University of Piraeus – Department of Informatics
Master Program in
«Informatics»

Postgraduate Thesis

Thesis Title	**(iOS application Security Analysis)**
Student First and Last Name	**Konstantinos Vlachos**
Father's Name	**Georgios**
Registration Number	**MPSP15011**
Supervisor	**Konstantinos Patsakis, Assistant Professor**

Delivery Date **November 20, 2017**

TABLE OF CONTENTS

ABSTRACT

The purpose of this research is to explain the nature of the Apple iOS applications and provide all the available Open Source tools for analyzing them, starting from decrypting any application's binary downloaded from the AppStore to reverse engineering it and even altering the flow of its running process on the actual device.

We start introducing the basic theory of the iOS operating system and its applications including the security mechanisms incorporated by Apple that are also the main targets of every iOS exploit or jailbreak developer. The next step is to describe the process of setting up the testing environment with a Macintosh OS and the Xcode IDE and/or an actual jailbroken iOS device (iPhone, iPad, iPod). The rest of the chapters describe the installation and usage of tools to implement the whole application security analysis procedure, starting from static to dynamic analysis, after having decrypted and reverse engineered the application's binary and even interacting in an unplanned manner with the running process to change its method calls or arguments.

A walk through on Data Protection on iOS follows, describing possible ways of data leakage on such devices. Finally, a chapter is dedicated to further in-depth explanation of important technical terms used throughout the whole document. The dissertation concludes, in the last phase. In this chapter, it is being assumed that for every step in application security analysis on the iOS platform the appropriate tools have been provided in this document. Further suggestions for research are being provided for those interested too.

DISCLAIMER: ***All the tools, software and commands used throughout this dissertation are strictly restricted to the scope of this research and have not been used, in any case, against Apple's commercial products like devices or software. Illegal use of any of the reported tools is not allowed and is your own responsibility!***

CHAPTER 1

INTRODUCTION

The default programming language used for developing iPhone / iPad applications 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 will be 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. However, Apple iOS devices (iPhone, iPad, iPod) also have their own vulnerabilities that are known to have been exploited like Kernel exploits (heap overflow, API Kext misuse), Sandbox misconfiguration, entitlements leverage and many more **[1]**.

Sometimes the vulnerabilities are found in the actual operating system and other times they are found in the installed software. To fully compromise such a device, the first entry point of interaction has to be exploited which is going to be most of the times a third party installed application or a system application or a combination of them through their entitlements interaction. Therefore, our main focus in this document is to create a baseline tutorial which provides the knowledge and tools to find any possible security flaw in iOS applications.

As commonly known, Apple puts a great amount of effort to provide its systems and end users with security and privacy. This leads to few detailed resources available online about the current topic regarding step by step guides on iOS application security analysis and potential exploitation. An effort to provide such a walk-through guide is also the current document's purpose.

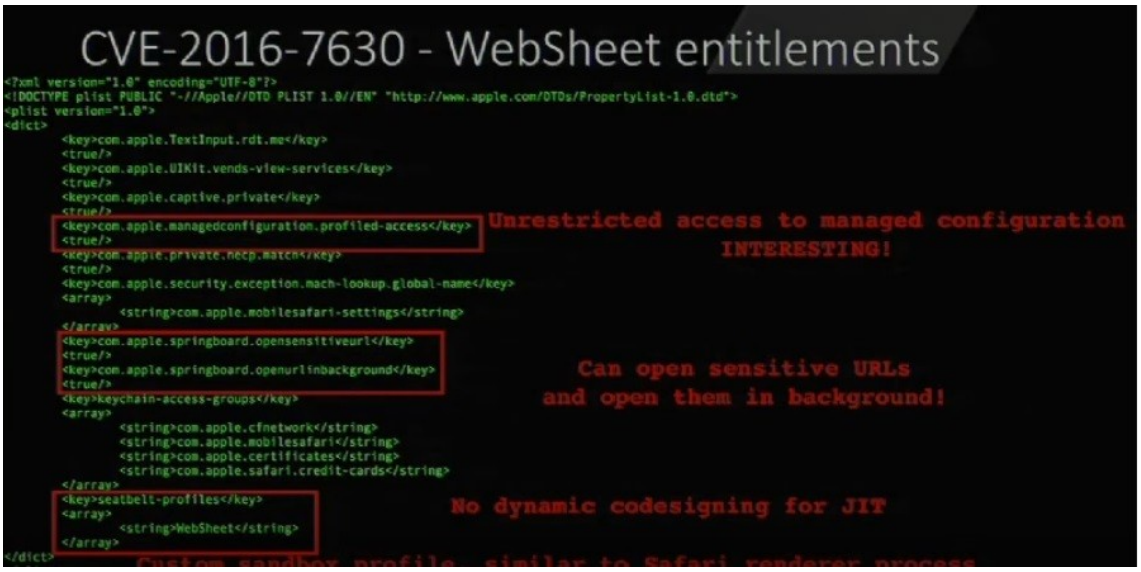
Firstly, the context in which applications run within the iOS must be broken down and understood. The iOS uses a very similar to the OSX kernel the XNU based on Darwin, that functions underneath almost the same way as Unix like Operating System. This leads to file and folder ***permissions of two types***:

- ***read-write:*** everyone reads or writes this file or directory.
- ***Read-only:*** only the user of the application (owner) writes to it.

The users available on the iOS are mobile (user running the application with restricted permissions "501" uid) and root (super user can be used by others and not the actual system only on jailbroken devices). **[5]**

Every application is running in a sandbox, an environment where code is deemed untrusted and is isolated from other resources and processes of the operating system. Apple also allows access to the device resources through classes to specific interfaces like Camera, microphone, GPS but prevents the application from directly accessing those components providing only an API that interacts with the Kernel extensions known as kexts **[2].** APIs are categorized as Private and Public ones. Public APIs are used from every developer and can be included in the source code of every third-party application. The Private ones are only used from system applications and are not allowed to be used from any third party as they would pose a security issue. They are commonly used with Theos on tweaks and Cydia though, for jailbroken device application development (these terms will be discussed later). **[3]**

Applications use entitlements for special capabilities that are like Android permissions and are included in the binary. It is a single right granted to a particular application or other executable that gives additional permissions beyond what it would ordinarily have. The term *entitlement* is most commonly used in the context of a sandbox. It is actually a piece of configuration information included in your app's code signature (in xml format as shown on the figure below) telling the system to allow your app to access certain resources or perform certain operations. In effect, an entitlement extends the sandbox and capabilities of your app to allow a particular operation to occur **[6]**. They are set during application development using Xcode and the final result can be viewed using the command:
"ldid -e /Applications/Whatever.app/"
An example of insecure entitlements is demonstrated below, that was leveraged to achieve an iOS Wi-Fi exploit (Marco Grassi CVE-2016-7630) **[7]**



CVE-2016-7630 - WebSheet entitlements

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>com.apple.TextInput.rdt.me</key>
        <true/>
        <key>com.apple.UIKit.vends-view-services</key>
        <true/>
        <key>com.apple.captive.private</key>
        <true/>
        <key>com.apple.managedconfiguration.profiled-access</key>
        <true/>
        <key>com.apple.private.necp.match</key>
        <true/>
        <key>com.apple.security.exception.mach-lookup.global-name</key>
        <array>
                <string>com.apple.mobilesafari-settings</string>
        </array>
        <key>com.apple.springboard.opensensitiveurl</key>
        <true/>
        <key>com.apple.springboard.openurlinbackground</key>
        <true/>
        <key>keychain-access-groups</key>
        <array>
                <string>com.apple.cfnetwork</string>
                <string>com.apple.mobilesafari</string>
                <string>com.apple.certificates</string>
                <string>com.apple.safari.credit-cards</string>
        </array>
        <key>seatbelt-profiles</key>
        <array>
                <string>WebSheet</string>
        </array>
</dict>
```

Unrestricted access to managed configuration
INTERESTING!
Can open sensitive URLs
and open them in background!
No dynamic codesigning for JIT
Custom sandbox profile, similar to Safari renderer process

WebSheet entitlements – Marco Grassi – CVE-2016-7630

Apple also uses a code signing mechanism. Once the iOS kernel has started, it controls which user processes and apps can be run. To ensure that all apps come from a known and approved source and haven't been tampered with, iOS requires that all executable code must be signed using an Apple-issued certificate. Apps provided with the device, like Mail and Safari, are signed by Apple. Third-party apps must also be validated and signed using an Apple-issued certificate making it difficult to run arbitrary unsigned code **[4]**. On top of that, Apple introduced the DEP (Data Execution Prevention) mechanism on iOS 2.0 and after iOS 4.3 the ASLR (Address Space Layout Randomization) was introduced which fully randomizes the position of an application's executable parts loaded in RAM if it is compiled as a PIE (position-independent-executable) **[8]**.

All the aforementioned information needs to be taken into consideration in order to achieve a full exploitation of an iOS system. Our focus in this document though; is mainly the applications of the Apple's mobile operating system as stated before and the ways of analyzing them from a security perspective using only ***Open Source*** tools. So, this is going to be a walk-through guide on how to decrypt, reverse engineer (to the best possible level for lateral analysis), statically and dynamically analyze and even hook into an application's process. The part of exploitation will not be covered as it is considered beyond the scope of this thesis.

Commercial iOS ***applications*** available on the AppStore are compressed (.ipa files described on Chapter 8 ***"Technical Terms"***) and encrypted files containing the application binaries and resources. On the other hand, applications from Apple's IDE "***Xcode"***, provide us with the full source code and project directory. The first case scenario is the most common in terms of analysis and the second one comes up in case the application owner or developers ask us for code auditing on their own application or we somehow manage to get our hands on the original application's Xcode project. Both options are going to be discussed, providing you with the available options and tools, whatever the case is.

As already stated Apple's IDE for iOS application development is ***Xcode***. We will cover the steps of installing it on a MAC OSX system and how to interact with it using its built in iOS simulator or a directly connected iOS device (iPhone or iPad) to run-debug an application. The IDE is useful in case we have to deal with the original application project (second option stated above). In any other case you need to interact with a jail broken (refer to ***"Technical Terms"***) iPhone device over ssh protocol using the described tools.

In most cases, we will have to deal with commercial applications downloaded from the AppStore. So, decryption and decompilation are key factors in a successful iOS application analysis and several tools and options are going to be described on that, before we are able to go through a static or dynamic analysis. Static analysis is more useful on Xcode projects, because on third party applications the initial source code level could never be reached but only to a point of assembly language and Objective-C message or NSobject calls and system interaction.

Finally, the most important part is dynamic analysis and here is where the most effort is applied in order to be able to analyze every commercial application available online. Options like heap inspection, message calls and even process hooking will be described providing you with interaction level of even changing the application's process behavior, calls and functionality.

CHAPTER 2

PREREQUISITES

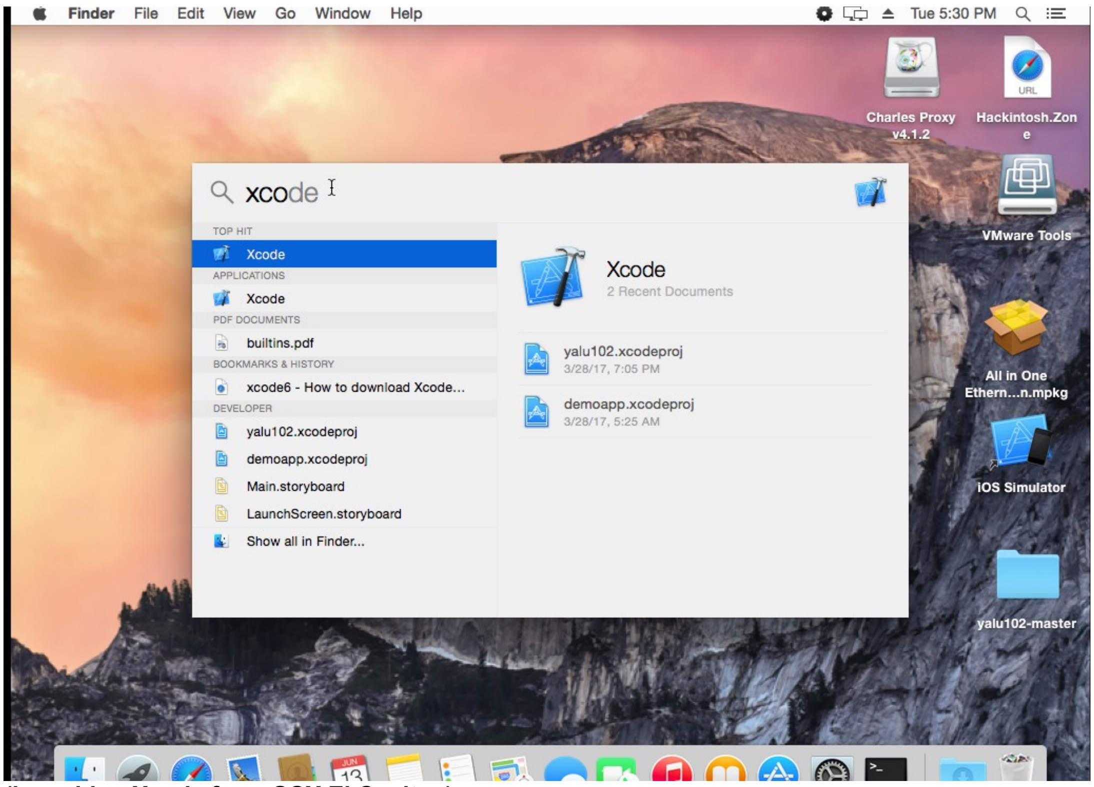
The required testing environment for iOS application analysis can be set up with minimum cost using an OSX environment and the XCode's iPhone simulator. An iOS device is always needed to interact with third party applications that run only on ARM architecture. To begin with, the basic equipment needed consists of the following:

- **Mac Book** running Snow Leopard 10.6.2 OS or above.
- **Xcode** 6.1 or greater (Apple's **iOS IDE**).
- **iOS SDK** and/or **iOS actual device** to run/debug the applications against (a jail broken device).
- Preferably an **apple ID – Account**, to download application binaries.

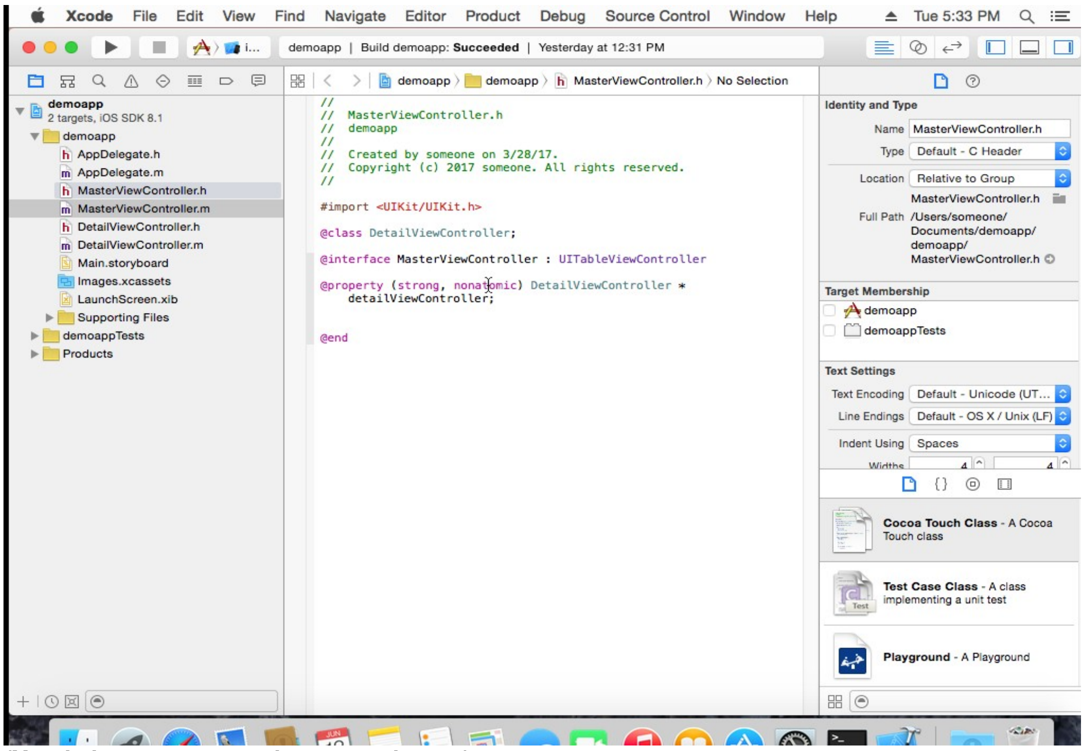
The steps to get into an OSX Desktop are not covered as this is a trivial task.

2.1 Install Xcode

Initially, the **Xcode** iOS IDE has to be installed on MAC OSX environment. After the file (xcode****.dmg) is downloaded, we install it and then we type XCode into the quick search to launch the IDE (shown on figure below) by selecting one of the already opened projects, same as Android Studio's start up process.



(launching Xcode from OSX El Capitan)


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Tue 5:33 PM
demoapp | Build demoapp: Succeeded | Yesterday at 12:31 PM
demoapp > demoapp > MasterViewController.h > No Selection
demoapp
2 targets, iOS SDK 8.1
demoapp
AppDelegate.h
AppDelegate.m
MasterViewController.h
MasterViewController.m
DetailViewController.h
DetailViewController.m
Main.storyboard
Images.xcassets
LaunchScreen.xib
Supporting Files
demoappTests
Products
Cocoa Touch Class - A Cocoa Touch class
Test Case Class - A class implementing a unit test
Playground - A Playground

```
//
//  MasterViewController.h
//  demoapp
//
//  Created by someone on 3/28/17.
//  Copyright (c) 2017 someone. All rights reserved.
//

#import <UIKit/UIKit.h>

@class DetailViewController;

@interface MasterViewController : UITableViewController

@property (strong, nonatomic) DetailViewController *
    detailViewController;

@end
```

Identity and Type
Name: MasterViewController.h
Type: Default - C Header
Location: Relative to Group
MasterViewController.h
Full Path: /Users/someone/Documents/demoapp/demoapp/MasterViewController.h

Target Membership
demoapp
demoappTests

Text Settings
Text Encoding: Default - Unicode (UT...
Line Endings: Default - OS X / Unix (LF)
Indent Using: Spaces
Widths: 4 4

(Xcode home screen – demoapp chosen)

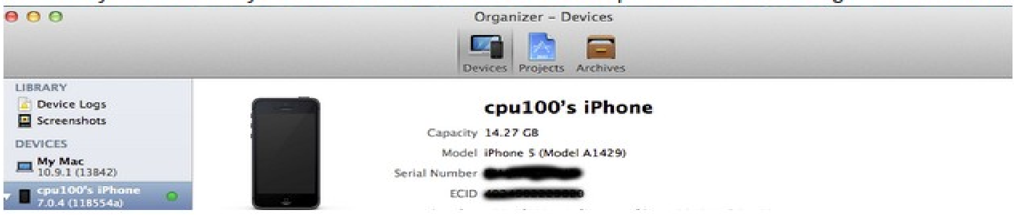
Regarding the **iOS** environment on which XCode applications can be debugged and executed, Apple does not allow the use of a packed simulator version as a standalone download. Although, Xcode includes a built-in simulator as already stated, that can be used after successfully installing the whole package (SDK).

*An important note here is that **only registered** Apple developers can download and use Xcode for signed projects (legitimate Apple applications). An **iOS device** can also be used (similar to Android studio with usb debugging mode on), while being connected over the wire (usb as shown in picture below). Both options are used in this document.*

Connecting iPhone to Xcode with Apple developer ID for production purposes

For ios7 and Xcode 5 and for testing it, follow the below steps:

1. You need to request a certificate (if this is your first time)
2. Connect you iPhone to your Mac and then click on "Development in Xcode's Organizer window"

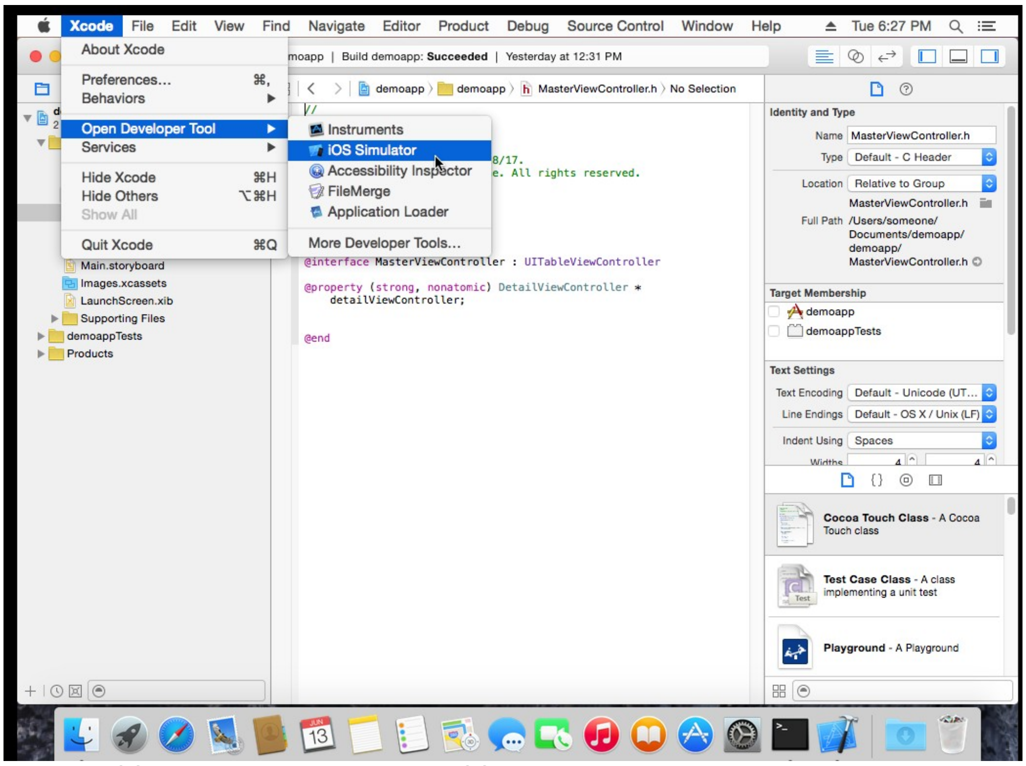


cpu100's iPhone
Capacity 14.27 GB
Model iPhone 5 (Model A1429)
Serial Number [redacted]
ECID [redacted]

3. Sign in with the Apple ID associated with your iOS Developer Program membership and Xcode will automatically generate your certificates.
4. in Xcode, Click on "Product" -->"Destination" --> you should see the name of your iPhone list there so click on it :)
5. in Xcode ,Click "Product" --> "Build for Running" --> "Allow"
6. in Xcode ,Click "Product --> "Run" --> "Allow"

2.2 iOS Simulator

The iOS **Simulator** included in the SDK (example of an iOS 6) can be accessed directly from Xcode under **Xcode → Open Developer Tool → iOS Simulator** menu entry, as shown in the figures below:



moapp | Build demoapp: **Succeeded** | Yesterday at 12:31 PM

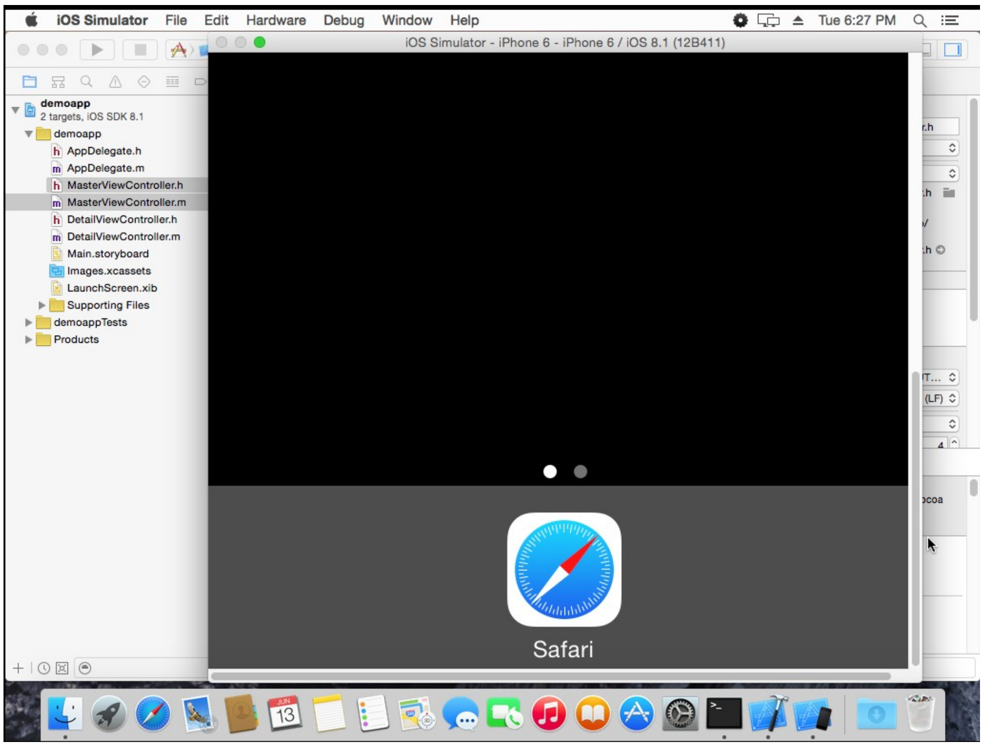
```
//
8/17.
e. All rights reserved.

@interface MasterViewController : UITableViewController

@property (strong, nonatomic) DetailViewController *
    detailViewController;

@end
```

Identity and Type
Name: MasterViewController.h
Type: Default - C Header
Location: Relative to Group
MasterViewController.h
Full Path: /Users/someone/Documents/demoapp/demoapp/MasterViewController.h

Target Membership
demoapp
demoappTests

Text Settings
Text Encoding: Default - Unicode (UT…
Line Endings: Default - OS X / Unix (LF)
Indent Using: Spaces

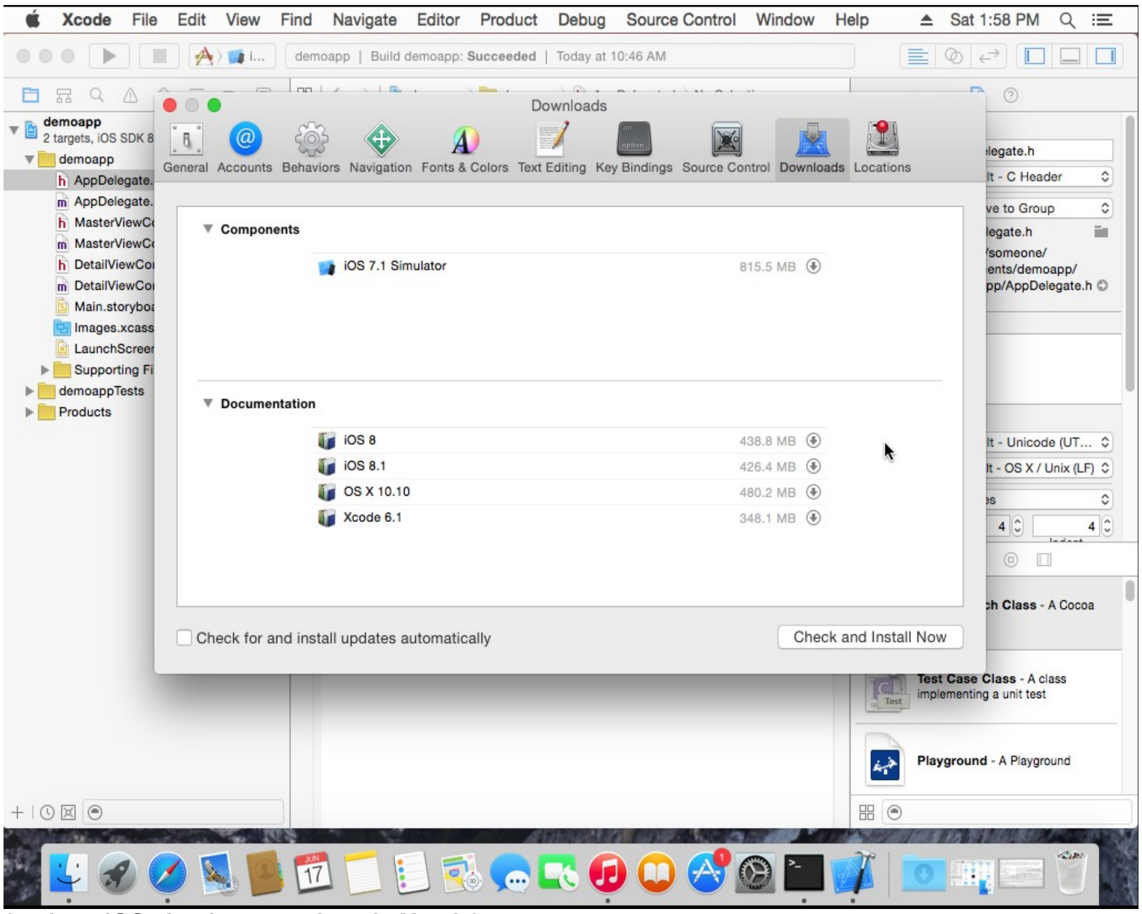
(launching iOS simulator – Xcode built in iOS 6 simulator)



(actual simulator screen – virtual iPhone with iOS 6)

Additionally, in modern versions of the OS X including the El Capitan, it can be immediately accessed from the finder by typing: /Applications/Xcode.app/Contents/Developer/Applications/Simulator.app This is automatically done when building/running a project/application in Xcode as the default output is the iOS simulator.

Later versions of iOS can be also downloaded for use as simulators from the menu entry:
Xcode → Preferences → Downloads


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Sat 1:58 PM
demoapp | Build demoapp: Succeeded | Today at 10:46 AM
Downloads
General Accounts Behaviors Navigation Fonts & Colors Text Editing Key Bindings Source Control Downloads Locations
Check for and install updates automatically
Check and Install Now

Components

iOS 7.1 Simulator	815.5 MB

Documentation

iOS 8	438.8 MB
iOS 8.1	426.4 MB
OS X 10.10	480.2 MB
Xcode 6.1	348.1 MB

(various iOS simulator versions in Xcode)

A full in-depth analysis of an iOS application requires the complete Xcode project to be delivered by the application owners. In most of the cases, this resource is not available so the use of alternative solutions based only on the application binaries are incorporated. These binary files can be downloaded from Apple's AppStore, either in an actual iPhone device or in a Macintosh OSX so that the binary can be reverse engineered. In any way, the steps covered in this chapter about the Xcode, constitute important knowledge for several situations. Some examples are the examination of a jail brake .ipa file or exporting a malicious application as a benign signed iOS application.

<u>Important note:</u> *AppStore applications* ***cannot run on iOS simulator*** *because it was built on x86 architecture instead of the application binaries in AppStore, which were compiled against ARM architecture to run on iOS devices equipped with ARM processors. This leaves only two real case scenarios of iOS applications analysis during runtime:*

a) Either analyze the application while running on an actual iPhone device through the connected Mac OSX using ssh.

b) Or obtain the application's **source code** or **.xcodeproj** file and load it into Xcode for analysis.

The second option requires the whole project to be loaded into Xcode and this will be explained on the next chapter. The rest of the analysis is dedicated to the former and most used option that requires to reverse engineer the application and dynamically analyze it during runtime. Similar to any reverse engineering process of a binary file on other platforms.

CHAPTER 3

RUN-LOAD AN APPLICATION INTO XCODE

Based on the latter option, obtaining the application's **source code** or **.xcodeproj** file and loading it into Xcode , the analysis will be conducted in this case using Xcode. The application can be loaded into Xcode by pasting its whole folder (see the figure below of the root directory of Xcode project and its contents) under the following directory:

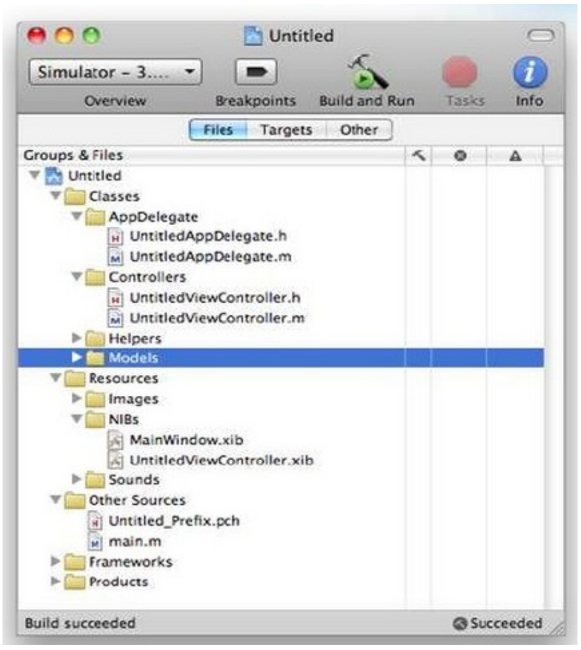
/Users/<username>/Library/Application Support/iPhone Simulator/<iOS version e.g. 3.2 (iPad) or 4.0.1 (iPhone)>/Applications/<folder with unique application id>

This action will also make the specific project tree available on Xcode's home screen as a recent/opened project. Once again, this option will probably only be used if the application owners request a source code auditing service.

Solutions for other cases that the Xcode project will not be available are:

- Decompile any application from the AppStore (this process will never reach the initial source code). Then work with the product assembly code for a static code analysis investigating on possible issues such as buffer overflows (iOS applications are written in objective C), method calls, data access and more using **static code** analyzers.
- Analyze Memory dumps, system-kernel calls, logs and method calls like obj_MsgSend() by **dynamically analyzing** the application during runtime.

These options will be discussed later. Keep in mind that application packages downloaded from the AppStore as ***.ipa*** files (*refer to* ***Technical Terms*** *section for further details on this*) are encrypted. There are ways of bypassing this obstacle like the Clutch tool discussed in the ***Decompilation*** section. The general application structure as an Xcode project is listed below (*this will not be further explained as it is considered beyond the scope of this dissertation*):



iOS application structure as an Xcode project

CHAPTER 4

DECOMPILING iOS APPLICATION

Working with applications installed on an iOS device, which are already compiled and no Xcode project is available, locating the installation directory is required to begin the decompilation process. Based on the fact that they are either pre-installed or downloaded from the AppStore on the iOS device, the binaries will be located in two different directories respectively:

- */Applications (pre-installed applications)*
- */var/mobile/Applications (downloaded from the AppStore)*

The demo application project "demoapp" of Xcode is going to be used as an example for the reverse engineering process. After being built in Xcode, its binary file is available. The main application file demoapp.app (every application contains its main file on a name convention of application_name.app) will be required for the tools used and it resides under:

```
Users/<username>/Library/Developer/Xcode/DerivedData/demoapp-frwksdfj…..d<random-name>/Build/Products/Debug-iphonesimulator/demoapp.app
```

This directory indicates that Xcode built the project to run against the iPhone simulator. Now that the application's binary is available too, the reversing process can begin to identify what kind of information can be gained from the initial source code of demoapp, pretending that the initial source code was not available in the first place. Due to the fact that this process is extensive and crucial enough, several tools are going to be presented.

The basic tools for the decompilation process are presented below and both demoapp.app and third-party applications are going to be used as examples. Third party applications are used for the decryption process because, as already stated, AppStore .ipa files are encrypted.

4.1 otool

Otool comes with Xcode and is directly available on OSX console too (like other Xcode console features). It is a powerful disassembling tool that provides information of object files or libraries. It can also provide a compiled application's assembly code **[9]**. All the options can be found with the command otool –help. In order to acquire an application's assembly dump file, the following terminal command is used:

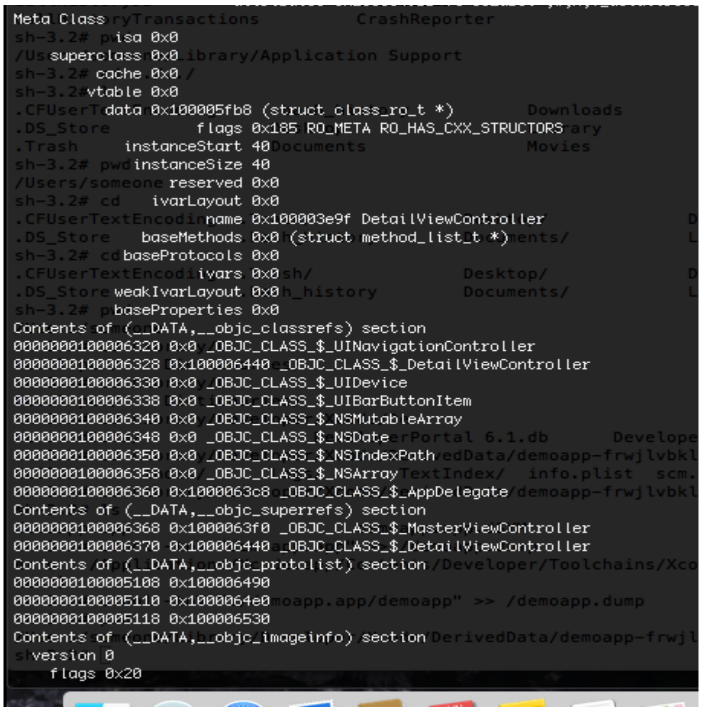
```
#otool -options "dir to .app/appname" >> output file
```

appname is used for the basic class of the application.

Regarding the demoapp, the command is going to be:

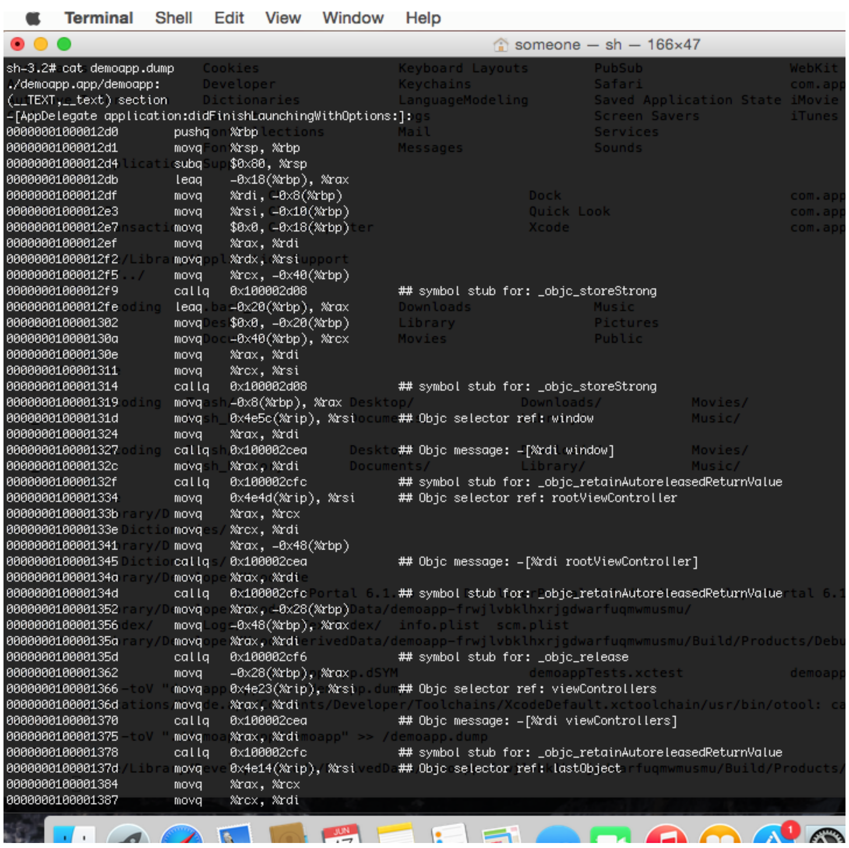
```
# otool -toV /Users/<username>/Library/Developer/Xcode/DerivedData/demoapp-frw…..kd<random name>/Build/ProductsDebug-iphonesimulator/demoapp.app/demoapp" >> /demoapp.dump
```

The expected output is then available under the root directory and some parts of the extracted assembly of the demoapp (beginning and end of the whole assembly code using head and tail commands) are listed below:



```
Meta Class
   isa 0x0
   superclass 0x0
   cache 0x0
   vtable 0x0
   data 0x100005fb8 (struct class_ro_t *)
            flags 0x185 RO_META RO_HAS_CXX_STRUCTORS
      instanceStart 40
      instanceSize 40
         reserved 0x0
       ivarLayout 0x0
            name 0x100003e9f DetailViewController
      baseMethods 0x0 (struct method_list_t *)
    baseProtocols 0x0
            ivars 0x0
    weakIvarLayout 0x0
    baseProperties 0x0
Contents of (__DATA,__objc_classrefs) section
0000000100006320 0x0 _OBJC_CLASS_$_UINavigationController
0000000100006328 0x100006440 _OBJC_CLASS_$_DetailViewController
0000000100006330 0x0 _OBJC_CLASS_$_UIDevice
0000000100006338 0x0 _OBJC_CLASS_$_UIBarButtonItem
0000000100006340 0x0 _OBJC_CLASS_$_NSMutableArray
0000000100006348 0x0 _OBJC_CLASS_$_NSDate
0000000100006350 0x0 _OBJC_CLASS_$_NSIndexPath
0000000100006358 0x0 _OBJC_CLASS_$_NSArray
0000000100006360 0x1000063c8 _OBJC_CLASS_$_AppDelegate
Contents of (__DATA,__objc_superrefs) section
0000000100006368 0x1000063f0 _OBJC_CLASS_$_MasterViewController
0000000100006370 0x100006440 _OBJC_CLASS_$_DetailViewController
Contents of (__DATA,__objc_protolist) section
0000000100005108 0x100006490
0000000100005110 0x1000064e0
0000000100005118 0x100006530
Contents of (__DATA,__objc_imageinfo) section
  version 0
    flags 0x20
```

(assembly product of otool 1)



```
sh-3.2# cat demoapp.dump
./demoapp.app/demoapp:
(__TEXT,__text) section
-[AppDelegate application:didFinishLaunchingWithOptions:]:
00000001000012d0        pushq   %rbp
00000001000012d1        movq    %rsp, %rbp
00000001000012d4        subq    $0x80, %rsp
00000001000012db        leaq    -0x18(%rbp), %rax
00000001000012df        movq    %rdi, -0x8(%rbp)
00000001000012e3        movq    %rsi, -0x10(%rbp)
00000001000012e7        movq    $0x0, -0x18(%rbp)
00000001000012ef        movq    %rax, %rdi
00000001000012f2        movq    %rdx, %rsi
00000001000012f5        movq    %rcx, -0x40(%rbp)
00000001000012f9        callq   0x100002d08             ## symbol stub for: _objc_storeStrong
00000001000012fe        leaq    -0x20(%rbp), %rax
0000000100001302        movq    $0x0, -0x20(%rbp)
000000010000130a        movq    -0x40(%rbp), %rcx
000000010000130e        movq    %rax, %rdi
0000000100001311        movq    %rcx, %rsi
0000000100001314        callq   0x100002d08             ## symbol stub for: _objc_storeStrong
0000000100001319        movq    -0x8(%rbp), %rax
000000010000131d        movq    0x4e5c(%rip), %rsi      ## Objc selector ref: window
0000000100001324        movq    %rax, %rdi
0000000100001327        callq   0x100002cea             ## Objc message: -[%rdi window]
000000010000132c        movq    %rax, %rdi
000000010000132f        callq   0x100002cfc             ## symbol stub for: _objc_retainAutoreleasedReturnValue
0000000100001334        movq    0x4e4d(%rip), %rsi      ## Objc selector ref: rootViewController
000000010000133b        movq    %rax, %rcx
000000010000133e        movq    %rcx, %rdi
0000000100001341        movq    %rax, -0x48(%rbp)
0000000100001345        callq   0x100002cea             ## Objc message: -[%rdi rootViewController]
000000010000134a        movq    %rax, %rdi
000000010000134d        callq   0x100002cfc             ## symbol stub for: _objc_retainAutoreleasedReturnValue
0000000100001352        movq    %rax, -0x28(%rbp)
0000000100001356        movq    -0x48(%rbp), %rax
000000010000135a        movq    %rax, %rdi
000000010000135d        callq   0x100002cf6             ## symbol stub for: _objc_release
0000000100001362        movq    -0x28(%rbp), %rax
0000000100001366        movq    0x4e23(%rip), %rsi      ## Objc selector ref: viewControllers
000000010000136d        movq    %rax, %rdi
0000000100001370        callq   0x100002cea             ## Objc message: -[%rdi viewControllers]
0000000100001375        movq    %rax, %rdi
0000000100001378        callq   0x100002cfc             ## symbol stub for: _objc_retainAutoreleasedReturnValue
000000010000137d        movq    0x4e14(%rip), %rsi      ## Objc selector ref: lastObject
0000000100001384        movq    %rax, %rcx
0000000100001387        movq    %rcx, %rdi
```

(assembly product of otool 2)

4.2 class-dump

This tool provides easily readable information of class declarations and structs. It is similar to the strings command output providing human readable strings, or to the output of “otool -ov” command except that the information is presented as Objective-C declarations with this tool **[10]**.

It can be downloaded from “http://stevenygard.com/projects/class-dump/”. Then using the bash in a terminal, the archive must be extracted and finally be used against the demoapp with the commands:

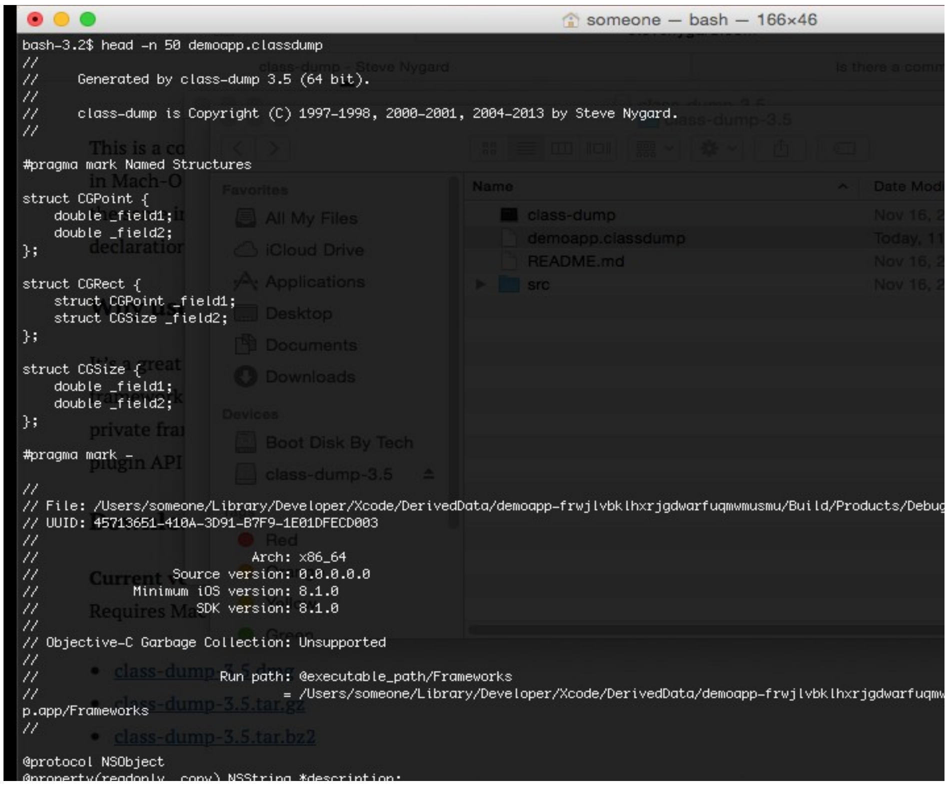
>macbook$ bash

>bash-3.2$ tar -xvf ~/Downloads/class-dump-3.5.tar

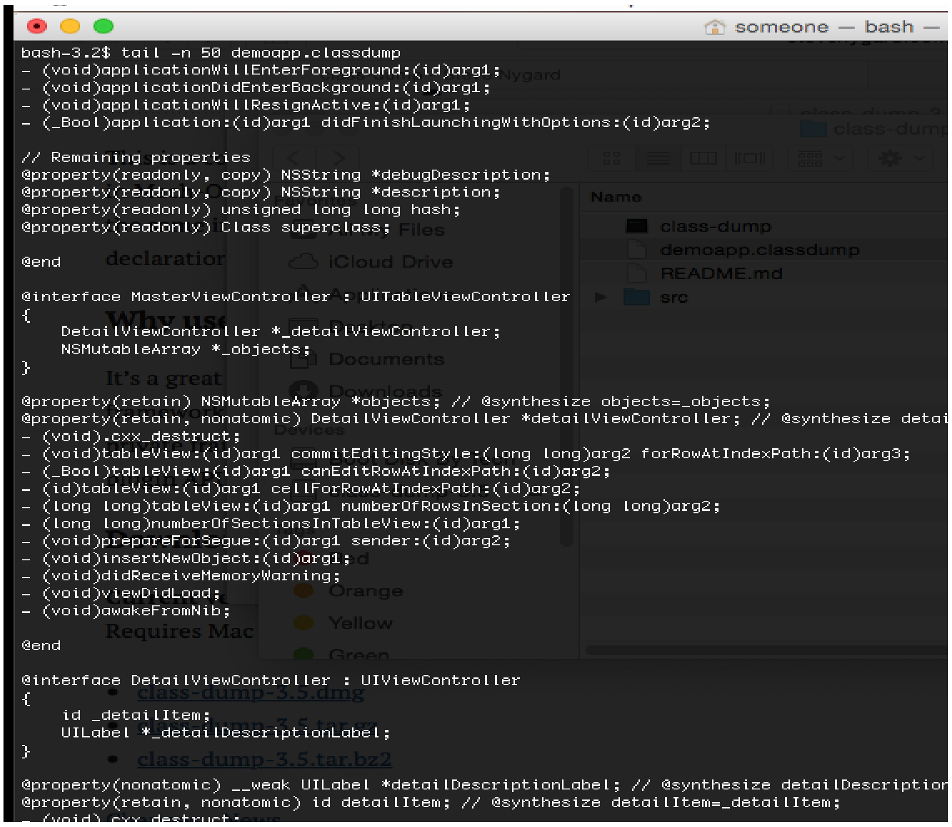
>bash-3.2$ ./class-dump “/Users/username/Library/Developer/Xcode/DerivedData/demoapp-fwrm………/Build/Products/Debug-iphonesimulator/demoapp.app” >> demoapp.classdump

*(general command is **./class-dump “dir to the .app file” >> output.file**)*

Valuable information can be gathered by using this tool like the use of a method and its arguments (it may be a vulnerable one) or the private Frameworks that the application may uses. The class-dumb output file of the demoapp is shown in the picture below:



```
bash-3.2$ head -n 50 demoapp.classdump
//
//     Generated by class-dump 3.5 (64 bit).
//
//     class-dump is Copyright (C) 1997-1998, 2000-2001, 2004-2013 by Steve Nygard.
//

#pragma mark Named Structures

struct CGPoint {
    double _field1;
    double _field2;
};

struct CGRect {
    struct CGPoint _field1;
    struct CGSize _field2;
};

struct CGSize {
    double _field1;
    double _field2;
};

#pragma mark -

//
// File: /Users/someone/Library/Developer/Xcode/DerivedData/demoapp-frwjlvbklhxrjgdwarfuqmwmusmu/Build/Products/Debug
// UUID: 45713651-410A-3D91-B7F9-1E01DFECD003
//
//                         Arch: x86_64
//               Source version: 0.0.0.0.0
//          Minimum iOS version: 8.1.0
//                  SDK version: 8.1.0
//
// Objective-C Garbage Collection: Unsupported
//
//                     Run path: @executable_path/Frameworks
//                             = /Users/someone/Library/Developer/Xcode/DerivedData/demoapp-frwjlvbklhxrjgdwarfuqmw
p.app/Frameworks
//

@protocol NSObject
@property(readonly, copy) NSString *description;
```



```
bash-3.2$ tail -n 50 demoapp.classdump
- (void)applicationWillEnterForeground:(id)arg1;
- (void)applicationDidEnterBackground:(id)arg1;
- (void)applicationWillResignActive:(id)arg1;
- (_Bool)application:(id)arg1 didFinishLaunchingWithOptions:(id)arg2;

// Remaining properties
@property(readonly, copy) NSString *debugDescription;
@property(readonly, copy) NSString *description;
@property(readonly) unsigned long long hash;
@property(readonly) Class superclass;

@end

@interface MasterViewController : UITableViewController
{
    DetailViewController *_detailViewController;
    NSMutableArray *_objects;
}

@property(retain) NSMutableArray *objects; // @synthesize objects=_objects;
@property(retain, nonatomic) DetailViewController *detailViewController; // @synthesize detai
- (void).cxx_destruct;
- (void)tableView:(id)arg1 commitEditingStyle:(long long)arg2 forRowAtIndexPath:(id)arg3;
- (_Bool)tableView:(id)arg1 canEditRowAtIndexPath:(id)arg2;
- (id)tableView:(id)arg1 cellForRowAtIndexPath:(id)arg2;
- (long long)tableView:(id)arg1 numberOfRowsInSection:(long long)arg2;
- (long long)numberOfSectionsInTableView:(id)arg1;
- (void)prepareForSegue:(id)arg1 sender:(id)arg2;
- (void)insertNewObject:(id)arg1;
- (void)didReceiveMemoryWarning;
- (void)viewDidLoad;
- (void)awakeFromNib;

@end

@interface DetailViewController : UIViewController
{
    id _detailItem;
    UILabel *_detailDescriptionLabel;
}

@property(nonatomic) __weak UILabel *detailDescriptionLabel; // @synthesize detailDescription
@property(retain, nonatomic) id detailItem; // @synthesize detailItem=_detailItem;
```

(first and last lines of classdump output for demoapp)

4.3 Clutch

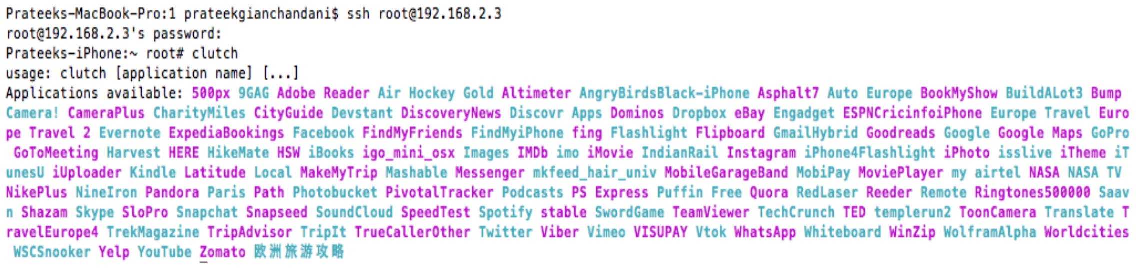
Most of the time, the analyzed application is going to be installed on an actual iOS device. These applications are either pre-installed on the device or downloaded from the AppStore. Two main things need to be clear now:

1. iOS applications are located either under ***/Applications/*** *(preinstalled)* or under ***/var/mobile/Applications/*** *(downloaded)* directories on the actual iOS devices.
2. Unlike pre-installed applications, the downloaded ones are **encrypted**, meaning that **decryption** is required before analyzing them.

The **decryption** process can be done using this specific tool called **Clucth**. It was once offered by *Hackulous* but now is only available online here:

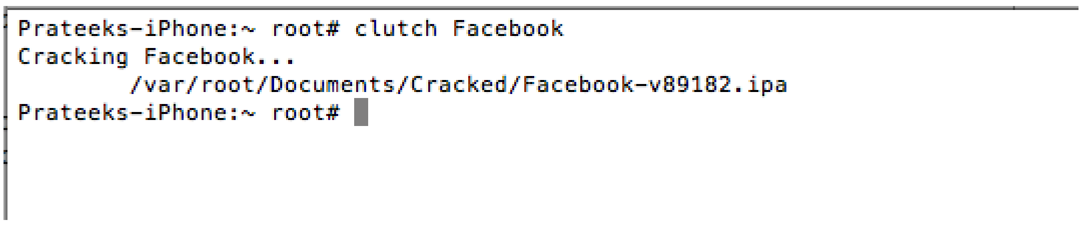
https://github.com/KJCracks/Clutch

After the tool is downloaded, it has to be uploaded to the device, using **sftp**, with the **put** command to the **/usr/bin/** directory. Then, while connecting to the device using ssh, the tool could be used on any installed application. The following pictures show the tool options listed from an iPhone device and the process of decryption:



```
Prateeks-MacBook-Pro:1 prateekgianchandani$ ssh root@192.168.2.3
root@192.168.2.3's password:
Prateeks-iPhone:~ root# clutch
usage: clutch [application name] [...]
Applications available: 500px 9GAG Adobe Reader Air Hockey Gold Altimeter AngryBirdsBlack-iPhone Asphalt7 Auto Europe BookMyShow BuildALot3 Bump
Camera! CameraPlus CharityMiles CityGuide Devstant DiscoveryNews Discovr Apps Dominos Dropbox eBay Engadget ESPNCricinfoiPhone Europe Travel Euro
pe Travel 2 Evernote ExpediaBookings Facebook FindMyFriends FindMyiPhone fing Flashlight Flipboard GmailHybrid Goodreads Google Google Maps GoPro
 GoToMeeting Harvest HERE HikeMate HSW iBooks igo_mini_osx Images IMDb imo iMovie IndianRail Instagram iPhone4Flashlight iPhoto isslive iTheme iT
unesU iUploader Kindle Latitude Local MakeMyTrip Mashable Messenger mkfeed_hair_univ MobileGarageBand MobiPay MoviePlayer my airtel NASA NASA TV
NikePlus NineIron Pandora Paris Path Photobucket PivotalTracker Podcasts PS Express Puffin Free Quora RedLaser Reeder Remote Ringtones500000 Saav
n Shazam Skype SloPro Snapchat Snapseed SoundCloud SpeedTest Spotify stable SwordGame TeamViewer TechCrunch TED templerun2 ToonCamera Translate T
ravelEurope4 TrekMagazine TripAdvisor TripIt TrueCallerOther Twitter Viber Vimeo VISUPAY Vtok WhatsApp Whiteboard WinZip WolframAlpha Worldcities
 WSCSnooker Yelp YouTube Zomato 欧洲旅游攻略
```

clutch options (within iPhone over ssh [11])



```
Prateeks-iPhone:~ root# clutch Facebook
Cracking Facebook...
        /var/root/Documents/Cracked/Facebook-v89182.ipa
Prateeks-iPhone:~ root#
```

decrypting Facebook application with clutch

After the tool completes the cracking process, it creates an .ipa file of the whole application bundle and normally saves it under **/var/root/Documents/Cracked/** directory. After decompressing the .ipa file using the untar command, the application files are available. This extracted directory contains used images, the plist file (property listing) and everything else that an unencrypted .ipa file contains. The class-dump tool can also be used now against the Facebook main application file with:

class-dump dir-to-extracted-ipa-file/Payload/Facebook.app/Facebook > Facebook-class

All the class information is accessible as if the application was not encrypted in the first place. This process demonstrates that even when dealing with an encrypted application from the AppStore, information can still be extracted using the aforementioned and the rest of the tools described in the document.

4.4 Dumpdecrypted (running application decryption)

This specific tool is used for application binaries installed on a device (that are of course encrypted) to generate a decrypted version of them for lateral analysis using for example the class-dump tool. It is available on the following link:

https://github.com/stefanesser/dumpdecrypted/archive/master.zip

The installation and usage of the tool to decrypt a third-party application starts by downloading the archive and decompressing it as usual and then using the "**make**" command to build it. Based on the **iOS** version of the device, the specific option in the "**Makefile**" must be adjusted before building the project:

Mac:~ user$ cd /path-to-decompressed-project/dumpdecrypted-master/

Mac: dumpdecrypted user$ make

Now a specific file generated under the current project's directory called "**dumpdecrypted.dylib**" (on the MAC OSX) is required along with the application's executable directory on the iOS device and the application's Document directory in which it must also be saved (on the iOS device). To make things clear here, the generated file is a dynamic library file (.dylib) that will be used in the sandbox directory of the analyzed application.

In order to locate the two directories, the analyzed application must be running on the iPhone device (kill all other instances from background) and then by using the "ps" command the following output is shown:

```
Mac:~ user$ ssh root@yourIP
iphone-5:~ root# ps -e
  PID TTY           TIME CMD
    1 ??         3:28.32 /sbin/launchd
……
5717 ??         0:00.21
/System/Library/PrivateFrameworks/MediaServices.framework/Support/mediaartworkd
 5905 ??        0:00.20 sshd: root@ttys000
 5909 ??        0:01.86 /var/mobile/Containers/Bundle/Application/03B61840-2349-4559-B28E-
0E2C6541F879/TargetApp.app/TargetApp
 5911 ??        0:00.07 /System/Library/Frameworks/UIKit.framework/Support/pasteboardd
 5907 ttys000    0:00.03 -sh
 5913 ttys000    0:00.01 ps –e
```

Since only the TargetApp is running on the device, the directory to TargetApp's executable is the one listed below:

"/var/mobile/Containers/Bundle/Application/03B61840-2349-4559-B28E-

Now in order to locate TargetApp's **Documents** directory too, Cycript must be used as follows from the ssh shell to the iPhone device:

```
iphone-5:~ root# cycript -p TargetApp
cy# [[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask][0]
#"file:///var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-
2146128611EE/Documents
```

and this is where the "dumpdecrypted.dylib" file must be copied to using the "scp" command:

```
Mac:~ user$ scp /Users/user/Code/dumpdecrypted/dumpdecrypted.dylib
root@192.168.2.2:/var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-
2146128611EE/Documents/dumpdecrypted.dylib
```

The reason why dumpdecrypted.dylib file must be copied under the analyzed application's Documents directory is because a running iOS application does not have the right to write outside of the Sandbox (all applications run Sand boxed within the iOS, acting only in their own context for security purposes). Additionally, dumpdecrypted runs with the same privileges as the running analyzed application and in order to write the final decrypted file, the only directory it could write to would be the application's Documents directory. This is why it is placed in the Documents directory.

The decryption process can start by using two commands on the iPhone device. The first one is "DYLD_INSERT_LIBRARIES" and it sets the settings of where is the analyzed application's Documents directory containing the .dylib file and where its executable file resides. The second one simply runs the tool against the application:

```
iphone-5:~ root# cd /var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-
2146128611EE/Documents/
iphone-5:/var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-
2146128611EE/Documents root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib
/var/mobile/Containers/Bundle/Application/03B61840-2349-4559-B28E-
0E2C6541F879/TargetApp.app/TargetApp
mach-o decryption dumper
```

```
DISCLAIMER: This tool is only meant for security research purposes, not for application crackers.

[+] detected 32bit ARM binary in memory.
[+] offset to cryptid found: @0x81a78(from 0x81000) = a78
[+] Found encrypted data at address 00004000 of length 6569984 bytes - type 1.
[+] Opening /private/var/mobile/Containers/Bundle/Application/03B61840-2349-4559-B28E-0E2C6541F879/TargetApp.app/TargetApp for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a plain MACH-O image
[+] Opening TargetApp.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Setting the LC_ENCRYPTION_INFO->cryptid to 0 at offset a78
[+] Closing original file
[+] Closing dump file
```

The “ls” command reveals the final decrypted product file:

```
iphone-5:/var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-2146128611EE/Documents root# ls
TargetApp.decrypted  dumpdecrypted.dylib OtherFiles
```

All standard operations of security analysis can be implemented now on the decrypted product file, like on any other not encrypted .app file. For example, the **class-dump** tool can be used against the decrypted TargetApp’s file without issues:

```
 Class-dump --arch armv7s Target.decrypted
```

Finally, the reason why the dumpdecrypted.dylib file must be under application’s Documents directory is the Sandbox permissions. In case it is not placed in application’s Documents directory but under the tmp directory for instance, by running the tool (in the figure below), an erno=1 occurs, meaning that this operation is not permitted (expected outcome due to Sandbox permissions).

```
iphone-5: /var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-2146128611EE/Documents root# mv dumpdecrypted.dylib /var/tmp/
iphone-5: /var/mobile/Containers/Data/Application/D41C4343-63AA-4BFF-904B-2146128611EE/Documents root# cd /var/tmp
iphone-5:/var/tmp root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /private/var/mobile/Containers/Bundle/Application/03B61840-2349-4559-B28E-0E2C6541F879/TargetApp.app/TargetApp
dyld: could not load inserted library 'dumpdecrypted.dylib' because no suitable image found.  Did find:
	dumpdecrypted.dylib: stat() failed with errno=1

Trace/BPT trap: 5
```

Taking all the above under consideration, dumpdecrypted is the tool that makes every iOS application available for analysis as if it was pre-installed and unencrypted or its Xcode project was available in the first place. The next chapter refers to the code analysis process (static or dynamic), that can now be easily done as far as the application binary can be decrypted and used for this purpose. More details on the described process and general usage of the dumpdecrypted tool can be found on the “Mobile Application Penetration Testing” **[12]** book.

Chapter 5

STATIC CODE ANALYSIS

Static code analysis is performed in most cases on the source code or the object code of an application. In the previous chapter, the way of decompiling an application and having the object or assembly code for static analysis was introduced. If the initial source code is available too then this type of analysis brings the best possible results.

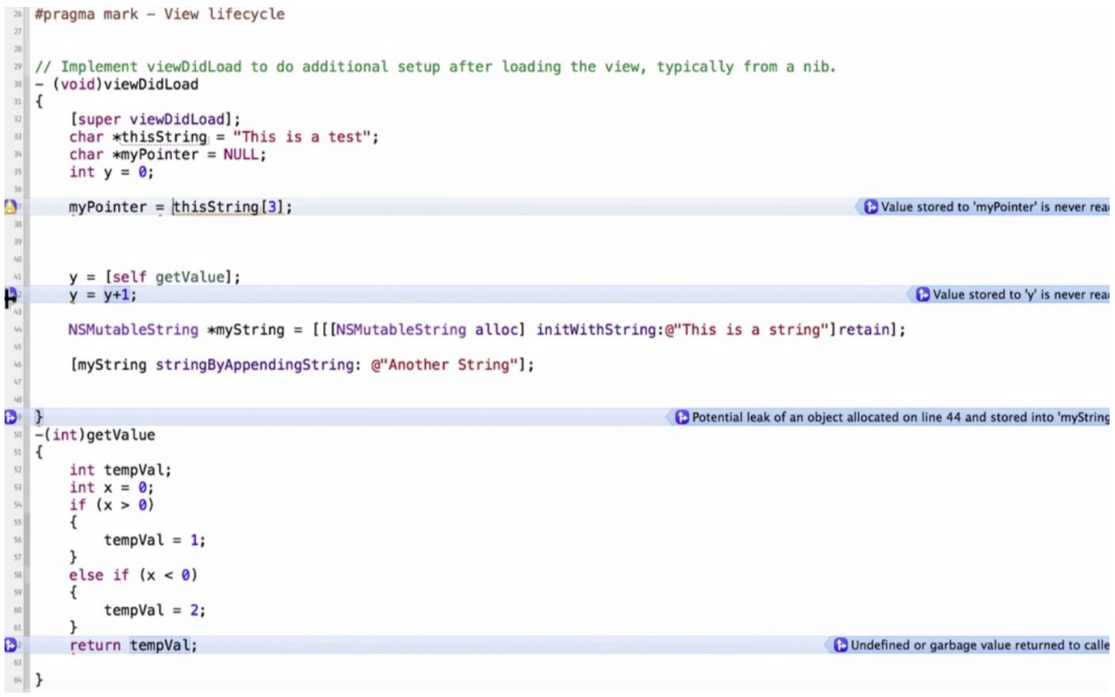
Static analysis can be leveraged to uncover issues such as memory leaks, uninitialized variables, dead code, type mismatch and buffer overflows among others. Should the **source code** for the application is available, this can be done by using **Xcode**. The static analyzer travels down each possible code path identifying logical errors such as memory leaks. Within the IDE this is performed by using ***Project > Analyze*** that brings up the **Xcode Analyzer.**

5.1 Xcode Analyzer

iOS applications may usually include/use native C language libraries such as strcpy. This gives the option to use the free tool "**Flawfinder"** as soon as the application uses the aforementioned C libraries. If it does not use them and depends on **Cocoa objects** such as **nsstring,** the use of **Clang** free tool is also an option, which is already located under ***/Applications/Xcode.app/Contents/Developer/usr/bin/clang***.

The static analyzer used by Xcode can be changed with another one from command line tools of the IDE by the following procedure:

https://clang-analyzer.llvm.org/xcode.html **[17]**
https://lowlevelbits.org/getting-started-with-llvm/clang-on-os-x/ **[18]**



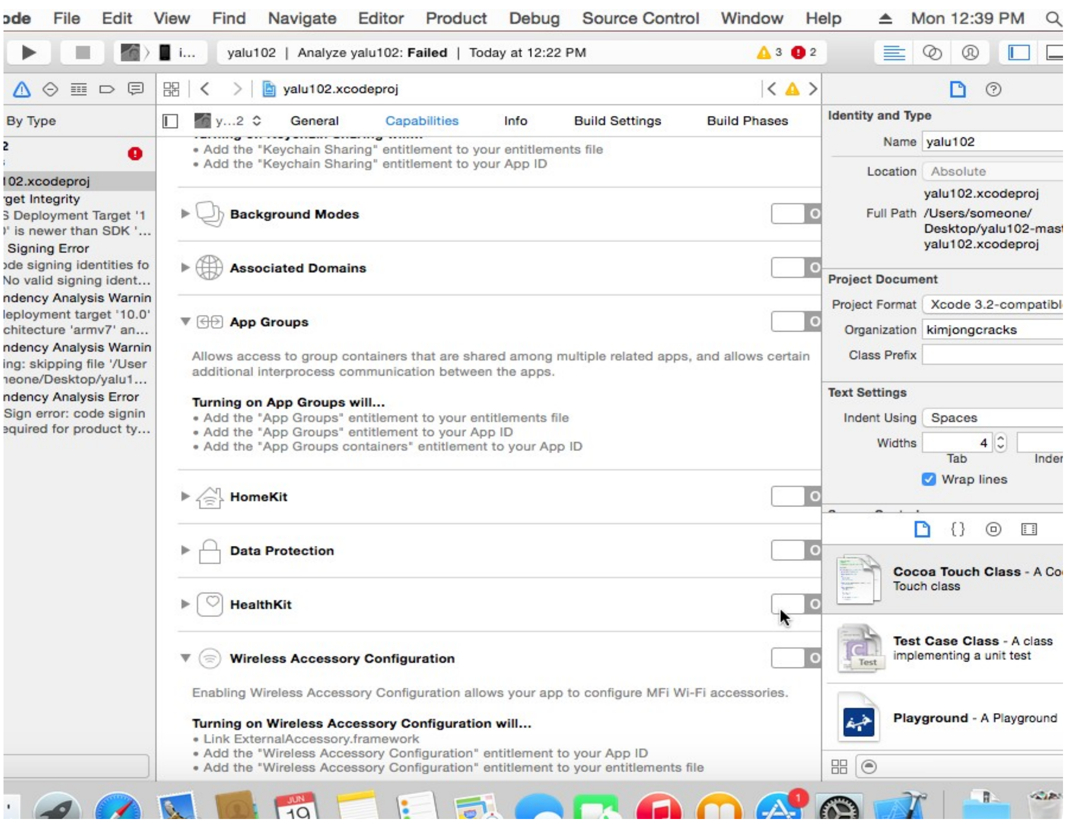
```
26  #pragma mark - View lifecycle
27
28
29  // Implement viewDidLoad to do additional setup after loading the view, typically from a nib.
30  - (void)viewDidLoad
31  {
32      [super viewDidLoad];
33      char *thisString = "This is a test";
34      char *myPointer = NULL;
35      int y = 0;
36
37      myPointer = thisString[3];                                   Value stored to 'myPointer' is never rea
38
39
40
41      y = [self getValue];
42      y = y+1;                                                     Value stored to 'y' is never rea
43
44      NSMutableString *myString = [[[NSMutableString alloc] initWithString:@"This is a string"]retain];
45
46      [myString stringByAppendingString: @"Another String"];
47
48
49  }                                            Potential leak of an object allocated on line 44 and stored into 'myString
50  -(int)getValue
51  {
52      int tempVal;
53      int x = 0;
54      if (x > 0)
55      {
56          tempVal = 1;
57      }
58      else if (x < 0)
59      {
60          tempVal = 2;
61      }
62      return tempVal;                                              Undefined or garbage value returned to calle
63
64  }
```

(Product → Analyze, Using XCode's built in Clang static analyzer on source code)

There is also some other important information available to the "***Analyze***" option from the IDE and these are listed below (*from the left side of the project tree, the master application's node must be selected to see the following tabs*):

- what the application requires/uses from the OS (**Capabilities**)
- Configuration and architecture options (**Build settings**)
- and the most important linked libraries, compile sources, bundle resources (**Build resources**)

Check the figures below for every option's information:


ode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Mon 12:39 PM
i...
yalu102 | Analyze yalu102: **Failed** | Today at 12:22 PM
3
2
yalu102.xcodeproj
By Type
y...2
General
Capabilities
Info
Build Settings
Build Phases

2
102.xcodeproj
get Integrity
S Deployment Target '1
' is newer than SDK '...
Signing Error
de signing identities fo
No valid signing ident...
ndency Analysis Warnin
leployment target '10.0'
chitecture 'armv7' an...
ndency Analysis Warnin
ing: skipping file '/User
eone/Desktop/yalu1...
ndency Analysis Error
Sign error: code signin
equired for product ty...

- Add the "Keychain Sharing" entitlement to your entitlements file
- Add the "Keychain Sharing" entitlement to your App ID

Background Modes

Associated Domains

App Groups

Allows access to group containers that are shared among multiple related apps, and allows certain additional interprocess communication between the apps.

Turning on App Groups will...
- Add the "App Groups" entitlement to your entitlements file
- Add the "App Groups" entitlement to your App ID
- Add the "App Groups containers" entitlement to your App ID

HomeKit

Data Protection

HealthKit

Wireless Accessory Configuration

Enabling Wireless Accessory Configuration allows your app to configure MFi Wi-Fi accessories.

Turning on Wireless Accessory Configuration will...
- Link ExternalAccessory.framework
- Add the "Wireless Accessory Configuration" entitlement to your App ID
- Add the "Wireless Accessory Configuration" entitlement to your entitlements file

Identity and Type

Name	yalu102
Location	Absolute
	yalu102.xcodeproj
Full Path	/Users/someone/ Desktop/yalu102-mas yalu102.xcodeproj

Project Document

Project Format	Xcode 3.2-compatibl
Organization	kimjongcracks
Class Prefix	

Text Settings

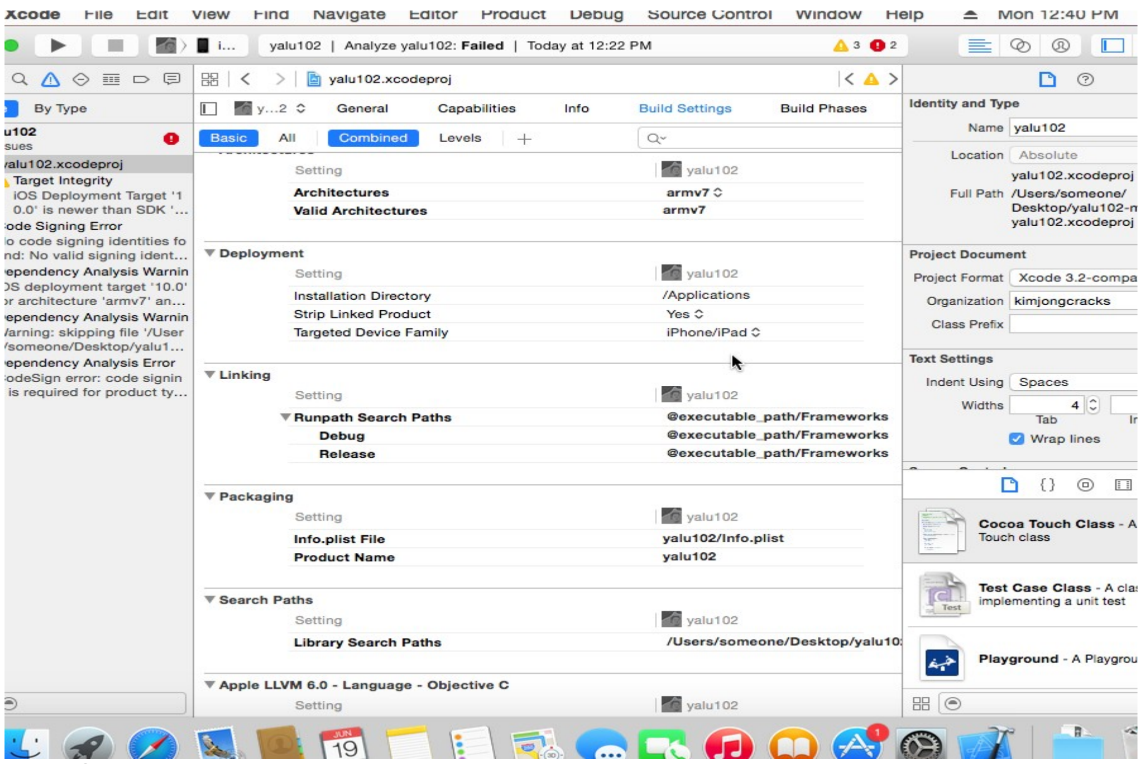
Indent Using	Spaces
Widths	4 (Tab) / (Inder)
	✓ Wrap lines

Cocoa Touch Class - A Co Touch class

Test Case Class - A class implementing a unit test

Playground - A Playground

(capabilities – Xcode → Analyze)


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Mon 12:40 PM
yalu102 | Analyze yalu102: Failed | Today at 12:22 PM
yalu102.xcodeproj
General Capabilities Info Build Settings Build Phases
Basic All Combined Levels

yalu102
Issues
yalu102.xcodeproj
Target Integrity
iOS Deployment Target '10.0' is newer than SDK '…
Code Signing Error
No code signing identities found: No valid signing ident…
Dependency Analysis Warning
iOS deployment target '10.0' for architecture 'armv7' an…
Dependency Analysis Warning
Warning: skipping file '/Users/someone/Desktop/yalu1…
Dependency Analysis Error
CodeSign error: code signing is required for product ty…

Setting	yalu102
Architectures	armv7
Valid Architectures	armv7

Deployment

Setting	yalu102
Installation Directory	/Applications
Strip Linked Product	Yes
Targeted Device Family	iPhone/iPad

Linking

Setting	yalu102
Runpath Search Paths	@executable_path/Frameworks
Debug	@executable_path/Frameworks
Release	@executable_path/Frameworks

Packaging

Setting	yalu102
Info.plist File	yalu102/Info.plist
Product Name	yalu102

Search Paths

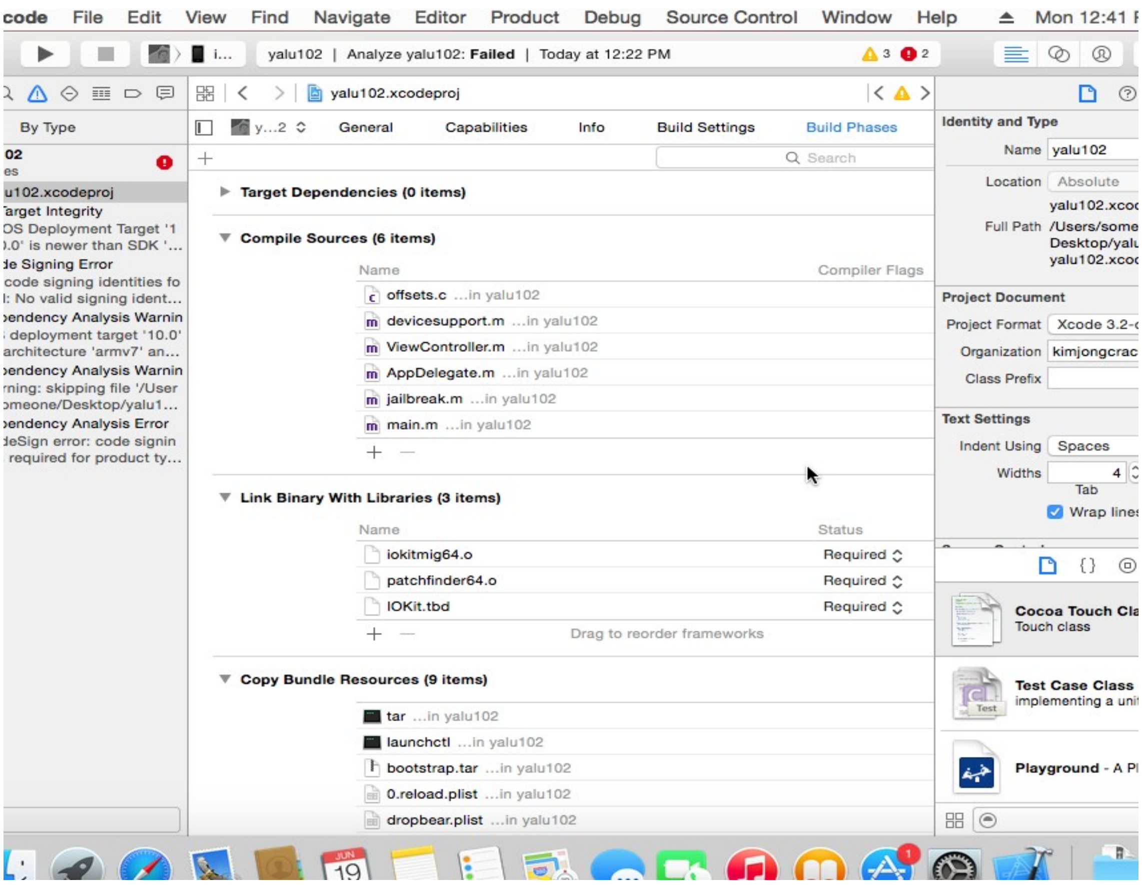
Setting	yalu102
Library Search Paths	/Users/someone/Desktop/yalu10

Apple LLVM 6.0 - Language - Objective C

Setting yalu102

Identity and Type
Name yalu102
Location Absolute
yalu102.xcodeproj
Full Path /Users/someone/Desktop/yalu102-m yalu102.xcodeproj
Project Document
Project Format Xcode 3.2-compa
Organization kimjongcracks
Class Prefix
Text Settings
Indent Using Spaces
Widths 4 Tab
Wrap lines
Cocoa Touch Class - A Touch class
Test Case Class - A class implementing a unit test
Playground - A Playground

(Build Settings – Xcode → Analyze)


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
Mon 12:41
yalu102 | Analyze yalu102: Failed | Today at 12:22 PM
yalu102.xcodeproj
General Capabilities Info Build Settings Build Phases
Search

Target Dependencies (0 items)

Compile Sources (6 items)

Name	Compiler Flags
offsets.c …in yalu102	
devicesupport.m …in yalu102	
ViewController.m …in yalu102	
AppDelegate.m …in yalu102	
jailbreak.m …in yalu102	
main.m …in yalu102	

Link Binary With Libraries (3 items)

Name	Status
iokitmig64.o	Required
patchfinder64.o	Required
IOKit.tbd	Required

Drag to reorder frameworks

Copy Bundle Resources (9 items)

tar …in yalu102
launchctl …in yalu102
bootstrap.tar …in yalu102
0.reload.plist …in yalu102
dropbear.plist …in yalu102

(Build Phases – Xcode → Analyze)

All those information, collected by static code analysis on the actual Xcode project, will not be available for the public applications from the AppStore due to the lack of project binaries or source code files. In such cases, different techniques and tools are used to find their dependencies, permissions, OS/kernel requests and interactions. A great static code analysis tool for third party applications is presented next.

5.2 iNalyzer for static analysis

The iNalyzer tool can be used for black box assessment of IOS applications. It shows class information, does runtime analysis and other things that will be presented below. Basically, it automates the tasks of decrypting the application, dumping class information and presents it in a more readable way. It can also hook into a running process and invoke methods during runtime. iNalyzer is developed and maintained by *Appsec Labs* and is also available open source **[13]**.

On the MAC OSX, *Graphviz* and *Doxygen* also need to be installed for iNalyzer to work. A jail broken iPhone device is also required in order to ssh to it as usual and retrieve the application data when they are exported by iNalyzer for further investigation in our MAC.

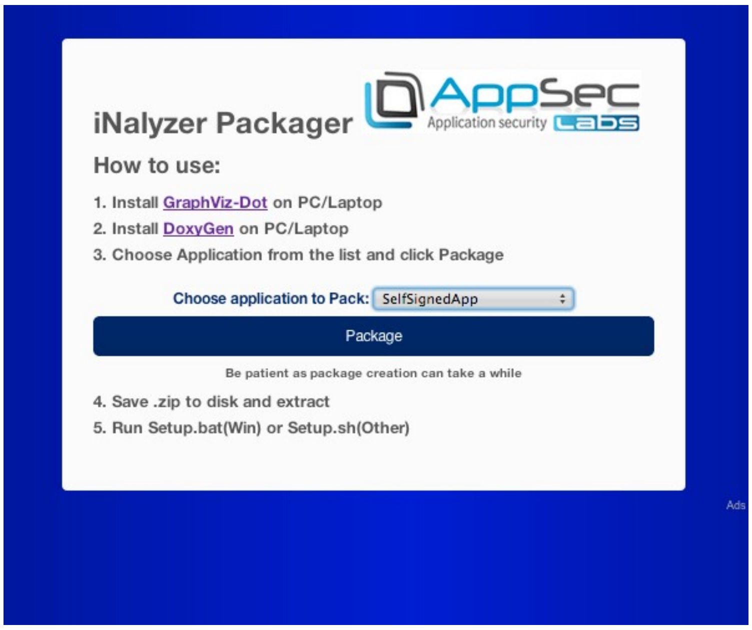
The installation and usage process consists of the following steps:

- Install iNalyzer on your jail broken iPhone device.
- ssh to the device and navigate to the iNalyzer application directory. iNalyzer is installed in the ***/Applications*** directory because it needs to run as a root.

  This is done by using the terminal commands:

  ```
  $ ssh root@<iphoneIP>
  $ cd /Applications/iNalyzerX.app/
  ```
- Run the iNalyzer

  ```
  $ ./iNalyxerX.app
  ```
- Visit the url **iphoneIP:port** from your browser (e.g. http://10.0.100.12:5443). After iNalyzer runs on the device, the port number is shown on the iPhone's screen on iNalyzer application's logo. Now a web interface is presented as shown in the figure below from where an application can be selected and iNalyzer will create a zip file and download it on your MAC for analysis.


iNalyzer Packager

How to use:

1. Install GraphViz-Dot on PC/Laptop
2. Install DoxyGen on PC/Laptop
3. Choose Application from the list and click Package

Choose application to Pack: SelfSignedApp

Package

Be patient as package creation can take a while

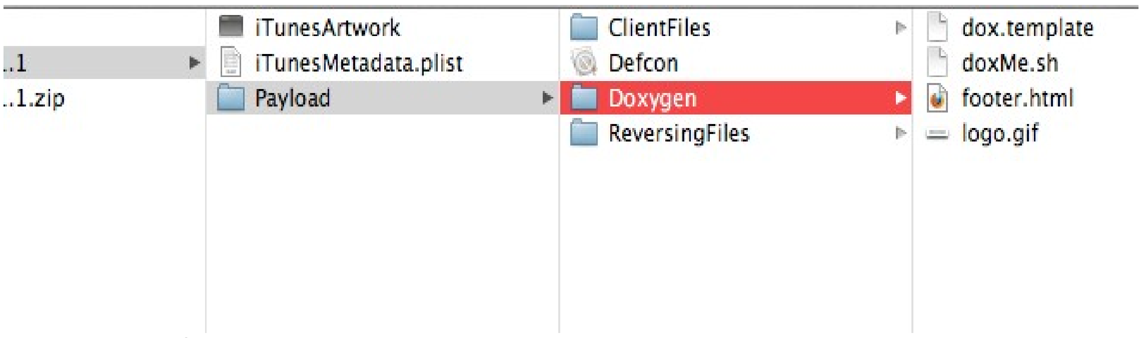
4. Save .zip to disk and extract
5. Run Setup.bat(Win) or Setup.sh(Other)

(iNalyzer web interface access from MAC)

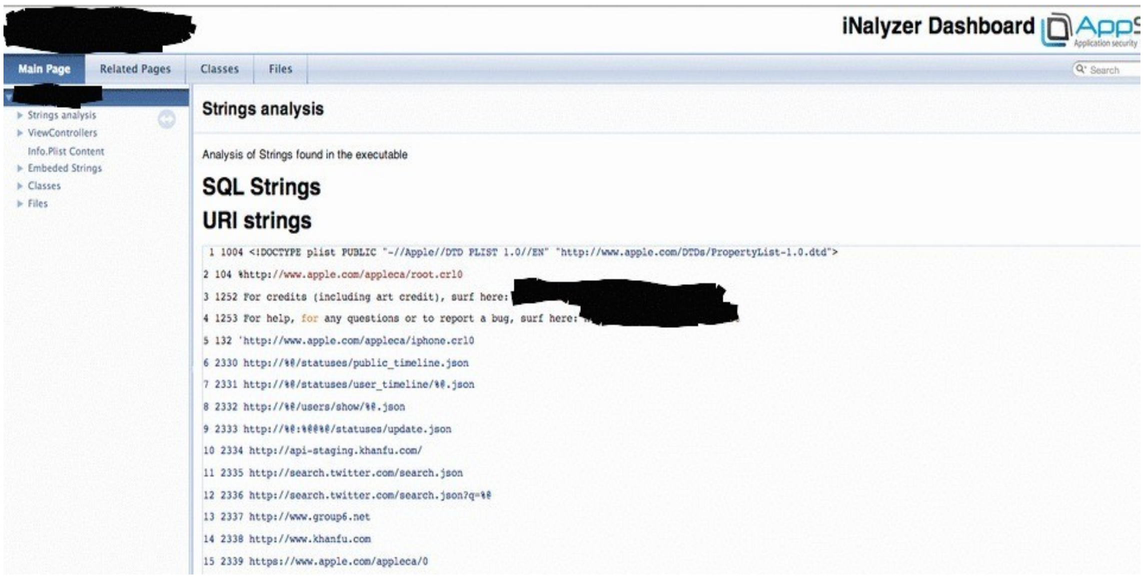
In case this option **does not work,** iNalyzer can still be used from the **command line** by using ssh access on the iPhone device, and run it with the desired application for analysis as argument (while being in **/Application/iNalyzer.app/** iPhone directory as root):

```
$ ./iNalyzerX NameOfAppToAnalyze
```

This outputs an **.ipa** file in the same directory that can easily be transferred to the MAC (using sftp) and then **renaming** it to **.zip** for extraction. The extracted folder contains a script under the directory **appname/Payload/Doxygen/doxMe.sh** which generates and finally opens an **index.html** file to represent all the gathered information using Graphviz. After running this script, the expected results are presented in the figure below **[14]**:



(.zip contents of iNalyzer containing doxME.sh script)


[redacted]

[redacted]

Strings analysis

Analysis of Strings found in the executable

SQL Strings

URI strings

```
1 1004 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
2 104 %http://www.apple.com/appleca/root.crl0
3 1252 For credits (including art credit), surf here: [redacted]
4 1253 For help, for any questions or to report a bug, surf here:
5 132 'http://www.apple.com/appleca/iphone.crl0
6 2330 http://%@/statuses/public_timeline.json
7 2331 http://%@/statuses/user_timeline/%@.json
8 2332 http://%@/users/show/%@.json
9 2333 http://%@:%@@%@/statuses/update.json
10 2334 http://api-staging.khanfu.com/
11 2335 http://search.twitter.com/search.json
12 2336 http://search.twitter.com/search.json?q=%@
13 2337 http://www.group6.net
14 2338 http://www.khanfu.com
15 2339 https://www.apple.com/appleca/0
```

(index.html file containing all the analyzed data of the chosen application, app name is hidden for security purposes)

Another way to analyze an iOS application is a dynamic analysis when it is actually running and interacting with the iOS and this is discussed on the next chapter.

Chapter 6

DYNAMIC ANALYSIS

Dynamic Analysis is the technique of assessing applications during execution. There are several tools provided by Apple for this purpose. Two of the main tools are the "**Instruments**" and "**Shark**", but Shark is left aside right now and its operations are all included within the Instruments tool. iNalyzer for black box dynamic analysis, **Dtrace** with Instruments and other tools will be presented in this chapter too.

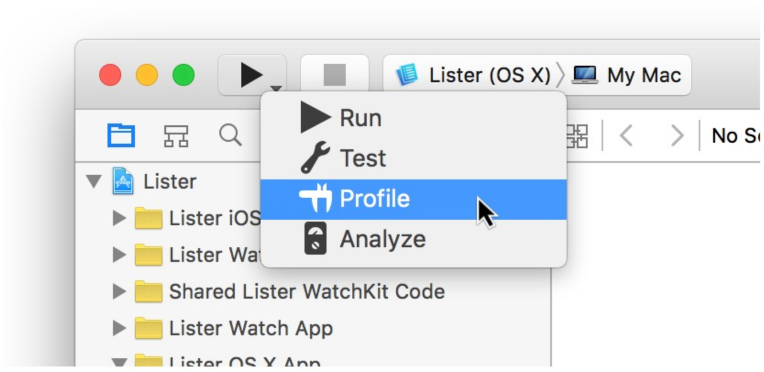
6.1 Instruments

The dynamic analysis starts with this tool that was introduced with Mac OS X v10.5. It provides a set of powerful options to assess the runtime behavior of the application **[15]**. It can be **compared** to several **SysInternals** tools used for application testing on the Microsoft Windows platform such as **procmon** and **netmon**. It can be launched from the directory **/Developer/Applications/Instruments**. Once launched, select the "Blank" template under the iPhone simulator section and then the instruments needed to use from the library. To inject this tool into a process select **Choose Target > Attach to Process > iPhone Simulator (<pid>)**. Finally, click **record** and start using the application in the simulator to generate the activity data **[16]**. The type of data being captured by the tool includes:

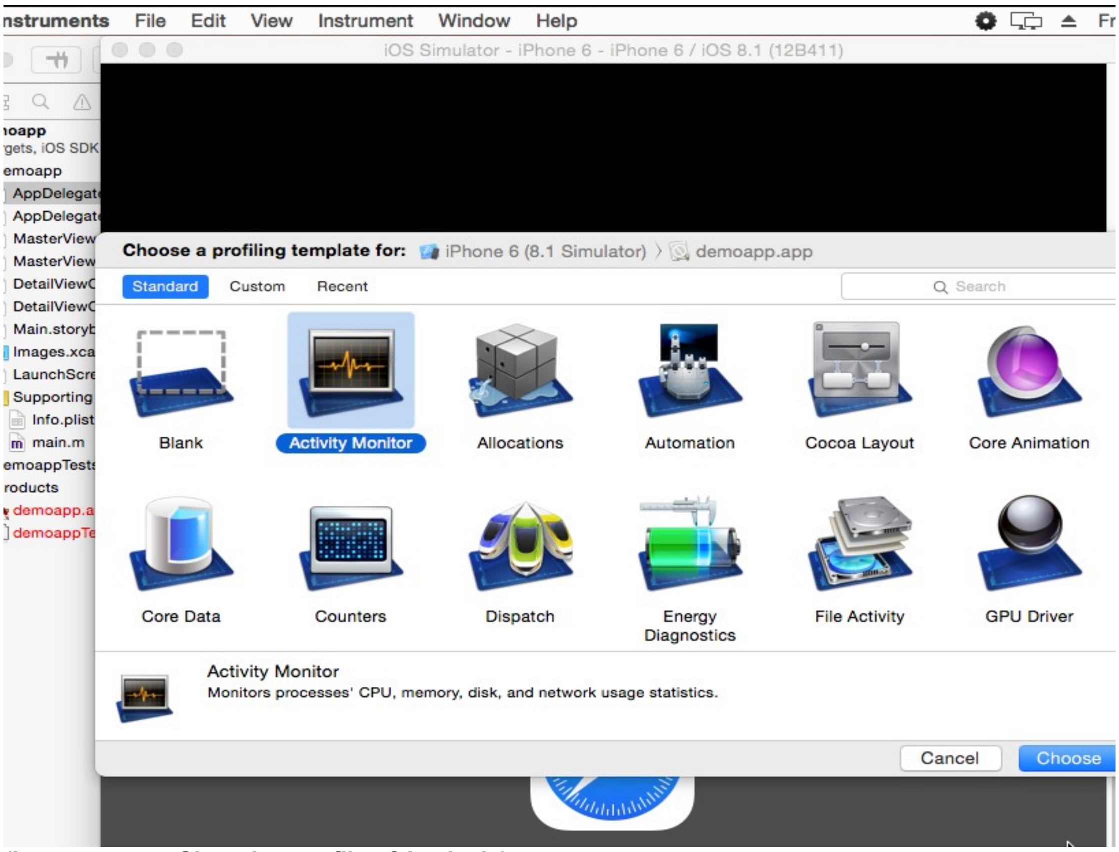
1. ***File Activity Monitoring***: This is similar to **filemon** in that it lets you identify the files generated and processed by the application. It is useful for identification of files that may be cached, or hidden files used by the application to store data on the client side.
2. ***Memory Monitoring:*** Helps to identify potential 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**".

Here are some figures showing the usage of **Instruments** with the **iPhone-simulator** while running an Xcode project compiled from the IDE.

First step is choosing the **Profile** option from the run button in Xcode:

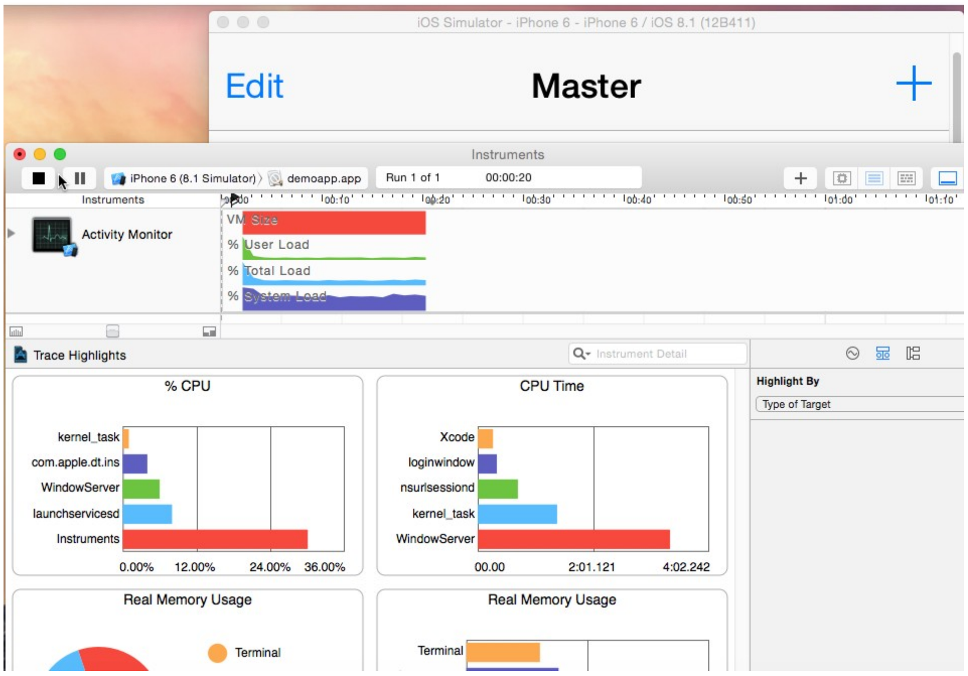


Then the monitoring options are chosen, for instance the "**Activity Monitor**" option:


Instruments File Edit View Instrument Window Help
iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.1 (12B411)
Choose a profiling template for: iPhone 6 (8.1 Simulator) › demoapp.app
Standard
Custom
Recent
Search
Blank
Activity Monitor
Allocations
Automation
Cocoa Layout
Core Animation
Core Data
Counters
Dispatch
Energy Diagnostics
File Activity
GPU Driver
Activity Monitor
Monitors processes' CPU, memory, disk, and network usage statistics.
Cancel
Choose

(Instruments – Choosing profile of Analysis)

Finally, by hitting the **record button**, the application runs into the simulator and data are simultaneously being gathered in the instruments console Graphs. In the end, all data can be **saved** when the **Stop** button is clicked:



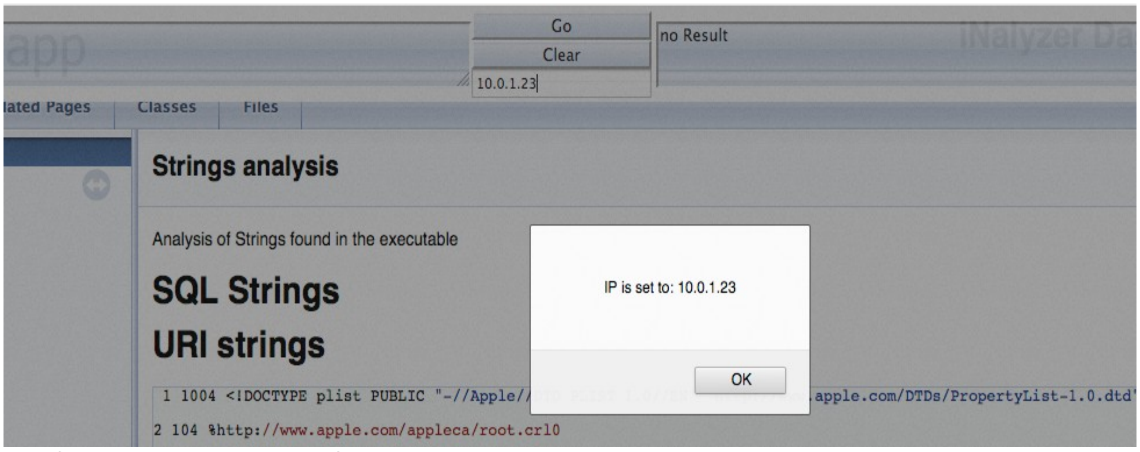
(Instruments recording "Activity Monitor" of demoapp at runtime)

6.2 iNalyzer

This tool was also introduced during static code analysis. It can be used for dynamic analysis too. **Dynamic analysis with iNalyzer** on IOS applications consists of invoking methods on runtime, finding the value of a particular instance variable at a particular time in the application and several other features.

Launching the tool's **runtime interpreter,** requires the **index.html** file generated by Doxygen to be opened (*see iNalyzer for static analysis section above*). **Firefox** is the recommended browser to work with for the runtime analysis, as suggested by the iNalyzer developer.

First of all, enter the IP address of your device (e.g. 10.0.1.23) on the box in the middle and then press enter.



Strings analysis

Analysis of Strings found in the executable

SQL Strings

URI strings

```
1 1004 <!DOCTYPE plist PUBLIC "-//Apple//                    apple.com/DTDs/PropertyList-1.0.dtd'
2 104 %http://www.apple.com/appleca/root.crl0
```

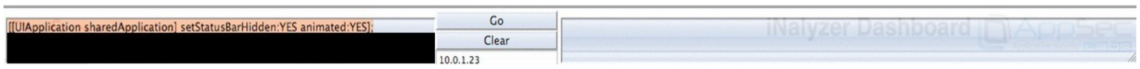
IP is set to: 10.0.1.23

OK

(IP of iPhone on index.html of iNalyzer)

Then any type of command can be typed on the left box and get the specific response on the right box. ***Important: The** analyzed application must be **running** on the iPhone preferably on the **foreground**.*

In order to hide the status bar of the app for example the command [[UIApplication sharedApplication] setStatusBarHidden:YES animated:YES]; is used on the left box. No response is received due to the nature of this method (void) but actually the status bar is hidden in the iPhone application's UI.



```
[[UIApplication sharedApplication] setStatusBarHidden:YES animated:YES];
```

It is obvious now that any function (*Cycript*) can be called on the running application by using either Objective C or JavaScript. There is a separate section dedicated to Cycript usage on iOS application later on this document under *Cycript (runtime analysis).*

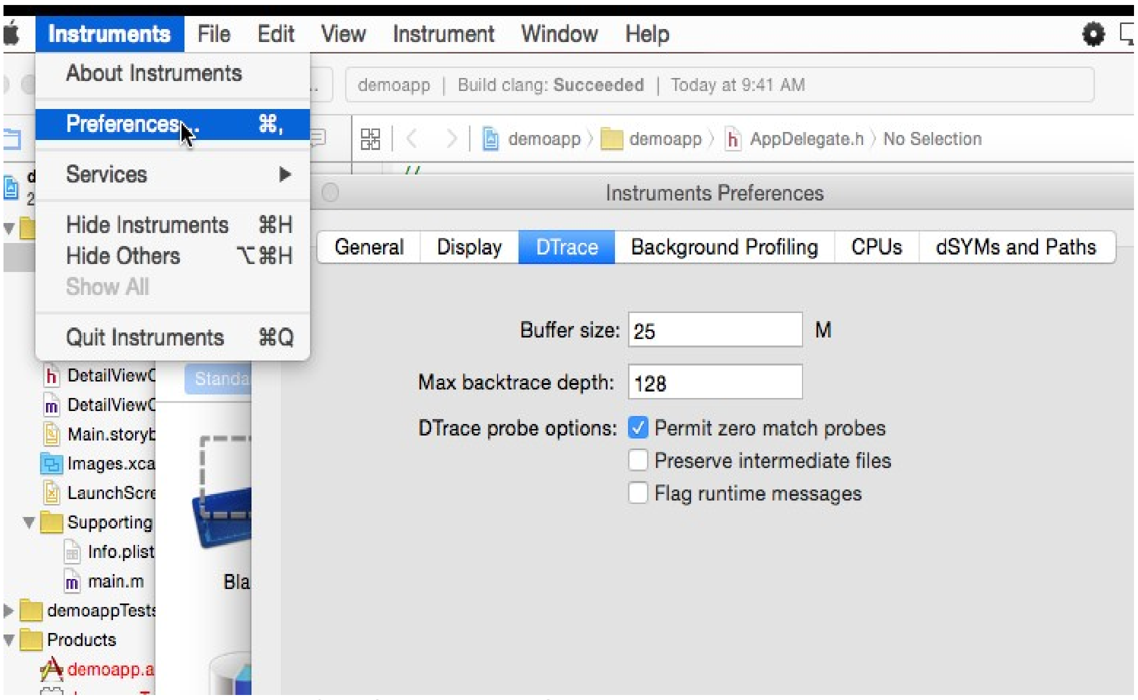
This is one of the best ways for black box dynamic analysis of an iOS application used so far, because the rest of the methods depend on the Xcode projects or run with the iPhone simulator. More methods of dynamically analyzing any iOS application from the AppStore like this one, are going to be presented on the next chapters.

6.3 Dtrace

DTrace is a dynamic tracing framework built by **San Microsystems** 10 years ago that provides *zero disable cost*, i.e. probes in code have no overhead when disabled. It is **only available** for **OS X** systems. Apple also uses DTrace on iOS in order to power tools such as **Instruments**. Although for third-party developers, it is only available on MAC OS X applications or the **iOS simulator** and that is the only way it can be used (*Apple already ported it to the iOS though but only internally for the company's labs and not for third party developers*).

DTrace is dynamic, meaning that attaching to an already running program and detaching from it again is possible without interrupting the program itself. There is also no need to recompile or restart the application. It is also a system wide tool that it can monitor with a single script what allocations where made by all processes on a system.

From the menu entry **Xcode → Open Developer Tools → Instruments** and then **Instruments *(menu on the upper left)* → Preferences** the Dtrace options for iPhone Simulator of the IDE can be adjusted. The rest of the functionality is built into the Instruments tool so it is actually being used transparently when **Instruments** is running leaving only the part of configuration tab available for editing purposes on the IDE. It can otherwise be used from a terminal as a command line tool **[19]** but this is out of the scope of this research.



Instruments Preferences

General | Display | DTrace | Background Profiling | CPUs | dSYMs and Paths

Buffer size: 25 M

Max backtrace depth: 128

DTrace probe options: ✓ Permit zero match probes
☐ Preserve intermediate files
☐ Flag runtime messages

(Dtrace within Instruments for XCode's iPhone Simulator)

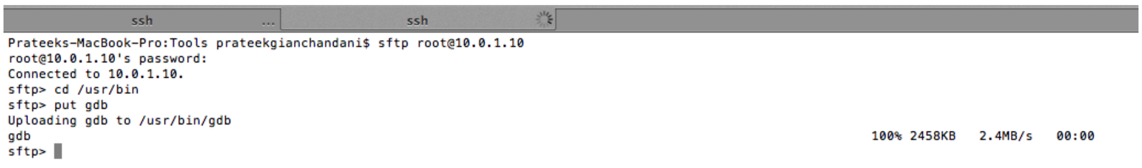
6.4 GDB (runtime analysis)

iOS applications mostly use the Objective-C language, meaning that they are **runtime oriented** and decisions for function calls are made during runtime and not during compilation of the source code or linking to libraries.

This behavior gives the opportunity to hook into a process of a running application using **GDB** for runtime analysis. First of all, a proper version of gdb needs to be installed on the device. The Cydia version of GDB does not work properly so another source like the following should be used:

https://github.com/swigger/gdb-ios

Then using sftp, the GDB must be uploaded to the device under the **/usr/bin/** directory as shown in the figure below:



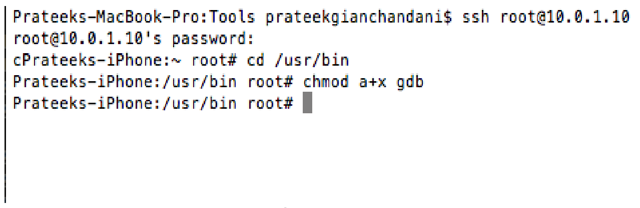
```
Prateeks-MacBook-Pro:Tools prateekgianchandani$ sftp root@10.0.1.10
root@10.0.1.10's password:
Connected to 10.0.1.10.
sftp> cd /usr/bin
sftp> put gdb
Uploading gdb to /usr/bin/gdb
gdb                                                                   100% 2458KB   2.4MB/s   00:00
sftp> 
```

uploading gdb to iPhone [20]

Now the file permissions have to be changed in order to be able to run the GDB tool (***chmod a+x /usr/bin/gdb***). The important information about the analyzed application is to locate its pid while it is running on the device, to hook into it (Google Maps will be used here). So, by using #***ps -aux | grep AppName*** the process id can be located and finally run gdb against it with the command: #***gdb -p PID.***

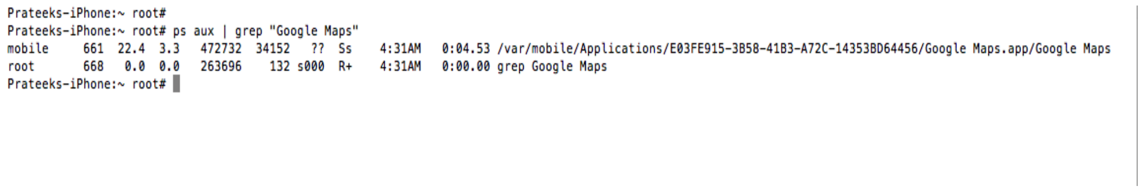
Please note that the application should be running in the foreground.

Demonstrated below are the commands used for this step:



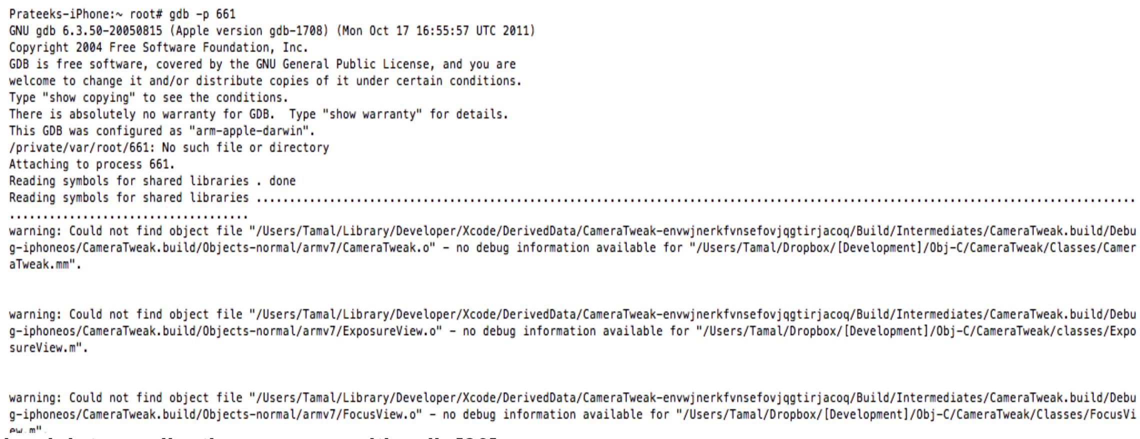
```
Prateeks-MacBook-Pro:Tools prateekgianchandani$ ssh root@10.0.1.10
root@10.0.1.10's password:
cPrateeks-iPhone:~ root# cd /usr/bin
Prateeks-iPhone:/usr/bin root# chmod a+x gdb
Prateeks-iPhone:/usr/bin root# █
```

changing permissions of gdb tool on iPhone [20]



```
Prateeks-iPhone:~ root#
Prateeks-iPhone:~ root# ps aux | grep "Google Maps"
mobile     661  22.4  3.3   472732  34152   ??  Ss    4:31AM   0:04.53 /var/mobile/Applications/E03FE915-3B58-41B3-A72C-14353BD64456/Google Maps.app/Google Maps
root       668   0.0  0.0   263696    132 s000  R+    4:31AM   0:00.00 grep Google Maps
Prateeks-iPhone:~ root# █
```

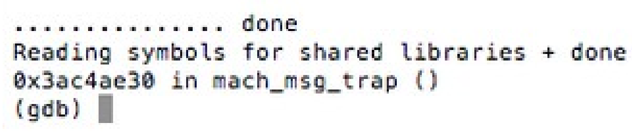
finding the application PID



```
Prateeks-iPhone:~ root# gdb -p 661
GNU gdb 6.3.50-20050815 (Apple version gdb-1708) (Mon Oct 17 16:55:57 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 "arm-apple-darwin".
/private/var/root/661: No such file or directory
Attaching to process 661.
Reading symbols for shared libraries . done
Reading symbols for shared libraries ...........................................................................................................
...................................
warning: Could not find object file "/Users/Tamal/Library/Developer/Xcode/DerivedData/CameraTweak-envwjnerkfvnsefovjqgtirjacoq/Build/Intermediates/CameraTweak.build/Debu
g-iphoneos/CameraTweak.build/Objects-normal/armv7/CameraTweak.o" - no debug information available for "/Users/Tamal/Dropbox/[Development]/Obj-C/CameraTweak/Classes/Camer
aTweak.mm".


warning: Could not find object file "/Users/Tamal/Library/Developer/Xcode/DerivedData/CameraTweak-envwjnerkfvnsefovjqgtirjacoq/Build/Intermediates/CameraTweak.build/Debu
g-iphoneos/CameraTweak.build/Objects-normal/armv7/ExposureView.o" - no debug information available for "/Users/Tamal/Dropbox/[Development]/Obj-C/CameraTweak/classes/Expo
sureView.m".


warning: Could not find object file "/Users/Tamal/Library/Developer/Xcode/DerivedData/CameraTweak-envwjnerkfvnsefovjqgtirjacoq/Build/Intermediates/CameraTweak.build/Debu
g-iphoneos/CameraTweak.build/Objects-normal/armv7/FocusView.o" - no debug information available for "/Users/Tamal/Dropbox/[Development]/Obj-C/CameraTweak/Classes/FocusVi
ew.m".
```

hook into application process with gdb [20]

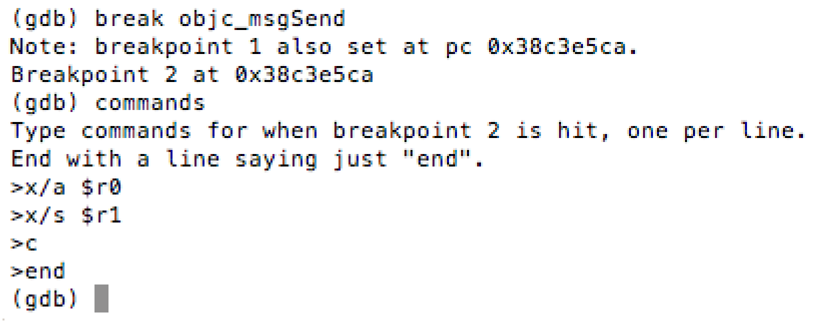
Several warnings might come up during the hooking process, but they can be ignored right now. After the successful hooking, the known gdb command line prompt is presented.



```
................ done
Reading symbols for shared libraries + done
0x3ac4ae30 in mach_msg_trap ()
(gdb) █
```

Gdb hooked into application process [20]

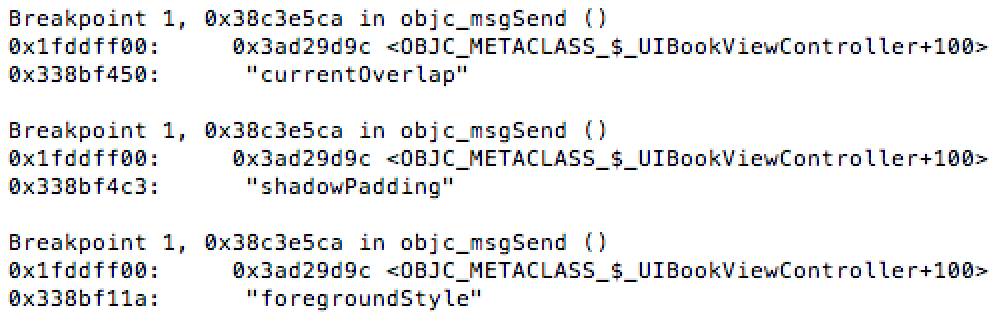
From now on, the running process can be treated as in any other case of runtime analysis using gdb. For example, the messaging mechanism of Objective-C that uses the obj_msgSend() method can be leveraged by adding a breakpoint whenever the specific method is called to show us the application flaw based on messaging and the actual selector/method that sent the message with the gdb command shown below:



```
(gdb) break objc_msgSend
Note: breakpoint 1 also set at pc 0x38c3e5ca.
Breakpoint 2 at 0x38c3e5ca
(gdb) commands
Type commands for when breakpoint 2 is hit, one per line.
End with a line saying just "end".
>x/a $r0
>x/s $r1
>c
>end
(gdb)
```

breakpoint on obj_msgSend() to print class and selector sending the message ($r0 for class, $r1 for class selector) [20]

Then typing the c command, reveals the values of each message sent with the help of the previous defined gdb variables.



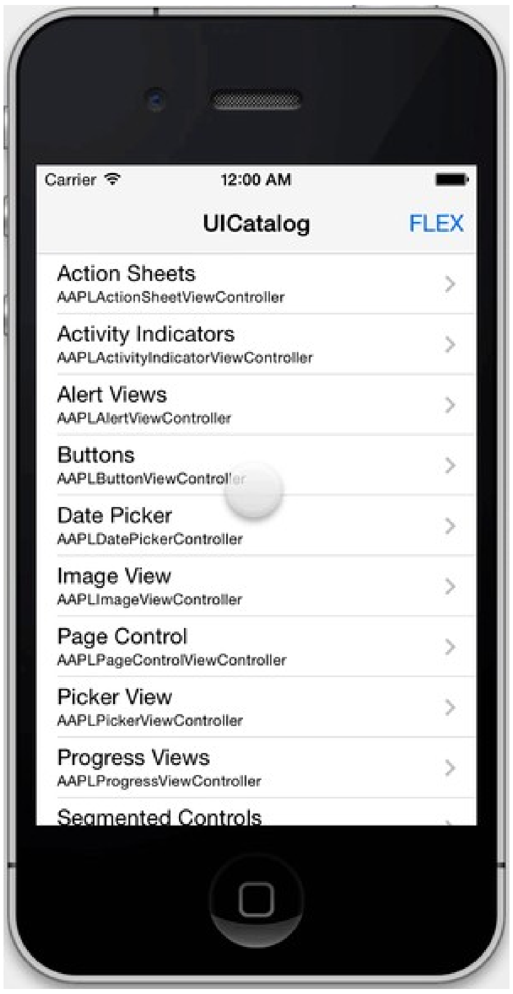
```
Breakpoint 1, 0x38c3e5ca in objc_msgSend ()
0x1fddff00:     0x3ad29d9c <OBJC_METACLASS_$_UIBookViewController+100>
0x338bf450:      "currentOverlap"

Breakpoint 1, 0x38c3e5ca in objc_msgSend ()
0x1fddff00:     0x3ad29d9c <OBJC_METACLASS_$_UIBookViewController+100>
0x338bf4c3:      "shadowPadding"

Breakpoint 1, 0x38c3e5ca in objc_msgSend ()
0x1fddff00:     0x3ad29d9c <OBJC_METACLASS_$_UIBookViewController+100>
0x338bf11a:      "foregroundStyle"
```

output of obj_msgSend() (class/selector) within gdb [20]

The capabilities of gdb-ios are as extensive as the ones of the common gdb. This chapter covers the fundamental steps of installation and hooking into a process with this debugger. Further analysis cannot be included as it reaches out of this document's purpose which is not to teach debugging skills rather than familiarize the reader with the available tools and procedure of iOS application security analysis. Following up, a more interactive tool is presented that includes a UI version of listing options and is called FLEX.

6.5 FLEX

FLEX (Flipboard Explorer) is a set of in-app debugging and exploration tools for iOS development. When presented, FLEX shows a tool bar that lives in a window above your application. From this tool bar, you can view and modify nearly every piece of state in your running application. You can inspect and modify views in the hierarchy, view and modify the properties and ivars on any object, dynamically call instance and class methods, view the file system within your application's sandbox, access any live object by a scan of the heap and more.



Flex options from its GitHub repository

It can be downloaded from the following link:

https://github.com/Flipboard/FLEX

As stated by the contributors it gives numerous options during runtime analysis of an application. FLEX is designed to be used for your own application analysis and not on third party ones. Nevertheless, a solution for this issue will be described after the actual options are presented, which makes FLEX available for every iOS application of a device. This tool can be used either with the iOS simulator of Xcode (iOS version must be greater than 7) or on an actual iOS device. The last option will be discussed in further detail and is also the more interesting. The first one is only useful for application developers.

This tool is a drop-in library that runs entirely inside the application without the need to connect to a debugger. The options and usage are presented clearly at the contributors GitHub project website. Only a short reference is demonstrated here about the capabilities of FLEX.

First of all, FLEX must be downloaded on a Macintosh device with the Xcode installed. Installation then can be achieved by either using the CocoaPods file (Xcode version 7,8) or by adding the files of the extracted FLEX project .zip under the **“Classes/”** directory of the analyzed **Xcode project**.

CocoaPods is a dependency manager for Swift and Objetive-C programming languages used for iOS development. For instance, with a simple file looking like a .yml configuration file and actually being a **Podfile,** the versions of JSON, StackView and several other options can be set as described in the CocoaPods website tutorial **[21].**

After FLEX is installed by using any of the aforementioned ways, it is ready for use with the iOS simulator but its iOS version has to be greater than 7 for FLEX to work.

Following up is a list of the capabilities this tool offers:

- ***Modify Views***

When a View (View is a Screen of an application, a UI element) is selected by using the floating FLEX tab, almost anything can be changed about it including the position of text and buttons, font style and generally all the View’s properties and method calls.

- ***Network History***

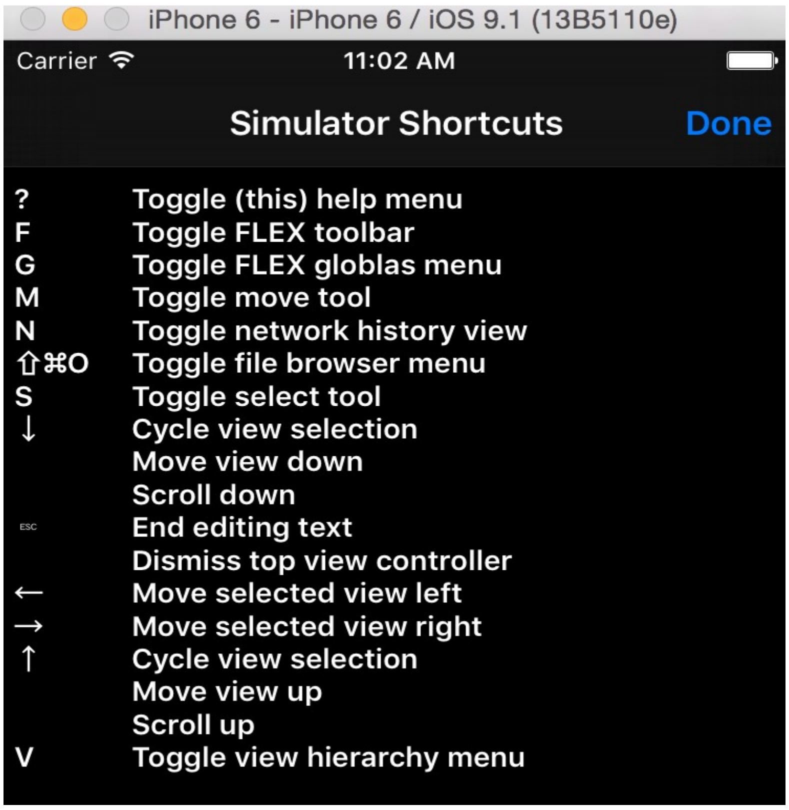
When enabled, network debugging allows you to view all requests made by using NSURLConnection or NSURLSession objects. Settings allow you to adjust what kind of response bodies get cached and the maximum size limit of the response cache. You can choose to have network debugging enabled automatically on app launch. This setting is persisted across launches, so any network call made by the selected application and its desired part of the HTTP response can be cached and saved.

- ***Objects on the Heap***

It can query all objects (NS objects like strings, arrays, variables) saved on the memory Heap.

- ***Simulator Keyboard shortcuts***

Within the iOS simulator several keyboard shortcuts are available as shown below:



Simulator Shortcuts

?	Toggle (this) help menu
F	Toggle FLEX toolbar
G	Toggle FLEX globlas menu
M	Toggle move tool
N	Toggle network history view
⇧⌘O	Toggle file browser menu
S	Toggle select tool
↓	Cycle view selection
	Move view down
	Scroll down
ESC	End editing text
	Dismiss top view controller
←	Move selected view left
→	Move selected view right
↑	Cycle view selection
	Move view up
	Scroll up
V	Toggle view hierarchy menu

FLEX keyboard shortcuts on iOS simulator

- ***File Browser***

Inspect everything within the application's sandboxed file system from .plist file to images and temporary generated files during the application execution. Information can be copied for lateral analysis from the file system to the user directory too.

- ***SQLite Browser***

SQLite database files (.db or .sqlite) can be explored using FLEX. The database browser allows you to view all tables, and individual tables can be sorted by tapping column headers.

- ***System Library Exploration***

The classes of system libraries being used can be inspected by creating an instance of them whether they are public or private and setting the desired values of their property.

- ***NSUserDefaults Editing***

These are the default settings of the user in the application like values to be displayed, the object types and their order within the application context. For instance, the language, date or the pop up keyboard layout are some of them and can be edited.

Finally, the option of using **FLEX** on **third party** applications is the most interesting feature. Steps to achieve this functionality are not provided by the contributors but will be presented and explained now as the last and most important part of FLEX.

This can be accomplished by using **code/dynamic library injection,** meaning that FLEX's dynamic library must be added into a jail broken iOS device's dynamic library directory.

Note: on iOS 7 or later, MobileSubstrate framework is provided after jailbreak for loading dynamic libraries. On previous versions, the environment variable DYLD_INSERT_LIBRARY can be used to add the dynamic library (not tested for FLEX).

Since XCode does not support the iOS dynamic library project development by default, it has to be enabled by copying from the folders

folder 1:

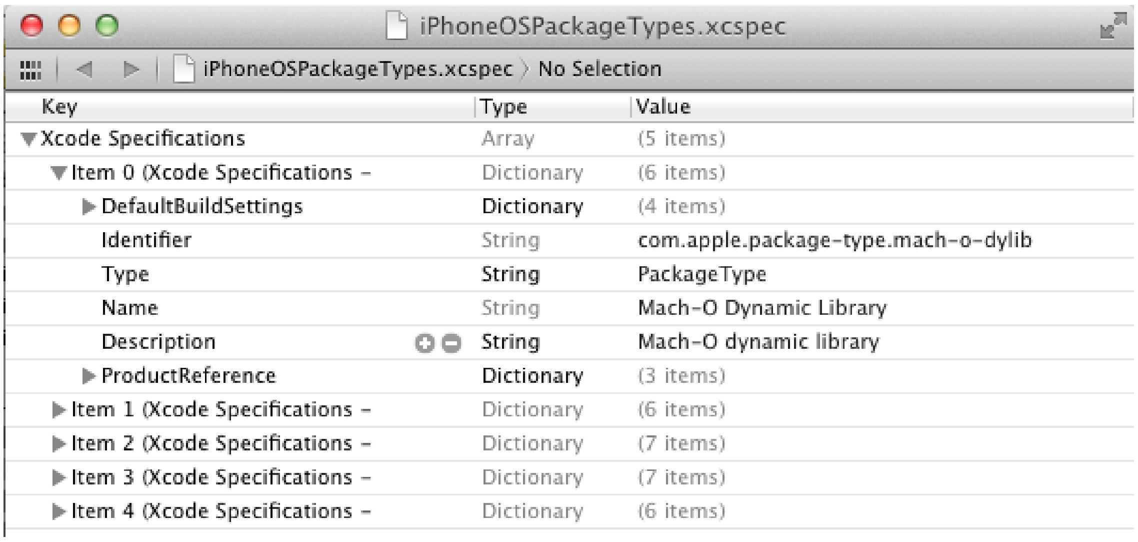
/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Specifications/

folder 2:

Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/Library/Xcode/Specifications/

to a write permission directory like Desktop, the **iPhoneOSPackageTypes.xcspec, iPhoneOSProductTypes.xcspec** and **MacOSXPackageTypes.xcspec, MacOSXProductTypes.xcspec** files respectively. Then these files must be opened with Xcode and:

- Drag Identifier for ***com.apple.package-type.mach-o-dylib*** from the ***MacOSXPackageTypes.xcspec*** to the ***iPhoneOSPackageTypes.xcspec*** so the last one looks like this:

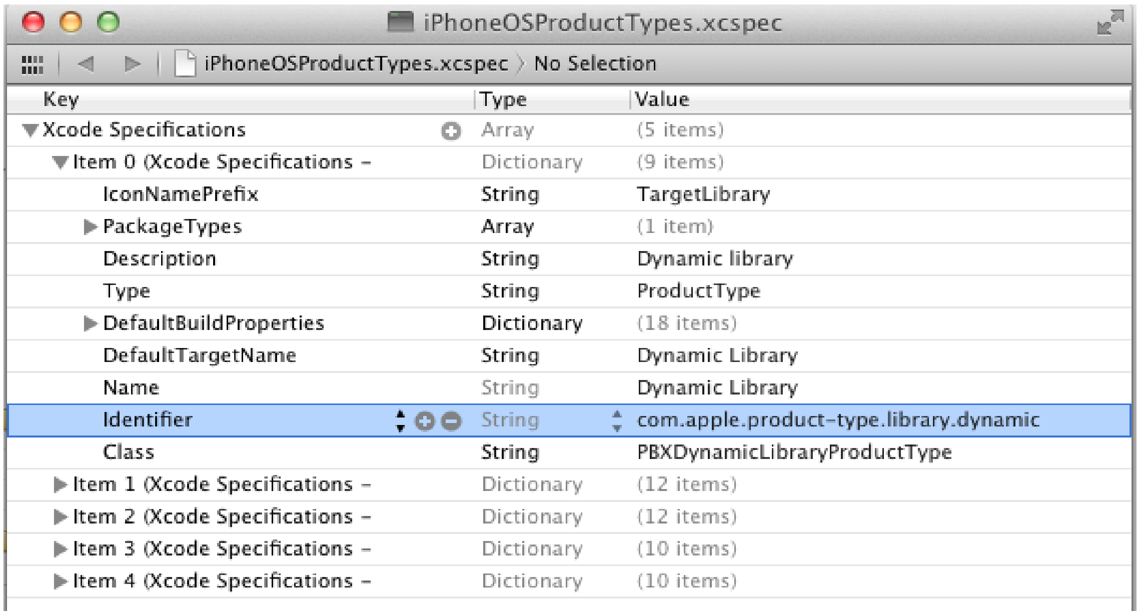


iPhoneOSPackageTypes.xcspec

iPhoneOSPackageTypes.xcspec 〉 No Selection

Key	Type	Value
▼Xcode Specifications	Array	(5 items)
▼Item 0 (Xcode Specifications –	Dictionary	(6 items)
▶DefaultBuildSettings	Dictionary	(4 items)
Identifier	String	com.apple.package-type.mach-o-dylib
Type	String	PackageType
Name	String	Mach-O Dynamic Library
Description	String	Mach-O dynamic library
▶ProductReference	Dictionary	(3 items)
▶Item 1 (Xcode Specifications –	Dictionary	(6 items)
▶Item 2 (Xcode Specifications –	Dictionary	(7 items)
▶Item 3 (Xcode Specifications –	Dictionary	(7 items)
▶Item 4 (Xcode Specifications –	Dictionary	(6 items)

- Drag Identifier for ***com.apple.product-type.library.dynamic from*** **MacOSXProductTypes.xcspec** to **iPhoneOSProductTypes.xcspec** so the last one looks as follows:

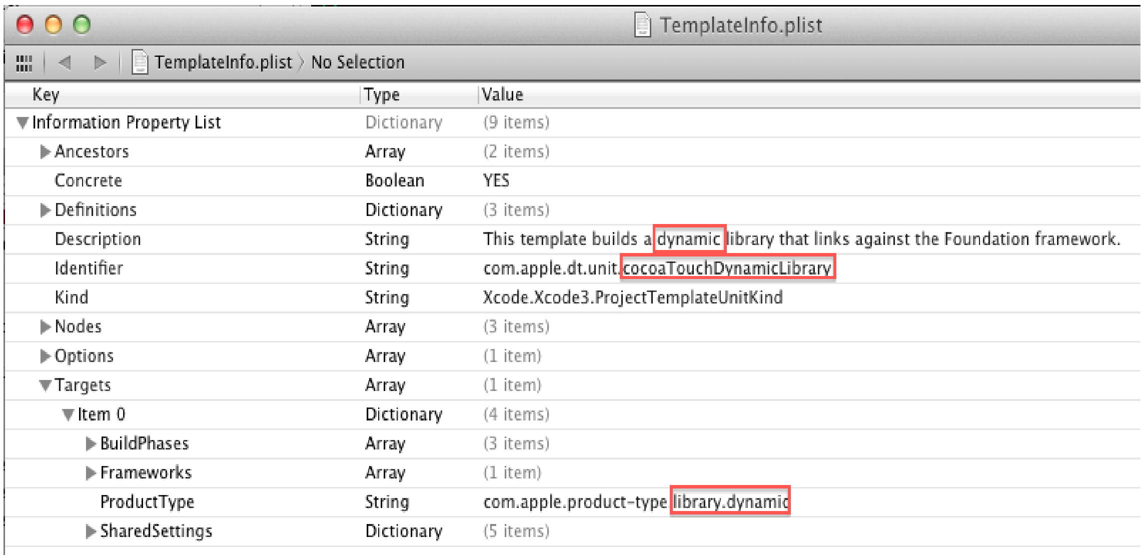


iPhoneOSProductTypes.xcspec

iPhoneOSProductTypes.xcspec 〉 No Selection

Key	Type	Value
▼Xcode Specifications	Array	(5 items)
▼Item 0 (Xcode Specifications –	Dictionary	(9 items)
IconNamePrefix	String	TargetLibrary
▶PackageTypes	Array	(1 item)
Description	String	Dynamic library
Type	String	ProductType
▶DefaultBuildProperties	Dictionary	(18 items)
DefaultTargetName	String	Dynamic Library
Name	String	Dynamic Library
Identifier	String	com.apple.product-type.library.dynamic
Class	String	PBXDynamicLibraryProductType
▶Item 1 (Xcode Specifications –	Dictionary	(12 items)
▶Item 2 (Xcode Specifications –	Dictionary	(12 items)
▶Item 3 (Xcode Specifications –	Dictionary	(10 items)
▶Item 4 (Xcode Specifications –	Dictionary	(10 items)

- Save the changes and copy the two iOS files ***iPhoneOSPackageTypes.xcspec*** and **iPhoneOSProductTypes.xcspec** back to their original directory *"/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/Library /Xcode/Specifications/"* and provide your password if prompted.
- Create a project template and copy to Desktop the entire folder *"/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/Library /Xcode/Templates/Project Templates / Framework & Library /* ***Cocoa Touch Static Library.xctemplate****"* now open the file ***TemplateInfo.plist*** *and edit the three highlighted entries as shown below:*

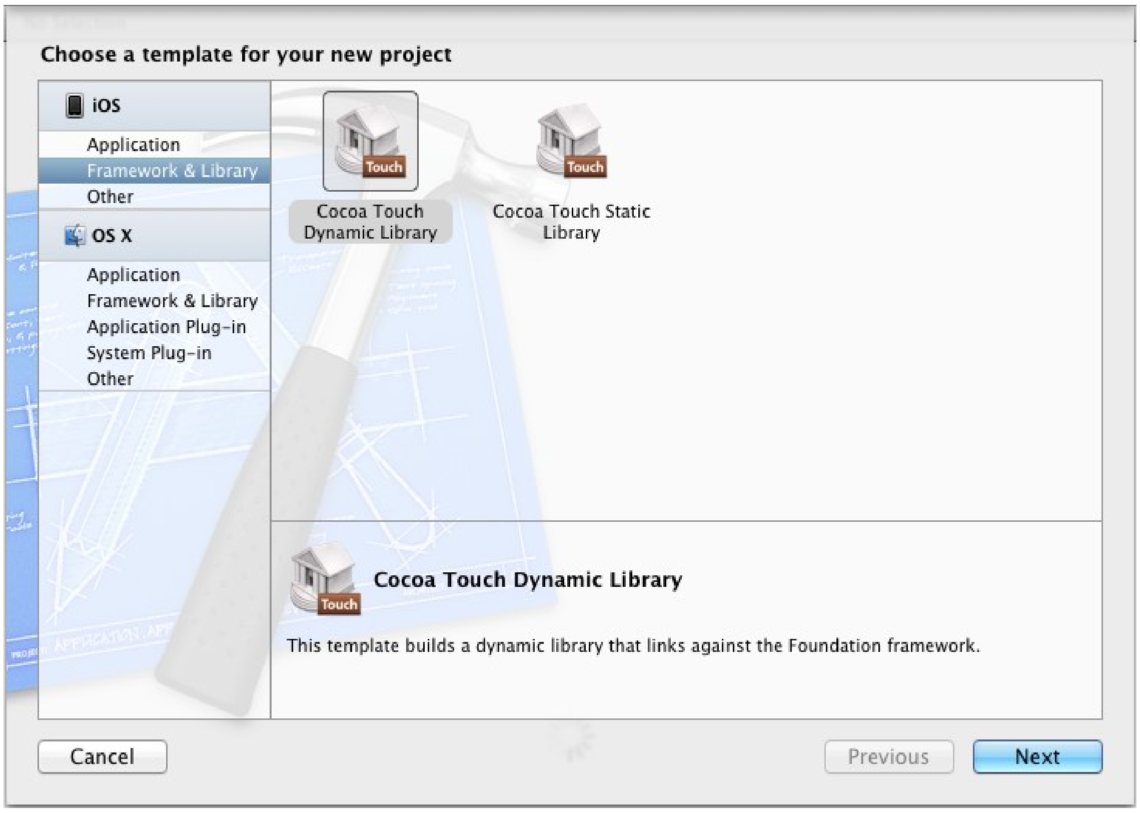


TemplateInfo.plist

Key	Type	Value
Information Property List	Dictionary	(9 items)
Ancestors	Array	(2 items)
Concrete	Boolean	YES
Definitions	Dictionary	(3 items)
Description	String	This template builds a dynamic library that links against the Foundation framework.
Identifier	String	com.apple.dt.unit.cocoaTouchDynamicLibrary
Kind	String	Xcode.Xcode3.ProjectTemplateUnitKind
Nodes	Array	(3 items)
Options	Array	(1 item)
Targets	Array	(1 item)
Item 0	Dictionary	(4 items)
BuildPhases	Array	(3 items)
Frameworks	Array	(1 item)
ProductType	String	com.apple.product-type.library.dynamic
SharedSettings	Dictionary	(5 items)

- Save and copy back the file **Cocoa Touch Dynamic Library.xctemplate** to *"/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/Developer/Library /Xcode/Templates/Project Templates / Framework & Library /".*

- Restart Xcode and select a new dynamic library Cocoa touch object (the initial goal is achieved and Xcode can be used for dynamic library development):



Choose a template for your new project

Cocoa Touch Dynamic Library

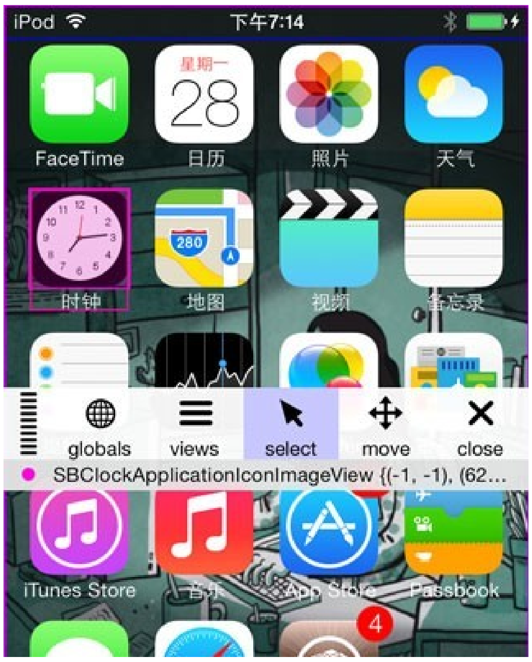
This template builds a dynamic library that links against the Foundation framework.

Now a new project called "**libFlex**" must be built by selecting just the "**Cocoa Touch Dynamic Library**". This option reveals that currently Xcode was correctly modified to support dynamic library projects.

Finally, import FLEX's code from GitHub into the project. The whole FLEX project is going to be built by using the dynamic library project option that was just enabled in order to acquire FLEX's dynamic library product file after compilation. Under the file "libFlex.m" add the following code snippet:

```
#import "libFlex.h"
#import <UIKit/UIKit.h>
#import "FLEXManager.h"

@implementation libFlex

- (id)init
{
    self = [super init];
    if (self) {
        [[NSNotificationCenter defaultCenter] addObserver:self
                                                 selector:@selector(appLaunched:)
                                                     name:UIApplicationDidBecomeActiveNotification
                                                   object:nil];
    }
    return self;
}

- (void)appLaunched:(NSNotification *)notification
{
    NSLog(@"====================== libFlex dylib show =======================");

    [[FLEXManager sharedManager] showExplorer];
}
@end

static void __attribute__((constructor)) initialize(void)
{
    NSLog(@"====================== libFlex dylib initialize =======================");

    static libFlex *entrance;
    entrance = [[libFlex alloc] init];
}
```

The project is ready to be **compiled.** When this is done and no errors come up, the product dynamic library needs to be copied to the iPhone device using "***scp -r libFlex.dylib* root@192.168.2.2*:/Library/MobileSubstrate/DynamicLibraries***". Finally, either restart the iPhone device or ssh to it and run "**killall SpringBoard**" to reload every Application together with FLEX. FLEX tool bar is available now on the device:



FLEX toolbar loaded on iPhone – itony.me [22]

The FLEX tool bar will be loaded for **each application** dynamically. If this is not required it can be adjusted from the ***/Library/MobileSubstrate/DynamicLibraries*** directory. In this place a file with the same name configuration file ***libFlex.plist*** must be created, so that only the specified applications will automatically load the FLEX tool bar. The following example of the “libflex.plist” file included the tool bar only for the four specified applications:

```
{
Filter = {
            Bundles = ("com.sina.weibo");
            Bundles = ("com.burbn.instagram");
            Bundles = ("com.apple.podcasts");
            Bundles = ("com.apple.iTunes");
  };
}
```

This now loads the FLEX tool bar dynamically only for the four specified applications. Hence, the way to use FLEX’s debugging powers on every single application has also been described. Interaction, modification and changing the properties of any installed application are some of the capabilities gained with this tool by simply injecting FLEX’s dynamic library onto the jail broken iOS device. Deeper inspection and analysis tools are presented on the next chapter providing the Objective C runtime method call hierarchy and arguments passed revealing valuable source code information for the reverse engineering process.

6.6 Inspective C

When the main goal is to discover the implementation of an application based on the Objective C source code including method calls and hierarchy, there is one main tool and this is Inspective C. This tool can currently observe Objective C objects, objects of given classes and their selectors and every single call of the obj_msgSend method.

Instead of the conventional way of analyzing the executable file using IDA and diving into its assembly code, Inspective C can be used to avoid this hassle. With InspectiveC, the following aspects can be inspected dynamically:

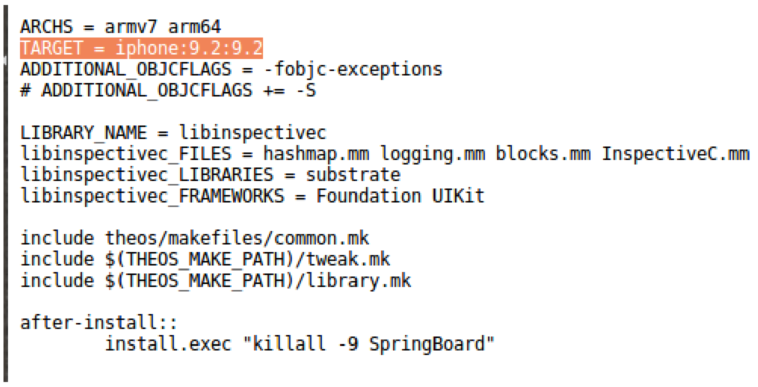
- Calls to a method of a class
- Every method a class may be using.
- Identify the instance object of a method call.
- The invocation of methods by their signature.

In order to install and use the tool against iOS applications, its project from the following GitHub repository must be first downloaded:

https://github.com/DavidGoldman/InspectiveC

Theos installation is required too (refer to Chapter 8 for further details).

Modification of the "***Makefile***" according to the actual iPhoneSDK version of the machine and the machine's CPU architecture is required prior to compiling the project:



```
ARCHS = armv7 arm64
TARGET = iphone:9.2:9.2
ADDITIONAL_OBJCFLAGS = -fobjc-exceptions
# ADDITIONAL_OBJCFLAGS += -S

LIBRARY_NAME = libinspectivec
libinspectivec_FILES = hashmap.mm logging.mm blocks.mm InspectiveC.mm
libinspectivec_LIBRARIES = substrate
libinspectivec_FRAMEWORKS = Foundation UIKit

include theos/makefiles/common.mk
include $(THEOS_MAKE_PATH)/tweak.mk
include $(THEOS_MAKE_PATH)/library.mk

after-install::
	install.exec "killall -9 SpringBoard"
```

Editing the Makefile of Inspective C project before compiling it [23]

So, the Theos tool is required now to create a .deb package file for the installation of the tool on the iOS device. "Make" the project using Clang on your MAC and then install the product .deb file into the iPhone device. After the installation, find libinspectivec.dylib under /usr/lib device directory and copy this file back to your Mac under /opt/theos/lib directory. This can be done by using scp from the iPhone to your Mac:

IPhone: ~ root # scp ./libinspectivec.dylib mac@

(On the Mac enable scp in the system preferences - shared - remote login)

and then copy **InspectiveC.h** too, into $THEOS/include.

Now you can either use inspectiveC with Cycript or with your own tweak. In case of Cycript you have to download it first from Cydia on your iPhone and then directly hook to the process of the desired application by using the following command:

root# cycript -p ApplicationProcess

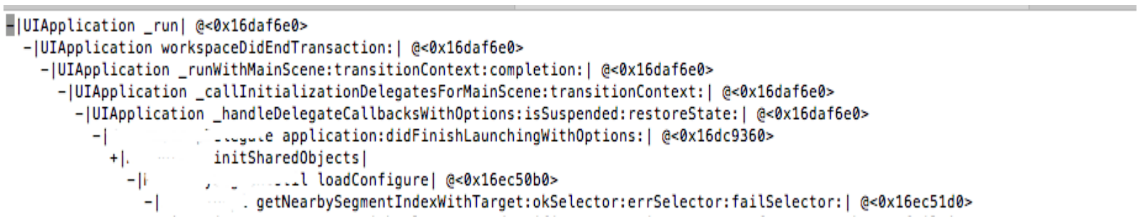
On the other hand, when it is used with Tweak you have to copy the "**InspCWrapper.m**" file from the GitHub project to the tweak directory, and then include this file in your tweak code with:

#include "InspCWrapper.m"

This gives the capability of dynamically using Inspective C every time the tweak runs. In order to proceed using the tool you have to refer to our chapter about tweak and Cycript usage. In any case the log output of Objective C calls recorded by Inspective C can be found and observed in the application's directory ***/Documents/InspectiveC*** as follows:

Tail -f AppDir/Documents/InspectiveC/12892_main.log

(Use tail by installing Core Utilities in Cydia)



```
-|UIApplication _run| @<0x16daf6e0>
  -|UIApplication workspaceDidEndTransaction:| @<0x16daf6e0>
    -|UIApplication _runWithMainScene:transitionContext:completion:| @<0x16daf6e0>
      -|UIApplication _callInitializationDelegatesForMainScene:transitionContext:| @<0x16daf6e0>
        -|UIApplication _handleDelegateCallbacksWithOptions:isSuspended:restoreState:| @<0x16daf6e0>
          -|[redacted]e application:didFinishLaunchingWithOptions:| @<0x16dc9360>
            +|[redacted] initSharedObjects|
              -|[redacted]l loadConfigure| @<0x16ec50b0>
                -|[redacted] getNearbySegmentIndexWithTarget:okSelector:errSelector:failSelector:| @<0x16ec51d0>
```

sample Objective C calls hierarchy logged output using InspectiveC [23]

By using Inspective C, a really close look of the initial source code can be achieved. The important part is that no assembly language is involved to this process. Following up, is the last tool of dynamic analysis that provides runtime manipulation and interaction with the application's running process and is called Cycript.

6.7 Cycript

Cycript is one of the most powerful assets for the iOS application security analysis. It basically is a Framework built on top of Cydia Substrate (refer to technical terms) that uses an interpreter of JavaScript and Objective C languages together, providing the ability to directly interact with the running application and modify it while running on the device. This functionality is achieved by using the underlying Substrate stated before, allowing interaction using only plain JavaScript and-or Objective C commands. Its official website provides all the information required to start using and learning this concept **[24]**.

The power of this Framework is its interaction with the running process of an application while being hooked into its functions. It can inspect, change class instance variables or even do method Swizzling and replace the actual executed code with a custom one.

Installation of Cycript on an iOS device, starts with downloading the latest version of its SDK from the official website **[24]** and then moving the downloaded deb package cycript.deb to the iOS device by using sftp:

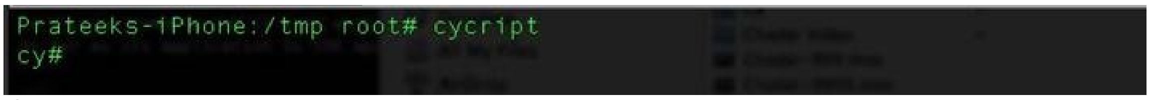
MAC# sftp root@192.168.2.2

sftp> put cycript.deb

Then with the dpkg command (like on a standard Debian environment) the package can be installed on the device either using ssh from the Mac or directly from the device with a terminal if such an option is available, using the command:

iPhone# dpkg -i cycript.deb

If everything went as expected Cycript must have been installed correctly on the device. Check this out by typing "Cycript" into a terminal on the iPhone. This should drop you into a cycript shell/prompt as shown in the figure below:

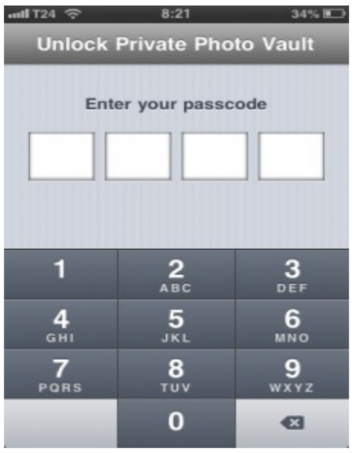


```
Prateeks-iPhone:/tmp root# cycript
cy#
```

Cycript prompt on iPhone [25]

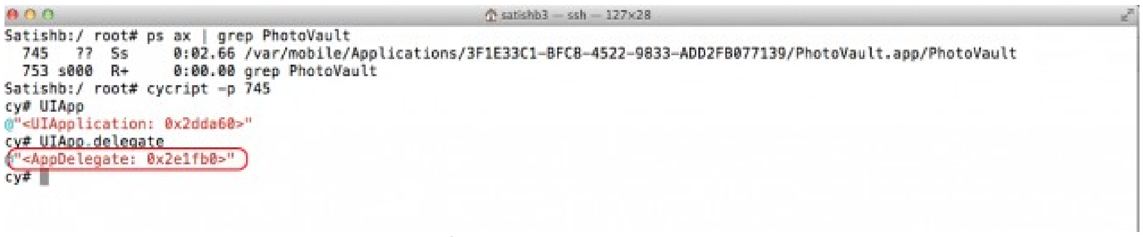
When the interpreter is correctly installed it can be used against running applications on the device. An interesting example is interacting with a ***Photovault*** application **[26]** using Cycript and manage to bypass the security code required to browse photos.

To begin with, the analyzed application must be running and be on the foreground. Any other state leaves Cycript with no hooking capabilities because the application would be paused. The home screen of this specific application prompts for the password input:



PhotoVault application home screen [26]

Its corresponding process id is required now (as far as the application is in a running state). This can be obtained using the command ***"ps -ax | grep "AppName"*** on the iPhone device over the ssh shell on the Mac OSX:



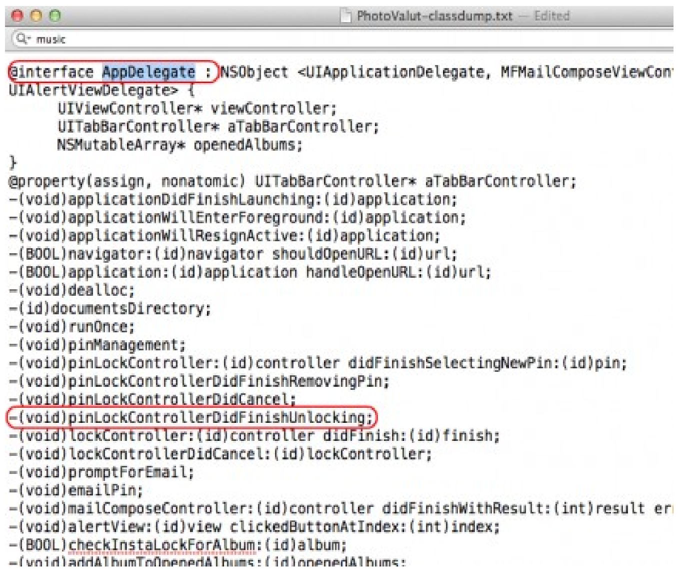
```
Satishb:/ root# ps ax | grep PhotoVault
  745   ??  Ss     0:02.66 /var/mobile/Applications/3F1E33C1-BFC8-4522-9833-ADD2FB077139/PhotoVault.app/PhotoVault
  753 s000  R+     0:00.00 grep PhotoVault
Satishb:/ root# cycript -p 745
cy# UIApp
@"<UIApplication: 0x2dda60>"
cy# UIApp.delegate
@"<AppDelegate: 0x2e1fb0>"
cy# 
```

hooking into application process with Cycript [26]

The first Objective C command that is always used in Cycript is ***UIApp*** (short for ***[UIApplication sharedApplication]***) on ***iOS*** and ***[NSApplication sharedApplication]*** on ***OSX*** applications. These are the basic application instance objects of Objective C that all methods and subclasses inherit from. An equivalent to the "Object" class in Java.

Having this object stored, you can later move deeper on the hierarchy of class and method calls to retrieve any possible child class object used by the application. In the previous figure for example, the delegate Class instance is obtained using the basic UIApp instance. The delegate class is defined for the application to conform to the UIApplication protocol and must implement some of the protocol's methods. The application object informs the delegate of significant runtime events like the application launch, low-memory warnings, and application termination, giving the opportunity to respond appropriately.

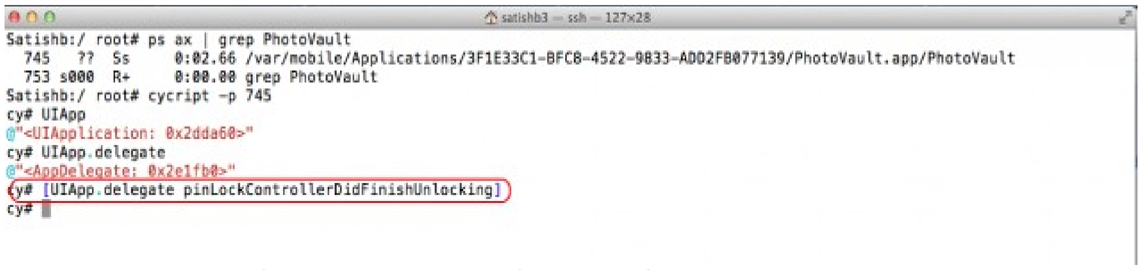
All the method calls can be discovered using the "***intermediate"*** delegate object's name "***AppDelegate***" against a class-dump file containing all the class and method calls available (created using the class-dump tool). By simply searching for the interface of AppDelegate class inside the class dump file all the method calls can be found:



```
@interface AppDelegate : NSObject <UIApplicationDelegate, MFMailComposeViewCon
UIAlertViewDelegate> {
        UIViewController* viewController;
        UITabBarController* aTabBarController;
        NSMutableArray* openedAlbums;
}
@property(assign, nonatomic) UITabBarController* aTabBarController;
-(void)applicationDidFinishLaunching:(id)application;
-(void)applicationWillEnterForeground:(id)application;
-(void)applicationWillResignActive:(id)application;
-(BOOL)navigator:(id)navigator shouldOpenURL:(id)url;
-(BOOL)application:(id)application handleOpenURL:(id)url;
-(void)dealloc;
-(id)documentsDirectory;
-(void)runOnce;
-(void)pinManagement;
-(void)pinLockController:(id)controller didFinishSelectingNewPin:(id)pin;
-(void)pinLockControllerDidFinishRemovingPin;
-(void)pinLockControllerDidCancel;
-(void)pinLockControllerDidFinishUnlocking;
-(void)lockController:(id)controller didFinish:(id)finish;
-(void)lockControllerDidCancel:(id)lockController;
-(void)promptForEmail;
-(void)emailPin;
-(void)mailComposeController:(id)controller didFinishWithResult:(int)result er
-(void)alertView:(id)view clickedButtonAtIndex:(int)index;
-(BOOL)checkInstaLockForAlbum:(id)album;
-(void)addAlbumToOpenedAlbums:(id)openedAlbums;
```

method calls of application using its delegate object instance name [26]

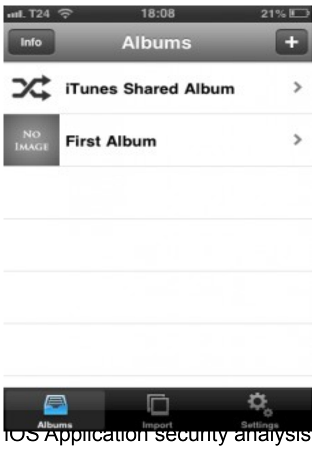
The interesting part found here is the call of method ***(void)pinLockControllerDidFinishlocking;*** that easily reveals its functionality of being ***called when the*** outcome of unlocking status check is true, probably somewhere within the rest of the application's code without any arguments. So, by calling this method directly by using Cycript (no arguments required), the application's process could be tricked into a complete and successful password check that never took place. With the following Cycript command the method is called directly as described:



```
Satishb:/ root# ps ax | grep PhotoVault
  745   ??  Ss     0:02.66 /var/mobile/Applications/3F1E33C1-BFC8-4522-9833-ADD2FB077139/PhotoVault.app/PhotoVault
  753 s000  R+     0:00.00 grep PhotoVault
Satishb:/ root# cycript -p 745
cy# UIApp
@"<UIApplication: 0x2dda60>"
cy# UIApp.delegate
@"<AppDelegate: 0x2e1fb0>"
cy# [UIApp.delegate pinLockControllerDidFinishUnlocking]
cy#
```

calling a method out of applications runtime flow using Cycript [26]

And eventually on the iPhone device, the application has changed state and opened up the photo albums meaning that the Cycript commands successfully skipped the password vault check with a simple method call:



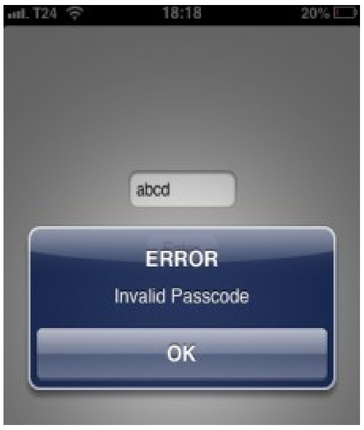
Password Vault step bypassed with Cycript on PhotoVault application [26]

This simple example of hacking an iOS application with Cycript, is based on Cycript's capability of directly by calling instances of methods during runtime. In any other case, this could not be achieved by changing the application's assembly for example, due to the validation mechanism present on every iOS application (code signing). Another very interesting way of using Cycript is method swizzling.

Method swizzling with Cycript

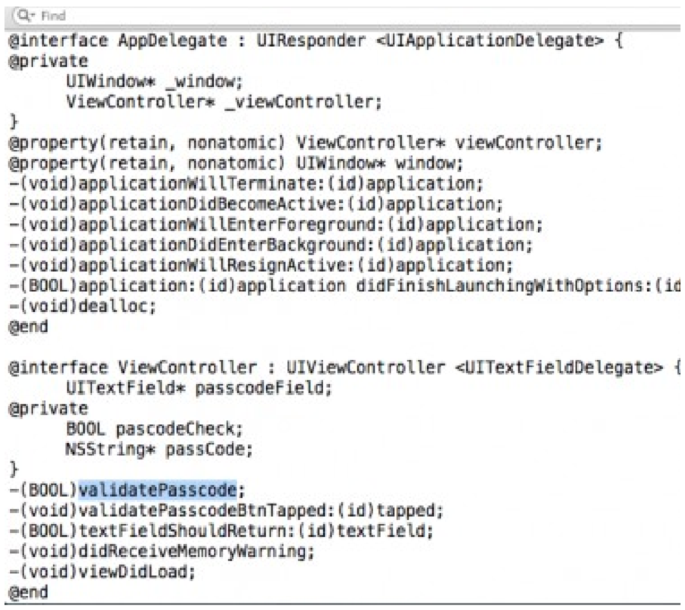
As already mentioned, Objective C is a ***runtime*** language. This ***allows modification*** or replacement of the existing methods source code dynamically, something known as ***method swizzling***. This feature can be leveraged using Cycript. For demonstration purposes, a demo application called "*HackTest*" that requires a password to login will be used. The login feature must be overridden by replacing the specific method of the application that implements this step.

This sample application is an unofficial .ipa file that can be found online for security testing purposes **[26].** After launching it on an iPhone device and providing a wrong password it arises the following error:



HackTest application Home screen [26]

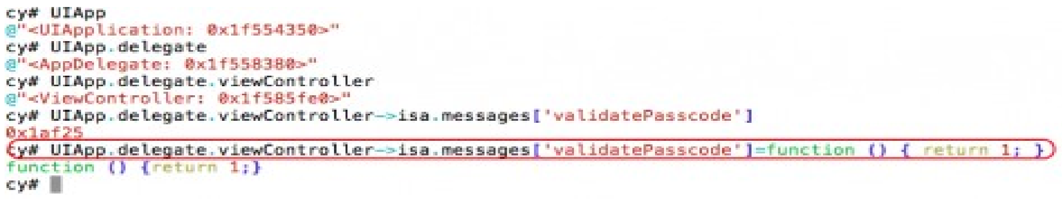
Now using an ssh connection to the iPhone device, Cycript must be launched against the application's process. After being hooked to the application's instance object UIApp, the ***delegate*** class instance must be also retrieved, in order to locate the interactive method calls by using this instance's name as a search term in the ***class-dump*** tool's exported .txt file. The interface of the AppDelegate object inside the class-dump.txt file section and its method calls are the following:



```
Find
@interface AppDelegate : UIResponder <UIApplicationDelegate> {
@private
        UIWindow* _window;
        ViewController* _viewController;
}
@property(retain, nonatomic) ViewController* viewController;
@property(retain, nonatomic) UIWindow* window;
-(void)applicationWillTerminate:(id)application;
-(void)applicationDidBecomeActive:(id)application;
-(void)applicationWillEnterForeground:(id)application;
-(void)applicationDidEnterBackground:(id)application;
-(void)applicationWillResignActive:(id)application;
-(BOOL)application:(id)application didFinishLaunchingWithOptions:(id
-(void)dealloc;
@end

@interface ViewController : UIViewController <UITextFieldDelegate> {
        UITextField* passcodeField;
@private
        BOOL pascodeCheck;
        NSString* passCode;
}
-(BOOL)validatePasscode;
-(void)validatePasscodeBtnTapped:(id)tapped;
-(BOOL)textFieldShouldReturn:(id)textField;
-(void)didReceiveMemoryWarning;
-(void)viewDidLoad;
@end
```

delegate object's Interface method calls found in class-dump export file [26]

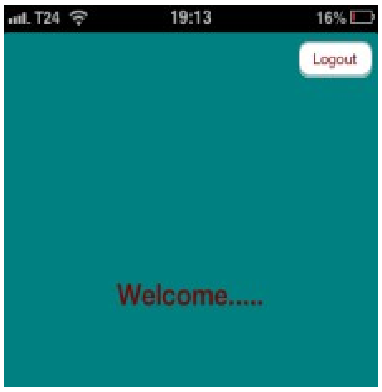
The method ***validatePasscode*** is declared in the ***ViewController*** interface and the *ViewController* instance is present in the *AppDelegate* interface. Hence the ***ViewController*** instance is required to access the validatePasscode method. Back on Cycript using the "**UIApp.delegate.viewController**" command, the ViewController instance is grabbed and then with *"UIApp.delegate.viewController->isa.messages['validatePasscode']=function (){return 1;}"* the function is overridden and always returns the value 1. **isa** is a pointer of the class structure and gives access to the method implementation in Cycript.



```
cy# UIApp
@"<UIApplication: 0x1f554350>"
cy# UIApp.delegate
@"<AppDelegate: 0x1f558380>"
cy# UIApp.delegate.viewController
@"<ViewController: 0x1f585fe0>"
cy# UIApp.delegate.viewController->isa.messages['validatePasscode']
0x1af25
cy# UIApp.delegate.viewController->isa.messages['validatePasscode']=function () { return 1; }
function () {return 1;}
cy# 
```

Overriding the ValidatePasscode function using cycript method swizzling [26]

The application changed state and the login screen is bypassed without providing the actual password:



HackTest application password login successfully overridden [26]

What was actually done here, is an interaction with the application's viewController object. This is responsible for the View interactions (touch and input) of the user and a change of the code being executed by its method for checking the password was done to return true (1). That tricked the viewController and its running flow to proceed with the next View of the application which is the main View.

This example demonstrates accurately how the ***runtime*** nature of Objective C (message based implementation during runtime) can be ***manipulated*** by using Cycript and less than 5 commands in total. The capabilities of this framework are many more but cannot be totally covered in this research. For further investigation and practice, the official website and even YouTube video series on this topic **[27]** are a great resource to understand and learn its numerous capabilities and options.

Runtime analysis ends with Cycript, one of the most powerful tools available for dynamic application analysis, to proceed with the chapter of data protection on the iOS.

CHAPTER 7

DATA PROTECTION

Data protection is another critical section when testing iOS applications. These applications are susceptible to loss and theft and sensitive information like cached data may get copied to syncing machines. Researches reveal that the iPhone does cache sensitive information like keystrokes and snapshots for extended periods of time. Additionally, the application itself may be storing sensitive information in form of temporary files, .plist files or in the client-side SQLite database.

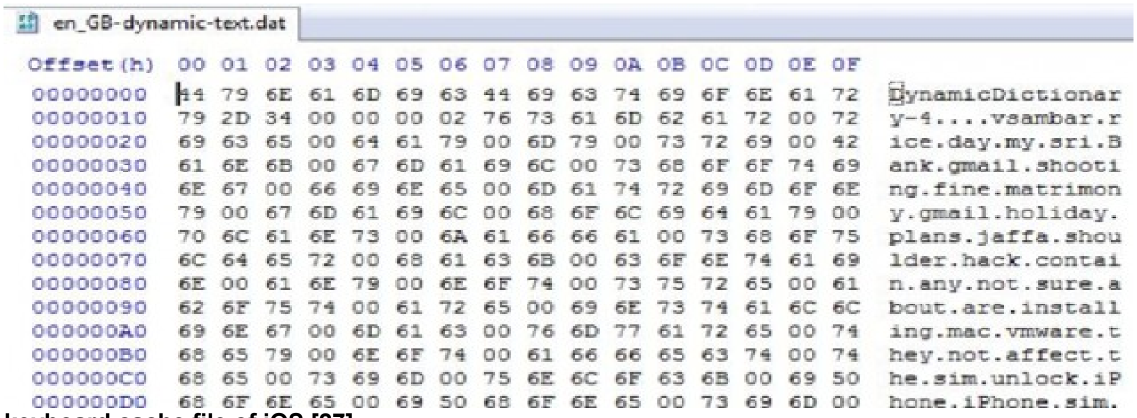
The basic sources that may expose sensitive information are sections where an application permanently or temporarily stores information. These sources may be the user input which is stored in the Keyboard cache for Autocomplete purposes, the SQLite database entries for permanent data storage and any of the temporary snapshots of an application window while it is moving to the background on the device home button press. The subject of this chapter is to demonstrate the ways in which these sources can be leveraged, in case the application developers did not take into consideration the appropriate security countermeasures.

7.1 Keyboard cache

The first and most important data source is the user input. An iPhone device can cache all the user keystrokes under:

This feature may be present on an application, based on the implementation techniques of the developers. It is similar to the ***AUTOCOMPLETE*** feature on other devices and web browsers. Regarding iOS, the AUTOCOMPLETE feature has to be set to off in UITextField, otherwise the text field input of the application will get cached on the aforementioned directory **[2]**.

The most important thing is that passwords never get cached for these fields regardless the value of AUTOCOMPLETE flags. Below is an example of a dynamic-tex.dat file containing keyboard cache and it is being viewed with a HEX editor:



```
en_GB-dynamic-text.dat

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  44 79 6E 61 6D 69 63 44 69 63 74 69 6F 6E 61 72  DynamicDictionar
00000010  79 2D 34 00 00 00 02 76 73 61 6D 62 61 72 00 72  y-4....vsambar.r
00000020  69 63 65 00 64 61 79 00 6D 79 00 73 72 69 00 42  ice.day.my.sri.B
00000030  61 6E 6B 00 67 6D 61 69 6C 00 73 68 6F 6F 74 69  ank.gmail.shooti
00000040  6E 67 00 66 69 6E 65 00 6D 61 74 72 69 6D 6F 6E  ng.fine.matrimon
00000050  79 00 67 6D 61 69 6C 00 68 6F 6C 69 64 61 79 00  y.gmail.holiday.
00000060  70 6C 61 6E 73 00 6A 61 66 66 61 00 73 68 6F 75  plans.jaffa.shou
00000070  6C 64 65 72 00 68 61 63 6B 00 63 6F 6E 74 61 69  lder.hack.contai
00000080  6E 00 61 6E 79 00 6E 6F 74 00 73 75 72 65 00 61  n.any.not.sure.a
00000090  62 6F 75 74 00 61 72 65 00 69 6E 73 74 61 6C 6C  bout.are.install
000000A0  69 6E 67 00 6D 61 63 00 76 6D 77 61 72 65 00 74  ing.mac.vmware.t
000000B0  68 65 79 00 6E 6F 74 00 61 66 66 65 63 74 00 74  hey.not.affect.t
000000C0  68 65 00 73 69 6D 00 75 6E 6C 6F 63 6B 00 69 50  he.sim.unlock.iP
000000D0  68 6F 6E 65 00 69 50 68 6F 6E 65 00 73 69 6D 00  hone.iPhone.sim.
```

keyboard cache file of iOS [27]

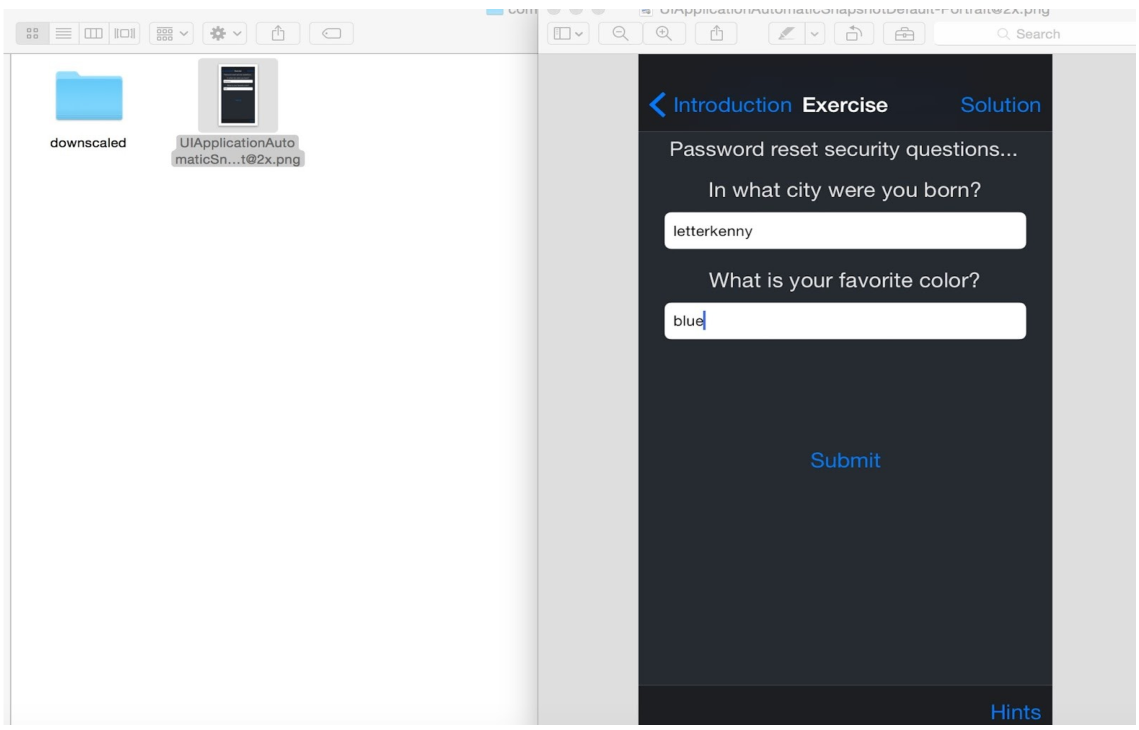
Potentially, almost any information provided in a text form as user input, may be directly accessible on an attempt to penetrate into an iOS device. Once again, everything except passwords (they are encrypted by default). Another way, in which user input or data could be gathered, is directly from the screen when a part of it is captured as a snapshot.

7.2 Snapshots

Snapshots are actual ***images/screenshots*** taken by the application, whenever a user pushes the Home button. This helps creating the shrinking effect while the active application's window disappears.

They are stored in the snapshots directory of the application, for example a "Sample" application stores them at *"~/Library/Application Support/iPhone Simulator/4.0.1/Applications/RANDOM-NUMBER /Library/Caches/Snapshots/com.yourcompany.Sample"*.

Applications may thus, not mask sensitive information on the screen sometimes, allowing not only shoulder surfing attacks but also information leakage from such snapshots. Below is an example of a snapshot from a sample application:



Password reset security questions...

In what city were you born?

letterkenny

What is your favorite color?

blue

application snapshot on iOS [28]

The type of data pictured in a snapshot may vary from username and credit card number input fields to extended messaging conversations or email contents. This is may be a great opportunity for information gathering. Applications also store data, either temporary or permanently. Temporary data are stored in the UIPasteBoard.

7.3 UIPasteBoard

This is a common ***storage space*** for applications to copy and paste temporary data being used. If an iOS application uses this feature, information could be obtained by other applications from the clipboard too. There is also an option for developers to use the ***persistent pasteboard*** in order to keep data available after application termination. This leads to copied data being stored unencrypted under: *"~/Library/Application Support/iPhone Simulator/4.0.1/Library/Caches/com.apple.UIKit .pboard"*

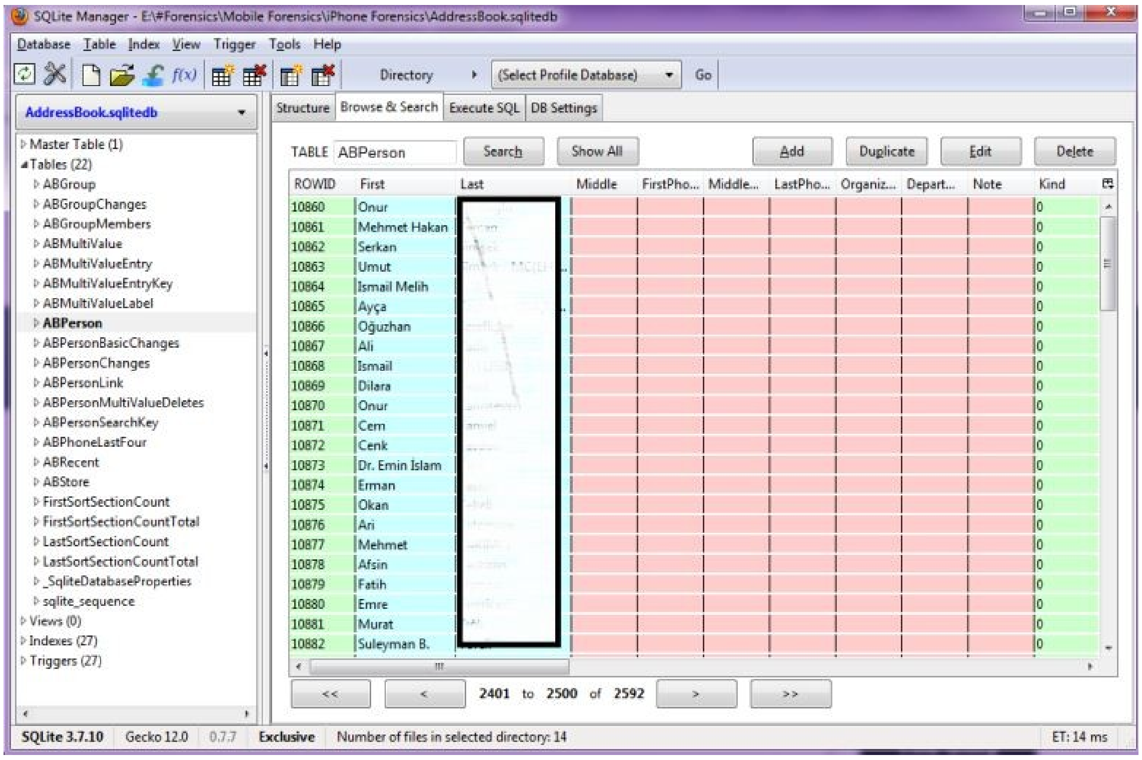
In cases that the pasteboard is being used, there is a good chance sensitive information can be obtained if no private pasteboard is being used instead. Finally, the permanent storage space on iOS is the SQLite database.

7.4 SQLite Database

Usually a well-designed iOS application stores most of the user data on the server side. In some cases though, data has to be stored on the ***client side*** (on the iOS device) and this is accomplished by using the ***SQLite database***, which also leads to ***sensitive information*** like user names and credit card numbers being saved, sometimes ***unencrypted***, on the device's file system.

To read an application's SQLite entries, any SQL client can be used like the Firefox add on "***SQLite Manager***". Moreover, in case the data is stored without encryption (*the use of SQLite storage with encryption becomes usually too complicated regarding the key management while developing an application*), much more interesting information can be found in the database **[27]**.

Here are some SQLite entries of an address book stored on an application's database unencrypted:



ROWID	First	Last	Middle	FirstPho...	Middle...	LastPho...	Organiz...	Depart...	Note	Kind
10860	Onur	[redacted]								0
10861	Mehmet Hakan	[redacted]								0
10862	Serkan	[redacted]								0
10863	Umut	[redacted]								0
10864	Ismail Melih	[redacted]								0
10865	Ayça	[redacted]								0
10866	Oğuzhan	[redacted]								0
10867	Ali	[redacted]								0
10868	Ismail	[redacted]								0
10869	Dilara	[redacted]								0
10870	Onur	[redacted]								0
10871	Cem	[redacted]								0
10872	Cenk	[redacted]								0
10873	Dr. Emin Islam	[redacted]								0
10874	Erman	[redacted]								0
10875	Okan	[redacted]								0
10876	Ari	[redacted]								0
10877	Mehmet	[redacted]								0
10878	Afsin	[redacted]								0
10879	Fatih	[redacted]								0
10880	Emre	[redacted]								0
10881	Murat	[redacted]								0
10882	Suleyman B.	[redacted]								0

Address book in SQLite [29]

This is an example of database information that can potentially be exposed unencrypted from an iOS application in its database files. There may be several other sources to look for sensitive data when analyzing an application, but the ones described in this "Data Protection" chapter, usually hold most of the valuable information and are the main targets of iOS hackers.

The rest of this document, is dedicated to an extensive break down and explanation of all the technical terms that were used throughout this research. Finally, in the conclusion chapter, the assumptions are made about the topic being discussed and further resources are also provided.

CHAPTER 8

TECHNICAL TERMS

This chapter aims to explain all the technical iOS terms that were used and referenced throughout this research. It should be used as a reference guide while studying the previous chapters and come across any of the following terms:

8.1 .ipa files

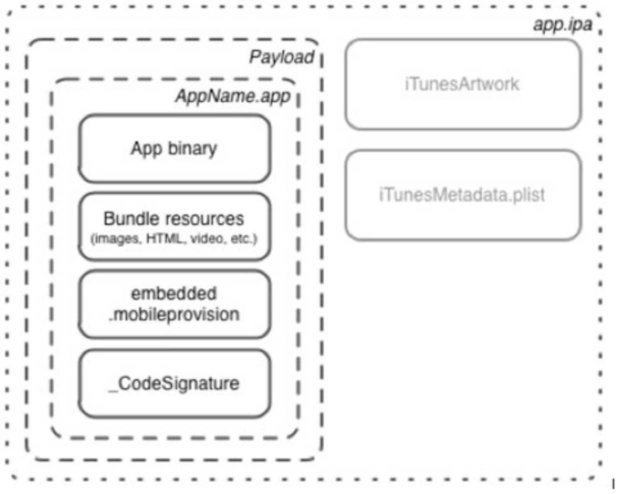
.ipa originally stood for ***iPhone Archive*** but after iOS was released it stands for ***iOS App Store Package.*** Every .ipa file is a compressed application archive including a binary file for the ARM architecture, that can only be installed on iOS devices. By changing the extension from .ipa to .zip and then unzipping it, all the contents of this archive can be viewed.

An ipa archive has a standard built-in structure for being recognized by iTunes and the App Store. This structure consists of the following:

/Payload/
/Payload/Application.app
/iTunesArtwork
/iTunesArtwork@2x
/iTunesMetadata.plist
/WatchKitSupport/WK

The Payload folder contains all the application data. The iTunes Artwork file is a 512×512 pixel PNG image, containing the application's icon being displayed in iTunes and the App Store. The iTunesMetadata.plist contains various bits of information, ranging from the developer's name and ID, the bundle identifier, copyright information, genre, the name of the application, release date and purchase date **[30]**.

The picture below graphically explains the structure of this compressed application file:



.ipa file structure [31]

An unsigned .ipa can be created by copying the folder with the extension ***.app*** from the Products folder of the application in ***Xcode*** to a folder called ***Payload*** and compressing the latter using the command:

```
zip -0 -y -r myAppName.ipa Payload/
```

It is then possible to install unsigned .ipa files on ***jail broken*** devices using third party software.

8.2 Tweak

iOS invasion cannot be separated from the development of jailbreak. The invasion is built on the basis of jailbreak. Without system-level authority (root) nothing special can be accomplished other than dealing with the flow of a specific application. Of course, free jailbreak crack patches exist on the market, but its development process is also based on the environment they are escaping from.

In the hacker community of iOS, to do crack or jailbreak development, ***tweaks*** are essential. Tweak is a variety of ***crack patch collection.*** The use of tweak as a search term on google, regarding jailbreak development information or open source crack patch code, brings up the best results.

iOS tweak is divided in two types:

- The first is released in the ***Cydia*** and requires a jailbroken device in order to be installed. Most of the .deb format installation packages will default to install a ***dynamic library*** called ***mobilesubstrate*** (see later on this chapter). Its role is to provide a ***system-level intrusion Pipeline*** on which tweak can rely on for development. The current mainstream ***tweak development tools*** are ***theos*** and ***iOSOpenDev***. The former is a compilation of Makefile framework, which provides a set of XCode project templates, it can directly use XCode development and can also be debugged. Even though this project has ***stopped updating and*** the latest version of Xcode support is not good, ***Theos*** remains the best option for developing and making your own tweaks, as community took over the update and maintenance of the whole project right now***.***
- The second type of tweak, is ***directly packaged*** into an ***ipa installation package*** which can use its own development certificate or enterprise certificate signature and can also be installed ***without jailbreak***, by directly downloading it from the developer's website. Dealing with devices without a jailbreak though, due to Permissions restrictions, there is no way to write ***system-level tweak***. These tweaks are mostly used for an application to ***modify its injection process***, and then re-sign and publish it. This is a similar process to the windows software cracking.

Summarizing, ***without escaping/jailbraking*** the machine and no mobilesubstrate dynamic system library available, there are only two options available. ***The first*** one is interacting directly with the application's library into its ***ipa file*** and using its API to achieve injection. ***The second*** is directly modifying the application's ***assembly code***.

The first option applies to more complex crack behavior, and the jailbreak tweak code can be reused. The second one, applies to cracking some if ... else ... like conditional statements or simple parts of the application's code through its assembly code.

Tweaks can be used to ***alter the implementation*** of an application method by hooking on it and are ***written*** and built by using the ***Theos*** framework that is described later on this chapter.

A great ***example*** of a ***tweak development*** with Theos for bypassing iOS' Safari restriction of viewing system files using the url bar, is described on the following link:

https://testeaxeax.blogspot.gr/2016/09/a-first-look-at-tweak-development.html

8.3 Mobile Substrate

Cydia (Mobile) Substrate, is a framework and development library used to make modifications to code written by other developers. This library is the foundation of most of the interesting hacks found on the iPhone.

It can also be used on Android (Native or Java). In any case, the first step to writing an extension is knowing where the code to be modified lives, and then making certain the modifications get loaded to those places. This will typically be done using the names of specific processes or libraries. Substrate extensions are compiled to dynamic libraries (using the extension .dylib) and are placed in the Substrate Dynamic Libraries folder on iOS/Darwin under "***/Library/MobileSubstrate/DynamicLibraries***".

It consists of 3 parts:

1. ***MobileHooker*** which is used to replace system functions.
2. ***MobileLoader*** that loads 3rd-party patching code into the running application using ***DYLD_INSERT_LIBRARIES*** environment variable.
3. ***Safe Mode*** used when an extension crashed the SpringBoard, MobileLoader will catch that and put the device into safe mode while disabling all other 3rd party extensions.

More information about its usage and configuration for iOS can be found at the following links:

(usage-manual)

http://iphonedevwiki.net/index.php/MobileSubstrate

(configuration)

http://www.cydiasubstrate.com/inject/darwin/

8.4 Jail broken device

Jail breaking is the privilege escalation process for ***removing software restrictions*** imposed by Apple on iOS while full execute and write access is obtained on all the partitions of the device. This is done by using a series of kernel patches similar to ***"rooting"*** an Android device. Jail breaking provides root access to the iOS, allowing a user to download and install additional applications, extensions, and themes that are not available on the official App Store.

When a device is booting, it loads Apple's own kernel initially. A jail broken device must be exploited and have the kernel patched each time it is being booted up. An ***untethered*** jailbreak uses exploits that are powerful enough to allow the user to reboot their device, and the kernel will be patched without the help of a computer on the next start up.

However, some jailbreaks are **tethered** meaning that they are only able to temporarily jailbreak the device during a single boot **[2]**.

8.5 Nsobjects

NSObject is the root class of the Objective-C class hierarchies, from which subclasses inherit a basic interface to the runtime system and the ability to behave as Objective-C objects. In other words, this is the base class of Objective C language and every other subclass object inherits from it. Therefore, any type of object like NSString is an Objective C subclass object that inherits from the basic NSObject class. It is like the Object class in Java programming language.

8.6 obj_MsgSend

This is a basic Objective C method that ***sends a message*** with a simple return value to an ***instance*** of a class. When it encounters a method call, the compiler generates a call to one of the functions objc_msg, objc_msg, objc_msg, or objc_msg. Hence, the actual mechanism that makes Objective C a ***runtime language*** and not pre-determined upon compilation is the use of this method.

8.7 Cydia

Cydia is a package manager mobile app for iOS that enables a user to find and install software packages on jail broken iOS devices. It also refers to the digital distribution platform for software on iOS accessed through Cydia software. Most of the software packages available through Cydia are free of charge but some require purchasing. It is like having a free version of the AppStore available for a jail broken device, equipped with third party applications that originally are not available for standard devices.

Compared to Android, it is similar to Aptoide for the end user but for developers it is a place to publish their third party applications, ranging from simple ones to jailbreak tweaks.

8.8 Theos

Theos is a ***cross-platform suite*** of tools for ***building*** and ***deploying software*** for iOS and other platforms. It was initially the "***iPhone-framework***", a project created to simplify building code from command line for iOS devices (primarily jail broken devices). It later underwent significant changes and became Theos, a flexible Make-based build system primarily for jailbreak software development that also provides complete support for building code for other supported platforms. Theos runs and can build projects for macOS, iOS, Linux, and Windows (under Cygwin or Windows Subsystem for Linux).

Information about Installation and configuration are available online at the following link:

https://github.com/theos/theos/

After installation is complete, Theos offers several template project options to start from. Start up by running its ***New Instance Creator (NIC)*** wizard with the command "***$THEOS/bin/nic.pl".*** This prompts with the information required to create a new project as shown below:

```
$ $THEOS/bin/nic.pl
NIC 1.0 - New Instance Creator
------------------------------
  [1.] iPhone/application
  [2.] iPhone/library
  [3.] iPhone/preference_bundle
  [4.] iPhone/tool
  [5.] iPhone/tweak
Choose a Template (required): 1
Project Name (required): iPhoneDevWiki
Package Name [com.yourcompany.iphonedevwiki]: net.howett.iphonedevwiki
Authour/Maintainer Name [Dustin L. Howett]:
Instantiating iPhone/application in iphonedevwiki/...
Done.
$
```

The rest of the process is clearly presented with an extensive example on the following link, that is included under the Tweak section too:

https://testeaxeax.blogspot.gr/2016/09/a-first-look-at-tweak-development.html

This cannot be fully explained, as it reaches out of the purpose of this research (which is not to teach coding of tweaks, Cycript or Objective C commands).

CHAPTER 9

CONCLUSION

This purpose of this research is to provide a step by step guide of the installation procedure and actual usage of the most essential available Open Source tools an iOS application security analyst would need to analyze and find possible security flaws on any specific iOS application.

First of all, the procedure of setting up the appropriate testing environment and reverse engineering any .ipa file or Xcode project package for further static or dynamic analysis is described. Then comes the part of the interaction with an application and the modification of its running flow by hooking into its process or doing a method swizzling to achieve the desired outcome.

In many cases, details about the reverse engineering process or the way a code snippet is written were left aside (*e.g. Cycript syntax, tweak development, Objective C code*) because this is not what this document aims to describe. This aims to be a ***guide*** about the ***Open Source*** tools the iOS application security analyst should use, the way they should be set up on the analysis environment and the crucial information and examples required to understand their functionality. A basic knowledge of the iOS operating system must be present by the reader though and some interaction with Objective C and its runtime behavior is considered enough.

After being familiar with these terms, this document can be easily used as a handbook for the tools that may come in handy by the time an iOS application must be analyzed by a security perspective.

Apple gives a great amount of effort to keep its operating system and users secure. This leads to iOS being one of the most secure mobile device operating systems and very little knowledge is publicly available about exploitation techniques and tutorials or complete and step by step guides on this topic.

The best resource can be any Chinese blog or repository website that still have valuable information available online about exploitation techniques and iOS application security analysis. Chinese hackers were responsible for the well-known ***Pegasus Exploit*** too, that can be found online (pdf file) **[32]** and is a good point to understand what it takes to fully exploit the iOS using an exploitation chain of even kernel exploits. The Chinese Pangu team could not be left aside too, as it is responsible for most of the jail brake exploits published for almost any iOS version **[33]**. To sum up, further research should also be based on Chinese resources along with any of the following resources provided at the end of this document.

There are some great books too, being listed below, in order to fully understand the iOS and the way applications can be hacked or even study known techniques on how a penetration test is delivered to them:

- **iOS Hacker's Handbook** (*Charlie Miller*, *Dion Blazakis*, *Dino DaiZovi*, *Stefan Esser*, *Vincenzo Iozzo*, *Ralf-Philip Weinmann*).
- **Mac OS X and iOS Internals: To the Apple's Core** (*Jonathan Levin*).
- **iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices** (*Andrew Hoog*, *Katie Strzempka*).
- **Hacking and Securing iOS Applications: Stealing Data, Hijacking Software, and How to Prevent It** (*Jonathan Zdziarski*)

Summarizing, this thesis proves that everyone interested in iOS security can easily get started, even with minimum cost (when the iOS simulator is used instead of an actual device), by setting up a complete Open Source testing environment. Another important note is that Apple already made iOS Open Source except the kernel that is nothing more than a Darwin kernel, like the one used in the OSX and as newer versions come up, it is easily observed that version numbers of these kernels are way too close, so is their source code. Rumors has it that iOS kernel will be Open Sourced too. In any way, there is a good chance that if a kernel exploit for the OSX kernel is found, it will be easily applicable to the iOS kernel too as newer versions are being released. This document is a simple entry point for security research on the iOS platform and it does not describe kernel exploitation, which already is a hot security topic due to its advanced security mechanisms.

As far as the iOS' programming language is concerned, this research is fully based on Objective C. Everything seems that Apple is planning on replacing it though, including the C language used for the kernel with the Swift programming language. All the described tools currently work with Objective C and even on iOS 11 they can still be used. On the other hand, a shift to the Swift language is going to be a necessary step for every iOS security researcher and developer in the future.

Finally, it should be acknowledged that all the information presented in this document are a product of online research and personal effort to implement every step on each section or collect and even take the actual screen shots in many cases using my own testing environment. For every information used by another source/author a corresponding reference is available under the "***Resources***" chapter.

Here are some more interesting resources for further research and in-depth analysis of iOS application security. One of the greatest is Prateek Gianchandani's website of Damn Vulnerable iOS Application (DVIA) **[34]**, a vulnerable by design iOS application for testing purposes. A wide variety of papers and presentations regarding iOS security researches can also be found on securitylearn.com **[35].** Last but not least, is the infosecinstitute website where several tutorials and topics are analyzed regarding iOS security **[36].**

RESOURCES

[1] 8/12/2015 iOS Exploits - iOS - EDG Confluence:
https://wikileaks.org/ciav7p1/cms/files/iOS%20Exploits%20-%20iOS%20-%20EDG%20Confluence.pdf

[2] ISBN-13: 978-1449318741 by Jonathan Zdziarski (Author): Hacking and Securing iOS Applications: Stealing Data, Hijacking Software, and How to Prevent It 1st https://www.amazon.com/Hacking-Securing-iOS-Applications-Hijacking/dp/1449318746

[3] iDevice Central YouTube channel https://www.youtube.com/user/leonteiz

[4] Apple iOS Security Guide https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[5] Usenix: Jekyll on iOS https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_wang-updated-8-23-13.pdf

[6] Application entitlements – developer.apple.com
https://developer.apple.com/library/content/documentation/IDEs/Conceptual/AppDistributionGuide/AddingCapabilities/AddingCapabilities.html

[7] CVE-2016-7630 iOS exploit via Wi-Fi – Marco Grassi (Blackhat Asia 2017)
https://www.youtube.com/watch?v=bP5VP7vLLKo

[8] iOS Hacker's Handbook – Charlie Miller - https://www.amazon.com/iOS-Hackers-Handbook-Charlie-Miller/dp/1118204123

[9] Otool – iphonedevwiki.net
http://iphonedevwiki.net/index.php/Reverse_Engineering_Tools#otool

[10] class-dump tool – Steve Nugard
http://stevenygard.com/projects/class-dump/

[11] iOS application security analysis - getting class information
http://highaltitudehacks.com/2013/06/16/ios-application-security-part-2-getting-class-information-of-ios-apps/

[12] Mobile Application Penetration Testing – Vijay Kumar Velu
https://www.packtpub.com/application-development/mobile-application-penetration-testing

[13] iNalyzer tool – GitHub
https://github.com/appsec-labs/iNalyzer

[14] infosecinstitute.com – Static Analysis of IOS Applications using iNalyzer
http://resources.infosecinstitute.com/part-15-static-analysis-of-ios-apps-using-inalyzer/

[15] developer.apple.com – Instruments User Guide
https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/index.html

[16] developer.apple.com – Launching Instruments
https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/LaunchingInstruments.html

[17] clang-analyzer
https://clang-analyzer.llvm.org/xcode.html

[18] Getting started with clang analyzer – lowlevelbits.org

https://lowlevelbits.org/getting-started-with-llvm/clang-on-os-x/

[19] developer.apple.com – Dtrace

https://en.wikipedia.org/wiki/DTrace

[20] highaltitudehacks.com - iOS application security – understanding the objective c runtime

http://highaltitudehacks.com/2013/06/16/ios-application-security-part-3-understanding-the-objective-c-runtime/

[21] cocoapods.org – Get Started with FLEX

http://cocoapods.org/?q=FLEX

[22] itony.me – FLEX with iPhone applications

https://itony.me/774.html

[23] Use Inspective C to view the call stack – bbs.iosre.com

http://bbs.iosre.com/t/inspectivec/1584

[24] Cycript

http://www.cycript.org/

[25] resources.infosecinstitute.com - Runtime analysis using Cycript

http://resources.infosecinstitute.com/ios-application-security-part-4-runtime-analysis-using-cycript-yahoo-weather-app/#gref

[26] www.securitylearn.net – Runtime analysis with Cycript

http://www.securitylearn.net/tag/pentest-iphone-applications/

[27] Keyboard Cache - securitylearn.net

http://www.securitylearn.net/tag/iphone-keyboard-cache/

[28] thehackpot.blogspot.gr - Unintended Data leakage

http://thehackpot.blogspot.gr/2015/08/unintended-data-leakage-application.html

[29] iOS forensics – bilgiguvelnik.net

http://www.bilgiguvenlik.net/2012/05/ios-forensic.html

[30] IPA File Format – theiphonewiki.com

https://www.theiphonewiki.com/wiki/IPA_File_Format

[31] The Structure of an iOS App – gowithfloat.com

https://gowithfloat.com/2011/11/re-signing-an-ios-app-without-xcode/

[32] outlook – Pegasus exploit analysis

https://info.lookout.com/rs/051-ESQ-475/images/pegasus-exploits-technical-details.pdf

[33] team Pangu

http://en.pangu.io/

[34] DVIA – Prateek Gianchandani

http://damnvulnerableiosapp.com/

[35] iOS hacking collection – securitylearn.net

http://www.securitylearn.net/2012/06/13/iphone-hacking-resource-collection/

[36] http://resources.infosecinstitute.com/