViewController; // 64 = 0x40
    CLLocation *_lastUpdatedLocation;   // 68 = 0x44
    BOOL _recentlyPurchased;    // 72 = 0x48
    UIWindow *_window;  // 76 = 0x4c
    NSString *_commerceOfferCode;   // 80 = 0x50
    SKProduct *_purchaseProduct;    // 84 = 0x54
    id _appBridgeRequest;   // 88 = 0x58
    id _backgroundSessionCompletionHandler; // 92 = 0x5c
    NSNotification *_delayedToastNotification;  // 96 = 0x60
}
@property(readonly, assign, nonatomic) EVEAccountContext *accountContext;
@property(retain, nonatomic) id appBridgeRequest;
@property(copy) id backgroundSessionCompletionHandler;
@property(retain, nonatomic) EVEBridgeProcessor *bridgeProcessor;
@property(retain, nonatomic) NSString *commerceOfferCode;
@property(retain, nonatomic) NSNotification *delayedToastNotification;
@property(readonly, assign, nonatomic) CLLocationManager *locationManager;
@property(readonly, assign, nonatomic) EVENoteViewController *noteViewController;
@property(readonly, assign, nonatomic) EVENotesContainerViewController *notesContainerController;
@property(retain, nonatomic) SKProduct *purchaseProduct;
@property(readonly, assign, nonatomic) BOOL recentlyPurchased;
@property(retain, nonatomic) UIWindow *window;
+ (void)initialize;
- (id)init;
- (void)initAppRater;
- (void).cxx_destruct;
- (void)accountContextInvalidatedCache:(id)cache;
- (void)actionSheet:(id)sheet didDismissWithButtonIndex:(int)buttonIndex;
- (void)addCommerceOperationForProduct:(id)product transaction:(id)transaction;
- (void)alertView:(id)view didDismissWithButtonIndex:(int)buttonIndex;
- (void)alertViewCancel:(id)cancel;
- (void)application:(id)application didDecodeRestorableStateWithCoder:(id)coder;
- (void)application:(id)application didFailToRegisterForRemoteNotificationsWithError:(id)error;
- (BOOL)application:(id)application didFinishLaunchingWithOptions:(id)options;
- (void)application:(id)application didReceiveLocalNotification:(id)notification;
- (void)application:(id)application didReceiveRemoteNotification:(id)notification fetchCompletionHandler:(id)handler;
- (void)application:(id)application didRegisterForRemoteNotificationsWithDeviceToken:(id)deviceToken;
- (void)application:(id)application handleEventsForBackgroundURLSession:(id)backgroundURLSession completionHandler:(id)handler;
- (BOOL)application:(id)application openURL:(id)url sourceApplication:(id)application2 annotation:(id)annotation;
```

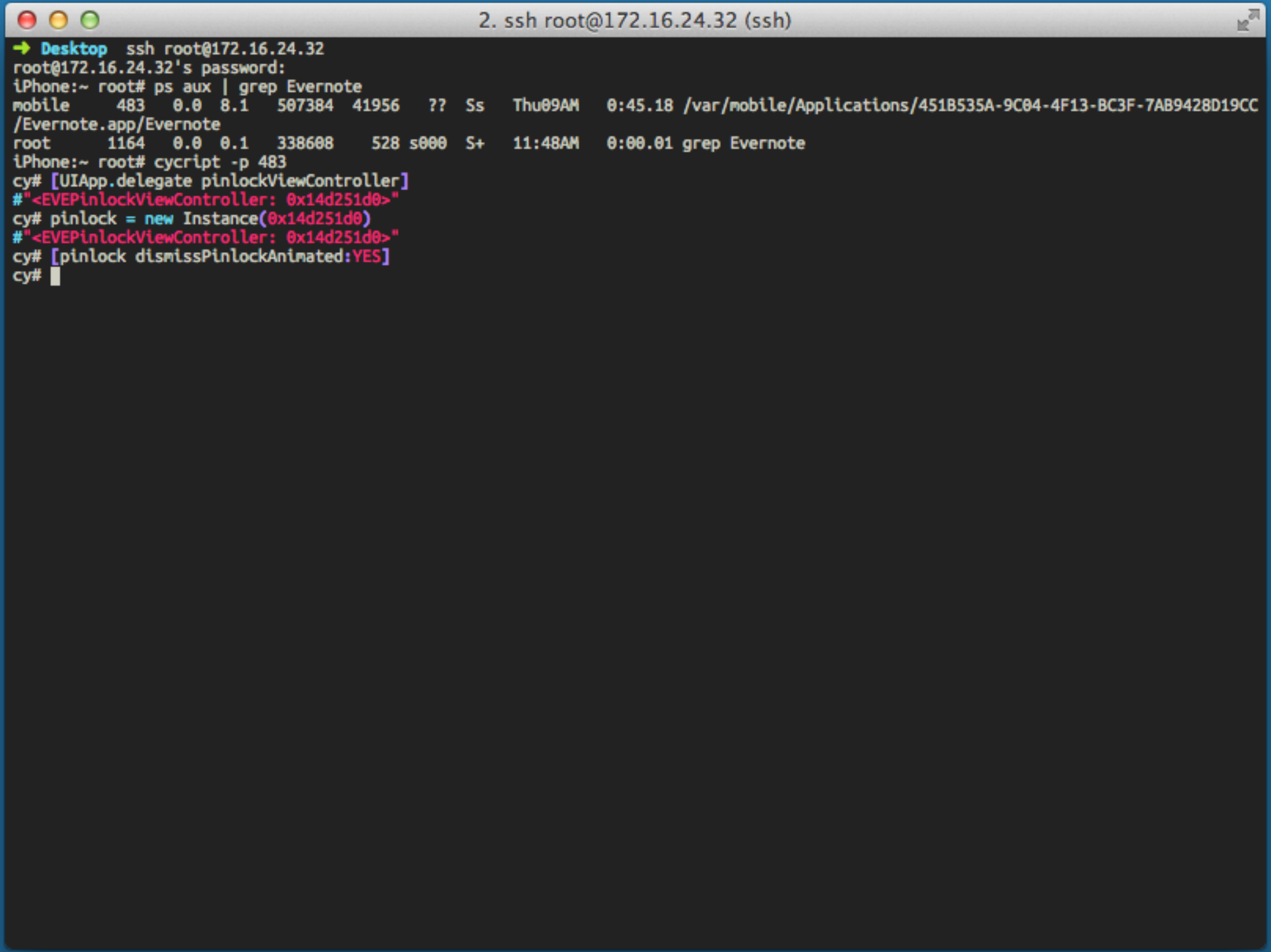
Runtime Analysis



```
__attribute__((visibility("hidden")))
@interface EVEPinlockViewController : XXUnknownSuperclass <EVEPinlockViewControllerDelegate, UIViewControllerAnimatedTransitioning, UIView
    BOOL _updatedConstraints;   // 158 = 0x9e
    BOOL _previousSessionActive;    // 159 = 0x9f
    BOOL _backgroundSnapshottingEnabled;    // 160 = 0xa0
    _EVEPinlockViewControllerInternal *_internalPinlock;    // 164 = 0xa4
    _EVEPinlockViewControllerInternal *_snapshottedPinlock; // 168 = 0xa8
    UIView *_firstResponder;    // 172 = 0xac
    UIWindow *_internalPinlockWindow;   // 176 = 0xb0
    UIWindow *_mainWindow;  // 180 = 0xb4
    UIViewController *_mainViewController;  // 184 = 0xb8
    id<EVEPinlockViewControllerDelegate> _delegate; // 188 = 0xbc
}
@property(assign, nonatomic) __weak id<EVEPinlockViewControllerDelegate> delegate;
@property(retain, nonatomic) UIViewController *mainViewController;
@property(readonly, assign, nonatomic) int numberOfFailures;
+ (void)initialize;
- (id)init;
- (void).cxx_destruct;
- (void)_addMainViewControllerIfNeeded;
- (id)_createPinlockWindowWithPinlock:(id)pinlock animated:(BOOL)animated;
- (void)_createSnapshotPinlockIfNeeded;
- (BOOL)_pinlockEnabledButShouldntShowYet;
- (void)_rotateSnapshotPinlockIfNeeded;
- (BOOL)_shouldTakeSnapshotWhenApplicationResignsActive;
- (void)animateTransition:(id)transition;
- (id)animationControllerForDismissedController:(id)dismissedController;
- (void)applicationDidBecomeActive:(id)application;
- (void)applicationDidEnterBackground:(id)application;
- (void)applicationDidFinishLaunching:(id)application;
- (void)applicationWillEnterForeground:(id)application;
- (void)applicationWillResignActive:(id)application;
- (id)childViewControllerForStatusBarHidden;
- (id)childViewControllerForStatusBarStyle;
- (void)dealloc;
- (void)decodeRestorableStateWithCoder:(id)coder;
- (void)didMoveToParentViewController:(id)parentViewController;
- (void)didReceiveMemoryWarning;
- (void)dismissPinlockAnimated:(BOOL)animated;
- (void)dismissPinlockUnlocking:(BOOL)unlocking animated:(BOOL)animated;
- (void)encodeRestorableStateWithCoder:(id)coder;
- (BOOL)isVisible;
- (BOOL)pinlockViewController:(id)controller shouldUnlockWithCode:(id)code;
- (void)pinlockViewControllerDidUnlock:(id)pinlockViewController;
- (BOOL)pinlockViewControllerShouldLock:(id)pinlockViewController;
- (id)restorationIdentifier;
- (void)restoreFirstResponder;
- (void)saveFirstResponder;
- (BOOL)shouldAutorotate;
- (void)showPinlockIfNeededAnimated:(BOOL)animated;
- (unsigned)supportedInterfaceOrientations;
- (double)transitionDuration:(id)duration;
- (void)viewDidAppear:(BOOL)view;
- (void)viewWillAppear:(BOOL)view;
- (void)viewWillLayoutSubviews;
@end

__attribute__((visibility("hidden")))
@interface EVEAccessPredictionContext : XXUnknownSuperclass {
```

```
EVEPinlockViewController
```

Runtime Analysis



```
➜ Desktop  ssh root@172.16.24.32
root@172.16.24.32's password:
iPhone:~ root# ps aux | grep Evernote
mobile     483   0.0  8.1   507384  41956   ??  Ss   Thu09AM   0:45.18 /var/mobile/Applications/451B535A-9C04-4F13-BC3F-7AB9428D19CC
/Evernote.app/Evernote
root      1164   0.0  0.1   338608    528 s000  S+   11:48AM   0:00.01 grep Evernote
iPhone:~ root# cycript -p 483
cy# [UIApp.delegate pinlockViewController]
#"<EVEPinlockViewController: 0x14d251d0>"
cy# pinlock = new Instance(0x14d251d0)
#"<EVEPinlockViewController: 0x14d251d0>"
cy# [pinlock dismissPinlockAnimated:YES]
cy# 
```

Runtime Analysis

- **Bypassing Runtime Protections - Jailbreak Detection/Prevention**
- It is now more common for applications to implement some form of jailbreak detection.
- This may be implemented to detect a jailbreak and initiate some tamper prevention/detection or to block a user for using the application on a jailbroken phone.
- No method will be completely secure from a determined attacker with access to the application and a debugger however this increases the difficulty for an attacker and will decrease the number of qualified attackers.

Runtime Analysis

- The techniques to defeat jailbreak detection will depend on how it’s implemented.
- Most of the time it’s a simple filesystem check. But can also be a sandbox integrity check or some combination of both.
- Couldn’t find an app that I hadn’t tested for a client that had jailbreak detection for a live demo.
- But I have screenshots.

Demo - Jailbreak Prevention Bypass

Runtime Analysis

- **Securing the runtime:**
- **Anti-debugging**
  - Periodically monitoring the process state flag to determine if application process is being debugged and then terminating the process.
  - Calling the ptrace request (DENY_ATTACH) which has the effect of denying future tracing of the process. This can be done with the following function call from within the application:
    - `ptrace(31,0,0,0)`

Runtime Analysis

- **Inline functions**
  - One of the simplest methods to hijack the behaviour inside an application is to hijack a particular function (as we saw before).
  - Turning sensitive functions into inline functions will complicate this style of attack.
  - Inline functions are functions in which the compiler inserts the function body within the code every place it is called.
  - Attackers have to find every occurrence of code and patch it out of memory, complicating the attack.

Runtime Analysis

- **Validating address space**
  - Validating the address space for critical methods increases the complexity of code injection attacks.
  - An attacker must find ways to inject code into the existing address space that the valid code lives in.
  - Use `dladdr` function to verify whether code has come from the application.

Assessing iOS Applications - Transport Security

Transport Security

- Apple iOS transport security

There is none

Transport Security

- I won’t be discussing goto fail-gate.
- Hubert Seiwert (@hubert3) is presenting an analysis on the SSL bug CVE-2014-1266 at Ruxmon tomorrow night at 6.

Transport Security

- The main issues from a penetration testing perspective:
  - Checking that traffic is encrypted (pretty standard)
  - Checking application level SSL validation
  - Dealing with certificate pinning

Transport Security

- **Bypassing SSL validation/cert pinning**
  - Manual way:
    - Patch core SSL validation methods
      - NSURLConnection
      - SecTrustEvaluate()
      - SSLSetSessionOption() and SSLHandshake() in SecureTransport

Transport Security

- Easy-mode:
    - Use ios-ssl-killswitch from ISEC Partners
        - Patches SSLSetSessionOption() and SSLHandshake() in SecureTransport
        - Available here -> https://github.com/iSECPartners/ios-ssl-kill-switch
        - Details on how it works -> http://nabla-c0d3.github.io/blog/2013/08/20/ios-ssl-kill-switch-v0-dot-5-released/

Transport Security

- Often times SSL validation will be a toggle in the application.
- This is usually a holdover from development where the development environment does not have valid certs.
- Typically encapsulated in a variable somewhere.
- Variables in Objective-C are easy to manipulate (as we saw before).

Demo

Assessing iOS Applications - Data Validation

Data Validation

- **Cross-Site Scripting**
  - UIWebView is built upon WebKit and uses the same core frameworks as Safari and MobileSafari.
  - UIWebViews can display remote content and also support JavaScript execution.
  - iOS are just as vulnerable to cross-site scripting if un-sanitised user input makes it into the web view.
  - There is no configurable option to disable Javascript within the API. As a result iOS apps can be affected by cross-site scripting.
  - Tested in pretty much the same way you would normally test it.

Data Validation

- **SQL Injection**
  - SQL injection occurs in an iOS when un-sanitised user input is used to construct a dynamic SQL statement.
  - It is rare for developers to sanitise user input in this case and SQL injection into client-side databases is common.
  - However it is not considered a particularly high risk given that an attacker would have access to the database directly.
  - Tested in pretty much the same way you would normally test it.

Other Tests

- Similar to a web app you should also be looking for things like:
  - Passwords controls
  - Session management
  - Authorisation etc.

References and Other Recommended Reading

- Hacking and Securing iOS Applications by Jonathan Zdiarski
- iOS Hacker's Handbook by Charlie Miller, Dion Blazakis, Dino DaiZovi, Stefan Esser, Vincenzo Iozzo and Ralf-Philip Weinmann
- iOS Application (In)Security by Dominic Chell
- Secure Development on iOS by David Thiel
- Auditing iPhone and iPad Applications by Ilja van Sprundel

Fin.

@mgianarakis on Twitter
eightbit.io on the Web