Global Information Assurance Certification Paper

iPwn Apps: Pentesting iOS Applications

GIAC (GPEN) Gold Certification

Author: Adam Kliarsky, adam.kliarsky@gmail.com
Advisor: Hamed Khiabani, Ph.D.

Accepted: May 4, 2014

Abstract

Since Apple's iPhone graced us with its presence in 2007, the mobile landscape has been forever changed. Smart phones and tablets and variations in between have permeated all demographics, almost to the point where not owning one is strange. The exponential growth has opened up new opportunities; letting users conduct computing activates while on the go…but so too has it opened up new security risks. It is important for penetration testers to be able to keep up with the ever-expanding user demand that brings in new devices, new apps and new risks. Understanding these devices, their apps, and the points of exposure they bring are absolutely essential. It is the goal of this paper to provide insight and methodology into penetration testing mobile devices and their applications.

1. Introduction

The growth of mobile device usage in both personal and professional environments continues to grow. Information from Pew Research show that as of January 2014, 90% of Americans have a cell phone, of which 58% are smart phones (Research, 2013). In an article published in February of 2013, Mashable blog author Samanta Murphy Kelly stated, "In 2012, the number of mobile-connected tablets grew 2.5 times to 36 million, and each tablet generated 2.4 times more traffic than the average smartphone. Android also beat iPhone levels of data usage in the U.S. and Western Europe." (Kelly, 2013). Mobile devices have become a staple computing device used in our lives today. Whether it is at home, at work, at school, or in transit, mobile devices including smart phones and tablets have enabled us to conduct computing tasks on the go. Mashable shows supporting information for this; mobile devices account for 17.4% of web traffic – globally – up 6% from the previous year (Fox, 2013).

The massive increased use in mobile devices brings a corresponding growth in mobile applications, driven partially from user demands as well as opportunistic developers looking to capitalize on this potential. According to Apple, "The App Store has more than 1 million apps and counting" (Apple.com, Apple - iPhone 5s - App Store, 2014). The apps are broken down into categories, such as games, productivity, education, business, entertainment and the list goes on. Gartner Research anticipates that mobile app downloads in 2014 will increase by 34,828 million, bringing the download count to 127,704 million from 92,876 million in 2013…and to 253,914 million by 2017 (Rivera & van der Meulen, 2013).

So what does this ever growing market mean in terms of risk to the end user, the enterprise or other organization? To better understand this, we need to think in terms of application security…and keep in mind that mobile environments map to full computing environments. Mobile devices run an operating system which themselves are subject to software security flaws. The apps that run on these platforms are likewise at risk from inherent programming flaws…but the apps suffer a bit more. Software Development Kits and development programs have been opened up to the public, leading to the explosive

growth of app development/distribution. Software development companies, private organizations, and individuals alike have donned their development hat and have become coders. Now each programmer has his/her own way of coding, ensuring no consistency whatsoever. And there is no standard SDLC (Software Development Life Cycle) governing development and assuring quality. The current app development process consists of three phases; the development phase, the test phase, and the distribute phase.

Figure 1 - Apple iOS Development Process

Apple does maintain ownership of the app distribution process, thus controlling which apps are available to the public via the iTunes App Store. As part of this process Apple conducts 'App Review' where they "review every app submitted based on a set of technical, content, and design criteria" (Apple.com, App Review, 2014).

So Apple has a process to review every app submitted. But what exactly is the process? And for that matter how thorough is it? Do they implement a backend SDLC testing process; conducting threat modeling at design, code analysis at implementation, and dynamic analysis/fuzzing during verification? Considering how many apps are submitted, and how many go to market, it's hard to imagine. In 2012 the Apple App Store was receiving 26,000 submissions each week (Sarno, 2012), which would be a daunting challenge for any review team/process. As for the approved apps, "The App Store added about 75,000 apps between Sept. 12, 2012 (the day of the iPhone 5 release) and Jan. 7, 2013. That constitutes 641 new apps in the App Store per day - and more than 19,000 new apps every month" (Rowinski, 2013). Those are just the Apple approved apps. On jail broken devices this process is moot; users can install what they want.

This leads us to the premise of this paper. Mobile apps extend the overall attack surface of connected users. Individuals and organizations alike are affected, creating potential attack points all over. Penetration testers need a systematic process by which they can assess a mobile app, identify potential attack vectors and subsequent risk to users and organizations.

2. Understanding iOS Security

2.1. Current Architecture

Before delving into penetration testing mobile platforms and the apps they run, it is imperative to understand what security measures exist. In May of 2012, Apple published a paper outlining the iOS Security architecture. Apple Platform Security team manager Dallas De Atley discussed this in Las Vegas at BlackHat the same year. Apple has since published two revisions, one as recently as February 2014 (Apple, iOS Security, 2014), which we'll use as to understand the current posture of iOS security. As described in the white paper, the components of iOS Security consist of system security, encryption and data protection, App security, network security, Internet services, and device controls (Apple, iOS Security, 2014).

2.1.1. System Security

The foundation of the iOS platform relies on its System Security. This consists of the Secure Boot Chain, System Software Authorization, Secure Enclave, and Touch ID.

The *Secure Boot Chain*, as the name implies is a chain, or a sequence of trusted events that occur during the boot process. The boot process begins with the Boot ROM which has immutable, trusted code built in. The Boot ROM contains Apple's Root Certificate Authority (CA) public key (Apple, iOS Security, 2014), which in turns verifies the authenticity of the Low-Level Bootloader (LLB). The chain of trust relies on this concept, where each element verifies the authenticity of the next for the boot process to continue. The Boot ROM loads the LLB, which then calls iBoot (next-stage bootloader), which then executes the iOS kernel. If the Secure Boot Chain is unable to complete, where any element fails authenticity, the iOS device will display a message to the user and then enter Device Firmware Upgrade (DFU) mode.

Figure 2 - Secure Boot Chain

System Software Authorization prevents iOS devices from being downgraded to run older, insecure code, which could then be exploited by attackers. Furthermore it draws from the *System Security* process the ability to ensure that all code executed on the device is signed by Apple based on the Root CA public key in the Boot ROM. During updates, the System Software Authorization validates the device and authenticates updates (using the public key) with the authorization server to further safeguard the device.

Secure Enclave is a new feature that was built in to handle the Touch ID fingerprint transactions. If a match is found, access control permits access to the device and its functions; if not well then standard access controls denying the user apply. The Secure Enclave is a coprocessor that exists within the A7 chip, maintains its own secure boot chain as well as software update authorization; completely separate from the main process. The Secure Enclave controls data protection key management and can be relied on even if the device is compromised at kernel level.

The *Touch ID* is the biometric fingerprint reader built into the iPhone 5s, used as an additional (but optional) security requirement for the phone. When Touch ID is enabled, it prompts the user for a fingerprint anytime the sleep/wake button is pressed. It can also be used to control App Store purchases, so users don't have to enter their password every time. This additional layer of security has a 1 in 50,000 chance of being matched to someone other than the owner (Apple, iOS Security, 2014). Once swiped, the fingerprint image is stored in encrypted memory while being verified, then discarded.

2.1.2. Encryption and Data Protection

In addition to the system security protection mechanisms, iOS has encryption and data protection, which exist to add additional layers of security even when the device is compromised, lost, or stolen.

Hardware Security Features include a dedicated crypto engine utilizing 256-bit AES encryption. This operates between flash storage and memory, which makes it efficient and does not tax the system, resorting in low battery life etc. Additionally the UID and GID keys are written directly into the application processor during manufacturing.

File Data Protection is another protection mechanism iOS uses to safeguard files and data on the device. A technology called 'Data Protection' generates a 256-bit key for every file on the system, which is then used to encrypt the file. The key is maintained in a wrapper in the file's metadata and accessed when the file is opened. The metadata itself is encrypted using a per device key that is created upon device initialization.

Passcodes are an optional way for users to secure their devices and data. iOS supports different options; 4 digit passcodes, or alpha numeric passphrases. The passcode is rather important; it enables Data Protection for the file level encryption mentioned above, it adds entropy for encryption keys, it is integrated with the UID to further deter password attacks, and uses an iteration scheme that would make brute force attacks less likely..."it would take more than 5½ years to try all combinations of a six-character alphanumeric passcode with lowercase letters and numbers" (Apple, iOS Security, 2014).

Data Protection Classes are assigned to files when created on the file system. These classes help enforce policy style access controls, protecting files different as needed. The `NSFileProtectionComplete` class for example implements complete protection on the file while the device is locked. The class `NSFileProtectionCompleteUnlessOpen` protects the file unless the file is open, regardless if the device is locked or not. Then there is `NSFileProtectionCompleteUntilFirstUserAuthentication`, which protects the file until the user authenticates with a passcode. This is the default class used for 3rd party app data. The class `NSFileProtectionNone` adds no additional protection, other than the default encryption used on all files.

Keychain Data Protection ensures the security and protection of the keychain used to store passwords and other sensitive items. The class structure used to protect the keychain data is similar to the data protection classes.

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

Figure 3 - Keychain Data Protection Classes (Apple, iOS Security, 2014)

Keybags are how iOS stores and manages keys used in both file and keychain data protection mechanisms. There are four keybags used on iOS devices:

1. System keybag: maintains wrapped class keys, such as NSFileProtectionComplete, which is used for password authentication.
2. Backup keybag: contains keys associated with iTunes backups
3. Escrow keybag: contains keys used with iTunes sync operations and mobile device management.
4. iCloud Backup keybag: contain keys associated with iCloud backup operations.

Also, all iOS crypto modules, with the exception of those associated with Bluetooth, are U.S. Federal Information Processing Standard (FIPS) 140-2 compliant.

2.1.3. App Security

As apps represent potential attack vectors to the iOS device and user, there are layers of protection put in place that limit app exposure, prevent unauthorized code execution, and protect user data.

App Code Signing is a fundamental requirement of installed apps by iOS devices and is checked at runtime by the device to ensure it is from an approved source. Every developer who wishes to write/develop apps must be registered with Apple. Individuals need to join Apple's iOS Developer Program to write apps for the App Store. Corporations looking to write apps for internal users need to join Apple's iOS Developer Enterprise Program. Both programs validate the developer and issue certificates, which are then used to sign the app code.

Adam Kliarsky, adam.kliarsky@gmail.com

Runtime Process Security is the next layer of security that follows application code signing. Once deemed legit, runtime process security ensures third party apps are properly sandboxed, and are unable to interact with data of other apps not authorized. In addition to sandboxing, memory exploitation is mitigated through address space layout randomization (ASLR). The ARM processor's 'Never Execute' (NX) feature also helps restrict memory based attacks by marking it read-only.

2.1.4. Network Security

Transport layer protocols Secure Sockets Layer (*SSL*) version 3 and Transport Layer Security (*TLS*) version 1.0 are used in native apps like Safari, Mail, and other internet-based apps that need to communicate with servers over the internet. "High-level APIs (such as CFNetwork) make it easy for developers to adopt TLS in their apps, while low-level APIs (SecureTransport)" provide fine-grained control (Apple, iOS Security, 2014).

Additionally, iOS devices support per-app virtual private network (VPN) connectivity through the following:

- Juniper Networks, Cisco, Aruba Networks, SonicWALL, Check Point, Palo Alto Networks, Open SSL, and F5 Networks SSL-VPN using the appropriate client app from the App Store. These apps provide user authentication for the built-in iOS support.
- Cisco IPSec with user authentication by Password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret and certificates. Cisco IPSec supports VPN On Demand for domains that are specified during device configuration.
- L2TP/IPSec with user authentication by MS-CHAPV2 Password, RSA SecurID or CRYPTOCard, and machine authentication by shared secret.
- PPTP with user authentication by MS-CHAPV2 Password and RSA SecurID or CRYPTOCard.

Figure 4 - Supported VPN Vendors (Apple, iOS Security, 2014)

Wi-Fi is implemented on iOS devices with standard 802.11i authentication and encryption protocols. Both pre-shared key (WPA2 PSK) and enterprise 802.1x authentication methods exist for home and enterprise users, respectfully. The 802.1x EAP methods supported are EAP-TLS, EAP-TTLS, EAP-FAST, EAP-SIM, PEAPv0, PEAPv1, and LEAP (Apple, iOS Security, 2014).

The list of provisions and security measures goes on, but it is clear that Apple has come a long way with their iOS security features and takes product security seriously.

2.2. Caveats

Though Apple has built these measures into its iOS, there are some additional things to consider. What happens when the security of an iOS device is bypassed; ie 'jailbroken'? How does that affect the device, how does that affect the app? Since app developers rely on these core security features to be available when their app executes, it is a fair assumption to say that some unexpected issues may arise when this is not the case. And what about the mobile app development process; how do end users know the app their downloading has been properly reviewed for security? These issues illustrate the need and set the stage for for mobile app security assessments.

3. The Mobile Test Platform: Setup

3.1. Network Connectivity

To efficiently and effectively analyze the mobile device, a connection between the analysis workstation and device needs to be setup. This can be done via the standard USB interface, or alternatively over the network. This paper will use the network method, as it lends itself to the traditional network penetration test environment.

3.2. Jailbreak

The next thing that needs to be done is jailbreak. Security restrictions put in place by Apple make it difficult to analyze iOS devices. Therefore in order to analyze an iPhone/iPad etc, a jailbreak will need to be performed on the target device. For this paper, an iPhone4 running iOS v7.0.4 will be used along with Evad3rs' evasi0n7 jailbreak from http://evasi0n.com. The evasi0n7 jailbreak is an untethered jailbreak, meaning it does not require connectivity to a computer to boot up each time. The jailbreak is fairly straightforward; download the file, run it, follow the prompts and within a minute or so the phone will be jailbroken and ready to go.

At this point it is important to understand the operating environment. The core operating system that iOS is derived from is Mac OS X, which has its roots in BSD Unix.

The iOS implementation however is stripped down, meaning many of the common utilities found in standard operating environments do not exist.

3.3. Install required software

Since the newly jailbroken device is missing some core utilities, and before any real analysis can begin, we need some software; both for analysis and for a basic *nix style command line environment. The first thing that is required will be SSH, to facilitate remote logins over the WiFi network. Using Cydia, the "…alternative to Apple's App Store for "jailbroken" devices…" (Freeman, 2014), we can install OpenSSH. Opening Cydia for the first time will initialize the file system and prep it for use. Choosing 'Developer' when prompted for user profile type will provide access to the full plethora of software available. OpenSSH can be found under Featured Apps → File Managers within Cydia.

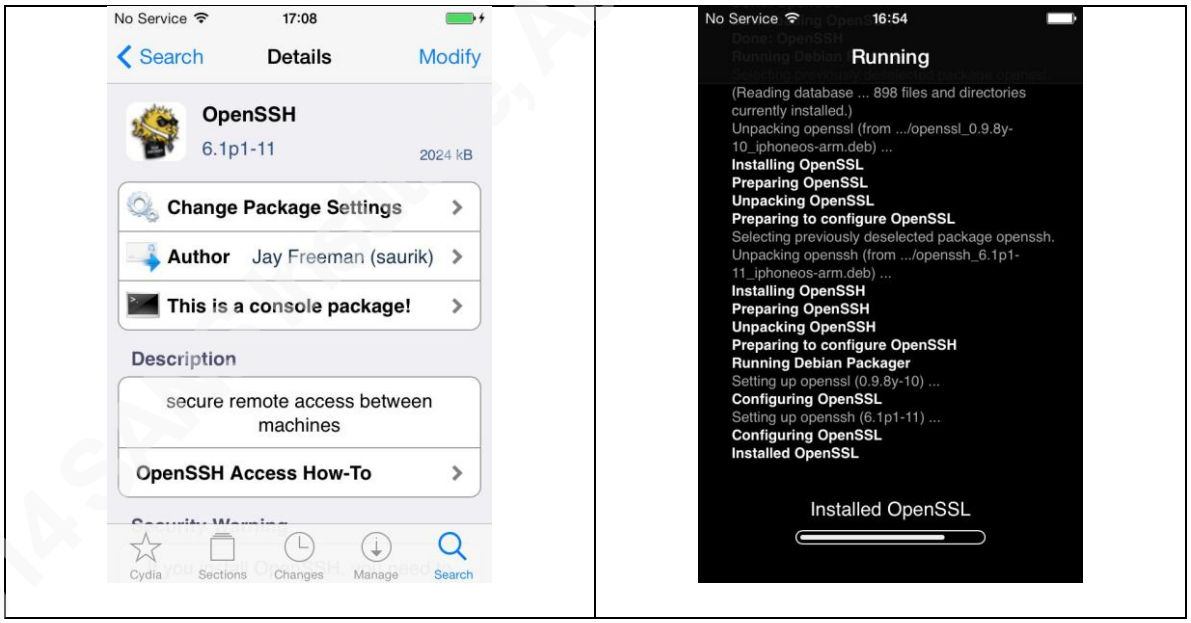

Figure 5 - Installing OpenSSH from Cydia

```
0x414141:~ adamkliarsky$ ssh -l root 192.168.1.139
The authenticity of host '192.168.1.139 (192.168.1.139)' can't be established.
RSA key fingerprint is 5d:69:77:58:e6:61:97:39:33:da:c3:68:d5:7e:3c:59.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.139' (RSA) to the list of known hosts.
root@192.168.1.139's password:
iph0wn:~ root#
```

Figure 6 - Connect to the iPhone over SSH

Next, use Cydia to find and install APT 0.6 Transitional. This will permit installation of additional packages from the command line (apt-get install <pkg>) while working in the phone environment. Then continue to install the following via Cydia (or internet):

- wget (Cydia)
- adv-cmds (Cydia)
- gdb (Cydia)
- class-dump (Cydia)
- Erica Utilities
- Snoop-it (Cydia → add 'repo.nesolabs.de' to the repositories)
- Introspy (http://isecpartners.github.io/Introspy-iOS/)
- unzip (Cydia)
- cycript (Cydia)
- Cydia Substrate (Cydia)
- Keychain Dumper
- clutch (https://github.com/KJCracks/Clutch/releases)

4. Analysis

When preparing to conduct a security assessment of an app, consider the goal; what is the intended outcome of the test: What exactly are we, as penetration testers, looking for? Some common analysis should include looking for broken authentication, memory flaws, encryption weaknesses, client-server communication, type of data transmitted etc. A penetration tester will need to be familiar with analyzing app traffic over the network, conducting both static and dynamic code analysis on the app binary, and looking at supporting data files of the app to truly gain a picture into the inner workings and potential risk an app might bring to the unsuspecting organization.

A penetration tester should develop a methodology that is suitable to his/her needs and can lend itself to consistent repeatable results. The OWASP IOS Application Security Testing Cheat Sheet has a nice list of tasks outlined in a well-mapped methodology that includes traffic analysis, code analysis, and other tasks

that might help discover security flaws (Cornea & Haddix, 2013).

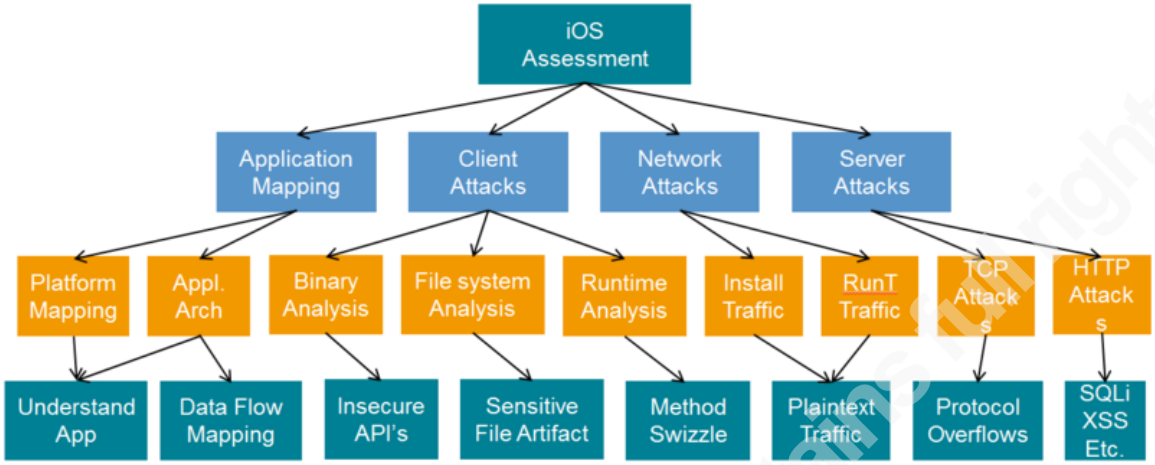

Figure 7 - OWASP IOS Application Security Testing Cheat Sheet

For the purposes of this paper, we'll focus on the following components that make up the basic app security tests:

- Static analysis
- Dynamic analysis
- Network Analysis
- Supporting File Analysis

4.1. Static Code Analysis

4.1.1. Identify The Path to the Target App on the iPhone

To begin analysis, we need to identify the file; the binary itself. The path of the application on the iOS device depends on whether it is a native iOS app (one that ships with the device) or an AppStore app. Native apps reside in the /Applications folder, while apps downloaded via the AppStore reside under /private/var/mobile/Applications. AppStore apps have a unique (and obscure) naming convention, which make it difficult to find at first. This information however can be found easily in the property list file *com.apple.mobile.installation.plist* file. Property list files (.plist) are used to store different types of data (Apple, About Property Lists, 2010); this one containing a list of installed applications and respective directories on the device. In order to read it, it will need to be converted using 'plutil' from the Erica Utilities suite. Copy the file com.apple.mobile.installation.plist from /private/var/mobile/Library/Caches/ to another location and run plutil with '-convert xml1' to convert the file.

```
→ iph0wn:~ root# cp /private/var/mobile/Library/Caches/com.apple.mobile.installation.plist .

→ iph0wn:~ root# plutil -convert xml1 com.apple.mobile.installation.plist
Converted 1 files to XML format
iph0wn:~ root#

→ iph0wn:~ root# cat com.apple.mobile.installation.plist | grep TargetApp
                <key>com.targetapp.com.targetapp</key>
                        <string>targetapp</string>
                        <string>targetapp</string>
                        <string>com.targetapp.com.TargetApp</string>
                        <string>TargetApp</string>
                        <string>com.TargetApp.usa.TargetApp</string>
                                <string>123ABC456C.com.targetapp.com.TargetApp</string>
                        <string>/private/var/mobile/Applications/0D5E8824-4598-4F96-AA0E-
E8ED4D907B00/TargetApp.app</string>
iph0wn:~ root#
```

Figure 8 - Conversion of com.apple.mobile.installation.plist

4.1.2. Decrypt the App

Since all non-native apps (apps available in the App Store) are encrypted (Apple, iTunes Connect Developer Guide, 2014), the target app will need to be decrypted for static code analysis.

There are different tools that can be used to decrypt iOS apps, but for this paper Clutch will be the tool of choice. The current version of clutch as of this paper is 1.4.3, and available via GitHub at the link above. Download it, copy it to the iPhone, change permissions to allow it to execute, and then let the fun begin:

```
0x414141:Downloads adamkliarsky$ scp Clutch-1.4.3 root@192.168.1.139:/var/root/.
root@192.168.1.139's password:
Clutch-1.4.3                                  100%  834KB 833.7KB/s   00:00
0x414141:Downloads adamkliarsky$

iph0wn:~ root# chmod 755 Clutch
iph0wn:~ root# ./Clutch-1.4.3
Clutch 1.4.3
--------------------------------

1) Twitter
2) Facebook
3) Pandora
4) TargetApp

iph0wn:~ root# ./Clutch-1.4.3 4
Clutch 1.4.3
--------------------------------
Cracking TargetApp...
Creating working directory...
Performing initial analysis...
dumping binary: analyzing load commands
dumping binary: obtaining ptrace handle
dumping binary: forking to begin tracing
dumping binary: successfully forked

----- <snip> ----------
```

This produces the unencrypted .ipa (archive) file /var/root/TargetApp-v1.1.4-ak-(Clutch-1.4.3).ipa. Unzipping this will dump the contents needed for analysis into a subdirectory called 'Payload'.

```
iph0wn:~ root# unzip TargetApp-v1.1.4-ak-\(Clutch-1.4.3\).ipa
Archive:  TargetApp-v1.1.4-ak-(Clutch-1.4.3).ipa
   creating: Payload/TargetApp.app/
 extracting: Payload/TargetApp.app/114.png
 extracting: Payload/TargetApp.app/120.png
 extracting: Payload/TargetApp.app/29.png
 extracting: Payload/TargetApp.app/50.png
----- <snip> ----------
```

4.1.3. Examine Objective C Class and Runtime Information

The next step is to analyze the decrypted app binary, to review the class information and look for anything worth attacking. Class Dump, according to the original author, Steve Nygard, on his project page, "can look at the design of closed source applications, frameworks, and bundles" (Nygard, 2013), which will be essential in static code analysis. The output of this tool can be quite extensive, so piping the output into a text file will allow for easy analysis.

```
iph0wn:~ root# class-dump /Payload/TargetApp.app/TargetApp > classdump.txt
```

Figure 9 - Running class-dump on the target app binary

The class-dump output can be filtered to look for keywords, such as 'authentication', 'login', 'username' or 'password'. Anything that might identify a method we can leverage to test authentication. Looking through the methods that exist in the app binary can help identify which ones might target a given functionality (such as authentication) that you as a penetration tester might want to look into.

```
iph0wn:~ root# cat TargetApp.txt | grep login
    HKLoginEntryView *_loginEntry;
    NSString *_loginID;
- (id)loginID;
    NSString *_loginErrorMessage;
    int _loginResultCode;
- (int)loginResultCode;
- (id)loginErrorMessage;
    int _loginDevice;
- (int)loginDevice;
    ECFLegacyLoginResponse *_loginResult;
- (id)loginResult;
- (void)processLoginResponse:(id)fp8 loginID:(id)fp12;
- (id)_login2012WithUserID:(id)fp8 scrambledPassword:(id)fp12;
    NSString *_loginMessage;
- (id)loginMessage;
+ (id)loginRequest2012;
    NSString *_loginID;
- (id)loginID;
```

```
- (id)initWithConnection:(id)fp8 loginID:(id)fp12 legacyResponse:(id)fp16;
    NSString *_loginID;
- (id)loginID;
    NSString *_loginMessage;
- (id)loginMessage;
```

Figure 10 - Searching class-dump output for interesting methods

4.2. Dynamic Runtime Analysis

4.2.1. Cycript

Cycript, pronounced "sssscript" ((saurik), 2014), is a dynamic analysis tool developed by Jay Freeman (saurik) that can be used to analyze apps on iOS devices. The tool works by hooking into the process of the running app by passing the "-p" flag to Cycript, followed by the app name.

```
iph0wn:~ root# ps -ef | grep test
  501   785     1   0   0:00.00 ??         0:05.65 /var/mobile/Applications/876D3137-7979-4E81-AF9F-
D89072733F6A/test.app/test
    0   794   749   0   0:00.00 ttys000    0:00.01 grep test
iph0wn:~ root# cycript -p test
cy#
```

Figure 11 - Identifying the target process and hooking it with Cycript

To get a list of instance variables used that might provide interesting information, there are a few code bits on the Cycript Tricks page of the iPhoneDevWiki website that will help enumerate variables and functions being used while the app is being run. Instance variables provide good information, and can be used with the following function code from the page:

```
function tryPrintIvars(a){ var x={}; for(i in *a){ try{ x[i] = (*a)[i];
} catch(e){} } return x; }
```

Copy and paste that into the terminal, and then invoke it for the current rootViewController to see what runtime variables can be accessed.

```
cy# function tryPrintIvars(a){ var x={}; for(i in *a){ try{ x[i] = (*a)[i]; } catch(e){} } return x; }
cy# tryPrintIvars(UIApp.keyWindow.rootViewController)
{isa:#"HKNavigationController",_view:#"<UILayoutContainerView: 0x17dd50a0; frame = (0 0; 320 480); autoresize = W+H; layer =
<CALayer:
0x17dd51a0>>",_tabBarItem:null,_navigationItem:null,_toolbarItems:null,_title:null,_nibName:null,_nibBundle:null,_parentViewController:n
ull,_childModalViewController:null,_parentModalViewController:null,_previousRootViewController:null,_modalTransitionView:null,_modalPr
eservedFirstResponder:null,_dimmingView:null,_dropShadowView:null,_currentAction:null,_storyboard:null,_storyboardSegueTemplates:n
ull,_externalObjectsTableForViewLoading:null,_topLevelObjectsToKeepAliveFromStoryboard:null,_savedHeaderSuperview:null,_savedFo
oterSuperview:null,_editButtonItem:null,_searchDisplayController:null,_modalTransitionStyle:-
1,_modalPresentationStyle:0,_lastKnownInterfaceOrientation:1,_popoverController:null,_containerViewInSheet:null,_contentSizeForViewIn
Popover:{width:320,height:1100},_formSheetSize:{width:0,height:0},_recordedContentScrollView:null,_afterAppearance:null,_explicitAppe
aranceTransitionLevel:0,_keyCommands:null,_retainCount:4,_ignoreAppSupportedOrientations:0,_viewHostsLayoutEngine:0,_storyboardI
dentifier:null,_transitioningDelegate:null,_modalPresentationCapturesStatusBarAppearance:0,_childViewControllers:@[#"<LoginViewContr
```

oller:
0x17dd2eb0>"],_customNavigationInteractiveTransitionDuration:0,_customNavigationInteractiveTransitionPercentComplete:0,_transitionDelegate:null,_customTransitioningView:null,_navigationControllerContentOffsetAdjustment:0,_topLayoutGuide:null,_bottomLayoutGuide:null,_topBarInsetGuideConstraint:null,_bottomBarInsetGuideConstraint:null,_sourceViewControllerIfPresentedViaPopoverSegue:null,_modalSourceViewController:null,_presentedStatusBarViewController:null,_edgesForExtendedLayout:15,__embeddedView:null,__embeddingView:null,__embeddedDelegate:null,_preferredContentSize:{width:0,height:0},_navigationControllerContentInsetAdjustment:{top:0,left:0,bottom:0,right:0},_contentOverlayInsets:{top:0,left:0,bottom:0,right:0},__embeddedViewFrame:{origin:{x:0,y:0},size:{width:0,height:0}},_containerView:#"<UILayoutContainerView: 0x17dd50a0; frame = (0 0; 320 480); autoresize = W+H; layer = <CALayer:
0x17dd51a0>>",_navigationBar:#"<UINavigationBar: 0x17dd53c0; frame = (0 -24; 320 44); hidden = YES; autoresize = W;
gestureRecognizers = <NSArray: 0x17d9a4e0>; layer = <CALayer:
0x17dd5550>>",_navigationBarClass:#"UINavigationBar",_toolbar:null,_navigationTransitionView:#"<UINavigationTransitionView:
0x17d505f0; frame = (0 0; 320 480); clipsToBounds = YES; autoresize = W+H; layer = <CALayer:
0x17da5160>>",_bottomInsetDelta:0,_statusBarHeightForHideShow:0,_disappearingViewController:null,_delegate:#"<HKAppDelegate:
0x17d9d760>",_savedNavBarStyleBeforeSheet:0,_savedToolBarStyleBeforeSheet:0,_backGestureRecognizer:null,_topPalette:null,_freePalette:null,_transitioningTopPalette:null,_interactiveTransition:0,__usingBuiltinAnimator:0,__barAnimationWasCancelled:0,_toolbarClass:nil,_customNavigationTransitionDuration:0,__transitionController:null,__cachedTransitionController:null,__interactionController:null,__cachedInteractionController:null,__toolbarAnimationId:null,__navbarAnimationId:null,__updateNavigationBarHandler:null,_builtinTransitionStyle:0,_builtinTransitionGap:20,_forceOrientationOnPush:0,_animationCurve:0,_overrideAnimation:0,_animationDuration:0}

Figure 12 - tryPrintIvars

Likewise there is code for a custom function to print out methods used in a specific class. Copy and paste the code from the website into the Cycript interface, and then invoke against a specific class to view its methods:

```
function printMethods(className) {    var count = new new Type("I");
var methods = class_copyMethodList(objc_getClass(className), 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);
free(count);    return methodsArray; }
```

Figure 13 - function 'printMethods'

```
cy# UIApp.keyWindow.rootViewController
#"<HKNavigationController: 0x17dd37e0>"
```

Figure 14 - Identify the rootViewController

cy# printMethods(HKNavigationController)
[{selector:@selector(setOverrideAnimation:),implementation:0x306db9},{selector:@selector(updateOrientation),implementation:0x306c7d},{selector:@selector(forceOrientationOnPush),implementation:0x306d69},{selector:@selector(setForceOrientationOnPush:),implementation:0x306d79},{selector:@selector(animationCurve),implementation:0x306d89},{selector:@selector(overrideAnimation),implementation:0x306da9},{selector:@selector(popViewControllerAnimated:),implementation:0x306721},{selector:@selector(setAnimationDuration:),implementation:0x306dd9},{selector:@selector(shouldAutorotate),implementation:0x306b25},{selector:@selector(supportedInterfaceOrientations),implementation:0x306ba1},{selector:@selector(setAnimationCurve:),implementation:0x306d99},{selector:@selector(preferredInterfaceOrientationForPresentation),implementation:0x306bf9},{selector:@selector(pushViewController:animated:),implementation:0x3065cd},{selector:@selector(popToViewController:animated:),implementation:0x306875},{selector:@selector(popToRootViewControllerAnimated:),implementation:0x3069b5},{selector:@selector(animationDuration),implementation:0x306dc9}]
cy#

Figure 15 – Pass as an argument to printMethods

4.2.2. GDB

The GNU debugger, aka 'GDB' is a popular debugger that usually comes packaged with most Unix/Linux distributions. As stated on the GDB website, "GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed." (GDB

developers, 2014). This is ideal for an app where debug symbols are available, such as for internally developed apps. For true black box style analysis, there will be some issues.

To use GDB, ssh into the iPhone, and run the command as follows:

```
iph0wn:~ root# gdb -p 936
GNU gdb 6.3.50.20050815-cvs (Fri May 20 08:08:42 UTC 2011)
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 "--host=arm-apple-darwin9 --target=".
/private/var/root/936: No such file or directory
Attaching to process 936.
0x3a72ea8c in ?? ()
(gdb)
```

Figure 16 - gdb with the '-p' option to specifiy PID

Once the debugger is attached to the process associated with the app, there are a few commands to execute to get some information about the program. Registers are important as they contain critical information as the program executes. The stack pointer (sp) shown below for example, stores the value of the last item placed on the stack. If assessing the app for bounds checking (buffer overflows etc), knowing how to pull information from registers is important.

```
(gdb) info registers
r0             0x10004005       268451845
r1             0x7000006        117440518
r2             0x0      0
r3             0xc00    3072
r4             0x1c03 7171
r5             0xffffffff -1
r6             0x0      0
r7             0x27de3e2c       668876332
r8             0x0      0
r9             0x1      1
r10             0x1c03          7171
r11             0xc00  3072
r12             0xfffffffe1            -31
sp              0x27de3dec      668876268
lr             0x3a72e88d       980609165
pc              0x3a72ea8c      980609676
cpsr           {0x60000010, n = 0x0, z = 0x1, c = 0x1, v = 0x0, q = 0x0, j = 0x0, ge = 0x0, e = 0x0, a = 0x0, i = 0x0, f = 0x0, t = 0x0,
  mode = 0x10}     {0x60000010, n = 0, z = 1, c = 1, v = 0, q = 0, j = 0, ge = 0, e = 0, a = 0, i = 0, f = 0, t = 0, mode = usr}
(gdb)
```

Figure 17 - 'info registers'

To get Mach specific information, there are a few gdb options. Using 'info mach-tasks' will display processes, PIDs, and task IDs.

```
(gdb) info mach-tasks
67 processes:
   gdb is 1719 has task 0x807
   bash is 1715 has task 0x2103
   sshd is 1714 has task 0x2203
<snip>
```

Figure 18 - 'info mach-tasks'

From the penetration testing point of view, finding writeable and executable areas of memory is essential to identify possible exploitation points. This can be done

using the gdb command 'info mach-regions'. This command will list all regions of mapped memory, and can be used to identify heap memory.

```
(gdb) info mach-regions
Region from 0xe7000 to 0xe9000 (---, max r-x; copy, private, not-reserved)
  ... from 0xe9000 to 0xea000 (---, max rw-; copy, private, not-reserved)
  ... from 0xea000 to 0xeb000 (---, max r--; copy, private, not-reserved)
  ... from 0xeb000 to 0xee000 (---, max r--; copy, private, not-reserved)
  ... from 0xee000 to 0xef000 (---, max rwx; copy, private, not-reserved)
  ... from 0xef000 to 0xf0000 (---, max rwx; copy, private, not-reserved)
  ... from 0xf0000 to 0xf1000 (---, max rwx; copy, private, not-reserved)
  ... from 0xf1000 to 0xf2000 (---, max rwx; copy, private, not-reserved)
  ... from 0xf2000 to 0xf3000 (---, max rwx; copy, private, not-reserved)
  ... from 0xf3000 to 0xf5000 (---, max rwx; copy, private, not-reserved) (2 sub-regions)
  ... from 0xf5000 to 0x102000 (---, max rwx; copy, private, not-reserved) (3 sub-regions)
  ... from 0x102000 to 0x10e000 (---, max rwx; copy, private, not-reserved) (2 sub-regions)
<snip>
```

Figure 19 - 'info mach-regions'

4.2.3. Snoop-it

The next tool is more of a suite of tools, all rolled up into one. Snoop-it is "a tool to assist security assessments and dynamic analysis of iOS Apps" (Kurtz, 2013). Snoop-it actually lends itself to static analysis as well as runtime analysis, by providing insight into class methods to illustrate more of what they do. It injects itself into the application while running, There is a good chance that when using Snoop-it, there will be some back and forth between static analysis of classes and method tracing of those classes (objects) in action. "Snoop-it is a tool to assist dynamic analysis and blackbox security assessments of mobile Apps by retrofitting existing apps with debugging and runtime tracing capabilities" (Kurtz, 2013).

To get started, launch Snoop-it from the home screen on the iPhone. After the splash screen appears upon initialization, you'll be presented with the configuration screen. Select the target app from within the Snoop-it configuration interface. Confirm the http port to use (default port is 12345), and then close the app. Launch the target app from the home screen on the iPhone, and from the analysis workstation, open a browser to the target device IP on the selected port, authenticating with user id and password 'snoop-it'.

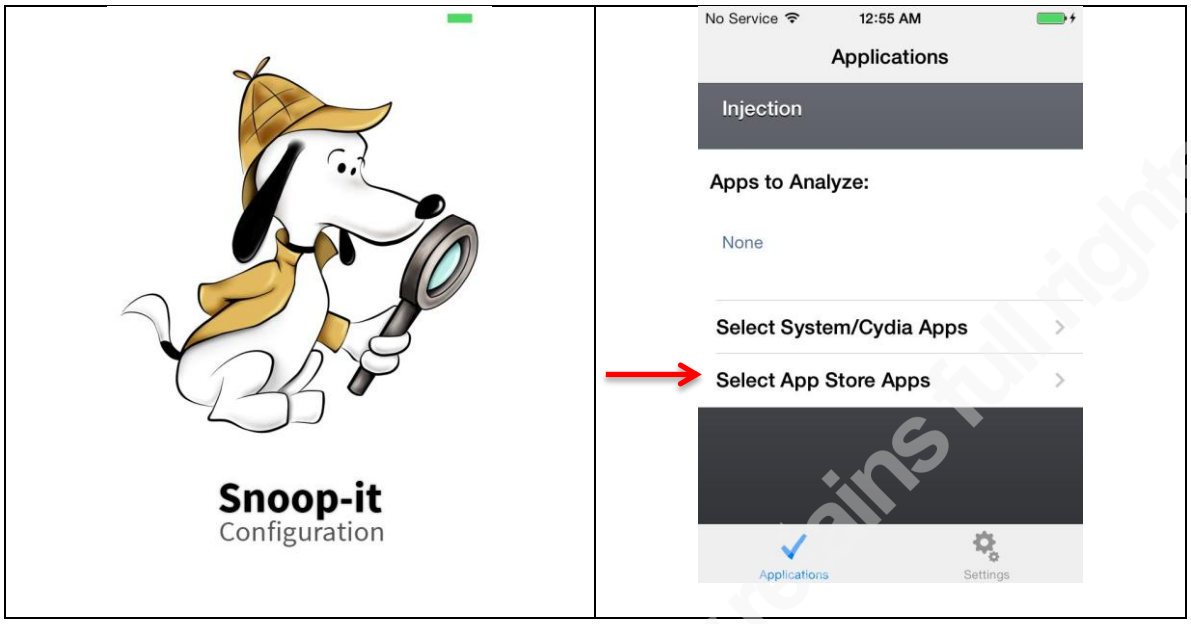

Figure 20 - Initial Configuration

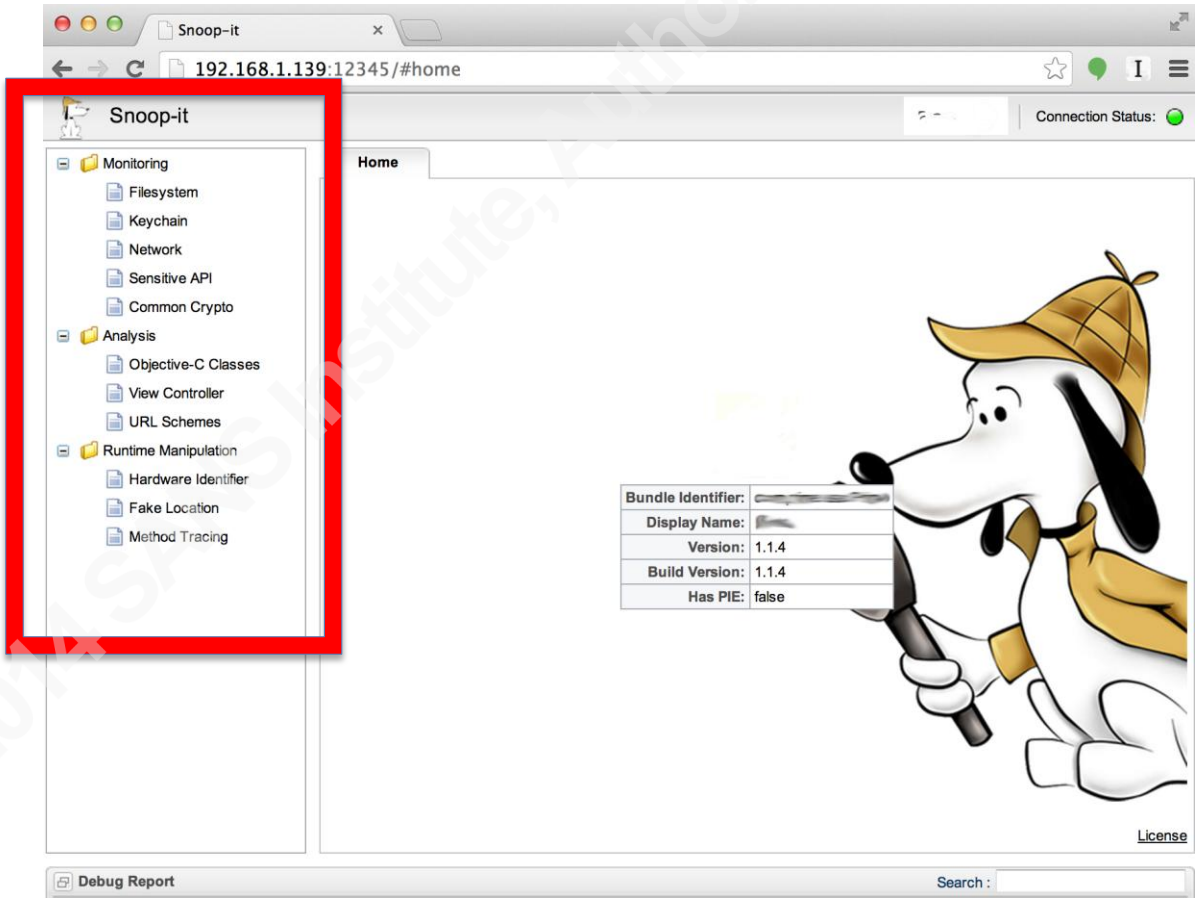

Figure 21 - Menu layout

There are three main parts of the Snoop-it interface; Monitoring, Analysis, and Runtime Manipulation. Each of these is broken down into a sub-component that can be used for analysis. Keep in mind that Snoop-it is injected into the target app, and will display real-time activity while the app is in use. So from the penetration testers perspective, think of how this app will normally be used, and likewise how it can be mis-used. Navigate through the app, authenticate where possible, and keep watching the Snoop-it interface. The 'Filesystem' submenu item under 'Monitoring' will show real time file activity, as well as the file name and path.

For dynamic runtime analysis, the method tracing is valuable (off by default, can be turned on by checking a box under 'Tracing' on the Method Tracing tab). As the app is being used, keep an eye on the method-tracing screen. Try logging into an app to see what is being called, and how the information is being handled. If there is too much information on screen, try downloading the current log files to parse through for relevant info.

```
Sun Apr 13 01:36:52 2014 (Thread  3): - [WPLoginViewController(0x18005410)
setAuthenticator:], args: <0x180421a0>
Sun Apr 13 01:36:52 2014 (Thread  3): - [WPLoginViewController(0x18005410) authenticator]
Sun Apr 13 01:36:52 2014 (Thread  3): - [WPLoginViewController(0x18005410) userID]
Sun Apr 13 01:36:52 2014 (Thread  3): - [WPLoginViewController(0x18005410) password]
Sun Apr 13 01:36:52 2014 (Thread  3): - [Authenticator(0x180421a0)
attemptLogin:password:], args: <__NSCFString 0x18050790: user>, <__NSCFString 0x18052360:
31337>
Sun Apr 13 01:36:52 2014 (Thread  3): - [Authenticator(0x180421a0) setUsername:], args:
<__NSCFString 0x18050790: user>
Sun Apr 13 01:36:52 2014 (Thread  3): - [Authenticator(0x180421a0) setPassword:], args:
<__NSCFString 0x18052360: 31337>
```

Figure 22 - Snoop-It Log File from Method Tracing Showing Login

4.3. Network Analysis

4.3.1. Network Activity via Snoop-it

Apps are sources of information, providing news, updates etc. from Internet sources to users. Additionally, hidden, back channel communication for app functionality is also taking place. It is important from the penetration testers perspective to understand what types of communication are taking place; which hosts the app is communicating to and what protocol. Snoop-it will show real-time network activity while the app is being used. The far left column keeps track of transaction sequences, the next column a

timestamp. There is protocol (http/https) information as well as URL and query string content.

4.3.2. Intercept Proxies

Intercept/attack proxies are great for viewing and analyzing client-server communication. Proxies can reside on the analysis computer, and provide insight into client requests and server responses by intercepting client requests, forwarding them to the destination server, and subsequently proxying the response in the same manner. Intercept proxies can identify protocols in use, types of information being sent, how authentication is handled, and if there are server-side redirects that might otherwise go unnoticed.

One popular proxy is Burp Suite from Portswigger.net. "Burp Suite is an integrated platform for performing security testing of web applications" (PortSwigger, LTD, 2014). Burp comes in two flavors; a free version with limited features, and a full-featured professional version; the proxy feature is available with both. To get started, launch Burp, go to Proxy → Options and change the IP of the interface from the loopback 127.0.0.1 to the network IP address, so that it can listen for connections. Next, change the intercept option as needed. For initial analysis, to simply identify traffic flow, turn it off. If request/responses are to be analyzed or modified, then leave it on.

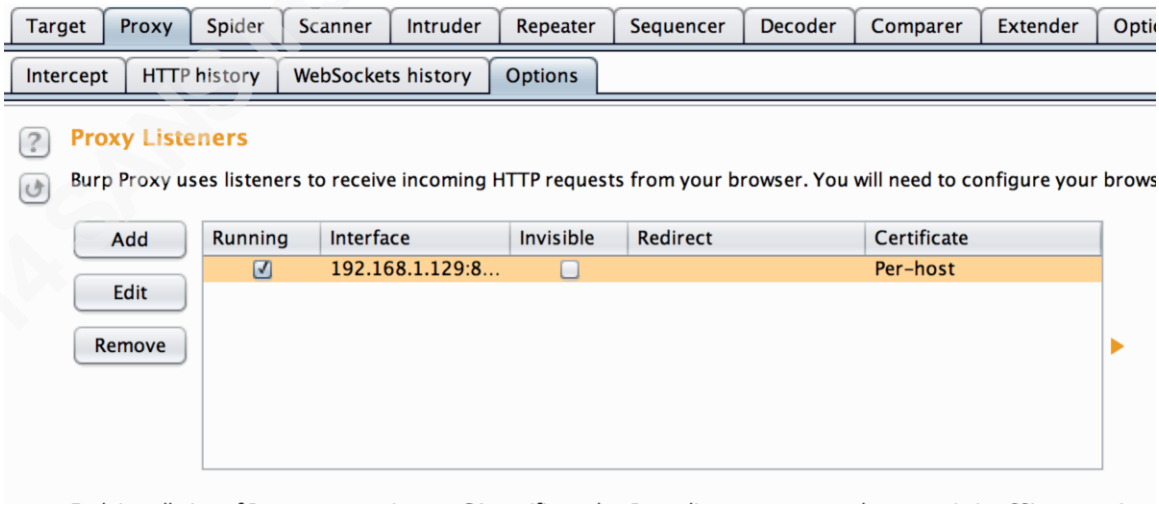

Figure 23 - Burp Suite Proxy Options

Configure the proxy on the iPhone by going to the Settings app → Wi-Fi → <network ID> → click on the information icon, and set the proxy IP and port.

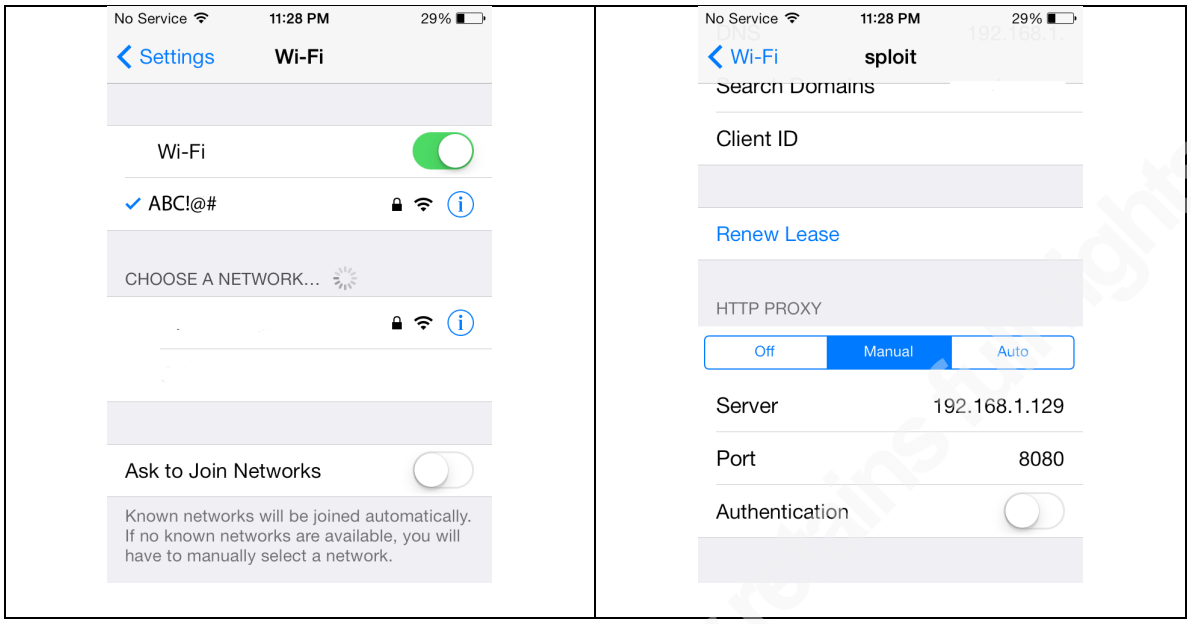

Figure 24 - Configuring Proxy Settings on iPhone

Start the app on the mobile device and start watching the proxy interface for traffic. The image below shows the initial hits up on launching the Twitter app. This illustrates the value of using a proxy to see what sites are actually being hit when using an app.

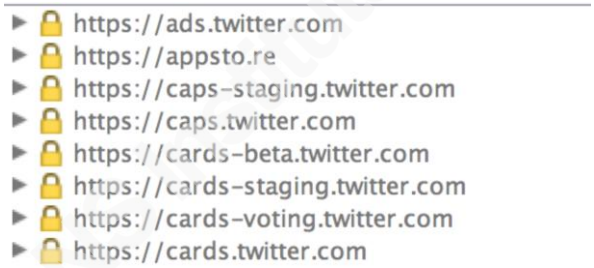

Figure 25 - Traffic From Initial Twitter App Launch

One caveat is dealing with SSL enabled sites; the Burp certificate will need to be downloaded to the iPhone; "to use Burp Proxy most effectively with HTTPS websites, you will need to install Burp's CA certificate as a trusted root in your browser" (PortSwigger, LTD, 2014). With the proxy enabled on the iPhone, launch Safari and type 'burp' in the URL. Click 'CA Certificate', and install the certificate presented by the proxy.

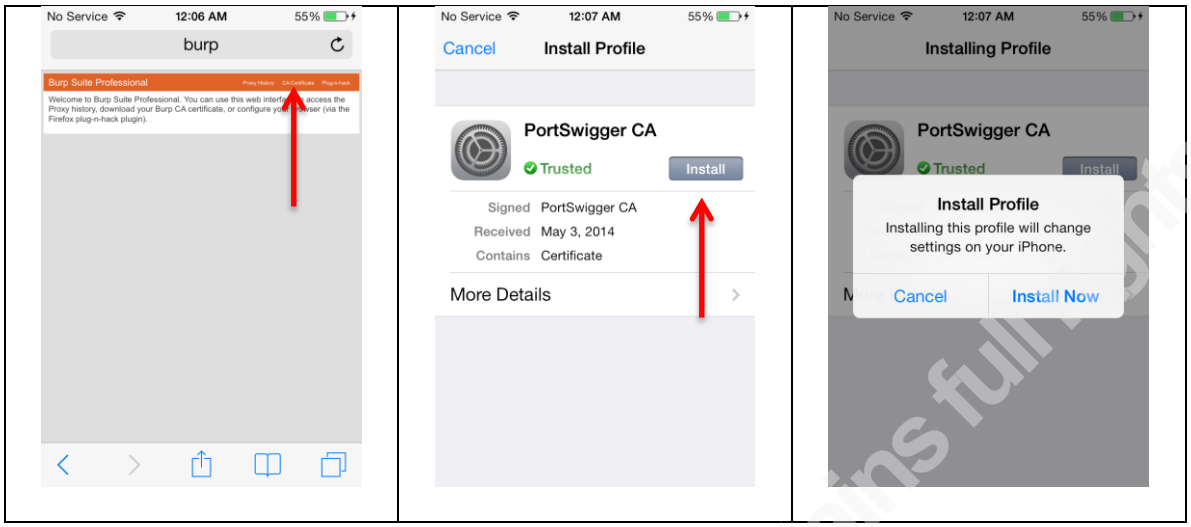

Figure 26 - Installing Burp's Certificate

To illustrate some of the analysis potential, let's look at an initial Facebook connection. When launching the Facebook mobile app, it initially sends an HTTP 'GET' request. The server responds then with a 302 redirect, sending subsequent requests via HTTPS.

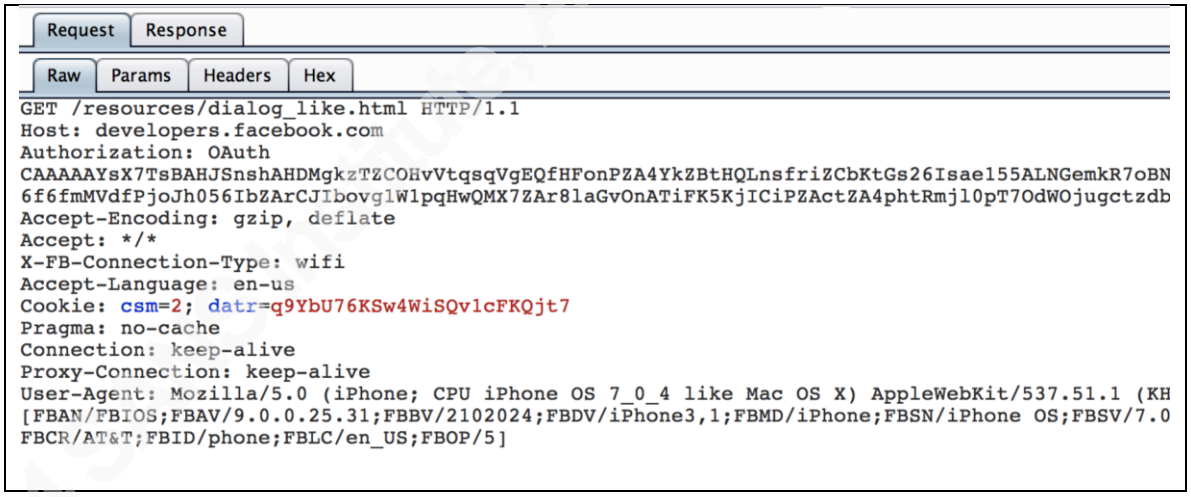

Figure 27 - Initial HTTP GET Request

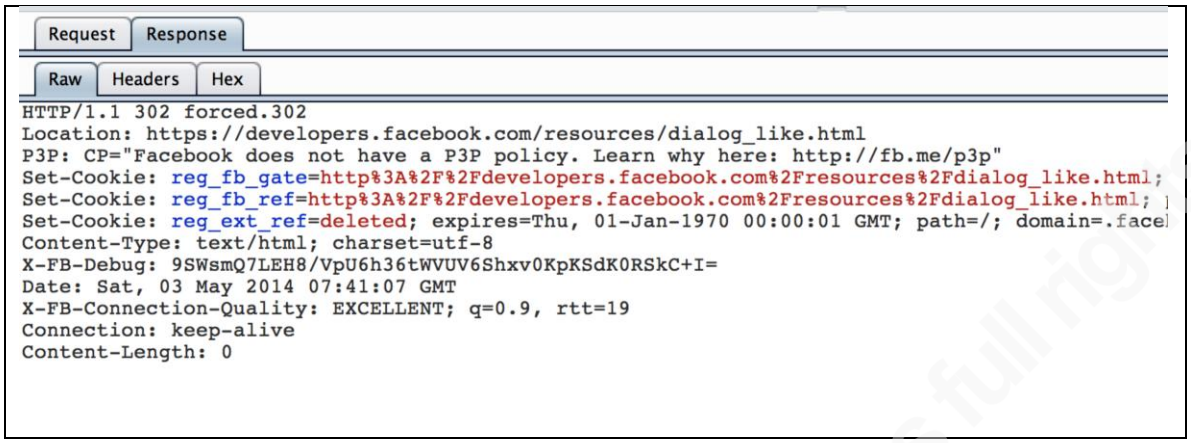

Figure 28 - Server 302 Redirect Reponse

As the client requests and server responses continue to flow, they will be displayed for easy analysis as the client/server communication image below shows.

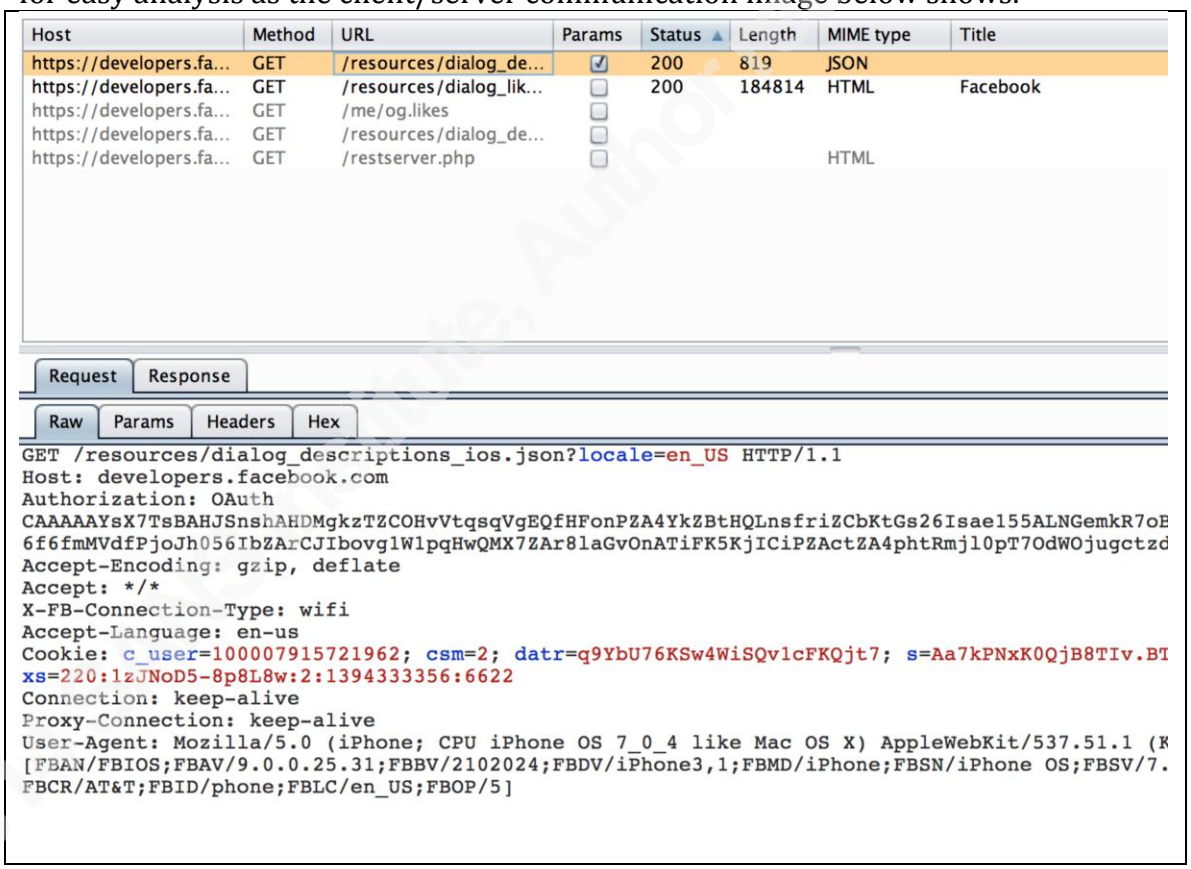

Figure 29 - Client/Server Communication

One thing to keep an eye on is the 'Scanner' tab; by default Burp is configured to do 'passive' scanning of traffic it intercepts. The initial Facebook connection the mobile app opened showed some interesting issues that a penetration tester might want to take note of:

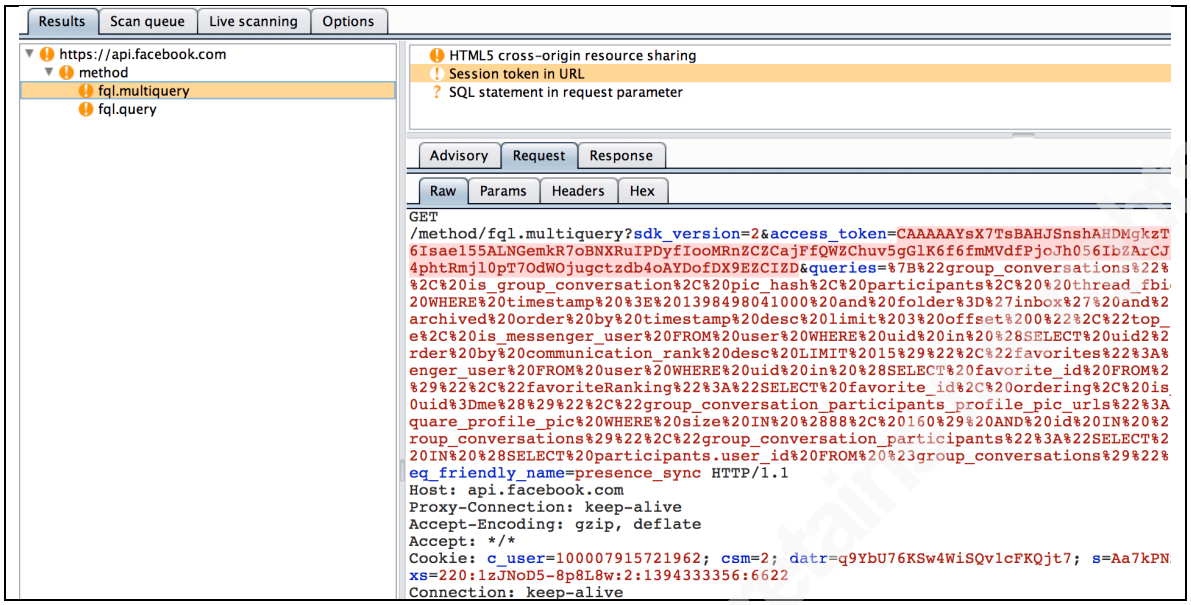

Figure 30 - Potential Issues with FQL Messages

Charles Proxy is another proxy that works great for analyzing traffic between the iOS app and its Internet resources. "Charles is an HTTP proxy / HTTP monitor / Reverse Proxy that enables a developer to view all of the HTTP and SSL / HTTPS traffic between their machine and the Internet" (Randow, 2014). Charles is setup the same way as Burp with the exception of default port; Burp uses 8080 and Charles uses 8888.

There are different proxies that can serve the same purpose. Whether it is Burp, Charles, ZAP, or any other proxy, viewing traffic between client/server is an essential step in penetration testing an iOS app.

4.3.3. Sniffers

Sniffers such as Wireshark and Tcpdump are great additions to traffic analysis. Tcpdump can be run with a snap length of 0 to capture the full packet and written to file with the "-w <filename>" option to ensure packets are not missed. Wireshark has a beautiful interface and can decode packets/protocols in an easy to understand (and analyze) manner. Once the file is saved from Tcpdump, it can be viewed and filtered with Wireshark. Like with the intercept proxies, if information is being sent clear text, there is a good chance the sniffer will pick it up, especially with the open (hub-like) nature of WiFi.

Adam Kliarsky, adam.kliarsky@gmail.com

4.4. Supporting File Analysis

Supporting files include database files, property files, as well as image and configuration files that support the app during operation. Start by looking through the folder in which the app binary resides. Expand the search to peruse the system to identify related files that might be hidden under Library or other system folders. Database files with '.db' or '.sqlite' extensions may offer a treasure trove of useful data. Consider the system database 'sms.db' used to store messages:

```
iph0wn:~ root# sqlite3 /private/var/mobile/Library/SMS/sms.db
SQLite version 3.7.13
Enter ".help" for instructions
sqlite> .tables
_SqliteDatabaseProperties  chat_message_join
attachment                 handle
chat                       message
chat_handle_join           message_attachment_join
sqlite> select * from message;
1|1094E694-33E9-4252-B4B9-F6BB79E89F91|Test1|0||2|||
                                        streamtyped???@???NSAttributedString|10|0|SMS|p:+13106232859|2A097B81-22A9-40BC-94B2-
49924136D297|4|420959839|0|0|0|1|0|1|0|0|0|0|0|0|0|0|0|0|0|0||1|0
2|C4A182E1-19C5-4A1C-BA85-2D8DD1789D03|Test2|0||4|||
                                        streamtyped???@???NSAttributedString|10|0|SMS|p:+13106232859|2A097B81-22A9-40BC-94B2-
49924136D297|4|420959848|0|0|0|1|0|1|0|0|0|0|0|0|0|0|0|0|0|0||1|0
3|B793407A-5FE2-4F89-B68F-F2FA9399D07B|Test3 |0||1|||
                                         streamtyped???@???NSMutableAttributedString|10|0|iMessage|e:ipwnd@hushmail.com|3BAC9055-
E8AD-41CF-8A8E-BC80A7E7E67E|0|420959872|420959904|0|1|1|0|0|0|0|0|1|0|0|0|0|0|0|0||1|0
4|1B7AE43D-D2BF-40E9-A8B8-50F6C96D3971|What up|0||1|||
                                         streamtyped???@???NSMutableAttributedString|10|0|iMessage|e:ipwnd@hushmail.com|3BAC9055-
E8AD-41CF-8A8E-BC80A7E7E67E|0|420959917|420959872|420959917|1|1|0|1|0|0|0|0|1|0|1|0|0|0|0|0|0||1|0
5|0ED228EF-6F19-44C2-BD7A-002CDCC81FAC|Nada|0||1|||
                                        streamtyped???@???NSMutableAttributedString|10|0|iMessage|e:ipwnd@hushmail.com|3BAC9055-
E8AD-41CF-8A8E-BC80A7E7E67E|0|420959872|420959923|0|1|1|0|0|0|0|0|0|1|0|0|0|0|0|0|0||1|0
sqlite>
```

Figure 31 - Contents of 'message' table in SMS.db

Looking through this file shows 8 tables. Using the SELECT statement, we can view private messages sent between devices via iMessage in the 'message' table and account info from the 'chat' table

```
sqlite> select * from chat;
1|iMessage;-;+15557554031|45|3|3BAC9055-E8AD-41CF-8A8E-
BC80A7E7E67E|bplist00?_#CKChatPreviousAccountsDictionaryKey?XiMessage_$8EF30FFF-A281-440A-A9A8-020684A10400
        14=|+15557554031|iMessage||E:ipwnd@hushmail.com|0|ipwnd@hushmail.com|
2|SMS;-;+15557554031|45|3|2A097B81-22A9-40BC-94B2-
49924136D297|bplist00?_#CKChatPreviousAccountsDictionaryKey?XiMessage_$8EF30FFF-A281-440A-A9A8-020684A10400
    14=|+15557554031|SMS||P:+15556232859|0||
3|iMessage;-;test360@gmail.com|45|3|8EF30FFF-A281-440A-A9A8-
020684A10400|bplist00?_#CKChatPreviousAccountsDictionaryKey?XiMessage_$8EF30FFF-A281-440A-A9A8-020684A10400
            14=|test360@gmail.com|iMessage||E:test360@icloud.com|0||
4|SMS;-;test360@gmail.com|45|3|2A097B81-22A9-40BC-94B2-
49924136D297|bplist00?_#CKChatPreviousAccountsDictionaryKey?XiMessage_$8EF30FFF-A281-440A-A9A8-020684A10400
            14=|test360@gmail.com|SMS||P:+15556232859|0||
```

Figure 32 - Contents of 'chat' table in SMS.db

5. Conclusion

With a solid understanding of static code analysis, runtime analysis, network analysis, and how supporting files can be used in application analysis, the penetration tester should be set and ready to go. Using a methodology, like the one published by OWASP, you can step through each test using elements of each analysis method.

While some penetration testing scenarios may require extensive analysis, others may conversely require minimal analysis; the situation will dictate. Analyzing authentication, for example could be very intensive. Parsing class-dump output for login methods, swizzling them with cycript to identify bypasses, identifying weaknesses with Snoop-it or Burp, only to uncover something missed to cycle back through the process. And of course possibly identifying a potential gold mine of stored data in one or more sqlite database files. The penetration tester will need both skill and patience on top of a solid and repeatable methodology.

Combining these analysis methods can facilitate a comprehensive test and understanding of the potential risk of an app. This information – what files the app uses with or interacts with, how it communicates over the network – can all be used to develop a picture of how it works…essentially a threat modeling the app – but in reverse. And with the growth of mobile devices and apps in the day to day corporate and private lives of our users, being able to assess these is an essential skill.

Adam Kliarsky, adam.kliarsky@gmail.com

6. References

(saurik), J. F. (2014, January 01). *Cycript Manual*. Retrieved March 06, 2014, from Cycript : http://www.cycript.org/manual/

Apple. (2010, March 24). *About Property Lists*. Retrieved March 6, 2014, from Mac Developer Library: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/PropertyLists/AboutPropertyLists/AboutPropertyLists.html#//apple_ref/doc/uid/10000048i-CH3-SW2

Apple. (2014, February 01). *iOS Security*. Retrieved March 07, 2014, from Apple: http://images.apple.com/iphone/business/docs/iOS_Security_Feb14.pdf

Apple. (2014, March 12). *iTunes Connect Developer Guide*. Retrieved March 15, 2014, from iOS Developer Library: https://developer.apple.com/library/ios/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/Chapters/SubmittingTheApp.html

Apple.com. (2014, February 28). *App Review*. Retrieved March 6, 2014, from Apple Developer: https://developer.apple.com/support/appstore/app-review/

Apple.com. (2014, 01 01). *Apple - iPhone 5s - App Store*. Retrieved 03 1, 2014, from Apple.com: http://www.apple.com/iphone-5s/app-store/

Cornea, O., & Haddix, J. (2013, October 7). *IOS Application Security Testing Cheat Sheet*. Retrieved March 8, 2014, from OWASP: https://www.owasp.org/index.php/IOS_Application_Security_Testing_Cheat_Sheet

Fox, Z. (2013, August 20). *17.4% of Global Web Traffic Comes Through Mobile*. Retrieved February 20, 2014, from Mashable: http://mashable.com/2013/08/20/mobile-web-traffic/

Freeman, J. (2014, 03 14). *Welcome to Cydia*. Retrieved 03 14, 2014, from Welcome to Cydia: https://cydia.saurik.com/

GDB developers. (2014, February 06). *GDB: The GNU Project Debugger*. Retrieved February 20, 2014, from GDB: The GNU Project Debugger: http://www.sourceware.org/gdb/

Adam Kliarsky, adam.kliarsky@gmail.com

Kelly, S. M. (2013, February 6). *Mobile Devices Will Outnumber People by the End of the Year*. Retrieved 12 29, 2013, from http://www.mashable.com: http://mashable.com/2013/02/06/mobile-growth/

Kurtz, A. (2013, August 20). *Project Home*. Retrieved March 2, 2014, from snoop-it: https://code.google.com/p/snoop-it/

Nygard, S. (2013, November 16). *Class-dump*. Retrieved February 20, 2014, from Steve Nygard: http://stevenygard.com/projects/class-dump/

PortSwigger, LTD. (2014, January 01). *Burp Suite*. Retrieved March 08, 2014, from Portswigger Web Security: http://www.portswigger.net/burp/

PortSwigger, LTD. (2014, January 01). *Installing Burp's CA Certificate*. Retrieved March 08, 2014, from Portswigger Web Security: http://portswigger.net/burp/Help/proxy_options_installingCAcert.html

Randow, K. v. (2014, January 01). *Charles Web Debugging Proxy Application* . Retrieved March 08, 2014, from Charles Web Debugging Proxy Application : http://www.charlesproxy.com/

Research, P. (2013, December 27). *Mobile Technology Fact Sheet*. Retrieved February 22, 2014, from Pew Research Internet Project: http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/

Rivera, J., & van der Meulen, R. (2013, September 19). *Gartner Says Mobile App Stores Will See Annual Downloads Reach 102 Billion in 2013*. Retrieved February 22, 2014, from Gartner: http://www.gartner.com/newsroom/id/2592315

Rowinski, D. (2013, January 7). *Apple iOS App Store Adding 20,000 Apps A Month, Hits 40 Billion Downloads*. Retrieved February 20, 2014, from Readwrite: http://readwrite.com/2013/01/07/apple-app-store-growing-by#awesm=~oy2N1l7HdSirc7

Sarno, D. (2012, March 14). *Apple's App Store receives 26,000 submissions every week*. Retrieved February 20, 2014, from Los Angeles Times: http://articles.latimes.com/2012/mar/14/business/la-fi-tn-apple-26000-20120314

7. Appendix A – TargetApp Decryption Process

```
iph0wn:~ root# ./Clutch-1.4.3
Clutch 1.4.3
--------------------------------

1) Twitter
2) Facebook
3) Pandora
4) TargetApp

iph0wn:~ root# ./Clutch-1.4.3 4
Clutch 1.4.3
--------------------------------
Cracking TargetApp...
Creating working directory...
Performing initial analysis...
dumping binary: analyzing load commands
dumping binary: obtaining ptrace handle
dumping binary: forking to begin tracing
dumping binary: successfully forked
dumping binary: obtaining mach port
dumping binary: preparing code resign
dumping binary: preparing to dump
dumping binary: ASLR enabled, identifying dump location dynamically
dumping binary: performing dump
dumping binary: patched cryptid
 [==================================================>] 100%
 dumping binary: writing new checksum
packaging: waiting for zip thread
packaging: compressing IPA
packaging: censoring iTunesMetadata
packaging: compression level 0
        /var/root/TargetApp-v1.1.4-ak-(Clutch-1.4.3).ipa
elapsed time: 4.08s

Applications cracked:

TargetApp

Total success: 1   Total failed: 0
continuing after int crackiph0wn:~ root#
```