Penetration Testing for iPhone / iPad Applications

Author:

Kunjan Shah
Security Consultant
Foundstone Professional Services

Table of Contents

Abstract

Mobile application penetration testing is an up and coming security testing need that has recently obtained more attention, with the introduction of the Android, iPhone, and iPad platforms among others. The mobile application market is expected to reach a size of $9 billion by the end of 2011[1] with the growing consumer demand for smartphone applications, including those for banking and trading. A plethora of companies are rushing to capture a piece of the pie by developing new applications, or porting old applications to work with smartphones. These applications often deal with personally identifiable information (PII), credit card and other sensitive data.

This paper focuses specifically on helping security professionals understand the nuances of penetration testing iPhone/iPad applications. It attempts to cover the key steps the reader would need to understand such as setting up the test environment, installing the simulator, configuring the proxy tool and decompiling applications. To be clear this paper does not attempt to discuss the security framework of the iPhone / iPad itself, identify flaws in the iOS, or try to cover the entire application penetration testing methodology.

[1] http://www.mgovworld.org/topstory/mobile-applications-market-to-reach-9-billion-by-2011

Background

Since the release of iPhone in June 2007, Apple has acquired 25% of the market for mobile phones[2]. This has meant that Apple has sold close to 60 million iPhones[3] since its release. Things have now become even more interesting as over 3 million iPads have now been sold to date. One of the big attractions of the iPhone / iPad is the availability of a variety of third party applications that span a range of categories from productivity and financial to games and entertainment. Currently, the Apple App Store contains over 225,000 third-party approved applications[4] which have been downloaded over 5 billion times. In addition to this about 10% of these devices[5] have gone through a process called "jailbreaking". Jailbreaking is a process that allows iPad/iPhone users to run third party unsigned code on their devices by unlocking the operating system and granting root privilege to them.

The programming language used for developing iPhone / iPad applications[6] is Objective C, which brings back the dreaded buffer overflows that were a non-issue for J2ME and mobile .NET environments. There have been several buffer overflow vulnerabilities already published against the iPhone operating system, as discussed below. These applications can also be a combination of native and web applications opening the possibility of both Cross Site Scripting (XSS) and Cross Site Request Forgery (XSRF) on top of the buffer overflows. Over and above these however, these devices bring their own variations of vulnerabilities such as tapjacking[7], smudge attacks[8], key stroke caching[9] and automated snapshots[10].

[2] http://comscore.com/Press_Events/Press_Releases/2010/2/comScore_Reports_December_2009_U.S._Mobile_Subscriber_Market_Share
[3] http://www.mobilecrunch.com/2010/07/20/apple-sold-8-4-million-iphones-last-quarter/
[4] http://en.wikipedia.org/wiki/App_Store
[5] http://www.saurik.com/id/12
[6] Throughout the rest of this paper for convenience we refer to "iPhone / iPad applications" as just "applications" or "iOS applications". If a distinction is necessary we will clarify as appropriate.
[7] http://www.technologyreview.com/communications/26057/
[8] http://www.zdnet.com/blog/security/researchers-use-smudge-attack-identify-android-passcodes-68-percent-of-the-time/7165?tag=mantle_skin;content
[9] http://www.security-faqs.com/did-you-know-that-the-iphone-retains-cached-keyboard-data-for-up-to-12-months.html
[10] http://www.wired.com/gadgetlab/2008/09/hacker-says-sec/

History

A quick survey of the news uncovers a number of categories of incidents with iOS applications from a security and privacy perspective. Some of these were quite obviously malicious while others were taking liberties with controls on the device.

Data Harvesting Incidents

- *MogoRoad*[11]: "Customers of ID Mobile's MogoRoad iPhone application are complaining that they're getting sales calls from the company, a process which turns out to be technically a piece of cake."
- *Storm8's iSpy*[12]: "A maker of some of the most popular games for the iPhone has been surreptitiously collecting users' cell numbers without their permission, according to a federal lawsuit filed Wednesday."
- *Aurora Feint*: The first application to be delisted on the Apple Store due to privacy concerns. This application looked through the contact list and sent it unencrypted to the servers to match their friends who are currently online.

Worms

- *ikee*[13]: "iPhone owners in Australia awoke this weekend to find their devices targeted by self-replicating attacks that display an image of 1980s heart throb Rick Astley that's not easily removed."
- *Dutch Ransom*[14]: The attacker in this case holds Dutch iPhones for ransom. The default SSH password on the jail broken iPhone was the cause of this issue.
- *iPhone/Privacy.A*[15]: This worm steals personal data such as emails, SMS, contacts, multimedia files, calendars etc.
- *ikee.B (DUH)*[16]: This worm tried to exploit ING Direct Bank's two factor authentication via SMS.

Vulnerabilities

- *libtiff*: It allows attackers to take over the iPhone through buffer overflow vulnerabilities found in the TIFF processing library of the Safari browser.
- *SMS Fuzzing*[17]: It allowed attackers to take over the phone using maliciously crafted SMS messages.

[11] http://www.theregister.co.uk/2009/09/30/iphone_security/
[12] http://www.theregister.co.uk/2009/11/06/iphone_games_storm8_lawsuit/
[13] http://www.theregister.co.uk/2009/11/08/iphone_worm_rickrolls_users/
[14] http://www.wired.com/gadgetlab/2009/11/iphone-hacker/
[15] http://www.softsailor.com/news/11697-worlds-second-iphone-worm-called-iphoneprivacy-a-steals-private-date-from-jailbroken-handsets.html
[16] http://mtc.sri.com/iPhone/

- *Jailbreakme*[18]: A security bug across all iOS4 devices that provides the attacker full access to the underlying device by simply viewing a malicious PDF file in the Safari browser.

Needless to say we can see a variety of attacks as well as malicious applications. It is therefore vital as you develop such applications or consider deploying third party applications within your organization it is essential that these be tested to ensure they provide the security assurance levels needed.

[17] http://www.scmagazineus.com/iphone-hacker-reveals-sms-vulnerability/article/139479/
[18] http://mobile.venturebeat.com/2010/08/03/apple-security-bug-gives-hackers-access-to-your-iphone-or-ipad-by-viewing-a-pdf/

Setting up the Test Environment

There are several ways to test mobile applications *e.g.*:

1. Using a regular web application penetration testing chain (browser, proxy).
2. Using WinWAP with a proxy[19].
3. Using a phone simulator with a proxy[20].
4. Using a phone to test and proxy outgoing phone data to a PC.

In this paper we will focus on using a phone simulator with a proxy as it is the easiest and cheapest option available for testing iPhone applications. For some platforms, this can be difficult but for iPhone/iPad applications, use of a simulator is easy and effective.

Pre-requisites:

- Mac Book running Snow Leopard 10.6.2 OS or above.
- Apple iOS 4.0.1 (for testing iPhone applications) and iOS 3.2 (for testing iPad applications).
- Charles Proxy[21].
- SQLite Manager.

[19] http://www.winwap.com/desktop_applications/winwap_for_windows
[20] http://speckyboy.com/2010/04/12/mobile-web-and-app-development-testing-and-emulation-tools/
[21] http://www.google.com/url?sa=t&source=web&cd=1&sqi=2&ved=0CBMQFjAA&url=http%3A%2F%2Fwww.charlesproxy.com%2F&rct=j&q=charles%20proxy&ei=p9WPTKq-Go-Bswab0NGLDA&usg=AFQjCNG_O70VsRrfb_q7F66Nkb9ZK6MNMA&cad=rja

Installing the iOS SDK

The iPhone/iPad simulator is not available for download, as an independent application. In order to use the simulator, it is necessary to install the iOS Software Development Kit (SDK). The simulator comes packaged with the SDK installer. However, only registered Apple developers can download the SDK[22]. For testing iPhone applications[23] download iOS 4.0.1 and iOS 3.2 for iPad applications since this is the only SDK that allows development and testing of iPad applications. The Apple Developer Center does not allow downloading archived versions of iOS. It can therefore be challenging to gain access to the iOS 3.2 installer. The SDK includes the Xcode IDE, an iPhone simulator (4.0.1), an iPad simulator (3.2) and other tools for development and testing.

Steps to install the SDK:

- After downloading the iOS installer, locate where the `.dmg` file is downloaded. Normally it is located on the Desktop or under the User > Downloads folder.
- Double click this file to open the disk image.
- Double click the installer and follow on screen instructions. Note this currently requires up to 6.53 GB of free space on the target system.

Figure 1: iPhone SDK Installer

[22] http://developer.apple.com/programs/register/
[23] http://developer.apple.com/iphone/index.action

- After successful installation a new "Developer" folder will be placed in the root directory of the hard drive. All the tools for iPhone development and testing are located under this directory.

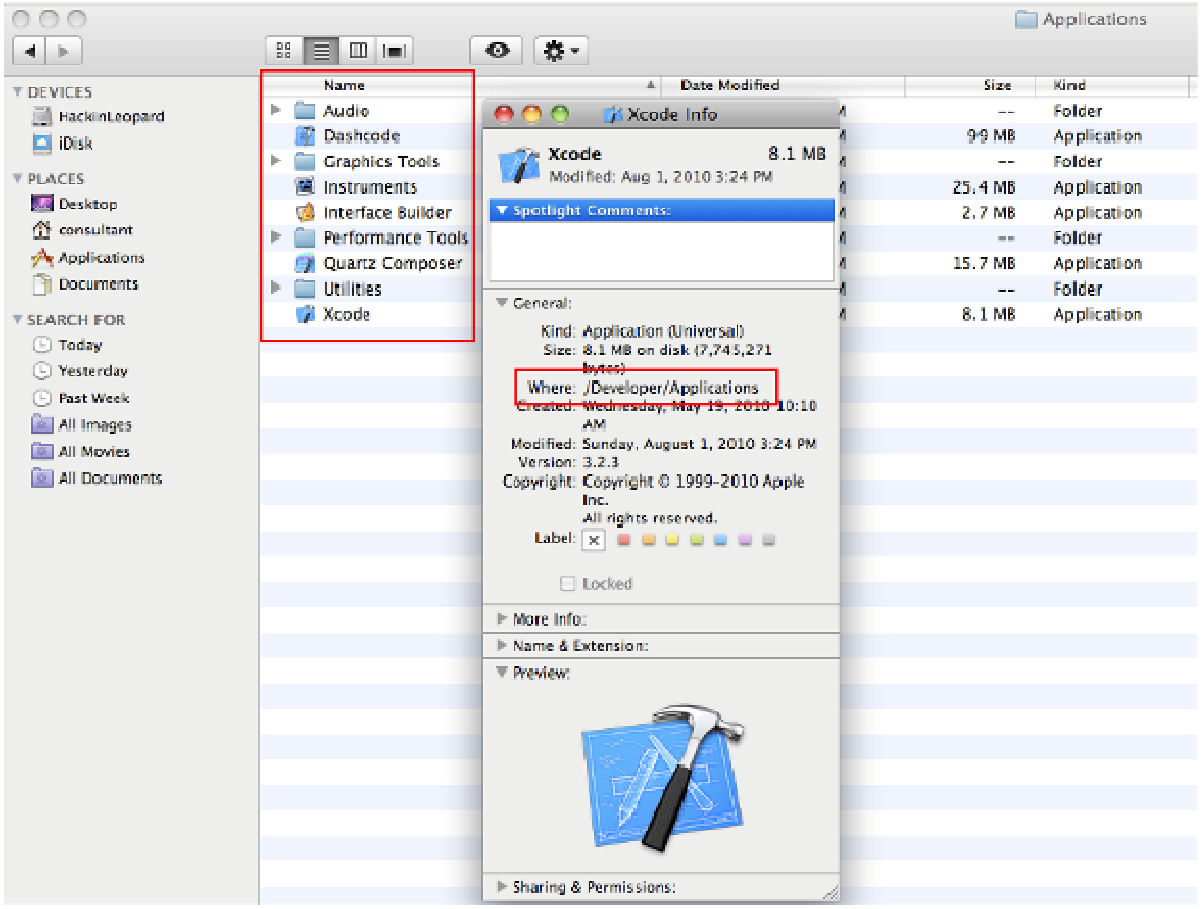

Figure 2: Location of all iPhone tools installed with the SDK

Using the Simulators

After successfully installing the SDK, the simulator can be launched from this location `/Developer/Platforms/iPhoneSimulator.platform/Developer/Applications`.

Figure 3: iPhone Simulator

To access the iPad simulator select this option under the Hardware > Device option as displayed below.

Figure 4: iPad Simulator

Getting Applications to Run within the Simulator

When developers successfully build the application using Xcode, it launches the application with the correct simulator for testing. However, the SDK does not provide a straightforward technique for packaging and transferring these binaries for testers to load. Based on our experience we recommend using the following technique[24] to obtain the binaries from development to the test environment.

Steps for the Developers:

- Launch the application project in Xcode and select Build > Go. This will compile the source code and create the binaries that can then be redistributed if the build was successful.
- Binaries created using the above step will be available at: `/Users/<username>/Library/Application Support/iPhone Simulator/<iOS version e.g. 3.2 (iPad) or 4.0.1 (iPhone)>/Applications/<folder with unique application id>`.
- Copy this folder and provide it to the testers for their analysis.

Steps for the Testers:

- Set up the test environment to match the development environment using the correct Mac OS X and iOS versions.
- Copy the binaries provided by the developers to the same location mentioned above.
- The newly copied application will now be available for testing when the simulator is launched.

[24] http://www.tuaw.com/2009/07/03/developer-to-developer-simulator-application-sharing-for-iphone/

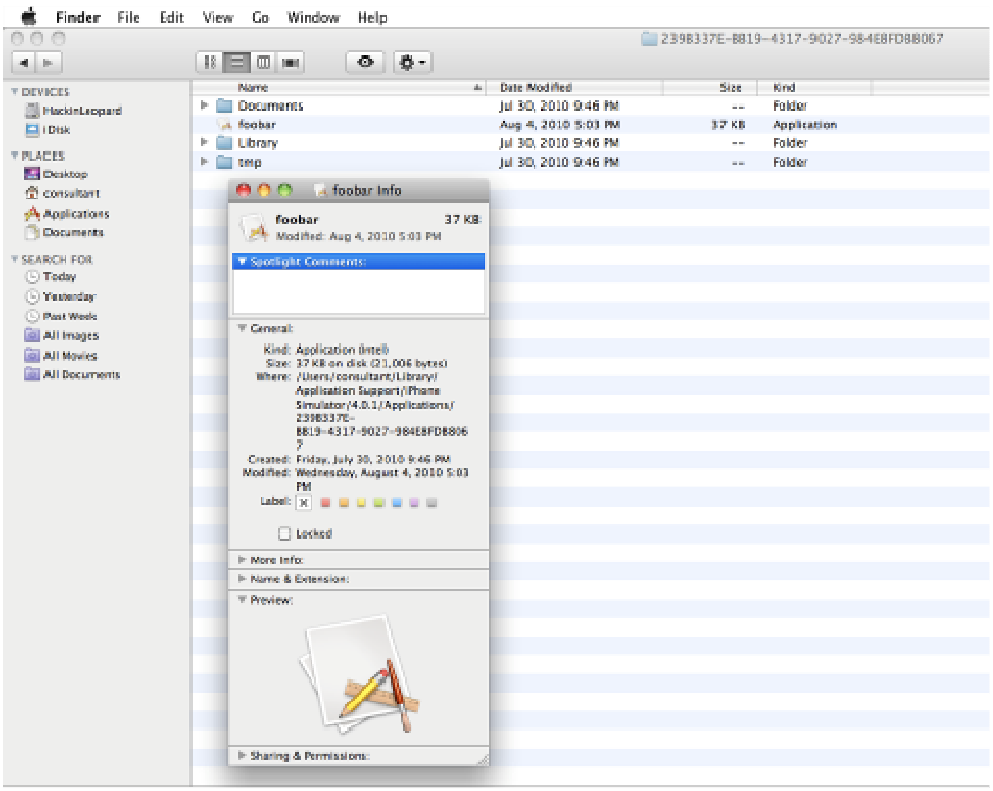

Figure 5: Location of a Sample iPhone Application

Alternatively, you could use the Simlaunch[25] application. It automates the steps mentioned above and makes transferring of the binaries easier and less error prone. This process builds custom executables to automatically launch an embedded iPhone/iPad simulator application using the correct SDK. Simlaunch works with both the iPhone and iPad simulators.

Steps:

- Install the Simulator Launcher application.
- Drag the application binary onto the "Simulator Bundler" icon.
- This will create a new Mac OS X application that bundles and launches the simulator application.
- The figure below shows that the "foobar application" was dropped on the Simulator Bundler icon which created the highlighted "foobar (iPhone Simulator) application". Double clicking this application launches it in the iPhone simulator as shown in the figure below.

[25] http://github.com/landonf/simlaunch/

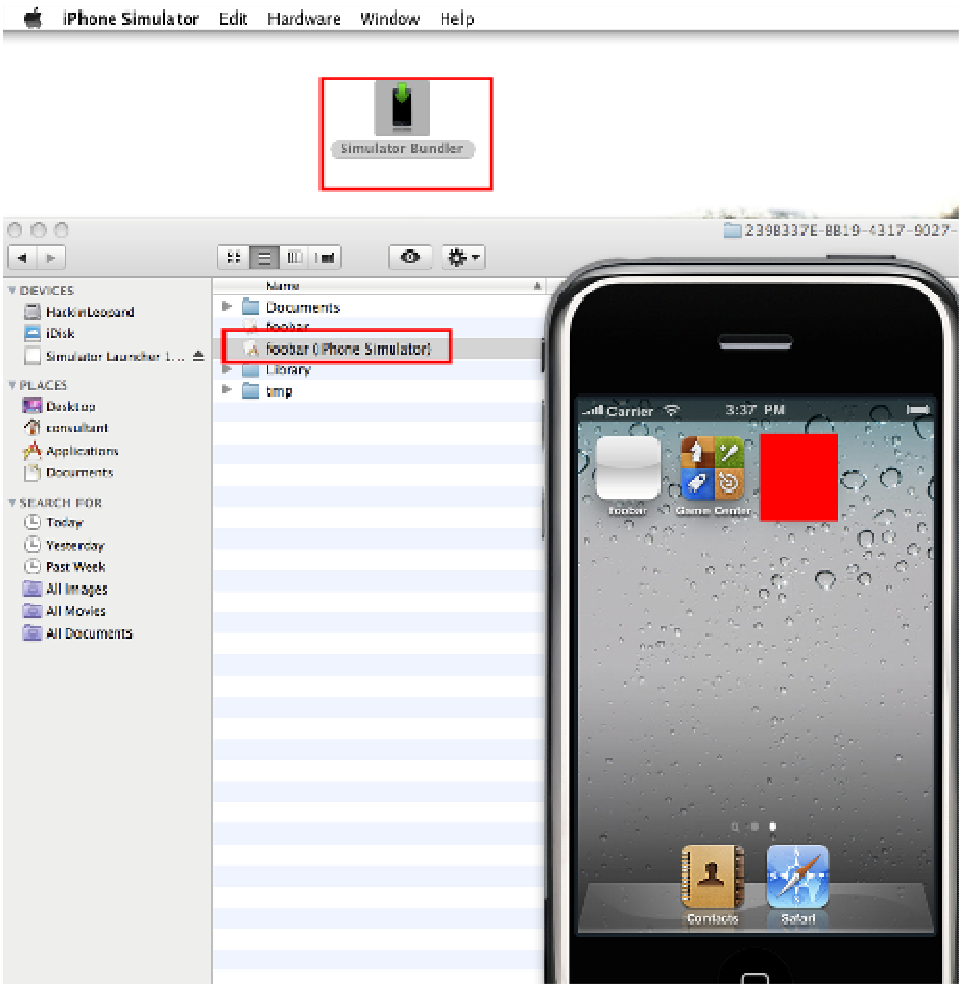

Figure 6: Dragging the foobar Application to the Simulator Bundler Icon Creates the foobar (iPhone Simulator) Application

Setting up a Proxy Tool

The first step in setting up your test environment should be setting up a proxy. Once you have done that a lot of the testing comes down to standard web application penetration testing techniques. There are several proxy tools available[26] for the Mac OS X. The most common choices are WebScarab, Paros, Burp and Charles. The Charles proxy is preferred for two main reasons. First, it provides an option to intercept data from every application running on Mac OS X without requiring manually changing of the proxy settings for each and every application. You just need to enable Proxy > Mac OS X Proxy option as displayed in the figure below. This will intercept all the HTTP(s) requests from the Safari browser, Simulators etc.

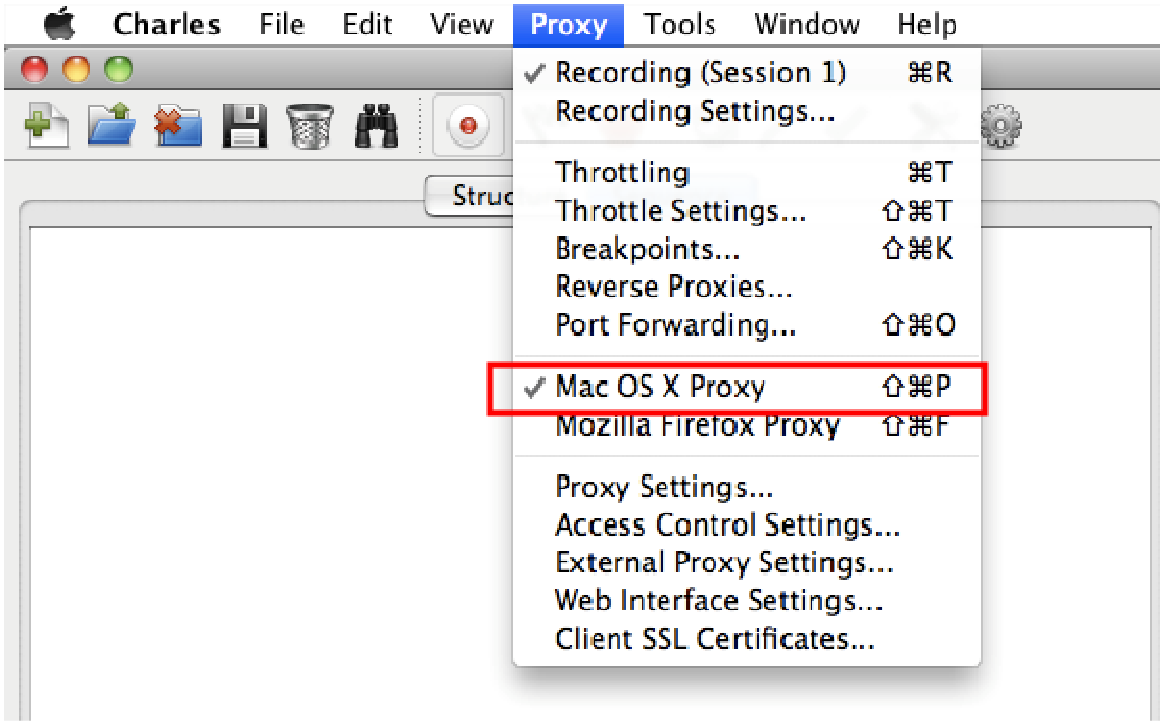

Figure 7: Setting to Intercept all HTTP(s) Requests from all Mac Applications

The second big advantage is that it is easy to setup[27] and works seamlessly with the iPhone/iPad simulators, especially if the application performs server certificate validation checks. It also provides a shell script[28] that could be executed to bypass this check. The script backs up the TrustStore.sqlite3 database and installs the Charles SSL certificate in the keychain for the iPhone/iPad simulator as displayed in the figure below.

[26] http://research.corsaire.com/tools/
[27] http://www.charlesproxy.com/documentation/faqs/#qa_177
[28] http://www.charlesproxy.com/assets/install-charles-ca-cert-for-iphone-simulator.zip

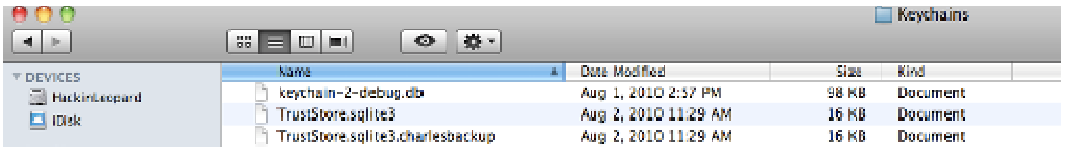

Figure 8: Execution of the Keychain Backup Script

This could also be achieved manually without the need of a script[29]. If TrustStore.sqlite3 database is opened using the SQLite Manager (discussed later in the paper) it can be observed that it stores a SHA1 hash of the server certificate in the `tsettings` table as displayed below.

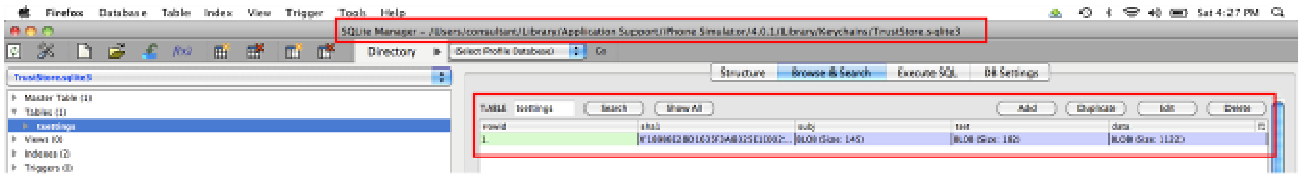

Figure 9: TrustStore.sqlite3 Database within SQLite Manager

The location of trusted certificates for iPhone simulator is: `/Users/<User Profile>/Library/Application Support/iPhone Simulator/4.0.1/Library/Keychains`

The location of trusted certificates for the iPad simulator is: `/Users/<User Profile>/Library/Application Support/iPhone Simulator/3.2/Library/Keychains`

It is possible to manually edit the `tsettings` table to replace the SHA1 hash with the Charles certificate hash. To find the hash for Charles proxy's certificate, install the certificate for it on the Mac using either Safari or Firefox. Open the certificate and find the hash value which can then be pasted into the `tsettings` table as shown in the figure below.

[29] http://stackoverflow.com/questions/347690/iphone-truststore-ca-certificates

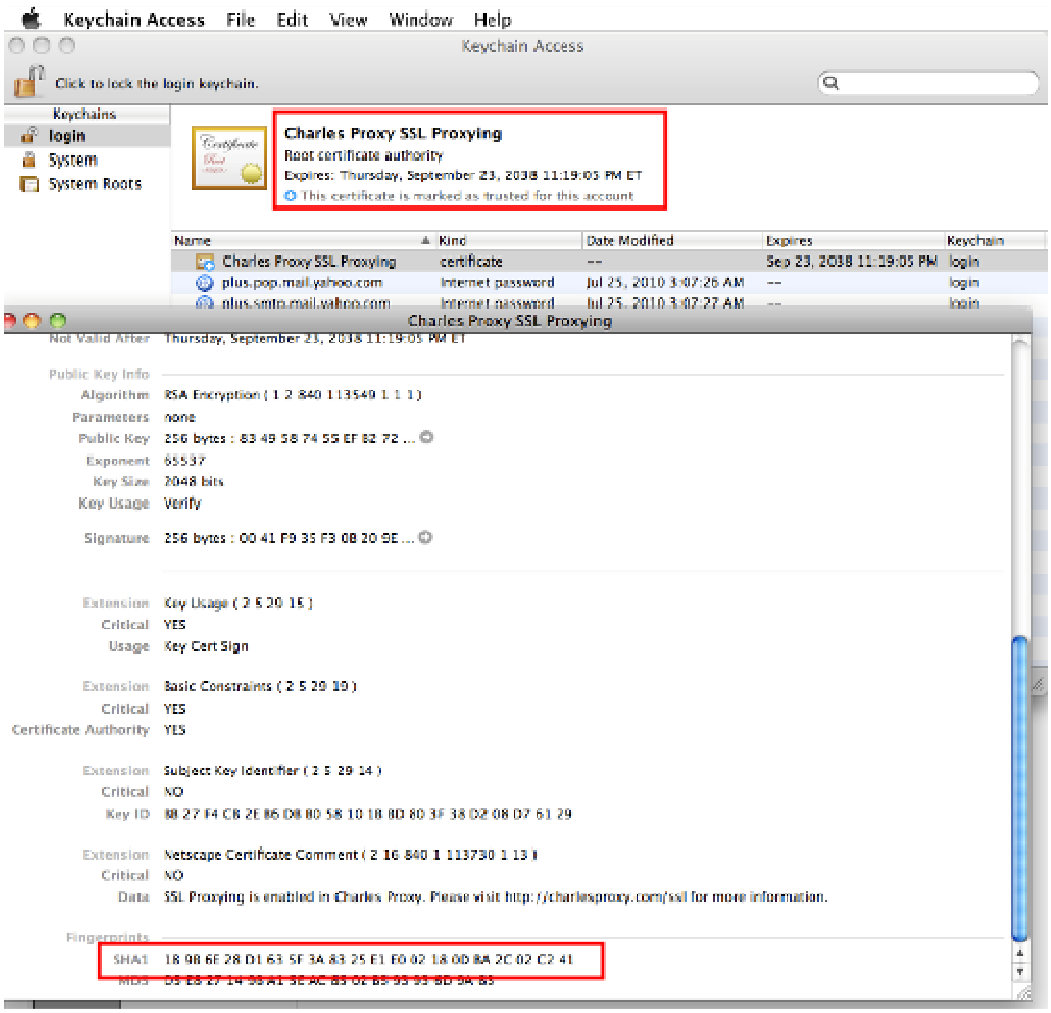

Figure 10: Obtaining SHA1 Hash of the Charles Certificate

Decompiling iPhone/iPad Applications

There are several benefits of decompiling the application when performing penetration testing. It helps one perform a more thorough security assessment by reviewing the code. It is also recommended to run the static source code analyzers mentioned later, on the decompiled code to identify issues such as buffer overflows.

Applications for the iPhone/iPad are written using objective-C, which is fairly easy to decompile. First obtain the application binaries by downloading them from the App Store and then transferring them to your Mac using iTunes. Once you have done this there are two tools available for performing the de-compilation. The first is the "otool" that comes with the Xcode.

Command:

```
otool -toV "/Users/consultant/Library/Application Support/iPhone
Simulator/4.0.1/Applications/744F3613-A728-4BD7-A490-
A95A6E6029F7/HelloWorld.app/HelloWorld" >> Helloworld.dump
```

Figure 11: Decompile an Application Using otool

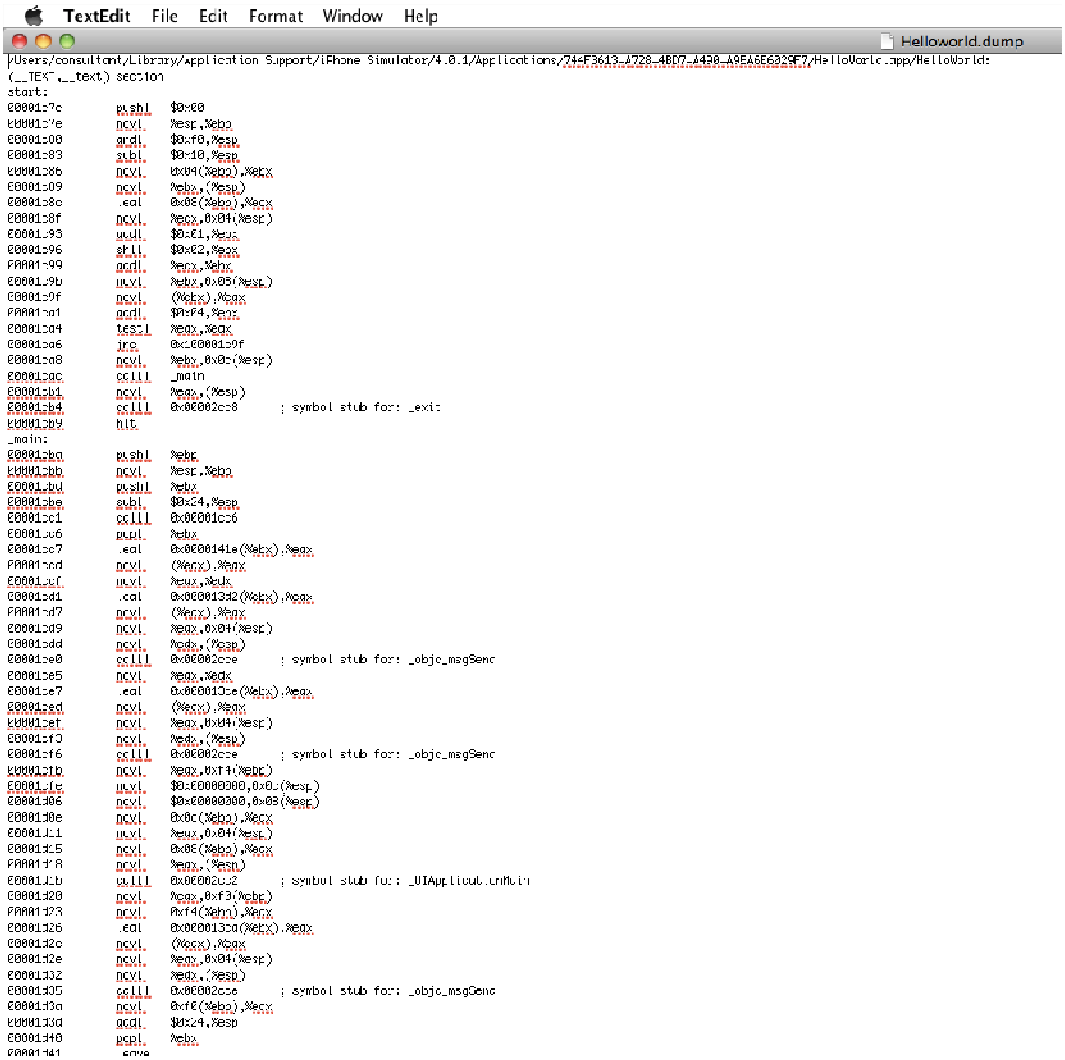

Figure 12: Decompiled Application Using otool

A second option to decompile an application is to use the class-dump-x[30] tool. This tool provides easily readable information on class declarations and structs.

Command:

```
>consultants-macbook-pro-17:Applications consultant$ cd /Applications

>consultants-macbook-pro-17:Applications consultant$ bash

>bash-3.2$ ./class-dump-x "/Users/consultant/Library/Application Support/iPhone
Simulator/4.0.1/Applications/744F3613-A728-4BD7-A490-
A95A6E6029F7/HelloWorld.app" >> Helloworld.classdump
```

[30] http://iphone.freecoder.org/classdump_en.html

TextEdit File Edit Format Window Help

Helloworld.classdump

```
/*
 *     Generated by class-dump 3.1.2.
 *
 *     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2007 by Steve Nygard.
 */

struct _NSZone;

/*
 * File: /Users/consultant/Library/Application Support/iPhone Simulator/4.0.1/Applications/744F3613-A728-4BD7-A490-A95A6E6829E7/HelloWorld.app/HelloWorld
 * Arch: Intel 80x86 (i386)
 */

@protocol NSObject
- (BOOL)isEqual:(id)fp8;
- (unsigned int)hash;
- (Class)superclass;
- (Class)class;
- (id)self;
- (struct _NSZone *)zone;
- (id)performSelector:(SEL)fp8;
- (id)performSelector:(SEL)fp8 withObject:(id)fp12;
- (id)performSelector:(SEL)fp8 withObject:(id)fp12 withObject:(id)fp16;
- (BOOL)isProxy;
- (BOOL)isKindOfClass:(Class)fp8;
- (BOOL)isMemberOfClass:(Class)fp8;
- (BOOL)conformsToProtocol:(id)fp8;
- (BOOL)respondsToSelector:(SEL)fp8;
- (id)retain;
- (oneway void)release;
- (id)autorelease;
- (unsigned int)retainCount;
- (id)description;
@end

@protocol UIApplicationDelegate <NSObject>
@end

@protocol UITextFieldDelegate <NSObject>
@end

@interface HelloWorldAppDelegate :  <UIApplicationDelegate>
{
    UIWindow *window;
    MyViewController *myViewController;
}

- (void)applicationDidFinishLaunching:(id)fp8;
- (void)dealloc;
- (id)myViewController;
- (void)setMyViewController:(id)fp8;
- (id)window;
- (void)setWindow:(id)fp8;

@end

@interface MyViewController :  <UITextFieldDelegate>
{
    UITextField *textField;
    UILabel *label;
    NSString *string;
}
```

Figure 13: Decompiled Application Using class-dump-x

Static Source Code Analysis

Static code analysis[31] is a technique for analyzing code without actually executing it. In most cases, analysis is performed on the source code or the object code. The technique of examining the application during runtime is known as dynamic analysis and is discussed later. As we already know by now it is trivial to decompile an iPhone/iPad application. Attackers thus, have the code and can use these tools to find flaws in the applications and thus it would be essential that we do the same during the testing.

Static Analysis for the applications could be performed using free tools such as Flawfinder[32] or Clang[33]. Flawfinder is only useful if the application uses native C libraries such as `strcpy` instead of Cocoa objects such as `nsstring`. If the application does not use such libraries, then Clang should be used. Static analysis techniques can be leveraged to uncover issues such as memory leaks, uninitialized variables, dead code, type mismatch and buffer overflows among others. This can be done using Xcode if source code for the application is available. The static analyzer travels down each possible code path, identifying logical errors such as memory leaks. Using the IDE this is performed using the Build > Build Analyze menu option as shown in the figure below.

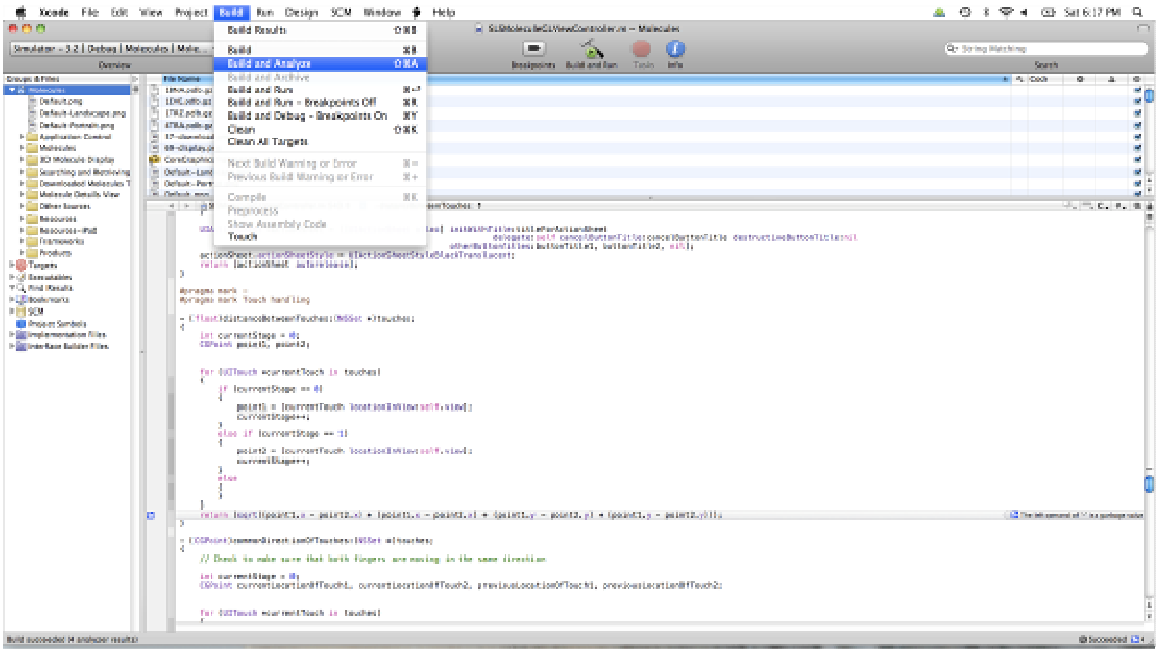

Figure 14: Using Static Analysis

[31] http://developer.apple.com/mac/library/featuredarticles/StaticAnalysis/index.html
[32] http://dwheeler.com/flawfinder/
[33] http://clang-analyzer.llvm.org/

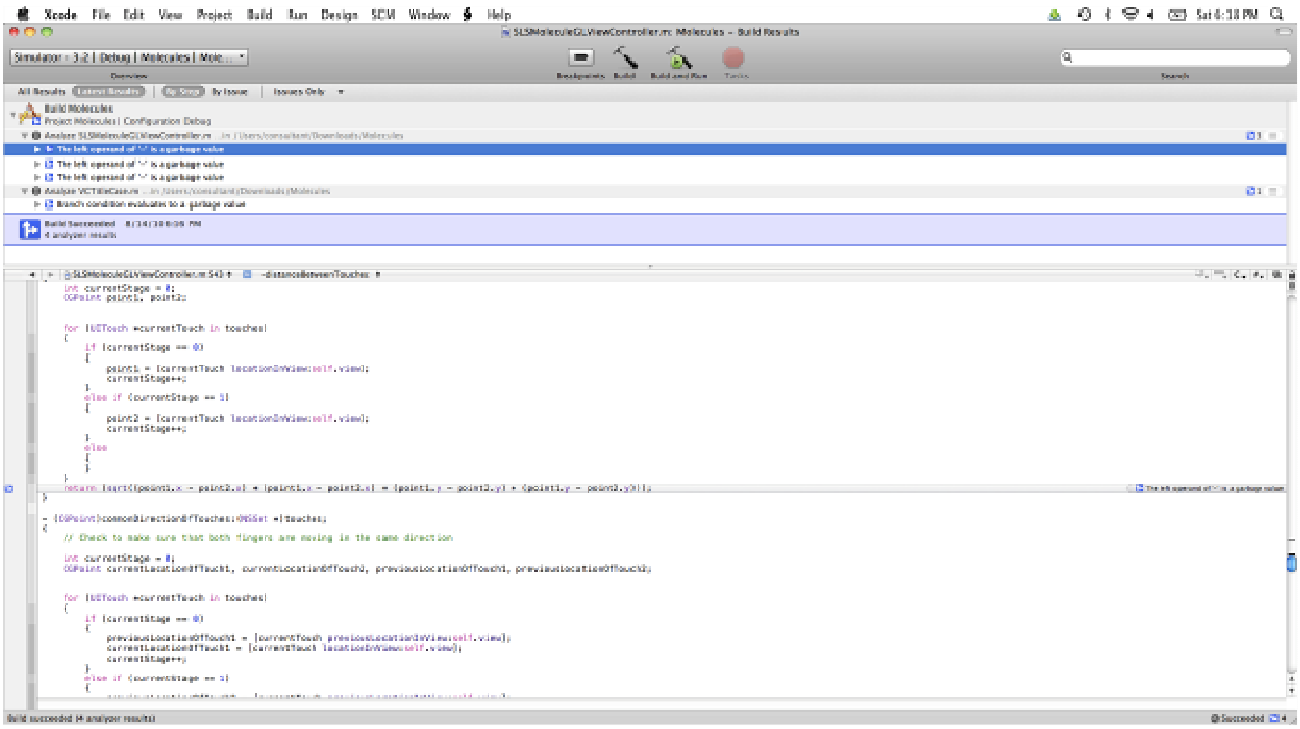

Figure 15: Results from the Analyzer

Dynamic Analysis

Dynamic Analysis refers to the technique of assessing applications during their execution. There are several tools that are provided by Apple for this purpose. The two main tools that we will be discussing in this paper are "Instruments" and "Shark". Detailed description of these and other tools can be found on the Apple website[34].

Instruments

The Instruments tool was introduced with Mac OS X v10.5. It provides a set of powerful tools to assess the runtime behavior of the application. This tool can be compared to several SysInternals[35] tools used for application testing on the Microsoft Windows platform such as procmon and netmon. It can be launched from `/Developer/Applications/Instruments`. Once launched, select the "Blank" template under the iPhone simulator section. Select the instruments needed to use from the library. To inject this tool into a process select Choose Target > Attach to Process > iPhone Simulator (<pid>). Click, record, and start using the application in the simulator to generate the activity data. The type of data then captured by the tool includes:

1. *File Activity Monitoring*: This is similar to filemon in that it lets you identify the files generated and processed by the application. It is useful for identification of files that may be cached, or hidden files used by the application to store data on the client side.
2. *Memory Monitoring:* Helps identify memory leaks.
3. *Process Monitoring*: This is similar to "Process Monitor" and shows real time process / thread activity.
4. *Network Monitoring*: Records network activity like "netmon".

34 http://developer.apple.com/iphone/library/documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html

35 http://technet.microsoft.com/en-us/sysinternals/default.aspx

Figure 16: Use of Different Instruments

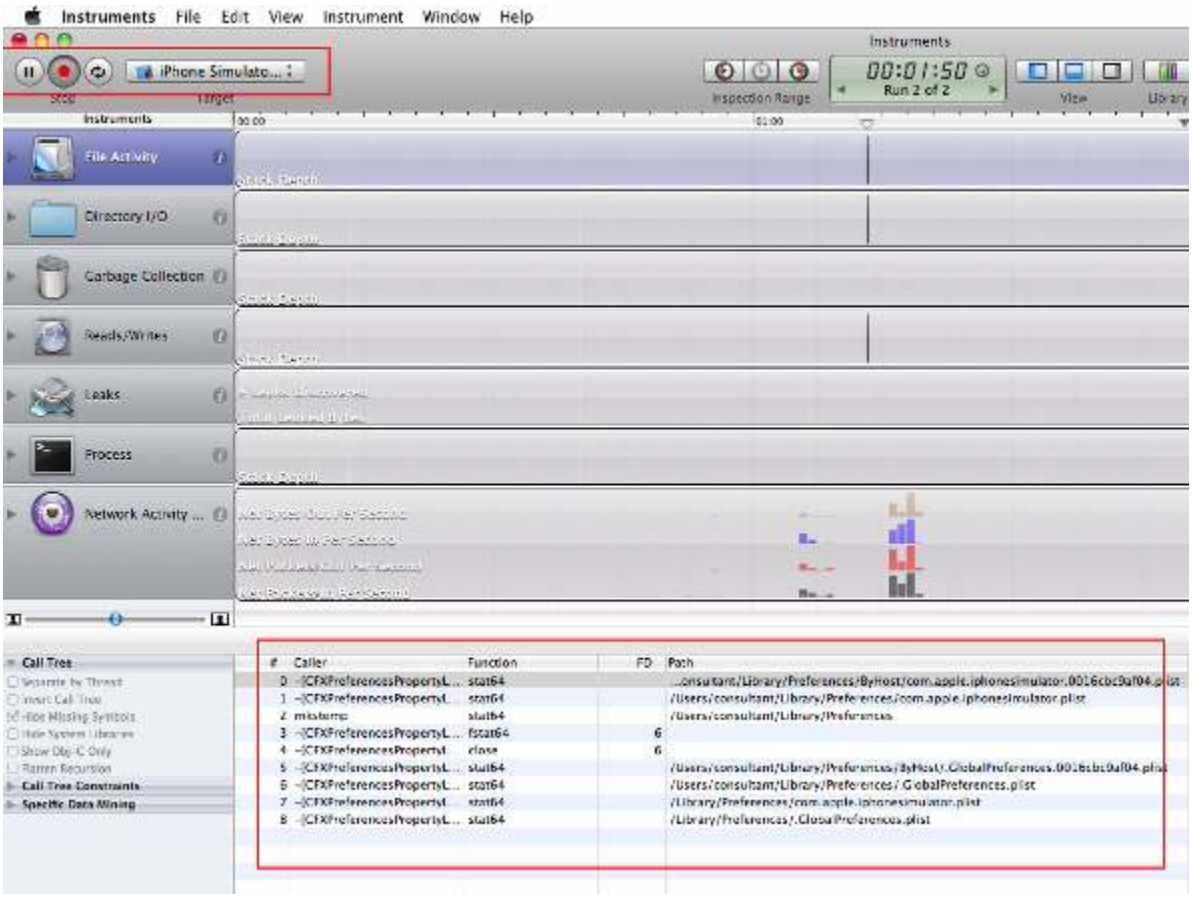

Figure 17: Instruments in Action Recording File Activity Data

Shark

Shark is mainly used for performance monitoring. But, in addition to this, it could also be used to analyze assembly level operations. For instance it could do the following:

1. Statistical sampling of the application over a period of time
2. System-level tracing
3. Malloc tracing
4. Static analysis
5. L2 Cache profiling
6. Java code analysis

It is shipped with every version of Mac OS X 10.3 or newer and comes as part of the Xcode Tools. It can be launched from `/Developer/Applications/Performance Tools/Shark`. After launching it, select what is needed for Shark to trace (e.g. static analysis in our example), specify the Process, and select iPhone simulator as shown in the figure below.

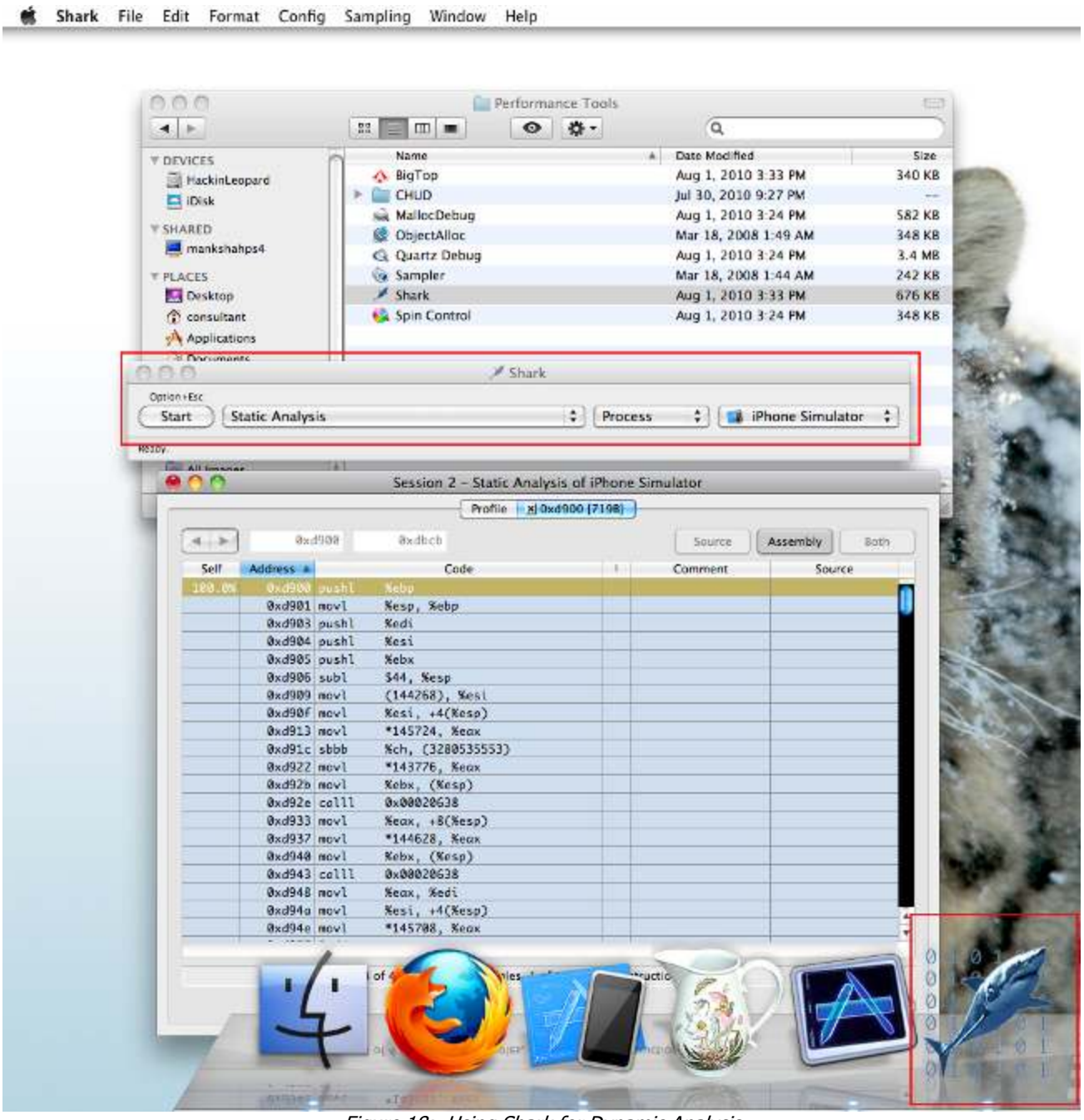

Figure 18: Using Shark for Dynamic Analysis

Data Protection

Data protection is an important category when testing mobile applications as they are more susceptible to loss and theft compared to regular computers. In addition to this, cached data may get copied to the machines that are used for syncing and could be stolen from there. Research has shown that the iPhone does cache sensitive information such as keystrokes and snapshots[36] often for extended periods of time. Moreover, the application itself may be storing sensitive information in form of temporary files, `.plist` files, or in the client side SQLite database. During security testing it is critical therefore to identify these risks and provide recommendations to mitigate them.

Keyboard Cache

All the keystrokes[37] entered on an iPhone could potentially get cached[38] in `~/Library/Application Support/iPhone Simulator/4.0.1/Library/Keyboard/dynamic-text.dat` for auto correction unless appropriate measures are taken. This issue is similar to the `AUTOCOMPLETE` for the web browsers. If `AUTOCOMPLETE` is not set to off for the `UITextField` then the text entered in these fields will get cached. It should be noted however that the iPhone does not store password fields at any time irrespective of these flags.

[36] http://www.telegraph.co.uk/technology/apple/7880155/How-your-Apple-iPhone-spies-on-you.html
[37] http://www.security-faqs.com/did-you-know-that-the-iphone-retains-cached-keyboard-data-for-up-to-12-months.html
[38] http://stackoverflow.com/questions/1955010/iphone-keyboard-security

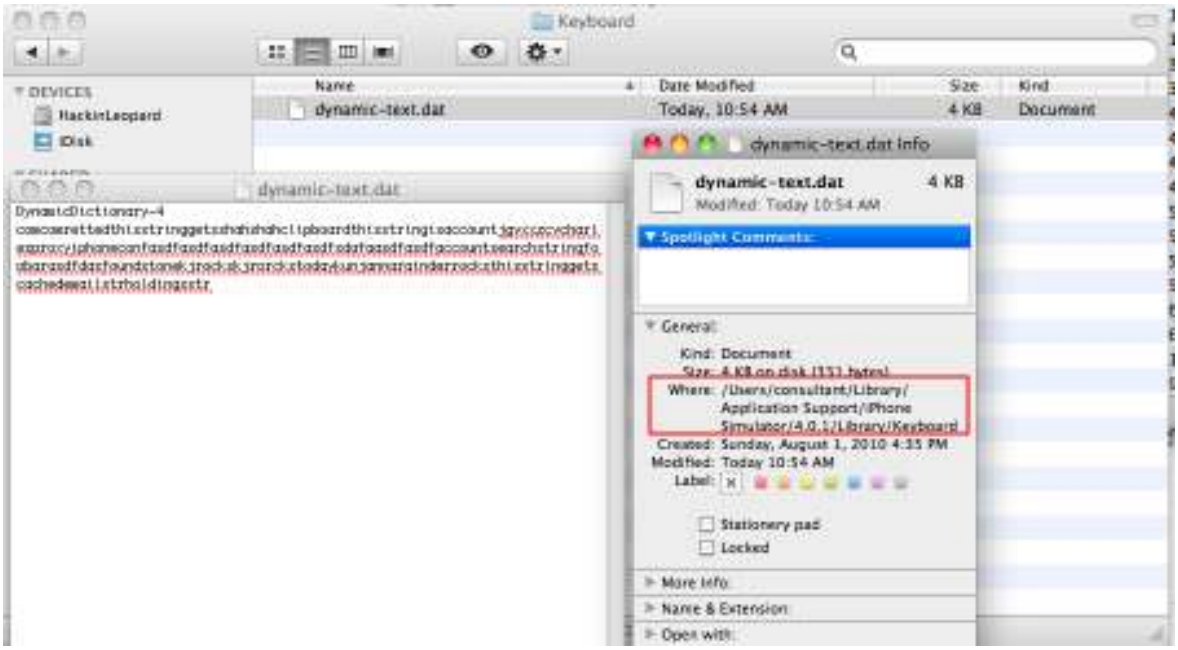

Figure 19: Cached Keystrokes in dynamic-text.dat

Snapshots

Every time the user taps the Home button, the window of the open application shrinks and disappears. In order to create this shrinking effect, iPhone takes an automatic screenshot[39]. These snapshots are stored in the snapshots directory of the application. For example the "sample Helloworld" application stores them at `~/Library/Application Support/iPhone Simulator/4.0.1/Applications/744F3613-A728-4BD7-A490-A95A6E6029F7/Library/Caches/Snapshots/com.yourcompany.HelloWorld`.

Applications should thus, mask sensitive information on the screen to, not only prevent it from shoulder surfing attacks but, also from getting leaked via such snapshots.

[39] http://www.wired.com/gadgetlab/2008/09/hacker-says-sec/ http://www.iphonefootprint.com/2008/09/iphones-privacy-flaw-it-takes-automatic-screenshots-of-all-your-latest-actions/

Figure 20: Automatic Screenshots and their Location

Individual users with privacy concerns could follow steps available online[40] to disable the screenshots on a jailbroken iPhone.

UIPasteBoard

If the iPhone application uses `UIPasteBoard` for copying and pasting objects, this information could be obtained by other applications from the clipboard. In addition to this if the persistent pasteboard property is used by the developer, the copied information will be stored unencrypted on the iPhone's file system and can be found at `~/Library/Application Support/iPhone Simulator/4.0.1/Library/Caches/com.apple.UIKit.pboard`. If the application contains sensitive information, it is therefore critical for them to use private pasteboards for copy and paste operations. Also, the persistent property should be used sparingly.

[40] http://www.iphone-hacks.com/2008/09/24/how-to-disable-the-iphones-automatic-screen-capture/

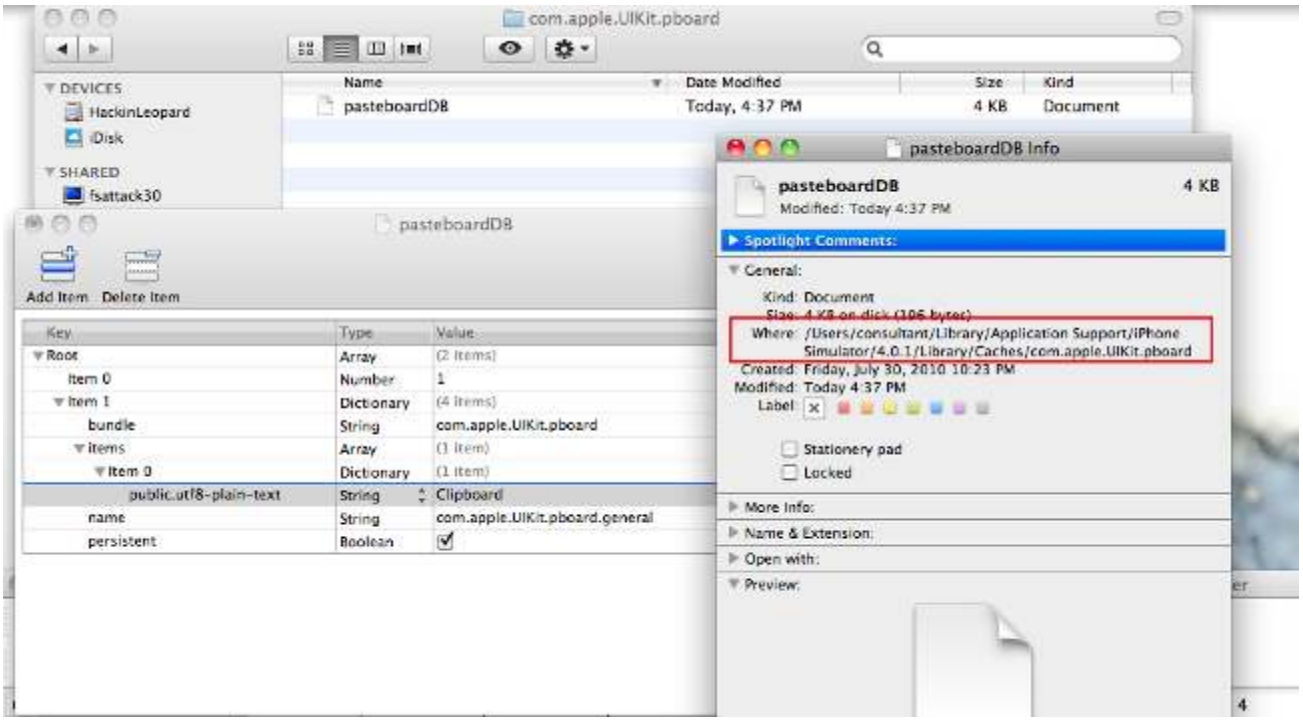

Figure 21: Location of the PasteBoard

Cached files

If the application displays PDF, Excel or other files, then it is possible that these files may also get cached on the device. These can then be found at "`/Users/<username>/Library/Application Support/iPhone simulator/3.2/Applications/<application folder>/Documents/temp.pdf`" as displayed in the figure below.

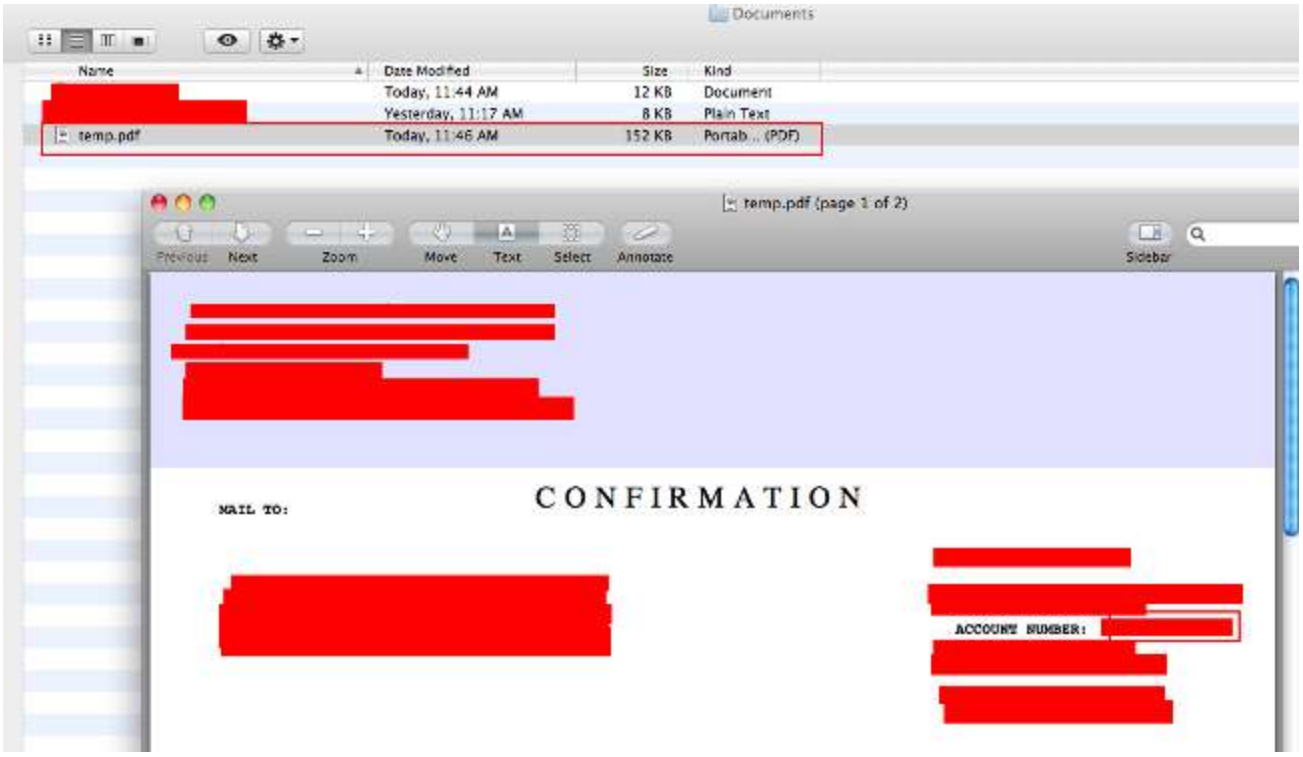

Figure 22: Cached PDF file with Account Number Information

SQLite Database

iOS applications store client side data in the SQLite database on the device. Information in this database is often not encrypted and can therefore contain sensitive information such as account numbers, SSN etc. It may also contain the application state information which could be altered to bypass the application logic. To read, or edit the SQLite database any of the available SQL clients can be used. For example, the SQLite Manager Firefox add-on[41] is a popular tool for this purpose. From a best practice perspective sensitive data should never be stored on the client side as far as possible. It should always be kept on the server side or at the very least stored in the keychain. Encryption of the data in the SQLite database should be used as a last resort as the implementation may become complex and require careful key management.

Figure 23: Account Number in the SQLite Database

Property list (.plist) files

[41] http://code.google.com/p/sqlite-manager/

Property list files are not a good place to store sensitive information either. Instead as discussed above, applications should store sensitive information in the keychain. Apple uses sandboxing mechanism to limit access from one application to another application's data. However, despite sandboxing, numerous application property files are in fact readable by other applications. This is because of the loose sandbox rules. In addition to this the file system can be browsed and files read using open source tools such as Fswalker[42] even on non-jailbroken devices.

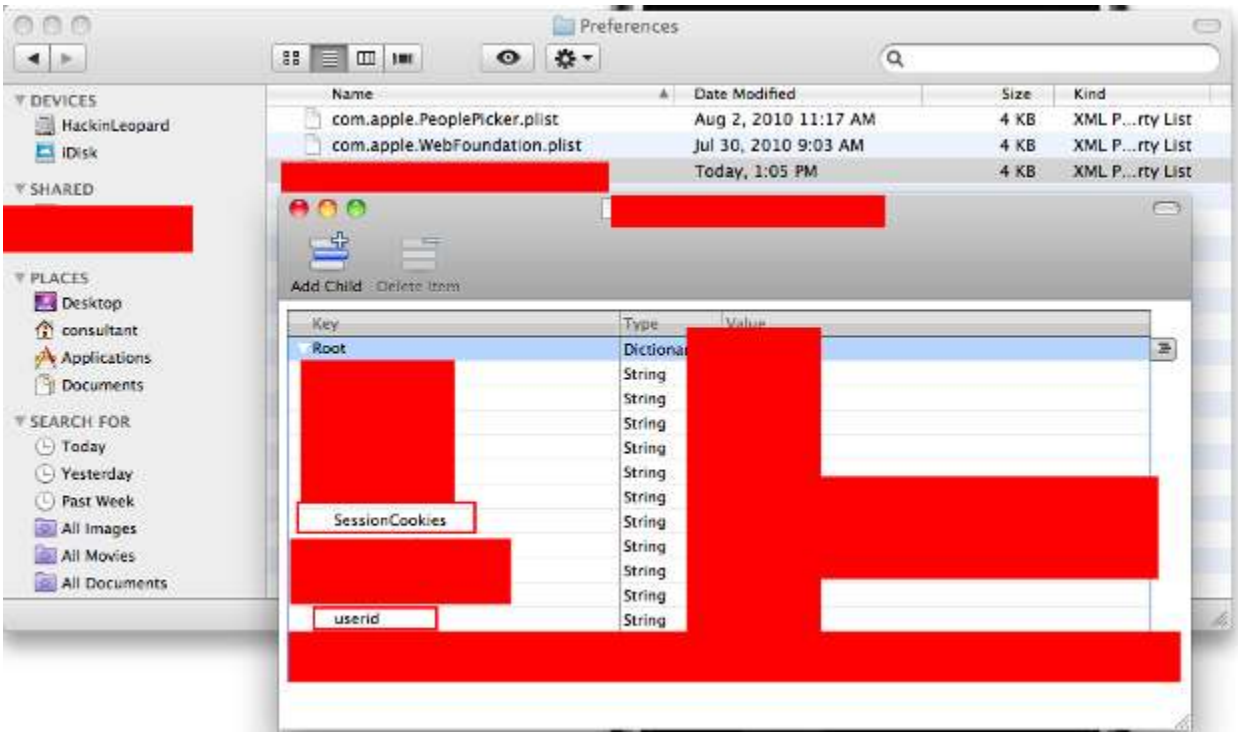

Figure 24: Userid Stored in the .plist File

[42] http://code.google.com/p/fswalker/

Log Files

Applications can generate excessive logs if the amount of logging is not toned down in production version of the application. Moreover, these log files may contain sensitive information that can be leaked. Logs for iOS applications are usually stored at the following locations:

- `~/Library/Logs/CrashReporter/MobileDevice/<DEVICE_NAME>`
- `/private/var/log/system.log`

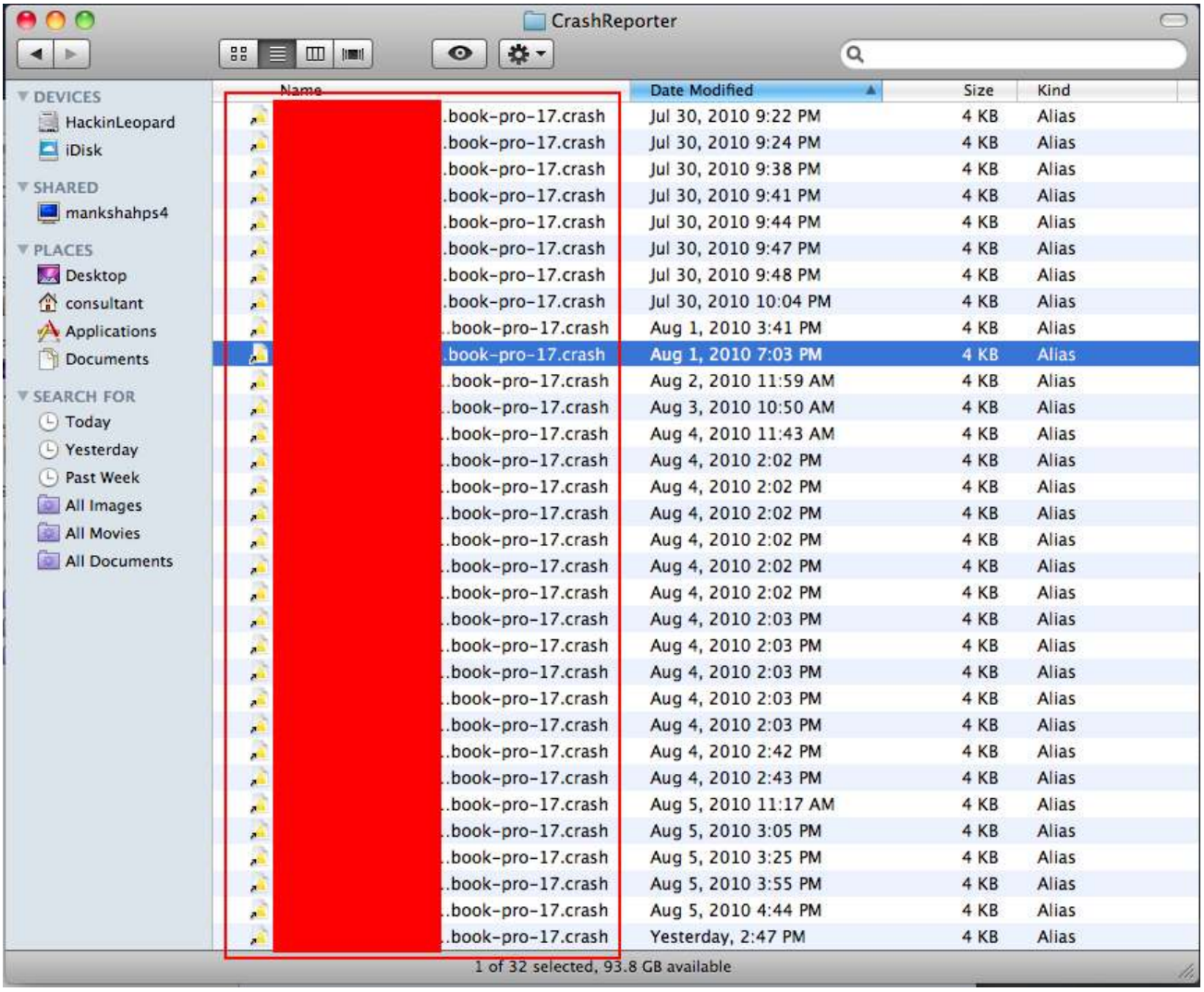

Figure 25: Crash Log Files

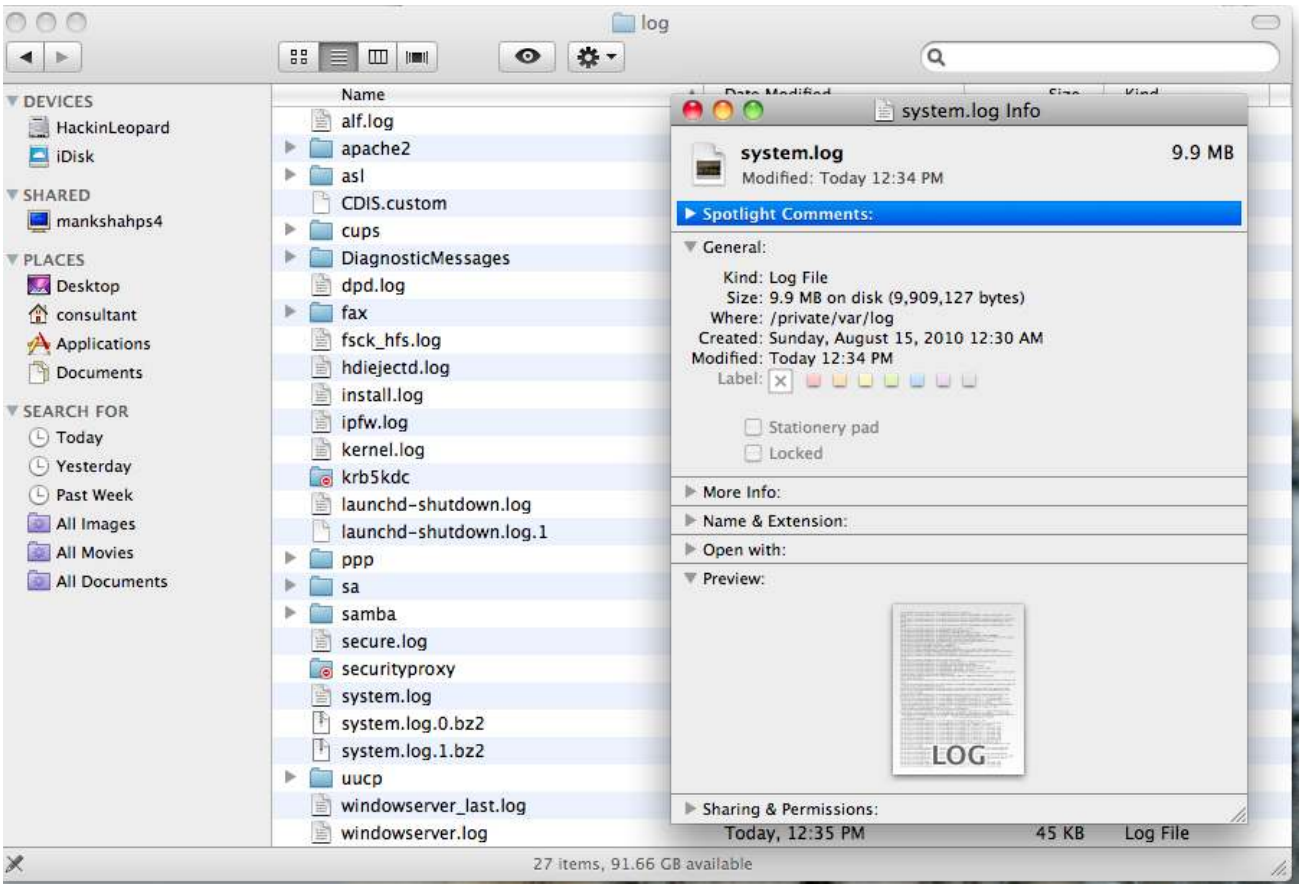

Figure 26: Location of the system.log

Conclusion

As security professionals responsible for penetration testing iOS applications it is important that we are aware of some of the inherent privacy risks with the device, the data protection issues with these applications and learn the tools and techniques available for testing them. It is also important to test for the new vulnerabilities specific to iOS applications or variants of the old vulnerabilities. This paper could serve as a guide when testing these applications.

About the Author

Kunjan Shah is a Security Consultant at Foundstone Professional Services, A division of McAfee based out of the New York office. Kunjan has over 5 years of experience in information security. He has dual Master's degree in Information Technology and Information Security. Kunjan has also completed certificates such as CISSP, CEH, and CCNA. Before joining Foundstone Kunjan worked for Cigital. At Foundstone Kunjan focuses on web application penetration testing, thick client testing, mobile application testing, web services testing, code review, threat modeling, risk assessment, physical security assessment, policy development, external network penetration testing and other service lines.

Acknowledgements

I would like to thank Rudolph Araujo and John D'Agostino for reviewing this paper and providing useful feedback, and suggestions on making it better.

About Foundstone Professional Services

Foundstone® Professional Services, a division of McAfee. Inc. offers expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, Foundstone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively. The company's professional services team consists of recognized security experts and authors with broad security experience with multinational corporations, the public sector, and the US military.