iOS Forensics & Data Leakage

UNIVERSITY OF PIRAEUS

Department of Digital Systems

Postgraduate Program Digital Systems Security

by

Menesidis Michail

Piraeus, October 2016

Supervisor

Christos Xenakis
Associate Professor, University of Piraeus

Approved by the Examining Committee

Christos Xenakis, Supervisor
Associate Professor, University of Piraeus

Konstantinos Lambrinoudakis
Associate Professor, University of Piraeus

Christoforos Ntantogian
Adjunct Lecturer, University of Piraeus

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Associate Professor Christos Xenakis who tirelessly helped me to prepare my master thesis. I would like to thank him for his encouragement in my research, his guidance, patience, help and support and for giving me enough space for exploring my ideas. His guidance helped me during the time of research and writing of this thesis. His enthusiasm and inspiring ideas have improved this work immeasurably.

Besides my supervisor, I would like to thank the rest of my thesis committee: Konstantinos Lambrinoudakis and Christoforos Ntantogian, for their encouragement, insightful comments, and hard questions.

Contents

Abstract

During the last years the evolution of mobile devices was unprecedented. With this increasing popularity of mobile devices and apps, security and privacy concerns have emerged as a salient area of inquiry for researchers. In this thesis, we focus on iOS devices due to, not only the lack of research in this area, but also by the Apple's restrictions which is far more challenging to investigate compared to Android. More specifically, we study the data leakage and privacy issues posed to iPhone users. This research is aim to gain sensitive information from a mobile device to point out that various iOS apps have data leakage issues. Using the latest forensic tools we present methodologies to acquire data from many renown apps. Our objective is to improve user's privacy highlighted the missing security mechanisms in order to avoid data and privacy leaks.

List of Figure

List of Tables

Abbreviations

ACL	Access Control List
API	Application Program Interface
GID	Group ID
GUI	Graphical User Interface
IDE	Integrated Development Environment
IDS	Intrusion Detection System
IV	Initialization Vector
JAR	Java Archive
NBG	National Bank of Greece
NIB	NeXT Interface Builder
SSID	Service Set Identifier
SSL	Secure Socket Layer
TLS	Transport Layer Security
UI	User Interface
UID	Unique Identifier
URL	Uniform Resource Locator
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access 2
XIB	XML Interface Builder
XML	Extensible Markup Language

Chapter 1 - Introduction

This chapter provides an introduction to the context of this research, by providing an overview of the main issues associated with the subject of the study. Then the aims and objectives of the research are established, followed by the research methodology. Finally, a brief summary of each chapter are also provided.

1.1 Introduction

Over the last years, there is no doubt that mobile phones have been rapidly evolved. The latest generations of smartphones are essentially small computers. These smartphones not only they offer the possibility to make phone calls and to send messages, but also they are a communication and entertainment platform for users to surf the web, send emails, and play games. Mobile phones are also ubiquitous, and allow anywhere, anytime access to information [1]. According to market research [2], the iPhone is one of the most common smartphones on the market today. As these devices grow in popularity, so does the interest in accessing all data these devices contain. However, the reputation and sophistication of smartphones have also enlarge concerns about data leakage and the privacy of users who operate these devices. These concerns have been exacerbated by the fact that the massive increased use in mobile devices brings a corresponding growth in mobile applications. For instance, mobile applications typically cache data locally for performance and availability reasons compared to traditional applications where data is stored on a tightly controlled server. Moreover, a common practice of developers is to use third-party libraries. All the above could lead to a potential security risk. In addition, mobile devices are by definition portable making them prone to theft and loss which opens up additional avenues for compromising data stored on the device. Thus, mechanisms are required to properly protect sensitive data against malicious applications. For that reason, Apple has introduced a vetting process. This vetting process should ensure that all applications conform to Apple's policy before they can be offered via the App Store. Apple screens each uploaded application to check whether it contains malicious code or violates Apple's privacy policy. Despite of these mechanisms there are plenty of apps that might leak personal data without the consent of the users.

In this thesis, we study the data leakage and privacy issues posed to iPhone users. The aim of this thesis is to present several methods that allow us to gain sensitive information from a mobile device. For our research we used various forensic tools to leak information such as keychain passwords, session ids etc. In addition, we provide various case studies and our evidence suggests that many renown apps, missing the appropriate security mechanisms in order to avoid data and privacy leaks. Our intention is to enhance the privacy and confidentiality of mobile users and specifically iPhone users.

1.2 Aims and Objectives

The general aim of this thesis has been to address the challenging issue of data and privacy leakage in iOS. Researchers until now tend to investigate the open-source Android instead of iOS. In this thesis, we motivated by the Apple's limitations and the closed-source iOS platform, which makes more difficult to investigate compared to Android.

Two primary aims were set for this thesis:

1. To evaluate significant importance apps such as mobile banking apps regarding to data and privacy leakage
2. To show various methodologies to collect sensitive data from apps

The aforementioned two aims were set due to the emerging security threats that mobile devices are facing and have profoundly changed the security designs of modern operating systems (OS).

1.3 Research Methodology

The review of the current literature will contribute to identifying the current state of the art on the field of data leaks in iOS apps. The current literature found in various research papers, surveys and books. In order to conduct our research methodology we lead on the existing techniques and evaluate various approaches for data and privacy leaks. From our experiments we demonstrate that the data leakage is still a problem in iOS devices and various security improvement needs to be done.

1.4 Thesis Outline

The structure of this research is the following.

- Chapter 2 explain the complex background of iOS forensics. This Chapter provides an overview of state-of-the-art techniques and an extensive literature review on the relevant aspects of the area. More specific, gives a thorough description of the related work and background, and finally compares the currently existing contrasting technologies.
- Chapter 3 provides the basic knowledge about iOS security. More precisely, the chapter gives a detailed description about the system security, the data protection, the network security and the application security.
- Chapter 4 presents a plethora of forensic tools and practical methods to collect information about iOS apps.
- Chapter 5 provides the methodology used for data leakage in iOS devices. More specifically, five scenarios have been conducted with escalated significance to extract information.
- This thesis ends with Chapter 6 which provides a summary of the main conclusions drawn for the whole research effort. In addition this chapter lists the main observations and contributions of our work and states research limitations.

Chapter 2 - Literature Review

This chapter introduces the state of the art and summarises the background relevant to this thesis for understanding the subsequent chapters. More specifically, the goal is to provide the reader with the relevant background information about iOS data and privacy leakage. Nowadays, the research community has contributed a lot of work to analyze and track how application leaks private data. A plethora of tools (see section 2.1) have been developed, which inspect applications for potential privacy leaks. A first step towards protecting user privacy is detecting apps that access sensitive data.

The two major contributions in this area are PiOS [1] and TaintDroid [3] for iOS and Android respectively. Authors in [1] employs static analysis of the App binaries, whereas TaintDroid [3] uses a dynamic taint analysis and is designed for the Android OS to monitor privacy leaks in real time. More specifically, the authors of PiOS construct a control flow graph of downloaded and decrypted iOS apps, to determine if there is a path from sources of sensitive data to destination where the information can leave the device. Their analysis is based on over 1400 iOS apps. From their results, they detected that many apps are sending the location and the address book, while more than a hundred apps sending the UDID. In contrast, TaintDroid [3] proposes modifying the Android OS such that 'taint' values can be assigned to sensitive data and their flow can be continuously tracked through each app execution, raising alerts when they flow to the network interface. The principal component of TaintDroid is variable level tracking, which is implemented in the Dalvik interpreter. TaintDroid imposes a runtime overhead because it runs continuously for all applications and hence the authors tested it on a set of thirty popular Android apps reporting that many of them leak privacy sensitive data.

Another work is LeakMiner [4] which is a points-to based static analysis for Android. It models the Android lifecycle to handle callback methods and the paper states that an app can be analyzed in 2.5 minutes on average. However, LeakMiner is context-insensitive which may lead to false positives. The authors of AndroidLeaks [5], present an automatic tool for detecting potential privacy leaks on Android system. AndroidLeaks adopts the existing analysis framework on java to the android applications by translating them into a Java Archive (JAR) file. However, the translation from Android application could lead to an incorrect analysis result.

More recently, authors in [6] compares and examines the difference in the usage of security/privacy sensitive APIs for Android and iOS. Their analysis revealed that iOS Apps access more privacy-sensitive APIs than Android Apps. Moreover, the authors in [7] study the RATP app for Paris subway using a combination of static and dynamic analysis techniques. They find that in addition to device identifiers, the RATP app transmits a list of apps running on the smartphone to third parties targeting mobile audiences. Their system is able to capture to which server the data is actually sent and thereby, is able to distinguish between first and third-party. Their work detects the private data leakage even if TaintDroid is not able to detect it.

The ProtectMyPrivacy (PMP) project [8] shows that access to privacy sensitive information such as the unique identifier, or user location or even the address book is ordinary in iOS apps. They developed a crowdsourced recommendation engine, which provides app privacy recommendations based on collected and analyzed user protection decisions. Apart from [1], the [9] is one work of the limited research area on iOS platform. The [9] addresses the open problem of preventing (not only detecting) privacy leaks and simultaneously strengthening security against runtime attacks on iOS. They present the design and implementation of PSiOS, a tool that features a novel policy enforcement framework for iOS. It provides fine-grained, application-specific, and user/administrator defined sandboxing for each third-party application without requiring access to the application source code. However, both the aforementioned research works don't provide any insight about the potential private data leakage over the network: they just deal with mere access to private data.

More recently, authors in [10] investigate the risk of privacy leakage through mobile analytics services. They show how an external adversary can extract individual's profile and mobile applications usage information, through mobile analytics services such as Google Mobile App Analytics and Flurry. In addition, they present a way to manipulate the user profiles to influence the ads served to users' devices by exploiting the vulnerability of analytics services. Last but not least the authors in [11] developed IccTA tool which statically analyzes app sets to detect flows of sensitive data. IccTA uses a single-phase approach that runs the full analysis monolithically.

Table 1 below summarises all the aforementioned related works for privacy and data leak detection frameworks. Our table is based on [12].

Tools/Frameworks	Platform	Technique	No. of Tested Apps	Year
TaintDroid [3]	Android	Dynamic taint analysis	30	2010
PiOS [1]	iOS	Static Data Flow	1,407	2011
LeakMiner [4]	Android	Static Data Flow	1750	2012
AndroidLeaks [5]	Android	Static Data Flow	25,976	2012
Comparing Mobile Privacy Protection [6]	Android & iOS	Static analysis techniques	2600	2013
RATP [7]	Android & iOS	Static and dynamic analysis techniques	-	2013
PMP [8]	iOS	Crowdsourcing	685	2013
PSiOS [9]	iOS	Static analysis & Binary rewriting and runtime enforcement techniques	-	2013
Information Leakage through Mobile Analytics Services [10]	Android & iOS	Static analysis techniques	-	2014
IccTA [11]	Android	Static intra component Analysis	3000	2014

Table 1 - Privacy and data leak detection frameworks

Chapter 3 - Understanding iOS Security

3.1 System Security

The foundation of the iOS platform relies on its System Security. This consists of the Secure Boot Chain, System Software Authorization, Secure Enclave, and Touch ID [13].

- *Secure Boot Chain* is a chain, or a sequence of trusted events that occur during the boot process to ensure integrity. Boot ROM is the read-only memory that used from the application processor after the iOS devices is turned on. The Boot ROM contains Apple's Root public key, which is used to verify the authenticity of the Low-Level Bootloader. After this first step in the chain of trust, each step ensures that the next is signed by Apple. This process ensures that the lowest levels of software are not tampered and allows iOS to run only on validated Apple devices. If the Secure Boot Chain is unable to complete, where any element fails authenticity, the iOS device will display a message to the user and then enter Device Firmware Upgrade mode.
- *System Software Authorization* prevents iOS devices from being downgraded to run older, insecure code, which could lead on exploitation. In addition, verify that all code executed on the device is signed by Apple Root CA public key. During an iOS upgrade iTunes connects to Apple's Authorization server to validate the upgrade process and the device.
- *Secure Enclave* is a coprocessor and it used to provide all cryptographic operation for Data Protection key management and maintains the integrity of Data Protection. Also, is responsible for processing fingerprint data from the TouchID sensor, determining whether there is a match or not against registered fingerprints to enable access on behalf of the user.
- *Touch ID* is the fingerprint sensing system that makes secure access to the device. Apart from the required passcode Touch ID is a complementary security mechanism to read fingerprint data avoiding the inconvenience of a passcode-based lock. This additional layer of security has a 1 in 50,000 chance of being matched to someone other than the

owner. However, Touch ID allows only five unsuccessful fingerprint match attempts before the user is required to enter a passcode to obtain access.

3.2 Encryption and Data Protection

The secure boot chain, code signing, and runtime process security ensure that only trusted code and apps can run on a device. iOS has encryption and data protection features to protect user's privacy even if third parties of the security infrastructure have been compromised.

3.2.1 Hardware Security Features

Every device, starting from iPhone 3GS, has a dedicated AES 256-bit crypto engine built in between the flash storage and the main system memory. The purposes of this processor are to accelerate the encryption and decryption operations and to protect user data so that they remain encrypted on the device's flash memory. Additionally, every device's unique ID (UID) and group ID (GID) AES 256-bit keys are written directly into the application processor during manufacturing and cannot be read directly.

3.2.2 Data Protection

3.2.2.1 File Data Protection

File Data Protection is a protection mechanism that iOS uses to protect files and data on the device. More specifically, Apple uses this Data Protection technology to further protect data stored in flash memory on the device. This technology allows the device to respond to common events such as incoming phone calls, but also enables a high level of encryption for user data. All the preinstalled apps, such as Messages, Mail, Calendar, Contacts, Photos, and Health data values use Data Protection by default, and third-party apps installed on iOS 7 or later receive this protection automatically. Data Protection is implemented by constructing and managing a hierarchy of keys, and builds on the hardware encryption technologies built into each iOS device. Data Protection is controlled on a per-file basis by assigning each file to a class; accessibility is determined by whether the class keys have been unlocked [13].

3.2.2.2 Architecture Overview

Every time a file on the data partition is created, Data Protection creates a new 256-bit key and gives it to the hardware AES engine, which uses the key to encrypt the file as it is written to flash memory using AES CBC mode. The process of encryption use SHA-1 hash - with an initialization vector (IV) - of the per-file key which is wrapped with one of several class keys. The wrapped per-file key is stored in the file's metadata. When a file is opened, its metadata is decrypted with the file system key, revealing the wrapped per-file key and a notation on which class protects it. The per-file key is unwrapped with the class key, then supplied to the hardware AES engine, which decrypts the file as it is read from flash memory. All wrapped file key handling occurs in the Secure Enclave; the file key is never directly exposed to the application processor. At boot, the Secure Enclave negotiates an ephemeral key with the AES engine. When the Secure Enclave unwraps a file's keys, they are rewrapped with the ephemeral key and sent back to the application processor. The metadata of all files in the file system is encrypted with a random key, which is created when iOS is first installed or when the device is wiped by a user. The file system key is stored in Effaceable Storage. Since it's stored on the device, this key is not used to maintain the confidentiality of data; instead, it's designed to be quickly erased on demand. Erasing the key in this manner renders all files cryptographically inaccessible. The content of a file is encrypted with a per-file key, which is wrapped with a class key and stored in a file's metadata, which is in turn encrypted with the file system key. The class key is protected with the hardware UID and, for some classes, the user's passcode. This hierarchy provides both flexibility and performance. [13].

3.2.2.3 Passcodes

By setting up a device passcode, the user automatically enables Data Protection. iOS supports six-digit, four-digit, and arbitrary-length alphanumeric passcodes. In addition to unlocking the device, a passcode provides entropy for certain encryption keys. The passcode is entangled with the device's UID, so brute-force attempts must be performed on the device under attack. The stronger the user passcode is, the stronger the encryption key becomes. Touch ID can be used to enhance this equation by enabling the user to establish a much stronger passcode than would otherwise be practical. This increases the effective amount of entropy protecting the encryption keys used for Data Protection, without adversely affecting the user experience of unlocking an

iOS device multiple times throughout the day. To further discourage brute-force passcode attacks, there are escalating time delays after the entry of an invalid passcode at the Lock screen. [13].

3.2.2.4 Data Protection Classes

When a new file is created on an iOS device, it's assigned a class by the app that creates it. App developers can use each class with different policies to determine when the data is accessible. The basic classes and policies are the following four [13]:

- Complete Protection (NSFileProtectionComplete): The class key is protected with a key derived from the user passcode and the device UID. Shortly after the user locks a device the decrypted class key is discarded, rendering all data in this class inaccessible until the user enters the passcode again or unlocks the device using Touch ID.
- (NSFileProtectionCompleteUnlessOpen): Some files may need to be written while the device is locked. A good example of this is a mail attachment downloading in the background.
- Protected Until First User Authentication (NSFileProtectionCompleteUntilFirstUserAuthentication): This class behaves in the same way as Complete Protection, except that the decrypted class key is not removed from memory when the device is locked.
- No Protection (NSFileProtectionNone): This class key is protected only with the UID, and is kept in Effaceable Storage. Since all the keys needed to decrypt files in this class are stored on the device, the encryption only affords the benefit of fast remote wipe.

3.2.2.5 KeyChain Data Protection

The Keychain is Apple's implementation of a secure storage for sensitive information such as login tokens, passwords, cryptographic keys, and digital certificates.

The keychain is implemented as a SQLite database stored on the file system. Keychain data is protected using a class structure similar to the one used in file Data Protection. These classes have behaviors equivalent to file Data Protection classes, but use distinct keys and are part of

APIs that are named differently.

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

Figure 1 - Keychain Data Protection Classes [13]

In addition, Keychain can use access control lists (ACLs) to set policies for accessibility and authentication requirements. Items can establish conditions that require user presence authentication such as Touch ID or passcode. ACLs are evaluated inside the Secure Enclave and are released to the kernel only if their specified constraints are met.

3.3 Network Security

Apple, in order to achieve secure communication adopted proven technologies and the latest standards for both Wi-Fi and cellular data network connections. For instance, Apple has incorporated many of the known solutions used in secure networking into iOS such as VPN, SSL/TLS transport encryption, and WEP/WPA/WPA2 wireless encryption and authentication. iOS achieves a reduced attack surface by limiting listening ports and removing unnecessary network utilities such as telnet, shells, or a web server compared to other platforms that use intrusion detection systems (IDS) to protect open communications. Many preinstalled apps such as Safari, Calendar etc. are use Transport Layer Security (TLS) to enable an encrypted communication channel between the device and network services.

3.4 Application security

On an application level, App Store applications are run in a sandbox. *Sandboxing* refers to an environment where code is deemed untrusted and is therefore isolated from other processes and resources available to the operating system. Apple's sandbox limits the amount of memory and CPU cycles an application can use, and also restricts it from accessing files from outside of its dedicated home directory. Another way of restricting the resources of an application is via signing to police the binary code allowed to run on the device. In order for an application to be permitted to run must be signed by Apple, as already aforementioned, to ensure that application has not been modified from the original binary.

Chapter 4 - Collect Information

In this section we present various tools and methods to gather information about iOS apps. More specifically the first subsection present the most known data leakage detection tools. The aim of this section is to give users an overview of the current and future research in data leakage detection tools. Therefore, this section do not provide details about the implementation of these frameworks. We focus on iOS forensic tools such as Clutch, Cycript, etc.. The following subsection explain in detail all the selected tools, whereas in the next section a typical usage is presented (see Chapter 5). The second subsection presents how to gather data stored on iPhone such as databases, plist files etc.

4.1 Data Leakage Tools

4.1.1 Clutch Tool

Clutch is a high-speed iOS decryption tool and is supported on iPhone, iPod Touch, and iPad as well as in all iOS versions, architecture types, and most binaries [14]. The clutch tool receives as input the application name and then decrypt it and store the decrypted IPA file in the /var/root/Documents/Cracked/ folder.

4.1.2 ibtool Tool

ibtool is a tool implemented in Python that attempts to reverse engineer the iOS Nib format (used for storing compiled interface files) [15]. Currently, ibtool supports only compiling XIB and storyboard files and printing NIB files in a readable way. For example, XIB is an XML Interface Builder which is a software application that allows you to develop GUI (Graphical User Interface) with the help of Cocoa and carbon APIs. The generated files are either stored as NIB or XIB files. However, ibtool works only with Interface Builder documents for iOS and not for OS X. In addition, the set of Interface Builder features supported by this tool is very limited, and requires specific functionalities to be manually added, so certain usages of unimplemented views, scenes, layout constraints, or size classes may fail to compile or result in NIBs that are missing functionality.

4.1.3 otool Tool

Otool is a debugging and analysis tool. More specifically, the otool utility (object file displaying tool), which also exists on the Mac OS X desktop, has been ported over to the ARM architecture, providing a number of mechanisms to display information about object files and dynamic libraries. This useful utility can be used to determine memory offsets and sizes of segments, object encryption, list dynamic dependencies, and much more. It can be combined with a debugger, such as *gdb*, to decrypt and analyze your application, and can even be used to disassemble some or all of your application [16].

4.1.4 Cycript Tool

Cycript is an implementation of JavaScript that can interact with Objective-C classes and objects. More precisely, we can write either Objective-C or javascript or even both in a particular command. One of the most useful functions of Cycript is its ability to attach directly to a process, much like *gdb*, and alter the state of the running application. With Cycript, you can manipulate existing objects already in your application's memory, or instantiate new objects, such as new view controller classes or windows. Cycript can access and change instance variables directly, send and intercept messages, access the run loop, override methods, and walk through an object's internal methods, properties, and instance variables. Cycript can be used to easily hijack and manipulate poorly written applications to bypass authentication screens, circumvent sanity checks, and perform a number of other hacking activities to make an application malfunction. [16]. As far as the usage on iOS application is concerned, here are some of the advantages of using Cycript:

1. We can hook into a running process and find the names of all classes being used, i.e the view controllers, the internal and third party libraries being used and even the name of the Application delegate.
2. For a particular class, i.e View Controller, App delegate or any other class, we can also find the names of all the methods being used.
3. We can also find the names of all the instance variable and their values at any particular time during the runtime of an application.
4. We can modify the values of the instance variable during runtime.

5. We can perform Method Swizzling, i.e replace the code of a particular method with some other implementation.
6. We can call any method in the application during runtime without it being in the actual code of the application.

4.1.5 KeyChain Dumper Tool

One of the most popular tools for dumping information from the keychain is Keychain dumper [17]. This tool contains a *keychain_dumper* binary which is allowed to be accessed by an application in the keychain if is specified in its entitlements. This binary is signed with a self-signed certificate with *wildcard* entitlements and hence it is able to access all the keychain items. There could also have been other ways to make sure all the keychain information is granted, like having the entitlements file contain all the keychain access groups or using a specific keychain access group that provides access to all the keychain data.

4.2 Information stored on device

4.2.1 sqlite database

Apple iOS devices make heavy use of database files to store information such as address book contacts, SMS messages, email messages, and other data of a sensitive nature. This is done using the SQLite database software, which is an open source, public domain database package. SQLite databases typically have the file extension .sqlitedb, but some databases are given the .db extension, or other extensions as well. It is important to note that the Core Data framework internally uses Sql queries to store its data and hence all the files are stored as database files. In Figure 24 we show how to find all the .db files.

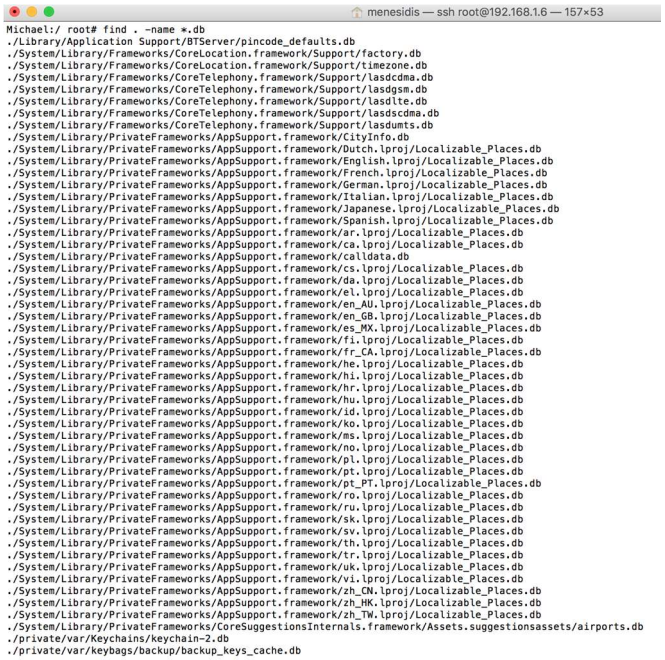


Figure 2 - Find .db files

As we can see we identify a list of all the databases files stored within the device. The contacts database would be of paramount importance because it contains sensitive data of the user. In this scenario we obtain the contacts of the user.

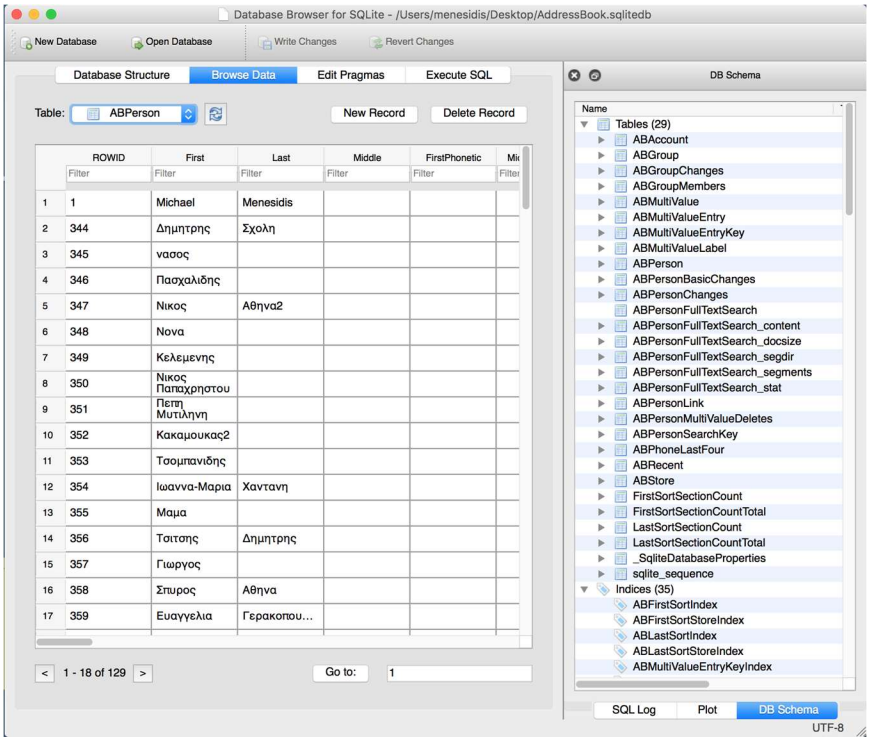


Figure 3 - Contacts of user

4.2.2 Plist files

Plist files are structured text files that are used for storing various settings and configuration for a particular app. Since the information is stored in a structured way in a plist file in key-value pairs, it is very easy to change this information and hence developers sometimes end up storing more information in these files than it should actually be used for. Even on a non-jailbroken device, plist files can be extracted by using the tool iExplorer. This tool gives access to the iPhone in disk mode and allows to browse the folders on the iPhone directly. In general every application stores the plist files inside library preferences folder.

4.2.3 Keychain mechanism

As already mentioned, the apps are isolated from other processes, known as sandbox environment. That doesn't mean we don’t have access to some security-critical services. The most common example is Apple’s keychain. Keychain is Apple’s credential management service, via which an app can store the user’s passwords, secret keys and certificates there. These keys will be automatically used after the user unlock the keychain through her password (single-sign on authentication). When the keychain is locked, all the credentials are encrypted and no one can access their content. Essentially, a default keychain is created for each user account and serves most system services and many popular apps. It is automatically unlocked whenever the user logs in, if its password is identical to that for login. Even though, keychain is not included in the Apple’s sandbox, can be considered as a secure storage system that provides a strong isolation between apps. Unless is permitted by the app’s creator in the access control list (ACL), each app does not have access to another’s keychain item [18].

4.2.4 NSUserDefaults

One of the most popular ways of saving data like user preferences or properties in an application, is by using NSUserDefaults. The information stored in NSUserDefaults persists even if you close the application and start it again. One typical example of saving information in NSUserDefaults is the logged in state of the user. We can save the logged in state of the user with BOOL values (YES or NO) in NSUserDefaults so that when the user closes the application and starts it again, the application can fetch data from NSUserDefaults and display different UI to the user depending on whether he is logged in or not. Other applications also use this feature to save

confidential information like the user's access token so that the next time the application launches, they can just use that access token to authenticate the user again. What most people do not realize is that the data saved by NSUserDefaults is not encrypted and can be easily viewed from the application bundle. It is stored in a plist file with the name as the *bundle Id* of the application.

Chapter 5 - Research Methodology

This chapter introduces the used methodology. In previous chapter we presented various forensic tools, now we use them to conduct various research scenarios. More precisely, we conducted five data leakage scenarios in different applications with escalated importance on iOS 9.0. But before we start our scenarios, there are two important things that we have to mention.

- The apps that come preinstalled with the device are located in:
    - /Applications
- The apps that we download from the Appstore are located in :
    - /var/mobile/Containers/Data/Application

5.1 Scenario 1 - 2048 App

In this scenario, we aim to collect information from the 2048 app, using the aforementioned Clutch tool (see Chapter 4.1.1). Firstly, we will try to gather information related to the app source code, such as libraries used and UI information etc. Figures below presents a typical usage of this tool. For instance, Figure 4 depicts all the possible apps choices to crack, while Figure 5 shows the selected app for our case is 2048 app.

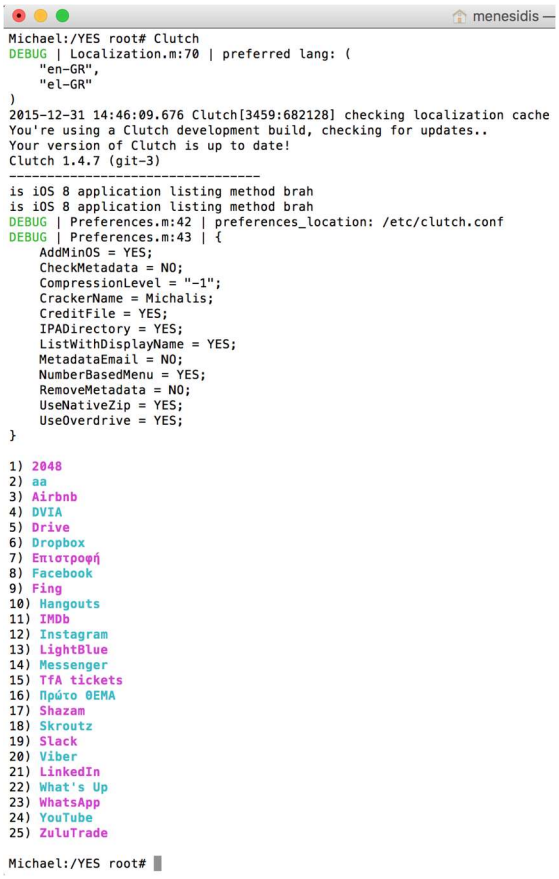


Figure 4 - Clutch choices

Figure 5 - Selected App

After cracking is finished, Clutch shows the location of the saved .ipa file. The .ipa file is an iOS application archive file which stores an iOS app [19]. Each .ipa file includes a binary for the ARM architecture (e.g. little-endian like x86) and can only be installed on an iOS device. Figure 6 presents the aforementioned procedure, while Figure 7 shows the leaked images of the app.

```
DEBUG | ApplicationLister.m:336 | cracked app ok
DEBUG | ApplicationLister.m:337 | this crack lol 207
DEBUG | Cracker.m:336 | Saved cracked app info!
        YES/2048-v2.0.7-Michalis-iOS6.0-(Clutch-1.4.7).ipa
elapsed time: 4.67s

Applications cracked:

2048

Total success: 1   Total failed: 0
Michael:/YES root#
```

Figure 6 - Path of .ipa file

```
Michael:/YES/YES root# unzip 2048-v2.0.7-Michalis-iOS6.0-\(Clutch-1.4.7\).ipa -d GameOf2048Folder
Archive:  2048-v2.0.7-Michalis-iOS6.0-(Clutch-1.4.7).ipa
   inflating: GameOf2048Folder/Payload/.com.apple.mobile_container_manager.metadata.plist
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon57x57.png
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon57x57@2x.png
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon60x60@2x.png
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon60x60@3x.png
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon72x72@2x~ipad.png
   inflating: GameOf2048Folder/Payload/2048.app/AppIcon72x72~ipad.png
```

Figure 7 - Unzip .ipa file

After we successfully dump the application file we may observe some .nib files which typical used in older version of iOS. A nib file is a special type of resource file that you use to store the user interfaces of iOS and Mac apps [20]. A nib file is an Interface Builder document. In our case the view controller (set of views) is called "ZTMT4DashboardViewController". To investigate it further, we can use the ibtool. Figure 8 show a usage example of this tool, where we can see that there are several leaks related to UI such as the dimensions of view, the autoresizingMask property which controls how a view responds to changes in its parent view's bounds etc.

```
ibtool-master — -bash — 157×59
MacBook-Pro-toutes-Menesidis:ibtool-master menesidis$ sudo python ibtool.py --dump /Users/menesidis/Desktop/Zulutrade/Payload/ZuluTrade.app/ZTMT4DashboardVie
wController.nib
Prefix: NIBArchive
Headers: 1
readClasses: Mystery value: 7 ( UIColor )
  0: NSObject
        UINibTopLevelObjectsKey = @7
        UINibObjectsKey = @9
        UINibConnectionsKey = @4
        UINibVisibleWindowsKey = @6
        UINibAccessibilityConfigurationsKey = @6
        UINibTraitStorageListsKey = @6
        UINibKeyValuePairsKey = @6
  1: UIRuntimeOutletConnection
        UILabel = @11
        UISource = @5
        UIDestination = @12
  2: NSString
        NS.bytes = IBFilesOwner
  3: UIProxyObject
        UIProxiedObjectIdentifier = @8
  4: NSArray
        NSInlinedValue = True
        UINibEncoderEmptyKey = @1
  5: UIProxyObject
        UIProxiedObjectIdentifier = @2
  6: NSArray
        NSInlinedValue = True
  7: NSArray
        NSInlinedValue = True
        UINibEncoderEmptyKey = @5
        UINibEncoderEmptyKey = @3
        UINibEncoderEmptyKey = @12
  8: NSString
        NS.bytes = IBFirstResponder
  9: NSArray
        NSInlinedValue = True
        UINibEncoderEmptyKey = @5
        UINibEncoderEmptyKey = @3
        UINibEncoderEmptyKey = @12
 10: UIColor
        UIColorComponentCount = 4
        UIRed = 1.0
        UIGreen = 1.0
        UIBlue = 1.0
        UIAlpha = 1.0
        NSRGB = 1 1 1
        NSColorSpace = 2
 11: NSString
        NS.bytes = view
 12: UIView
        UIBounds = (0.0, 0.0, 320.0, 548.0)
        UICenter = (160.0, 274.0)
        UIBackgroundColor = @10
        UIOpaque = True
        UIAutoresizeSubviews = True
        UIAutoresizingMask = 18
        UIViewSemanticContentAttribute = 0
MacBook-Pro-toutes-Menesidis:ibtool-master menesidis$
```

Figure 8 - ibtool usage

The next step to collect information is to find which libraries have been used in the app. We can succeed it using the otool.

```
Michael:/var/mobile/Containers/Bundle/Application/338F432D-2BFA-4147-BDBB-4519247A2DDC/2048.app root# otool -l 2048
2048:
Load command 0
      cmd LC_SEGMENT_64
  cmdsize 72
  segname __PAGEZERO
   vmaddr 0x0000000000000000
```

Figure 9 (a) - otool command

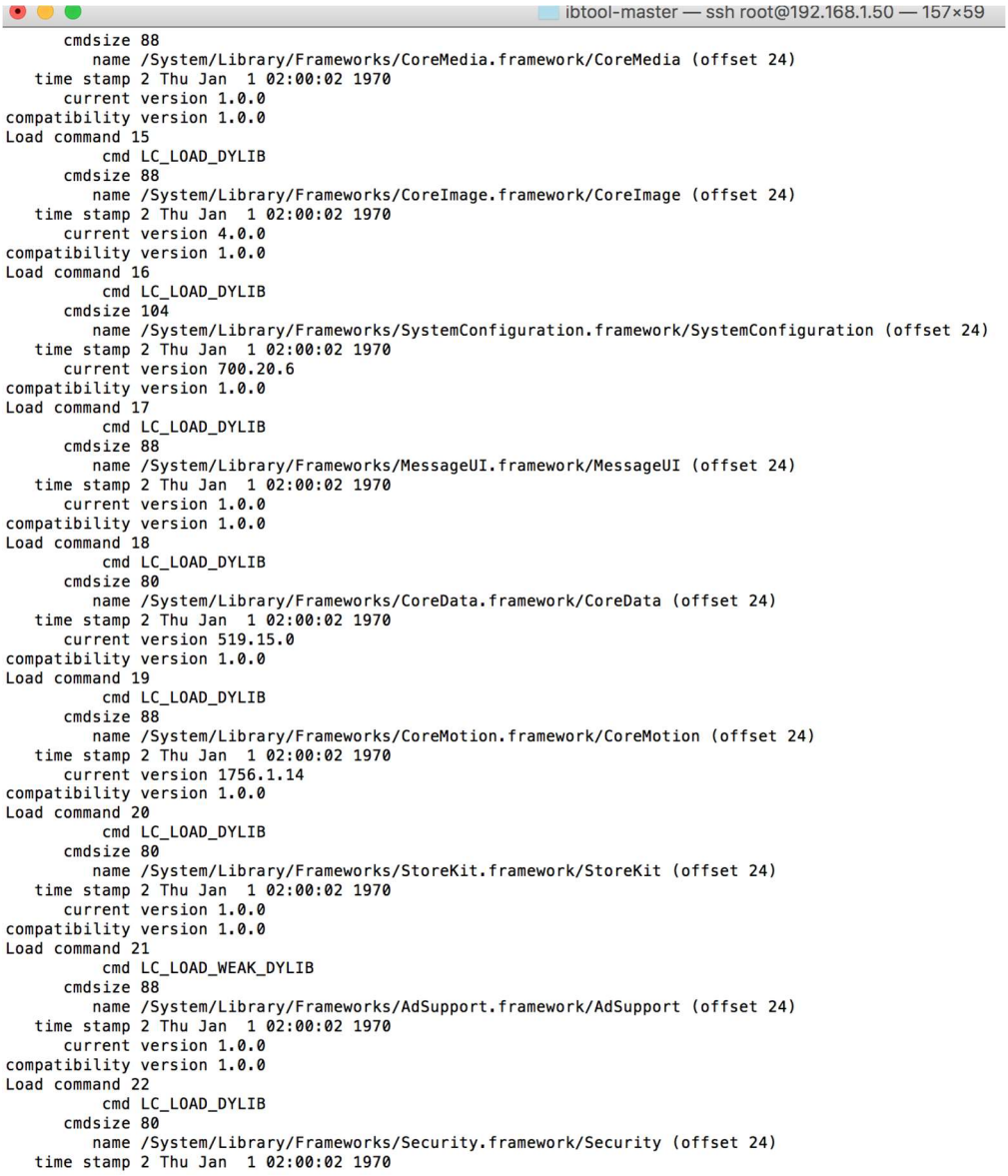


Figure 9 (b) - otool results

As we can see in Figure 9 (b) this app is using quite a lot of known frameworks and libraries. For instance we observe that this app is using the Security.framework which contains interfaces for managing certificates, public or private keys and trust policies. In addition, in Figure 10 we observe the objc-runtime library which makes runtime manipulation possible in Objective-C. By default, it is included in all iOS apps.

```
          cmd LC_LOAD_DYLIB
      cmdsize 56
         name /usr/lib/libobjc.A.dylib (offset 24)
   time stamp 2 Thu Jan  1 02:00:02 1970
      current version 228.0.0
compatibility version 1.0.0
Load command 41
          cmd LC_LOAD_DYLIB
      cmdsize 56
```

Figure 10 - Objective-C runtime library

5.2 Scenario 2 - Yahoo Weather App

In this scenario, we will try to perform runtime manipulation on the Yahoo Weather app. In order to achieve the manipulation we must first find information related to the source code of the app, such as classes' names, delegate files etc. For the manipulation part, we will try to inject our own code to modify the flow of the app. Code injection could be very useful to discover sensitive data of the app. The Yahoo Weather app provides information about the weather of different places. The first step is to make sure that the app is in foreground mode. This is because once the app is in the background state it paused and could not perform runtime analysis manipulation. Once the app is running, we can directly hook into the running process by finding the PID of the app and then using the cycript to hook into the process.

```
Michael:~ root# ps aux | grep "weather"
mobile    4991   0.0  3.1   912112  63232   ??  Us    8:23PM   0:05.55 /var/mobile/Containers/Bundle/Application/B369FA6C-6AD8-4F4C-8C11-909D8B37FF23/com.yahoo.weather-35
root      4997   0.0  0.1   546624   1216 s000  S+    8:25PM   0:00.01 grep weather
Michael:~ root# cycript -p 4991
cy# [UIApplication sharedApplication]
#"<UIApplication: 0x137d26700>"
cy# var a = [UIApplication sharedApplication]
#"<UIApplication: 0x137d26700>"
cy# a
#"<UIApplication: 0x137d26700>"
cy# a.delegate
#"<YWAppDelegate: 0x137e24640>"
```

Figure 11 - Yahoo delegate viewcontroller name

First of all, we get the instance of the yahoo app and after we identify the delegate class name which in our case is YWAppDelegate, so we conclude that the delegate files are defined as YWAppDelegate.h and YWAppDelegate.m. In Figure 12 we can see the initial screen of the app

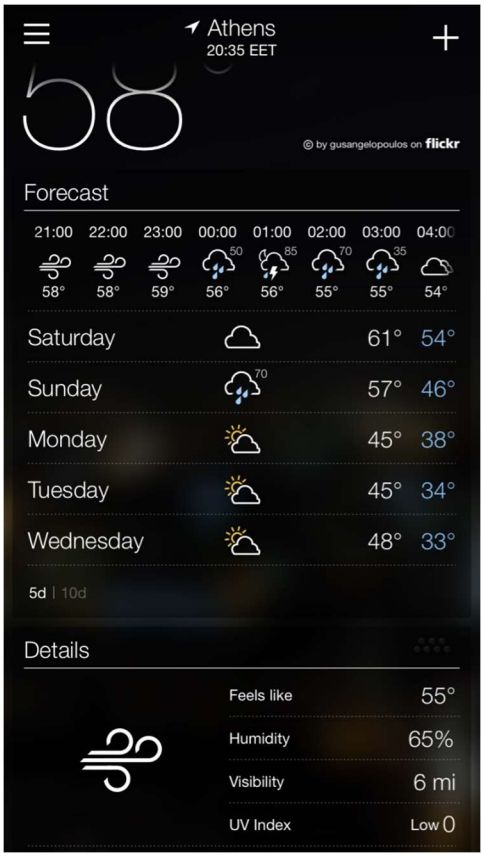


Figure 12 - Hidden status bar

As we can see, the status bar of the app is hidden. We can call a method in the application that shows the battery status bar.

```
Michael:~ root# ps aux | grep "weather"
mobile    5018   0.9  5.0   851360 102976   ??  Ss    8:33PM   0:04.23 /var/mobile/Containers/Bundle/Application/B369FA6C-6AD8-4F4C-8C11-909D8B37FF23/com.yahoo.weather-35245-distribution.app/yweather
root      5023   0.0  0.1   538432   1280 s000  S+    8:34PM   0:00.01 grep weather
Michael:~ root# cycript -p 5018
cy# [[UIApplication sharedApplication] setStatusBarHidden:NO animated:NO]
cy# [[UIApplication sharedApplication] setStatusBarHidden:YES animated:NO]
```

Figure 13 - Set status bar

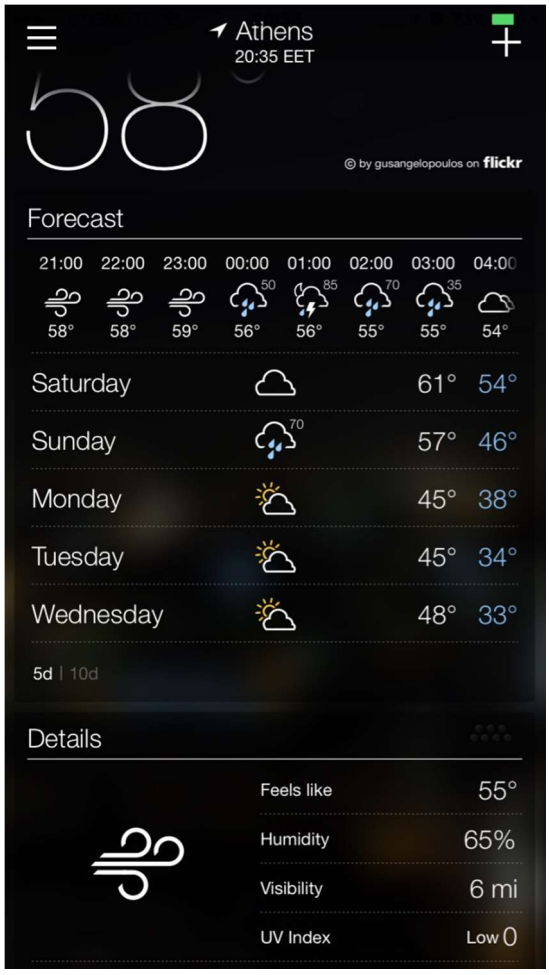


Figure 14 - Unhidden status bar

As we can see, the status bar is now visible. Secondly, we will try to modify the badge count of this particular application. A badge count is the number shown on the top-right of an application icon image. It usually refers to the amount of push notification received from the application. For instance, in mail apps, it can also refer to the amount of unread mails. In this app, there is no concept of push notifications and hence there is no count shown on the top-right of its app icon. The thing is that the application badge number can be set locally in the app through a simple function call as well as remotely through a push notification from the server. In this scenario we will modify the badge count number from 0 to 799. Figure 15 depicts the method that we should call to succeed it.

```
cy# [[UIApplication sharedApplication] setApplicationIconBadgeNumber:799];
cy#
```

Figure 15 - Set badge number

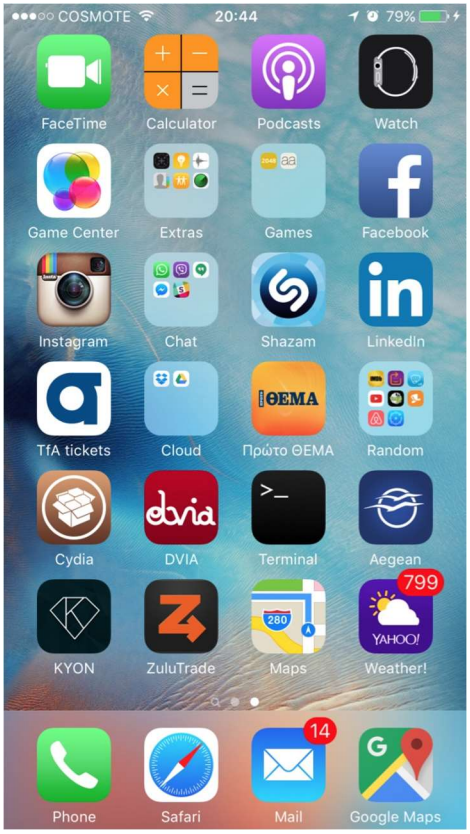


Figure 16 - Screen of Weather app after manipulation

Another attempt would be to find out the current view controller of the app. In order to succeed it we must first find out the keyWindow property. A keyWindow is the window which is currently accepting user interaction (touch events) from the user. For instance to find out the windows in an app we should run the following command (Figure 17).

```
cy# UIApp.windows
@[#"<UIWindow: 0x13c592500; frame = (0 0; 375 667); gestureRecognizers = <NSArray: 0x13c593b90>; layer = <UIWindowLayer: 0x13c592b80>>",#"<UITextEffectsWindow:
e = NO; autoresize = W+H; layer = <UIWindowLayer: 0x13dba6660>>"]
cy#
```

Figure 17 - Windows in the app

In Figure 18 we can see how to get the current keyWindow of application.

```
cy# UIApp.keyWindow
#"<UIWindow: 0x13c592500; frame = (0 0; 375 667); gestureRecognizers = <NSArray: 0x13c593b90>; layer = <UIWindowLayer: 0x13c592b80>>"
cy#
```

Figure 18 - Current keyWindow

In addition, we would try to find the root view controller for this window by using the property of the keyWindow. The root view controller provides the content view of the window. Assigning a view controller to this property (either programmatically or using Interface Builder) installs the view controller's view as the content view of the window. The new content view is configured to track the window size, changing as the window size changes [21].

```
cy# UIApp.keyWindow.rootViewController
#"<YMNMenuPresentationViewController: 0x13c696f10>"
cy#
```

Figure 19 - root view controller

Another usage would be to perform runtime manipulation and to modify the flow of the app. In Figure 20 we can see an example of this modification, where we injected an alert message in weather app. Similarly in Figure 21 we force the app to make a phone call in a specific number and in Figure 22 we force the app to go at specific URL. This is a typical scenario to lure the user by visiting a phishing website.

```
cy# [[[UIAlertView alloc] initWithTitle:@"Hello its me Michalis :)." message:nil delegate:nil cancelButtonTitle:@"Nice job!" otherButtonTitles:nil]show];
```

Figure 20 (a) - Command to inject Alert

Figure 20 (b) - View of injected alert

```
cy#  [[UIApplication sharedApplication] openURL:[NSURL URLWithString:@"tel:3069111111115"]];
true
cy#
```

Figure 21 (a) - Command to force a phone call

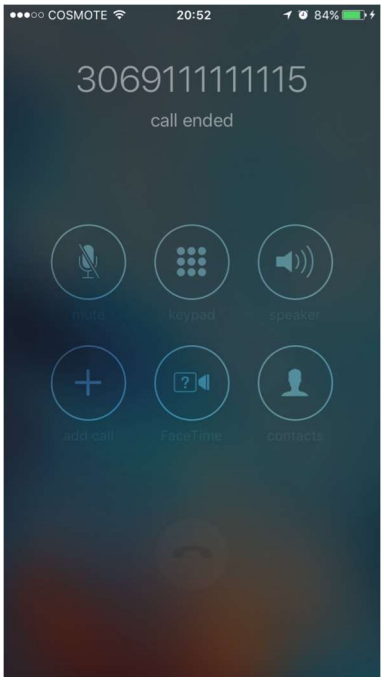


Figure 21 (b) - View of forced call

```
cy#  var app = [ UIApplication sharedApplication ]
#"<UIApplication: 0x13c62acd0>"
cy# [ app openURL: [ NSURL URLWithString: @"http://www.unipi.gr" ] ]
```

Figure 22 - Command to open a URL

Our next attempt would be to find unique details about the application. Firstly, we would try to get the *identifierForVendor*. This is an alphanumeric string that uniquely identifies a device to the app's vendor. The value of this property is the same for apps that come from the same vendor running on the same device. Normally, the vendor is determined by data provided by the App Store [22]. In Figure 23 we get the identifierForVendor of the app.

```
cy# [[UIDevice currentDevice] identifierForVendor];
#"<__NSConcreteUUID 0x13dbfa460> 4D8F1C75-1322-48D1-B19F-988C5BB811D4"
cy#
```

Figure 23 - IdentifierForVendor

Secondly, we would try to discover the bundle ID of the app (Figure 24). A *bundle ID* precisely identifies a single app. A bundle ID is used during the development process to provision devices

and by the operating system when the app is distributed to customers. For example, Game Center and In-App Purchase use a bundle ID to identify your app when using these app services. The preferences system uses this string to identify the app for which a given preference applies. Similarly, Launch Services uses the bundle ID to locate an app capable of opening a particular file, using the first app it finds with the given identifier. The bundle ID is also used to validate an app's signature. The bundle ID string must be a uniform type identifier (UTI) that contains only alphanumeric characters (A-Z,a-z,0-9), hyphen (-), and period (.). The string should be in reverse-DNS format [23].

```
cy# [[NSBundle mainBundle] bundleIdentifier];
@"com.yahoo.weather"
cy#
```

Figure 24 - Bundle ID of app

```
cy# [[[NSBundle mainBundle] infoDictionary] objectForKey:@"CFBundleShortVersionString"];
@"1.9.0"
cy#
```

Figure 25 - Release version of the app

- The *CFBundleShortVersionString* specifies the *release version number* of the bundle, which identifies a released iteration of the app (see Figure 25).
- The *CFBundleVersion* specifies the build version number of the bundle, which identifies an iteration (released or unreleased) of the bundle (see Figure 26).
- The *CFBundleDevelopmentRegion* is the default language and region for the bundle, as a language ID (see Figure 26).

```
cy# [[[NSBundle mainBundle] infoDictionary] objectForKey:@"CFBundleVersion"];
@"35245"
cy# [[[NSBundle mainBundle] infoDictionary] objectForKey:@"CFBundleDevelopmentRegion"];
@"en"
cy#
```

Figure 26 - Build version number and default language

5.3 Scenario 3 - WhatsApp App

In this scenario we will investigate the famous whatsApp app which be used from over a billion users. After we locate the .plist file we can open it via Xcode or convert it in xml format to be in a readable form. Because many users don't have Mac OS we selected the second option (see Figure 27). There are many OS-X-native, source utilities that interact with .plist files including the basic command line utility called plutil. Pete M. Wilson has developed this utility in a perl script that can convert binary plist files to their plain text equivalent [24]. His script plutil.pl basicaly parses the file and outputs a plain text version in the same directory.

```
./Media/iTunes_Control/iTunes/iTunesSyncDeletes.plist
Michael:/var/mobile root# plutil -convert xml1 Containers/Bundle/Application/BC302F79-8129-4B8D-ACD9-18C000FA4EEC/WhatsApp.app/Info.plist
Converted 1 files to XML format
Michael:/var/mobile root#
```

Figure 27 - Convert .plist file to xml

```
menesidis — ssh root@192.168.1.6 — 164×58
Michael:/var/mobile/Containers/Bundle/Application/BC302F79-8129-4B8D-ACD9-18C000FA4EEC/WhatsApp.app root# plutil Info.plist
{
    BuildMachineOSBuild = 14F1021;
    CFBundleDevelopmentRegion = en;
    CFBundleDisplayName = WhatsApp;
    CFBundleDocumentTypes =     (
            {
            CFBundleTypeIconFiles =             (
                Icon,
                "Icon@2x"
            );
            CFBundleTypeName = "WhatsApp Image";
            LSHandlerRank = Alternate;
            LSItemContentTypes =             (
                "public.image"
            );
        },
            {
            CFBundleTypeIconFiles =             (
                Icon,
                "Icon@2x"
            );
            CFBundleTypeName = "WhatsApp Image Exclusive";
            LSHandlerRank = Owner;
            LSItemContentTypes =             (
                "net.whatsapp.image"
            );
        },
            {
            CFBundleTypeIconFiles =             (
                Icon,
                "Icon@2x"
            );
            CFBundleTypeName = "WhatsApp Audio";
            LSHandlerRank = Alternate;
            LSItemContentTypes =             (
                "public.mp3",
                "public.mpeg-4-audio",
                "public.aif-audio",
                "public.aifc-audio",
                "com.apple.coreaudio-format"
            );
        },
            {
            CFBundleTypeIconFiles =             (
                Icon,
                "Icon@2x"
            );
            CFBundleTypeName = "WhatsApp Audio Exclusive";
            LSHandlerRank = Owner;
            LSItemContentTypes =             (
                "net.whatsapp.audio"
            );
        },
```

Figure 28 (a) - Info.plist WhatsApp

As we can see there is plenty of information within the .plist file. Some of the information is summarized below:

- *BuildMachineOSBuild* - Refers to the OS that the app has been compiled which in our case is the OS X Yosemite 10.10.5 (14F1021).
- *CFBundleDevelopmentRegion* - specifies the default language and region for the bundle, as a language ID.
- *CFBundleDisplayName* - specifies the display name of the bundle. Furthermore, we have access to the app icon image weather it is retina or not.
- *LSItemContentTypes* - contains an array of strings with the uniform type identifier (UTI) types that represent the supported file types in this group. In other words, the LSItemContentTypes key identifies the UTI associated with the file, which in our case are audio and video files.

```
            {
        CFBundleTypeIconFiles =             (
            Icon,
            "Icon@2x"
        );
        CFBundleTypeName = "WhatsApp Movie";
        LSHandlerRank = Alternate;
        LSItemContentTypes =             (
            "public.movie"
        );
    },
            {
        CFBundleTypeIconFiles =             (
            Icon,
            "Icon@2x"
        );
        CFBundleTypeName = "WhatsApp Movie Exclusive";
        LSHandlerRank = Owner;
        LSItemContentTypes =             (
            "net.whatsapp.movie"
        );
    }
);
CFBundleExecutable = WhatsApp;
CFBundleIcons =     {
    CFBundlePrimaryIcon =         {
        CFBundleIconFiles =             (
            AppIcon29x29,
            AppIcon40x40,
            AppIcon57x57,
            AppIcon60x60
        );
    };
};
CFBundleIdentifier = "net.whatsapp.WhatsApp";
CFBundleInfoDictionaryVersion = "6.0";
CFBundleName = WhatsApp;
CFBundlePackageType = APPL;
CFBundleShortVersionString = "2.12.13";
CFBundleSignature = "????";
CFBundleSupportedPlatforms =     (
    iPhoneOS
);
CFBundleURLTypes =     (
            {
        CFBundleURLName = "net.whatsapp.WhatsApp";
        CFBundleURLSchemes =             (
            whatsapp
        );
    }
);
CFBundleVersion = "2.12.13.20";
DTCompiler = "com.apple.compilers.llvm.clang.1_0";
DTPlatformBuild = 13C75;
DTPlatformName = iphoneos;
DTPlatformVersion = "9.2";
DTSDKBuild = 13C75;
```

Figure 28 (b) - Info.plist WhatsApp

In Figure 28 (b) we can see some basic information of app as the bundle identifier, the version of the app the supported platform version etc. Apart from these, some important parameters are:

- *CFBundleURLName* - is a string containing the abstract name of the URL scheme. This string you specify is also used as a key in your app's InfoPlist.strings file. The value of the key is the human-readable scheme name.
- *CFBundleURLSchemes* - is an array of strings containing the URL scheme names - for example, http, mailto, tel, and sms. More specific, the URL Schemes is the start of the URL e.g 'appname'. When you call this as a URL it targets the bundle identifier which launches the app.

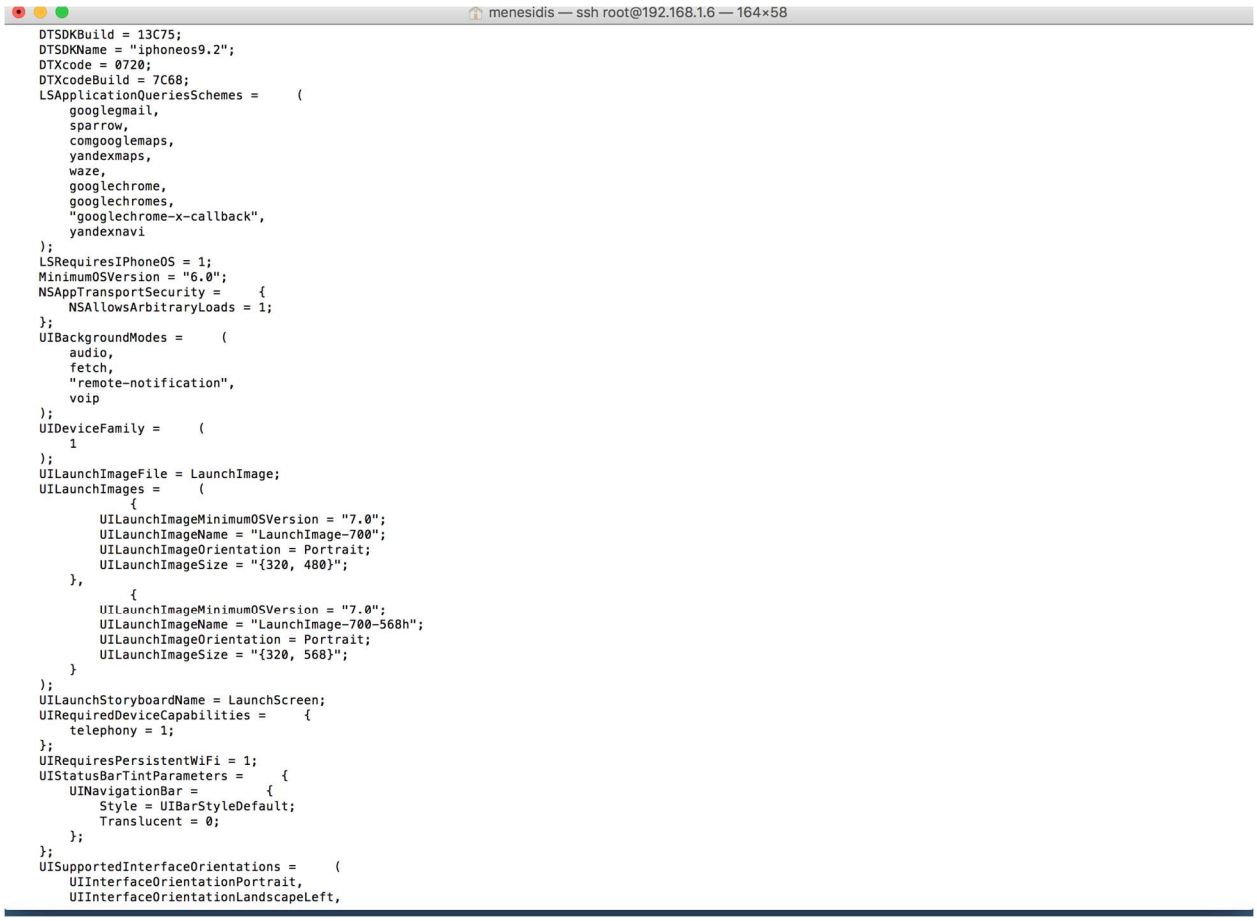


Figure 28 (c) - Info.plist WhatsApp

In addition, from Figure 28 (c) and (d) we can collect information about the development environment and various services.

- *DTSDKName* - shows that the app has been built on iOS 9.2 .
- *DTXcode* - shows that the version of IDE Xcode is 7.2.
- *LSApplicationQueriesSchemes* - specifies the URL schemes you want the app to be able to use with the canOpenURL: method, which in our case google gmail, google chrome, waze etc. The LSMinimumSystemVersion indicates the minimum version of iOS required for this app to run which in our case is iOS 6.0.
- *UIBackgroundModes* - provides specific background services and must be allowed to continue running while in the background which in our case are audio, voip, remote notifications and fetch.
- *UISupportedInterfaceOrientations* - specifies the interface orientations your app supports which in our example are both portrait and landscape.
- *UTTypeTagSpecification* - is a dictionary defining one or more equivalent type identifiers. The key-value pairs listed in this dictionary identify the filename extensions, MIME types, OSType codes, and pasteboard types that correspond to this type.

```
    UISupportedInterfaceOrientations =     (
        UIInterfaceOrientationPortrait,
        UIInterfaceOrientationLandscapeLeft,
        UIInterfaceOrientationLandscapeRight
    );
    UIViewGroupOpacity = 0;
    UTExportedTypeDeclarations =     (
                {
            UTTypeDescription = "WhatsApp Image Exclusive";
            UTTypeIdentifier = "net.whatsapp.image";
            UTTypeTagSpecification =             {
                "public.filename-extension" = wai;
                "public.mime-type" = "image/*";
            };
        },
                {
            UTTypeDescription = "WhatsApp Audio Exclusive";
            UTTypeIdentifier = "net.whatsapp.audio";
            UTTypeTagSpecification =             {
                "public.filename-extension" = waa;
                "public.mime-type" = "audio/*";
            };
        },
                {
            UTTypeDescription = "WhatsApp Movie Exclusive";
            UTTypeIdentifier = "net.whatsapp.movie";
            UTTypeTagSpecification =             {
                "public.filename-extension" = wam;
                "public.mime-type" = "video/*";
            };
        }
    );
}
Michael:/var/mobile/Containers/Bundle/Application/BC302F79-8129-4B8D-ACD9-18C000FA4EEC/WhatsApp.app root#
```

Figure 28 (d) - Info.plist WhatsApp

5.4 Scenario 4 - Dump Keychain

As we already mentioned in 4.1.5 subsection, the aim of keychain dumper is to check which keychain items are available to an attacker.

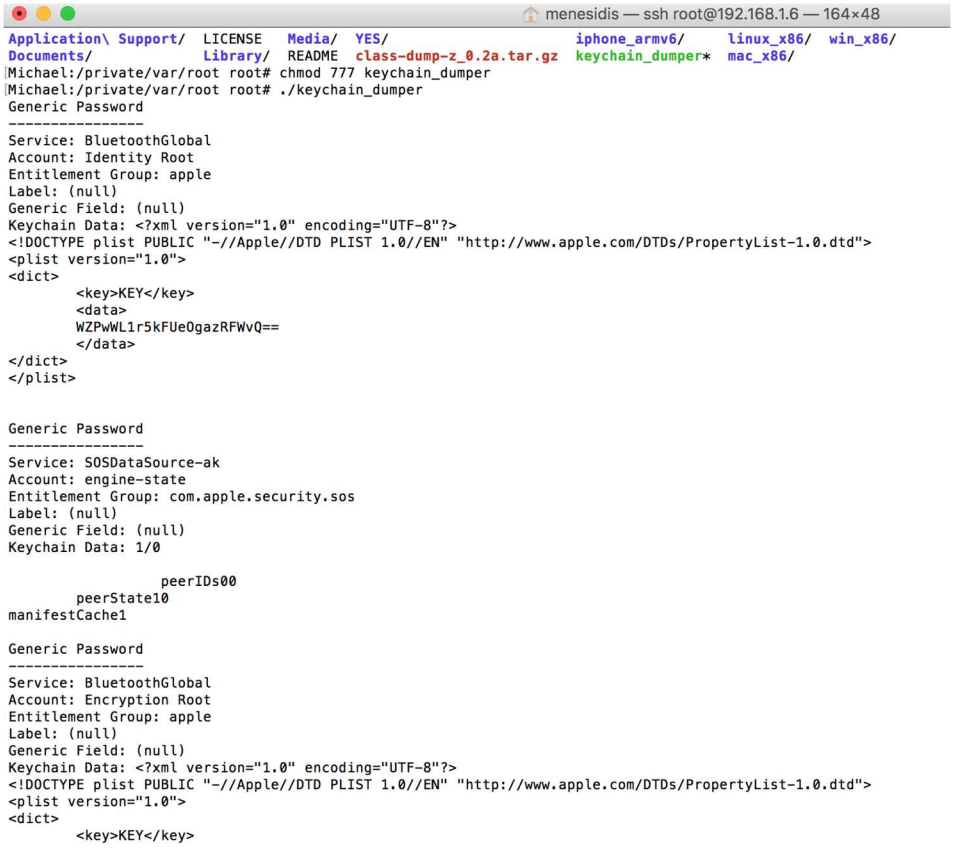


Figure 29 (a) - Keychain dumper

As we can see, this tool dumps all the keychain information. For instance in BluetoothGlobal service we observe the key that might be plain text or obfuscated. The entitlements of this service are owned by Apple. Similarly, we dump another three keys from SOSDataSource-ak service which also owned from a group of Apple.

```
----------------
Service: AirPort
Account: GRAMMESTONORIZONTON
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: 2107775050#

Generic Password
----------------
Service: AirPort
Account: AAAFXGUEST2
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: AAAFXWIFIGUESTPASS2012

Generic Password
----------------
Service: com.apple.assistant
Account: 191F217D-D0F4-42E7-A1AF-CBDC18A42D04 - Server Certificate Data
Entitlement Group: com.apple.assistant
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: com.google.sso.GetAppIdentifierPrefix
Account: GetAppIdentifierPrefix
Entitlement Group: EQHXZ8M8AV.com.google.Maps
Label: (null)
Generic Field: (null)
Keychain Data:

Generic Password
----------------
Service: com.apple.assistant
Account: 191F217D-D0F4-42E7-A1AF-CBDC18A42D04 - Validation Data
Entitlement Group: com.apple.assistant
Label: (null)
Generic Field: (null)
Keychain Data: (null)
```

Figure 29 (b) - Keychain dumper

Figure 29 (b) shows various keychain data in plain text form. This can be verified because of the knowledge of actual passwords from some wi-fi networks. The service AirPort is a Utility built by Apple to manage Wi-Fi networks and AirPort base stations [25]. In our case, the account 'GRAMMESTONORIZONTON' is the SSID of the Wi-Fi network and the keychain data is the corresponding password in plain text form. Similarly, the account "AAAFXGUEST2" is the SSID and the corresponding password is in plain text form too.

```
Service: gr.eurobank.epistroficom.flurry.analytics
Account: <466c7572 72795365 7373696f 6e54696d 65737461 6d704b65 79>
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: <466c7572 72795365 7373696f 6e54696d 65737461 6d704b65 79>
Keychain Data: (null)

Generic Password
----------------
Service: service
Account: gr.eurobank.epistrofi.NBAPIKey
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: (null)
Keychain Data: bdbae4d4acd011e5aca30003ff436ff8

Generic Password
----------------
Service: service
Account: gr.eurobank.epistrofi.NBWebID
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: (null)
Keychain Data: bdbaed3aacd011e5aca30003ff436ff8

Generic Password
----------------
Service: service
Account: gr.eurobank.epistrofi.EpistrofiUserName
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: (null)
Generic Password
----------------
Service: service
Account: gr.eurobank.epistrofi.EpistrofiEmail
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: (null)
Keychain Data: menesidis@gmail.com

Generic Password
----------------
Service: service
Account: gr.eurobank.epistrofi.EpistrofiCardNumber
Entitlement Group: 37L692QAGF.gr.eurobank.epistrofi
Label: (null)
Generic Field: (null)
```

Figure 29 (c) - Keychain dumper

In figure 29 (c) we dump information from a banking application. More precisely, the account 'gr.eurobank.epistrofi.NBAPIKey' is probably an API to communicate with the bank and in the keychain is the credential to authenticate in this service. Another assumption would be that the account "gr.eurobank.epistrofi.NBWebID" is a service to authenticate the ID of the user and the credential is located in keychain data. We can verify that the account "gr.eurobank.epistrofi.EpistrofiEmail" store the email of the user. This app will be examined thoroughly later in this chapter.

5.5 Scenario 5 - e-Banking Apps

In this Scenario we examine various known e-Banking apps. More specifically, we focus on the fourth largest banks in Greece.

5.5.1 eurobank epistrofh App

Eurobank epistrofh App app is a known bank application, which returns money after specific purchases with cooperating companies. After a detailed examination of the app we manage to obtain several data. In Figure 30, we observe essential information about the app such as:

- *Last_update* - when was the latest update
- *EpistrofiPoints* - how much was the returning amount
- *IOS_GEOFENCING_ENABLED* - whether geofence (virtual perimeter) is enabled or not
- *uuidsAlreadySent* - whether uuid is already sent or not
- *EpistrofiUserSession* - whether a user has session or not

gr.eurobank.epistrofi.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(44 items)
hasAlreadyDisplayedTutorial	Boolean	YES
kAppiraterDeclinedToRate	Boolean	NO
▶ main_menu	Array	(9 items)
WARPLY_ENABLED	Boolean	YES
last_update	Date	26 Feb 2016 9:52:36 PM
kAppiraterRatedCurrentVersion	Boolean	NO
Health	Number	2
IOS_LOCATION_BACKGROUND_MODE	Number	1
USER_TAGGING_ENABLED	Boolean	YES
WebKitShrinksStandaloneImagesToFit	Boolean	YES
EpistrofiPoints	Number	2,84
kAppiraterFirstUseDate	Number	1.456.314.048,50215
CONSUMER_DATA_ENABLED	Boolean	YES
IOS_FOREGROUND_DISTANCE_FILTER	Number	50
WebKitLocalStorageDatabasePathPref...	String	/var/mobile/Containers/Data/Application/6E4DCCCA-ED7E-487E-B429-43C94B45308F/Library/Caches
IOS_GEOFENCING_ENABLED	Boolean	YES
uuidsAlreadySent	Boolean	YES
USER_SESSION_ENABLED	Boolean	YES
IOS_LOCATION_FOREGROUND_MODE	Number	1
NBAppUninstalled	Boolean	YES
automoto	Number	1
Travel-Hotels	Number	3
CUSTOM_ANALYTICS_ENABLED	Boolean	YES
EpistrofiFirstTime	Boolean	NO
NBAPPUuidChanged	Boolean	NO
APPLICATION_DATA_ENABLED	Boolean	YES
EpistrofiLoggedIn	Boolean	YES
OFFERS_ENABLED	Boolean	YES
Malls	Number	3
kAppiraterUseCount	Number	5
kAppiraterCurrentVersion	String	6701
FEATURES_CHECK_INTERVAL	Number	259.200
EpistrofiUserSession	Boolean	YES
GEOUsageSessionID	Data	<08d8cdcc adade0d3 a615108a f7eecfc6 e68f91bc 01>
kAppiraterReminderRequestDate	Number	0
kAppiraterSignificantEventCount	Number	0
WebDatabaseDirectory	String	/var/mobile/Containers/Data/Application/6E4DCCCA-ED7E-487E-B429-43C94B45308F/Library/Caches
lastFeaturesUpdateTimestamp	Date	27 Feb 2016 3:11:07 AM
WebKitOfflineWebApplicationCacheEn...	Boolean	YES
IOS_BACKGROUND_DISTANCE_FILTER	Number	100
GEOUsageSessionIDGenerationTime	Number	475.856.993,15698
DEVICE_INFO_ENABLED	Boolean	YES
LIFECYCLE_ANALYTICS_ENABLED	Boolean	YES

Figure 30 - gr.eurobank.epistrofi.plist

Apart from the plist file we locate a database called ‘Cache.db’. Using sqlite browser we open the database and we investigate all the tables to discover sensitive information. The first table called “*cfurl_cache_response*”,which contains sensitive data in “*request_key*” field.

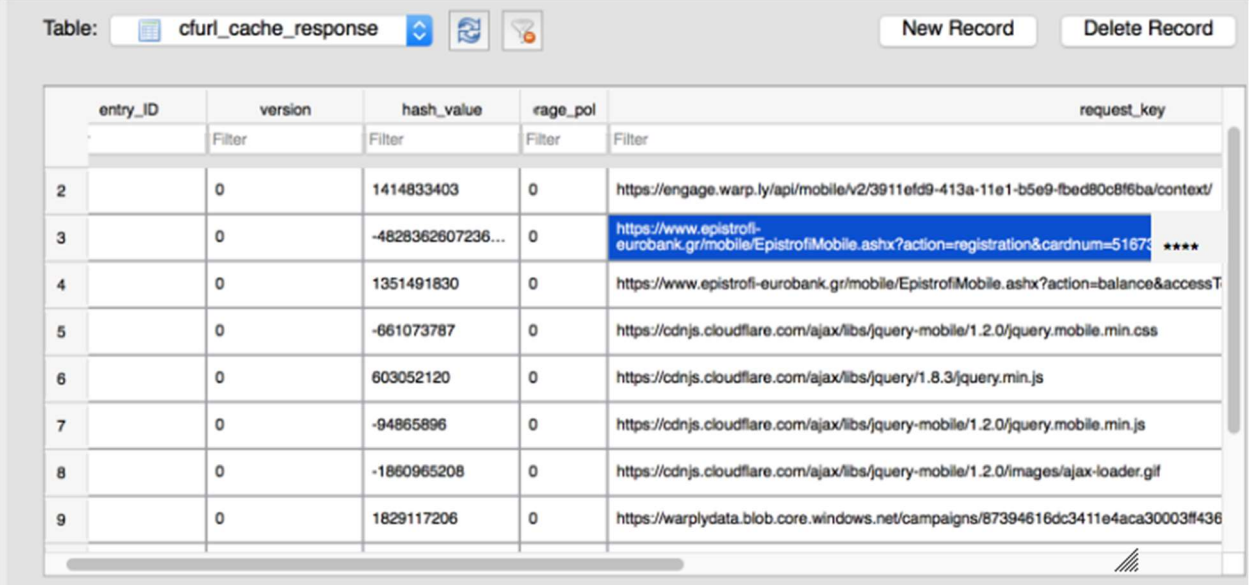


Figure 31 - cfurl_cache_response table

More specific, in Figure 31 we observe some links that may have access to sensitive data like card number, mobile_id etc. The first attempt was to check the link on the browser. As we can see in Figure 32, the action of the service, the card number and mobile_id are in a plaintext form. The response of this service was the access token (AccessToken) which is used to authenticate the user. AccessToken is in paramount importance because a hacker could use it to get access to user's account.

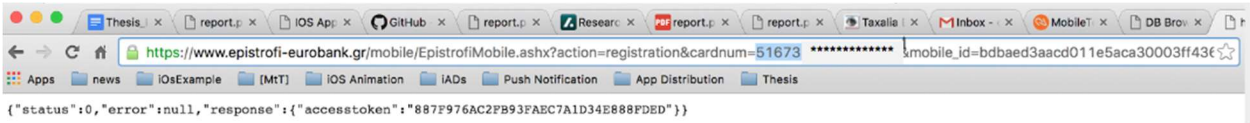


Figure 32 - Collect access token

Secondly, we check another link related to balance information.

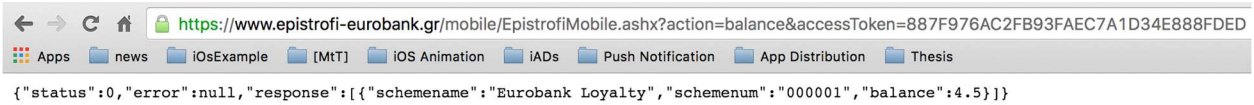


Figure 33 - Collect amount of balance

As we can see in Figure 33, the type of service is "balance" and it uses the already leaked accessToken to get the current balance. The response of this service returns the scheme name and the balance of returning amount of current user. This can be verified in Figure 34.

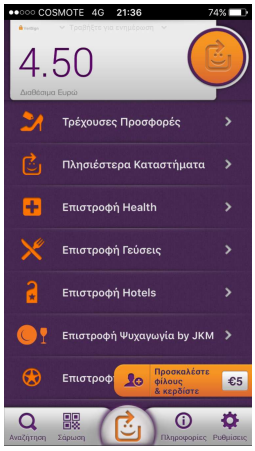


Figure 34 - Balance of App

In addition, we investigate further another “request_key” which give us whether the device was registered or not and also the api_key and web_id. Such information could used to get access to company api in order to leak further information.

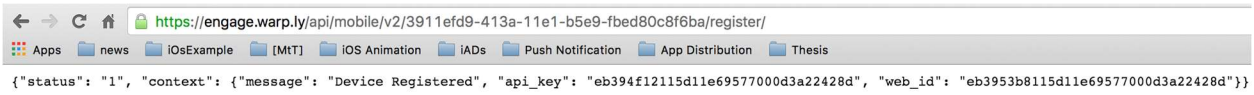


Figure 35 - Collect api key and web identifier

The second investigated table called “cfurl_cache_receiver_data” and probably stores the receiving data of various services (see Figure 36). As we can see, there is plenty of information which is unknown and further investigation is needed. We select to investigate the data that used the card number because it is sensitive.

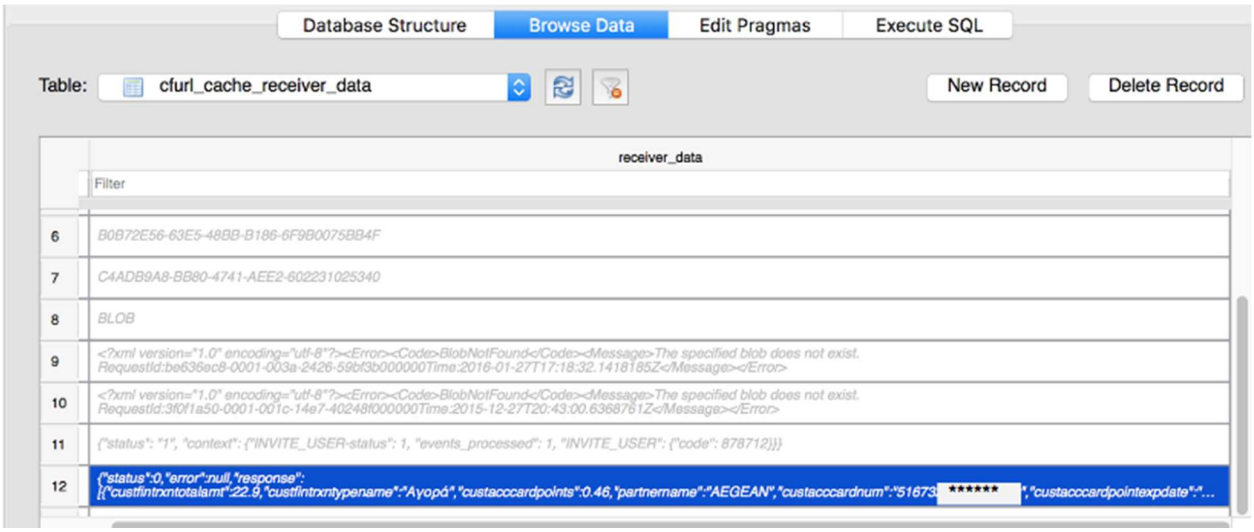


Figure 36 - cfurl_cache_receiver_data table

After further investigation we manage to access in all purchases that have been made with this card number. Figure 37 shows some of transactions that have been made with the specific card number. More specifically we observe :

- *custfintrxntotalant* - The amount of transaction
- *custfintrxntypename* - The type of transaction
- *custacccardpoints* - The returning points of transaction
- *partnername* - The partner name company that the transaction has been made
- *custacccardnum* - The card number
- *custaccardpointexpdate* - The date of points expiration
- *Custfintrxndate* - The date of transaction
- *Custacccardremainingpoints* - The card remaining points

```
{
   "custfintrxntotalamt":11.9,
   "custfintrxntypename":"Αγορά",
   "custacccardpoints":0.12,
   "partnername":"ΒΕΡΟΠΟΥΛΟΣ SUPER MARKET",
   "custacccardnum":"516732 **********
   "custacccardpointexpdate":"2017-02-14T19:06:10+02:00",
   "custfintrxndate":"2016-02-15T19:06:10+02:00",
   "custacccardremainingpoints":0.12
},
{
   "custfintrxntotalamt":26.95,
   "custfintrxntypename":"Αγορά",
   "custacccardpoints":0.27,
   "partnername":"ΣΚΛΑΒΕΝΙΤΗΣ",
   "custacccardnum":"516732 **********
   "custacccardpointexpdate":"2017-02-04T19:43:49+02:00",
   "custfintrxndate":"2016-02-05T19:43:49+02:00",
   "custacccardremainingpoints":0.27
},
{
   "custfintrxntotalamt":23.68,
   "custfintrxntypename":"Εξαργύρωση",
   "custacccardpoints":-23.68,
   "partnername":"AEGEAN",
   "custacccardnum":"516732 **********
   "custfintrxndate":"2016-01-25T18:49:37+02:00",
   "custacccardremainingpoints":0
},
{
   "custfintrxntotalamt":0,
   "custfintrxntypename":"ΔΩΡΟ ΕΥΡΩ ΕΠΙΣΤΡΟΦΗ",
   "custacccardpoints":10,
   "partnername":"EUROBANK",
   "custacccardnum":"516732 **********
   "custacccardpointexpdate":"2017-01-19T11:27:37+02:00",
   "custfintrxndate":"2016-01-20T11:27:37+02:00",
   "custacccardremainingpoints":1.99
}
```


Figure 37 - Collect transactions information

All this information is sensitive because we can violate the privacy of the owner. Furthermore, we discovered another file called “settings”. A part of this file is depicted in Figure 38. This file stores information of a well known service called “crashlytics “ which is used to provide reports to developers about app’s crashes.

Figure 38 - Settings.json

In addition, we also found a plist file called “CLSUserDefaults.plist”.

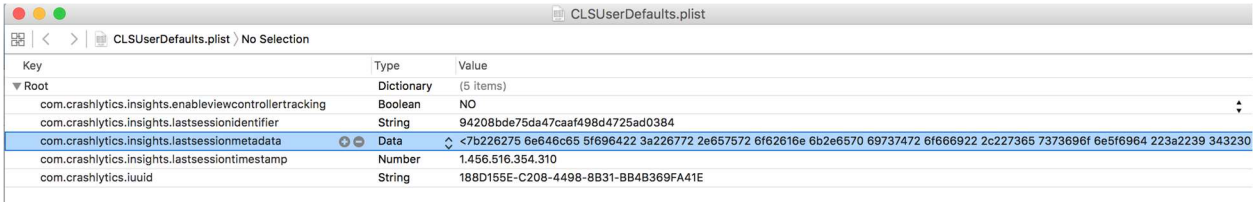


Figure 39 - Crashlytics plist

Various information is presented in such file in both plaintext and encoded form. For instance, the (universally unique identifier) UUID is in plain text. We can also collect information such as the latest session identifier and weather the current view controller is being tracking or not. Apart from the aforementioned plain text information, we observed raw data. The first thought was to

investigate whether these data are in a HEX form, which is proved true so we converted the hex string to the following json.


```
{
   "bundle_id":"gr.eurobank.epistrofi",
   "session_id":"94208bde75da47caaf498d4725ad0384",
   "advertising_tracking_enabled":true,
   "vendor_id":"1839A193-AB26-4056-B222-03FA3B285B03",
   "locale":"en_GR",
   "os_build":"13A342",
   "links_ad_support":true,
   "platform":"iOS",
   "install_id":"188D155E-C208-4498-8B31-BB4B369FA41E",
   "platform_code":1,
   "cores":2,
   "generator":"Answers iOS SDK\/1.1.0",
   "instance_id":"019bb2f7115d9f1166b3d90bf12fb9f0d0012e9d",
   "advertising_id":"E1E37777-A55D-4223-BE79-73084723BE4A",
   "os_version":"9.0.0",
   "jailbroken":true,
   "started_at":1456516354,
   "model":"iPhone8,1",
   "bundle_version":"6701",
   "api_key":"cec9993f32a3782c456b7ad024144c515bd4248e",
   "machine":"iPhone8,1",
   "bundle_short_version":"3.3.1"
}
```


Figure 40 - Crashlytics json

This json contains various data such as session_id, installation_id, instance_id, advertising_id, api_key as well as much information related to the hardware specifications (see Figure 40). As already mentioned session_id and api_key could lead to user's sensitive data.

5.5.2 NBG App

In this sub-scenario we tried to collect information from NBG (National Bank of Greece) App. Our first attempt, to gather information from the app, was to locate the plist files that contains data in a cleartext form. Figures 41 (a) and (b) present the plist configuration files that have been found. For instance, we locate:

- *Bundle identifier*
- *Localized Strings in both languages*

- *Images*
- *Build Version, Xcode Version*
- *Nib files*
- *Location usage*
- *URL scheme*

Key	Type	Value
▼ Root	Dictionary	(4 items)
MCMMetadataIdentifier	String	gr.nbg.mobile.mbanking
▼ MCMMetadataInfo	Dictionary	(1 item)
com.apple.MobileInstallation.Conte...	Number	0
MCMMetadataContentClass	Number	2
MCMMetadataUUID	String	3B69F7ED-F1C6-41AE-BCBD-57223E07F326

Figure 41 (a) - plist file

Key	Type	Value
▼ Information Property List	Dictionary	(35 items)
▼ URL types	Array	(1 item)
▼ Item 0	Dictionary	(4 items)
Document Icon File Name	String	Images.xcassets/AppIcons.appiconset/i-bank_58x58
▼ URL Schemes	Array	(1 item)
Item 0	String	nbgibank
CFBundleURLTypes	String	None
URL identifier	String	gr.nbg.mobile.mbanking
UIRequiresFullScreen	String	YES
InfoDictionary version	String	6.0
▶ Supported interface orientations (i...	Array	(3 items)
DTPlatformVersion	String	9.3
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
Bundle name	String	NBG
DTSDKName	String	iphoneos9.3
▶ Icon files (iOS 5)	Dictionary	(1 item)
Status bar style	String	UIStatusBarStyleLightContent
Bundle display name	String	NBG
Application requires iPhone enviro...	Boolean	YES
Privacy - Location When In Use Us...	String	See our branches and ATMs around you
DTSDKBuild	String	13E230
Bundle versions string, short	String	6.0.1
▶ CFBundleSupportedPlatforms	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
Privacy - Location Always Usage D...	String	See our branches and ATMs around you
BuildMachineOSBuild	String	15F34
DTPlatformBuild	String	13E230
Bundle OS Type code	String	APPL
DTXcodeBuild	String	7D1014
Localization native development re...	String	en
▼ Fonts provided by application	Array	(1 item)
Item 0	String	Fonts/FontAwesome.ttf
Bundle version	String	16080201
MinimumOSVersion	String	7.0
▶ UIDeviceFamily	Array	(2 items)
Bundle identifier	String	gr.nbg.mobile.mbanking
DTXcode	String	0731
Privacy - Location Usage Descripti...	String	See our branches and ATMs around you
Executable file	String	NBGios
Bundle creator OS Type code	String	????
▶ CFBundleIcons~ipad	Dictionary	(1 item)
DTPlatformName	String	iphoneos
▶ com.xamarin.ios	Dictionary	(1 item)

Figure 41 (b) - plist file

In Figure 41 (c) we can see the date and time of the last user's login and whether the password was saved or not (see the key "rememberme"). In addition, we observe the identifier for Push Notifications (see the key "apnRegId") and the current language of app (see the key "NbgMobileLanguage"). Also, a path with various cached data is presented and after further investigation of the path we found that it contains obfuscated data. In general, we found many obfuscated files in different paths in the app but we didn't manage to deobfuscate any of them.

gr.nbg.mobile.mbanking.plist 〉 No Selection

Key	Type	Value
▼ Root	Dictionary	(10 items)
4288377-lastlogin	String	27/09/2016 20:25
rememberme	Boolean	NO
apnregId	String	43aa637706be99d4393c606100a80611894a2cb564c2b16cf5426e633490957d
NbgMobileLanguage	String	el
WebKitLocalStorageDatabasePathPref...	String	/var/mobile/Containers/Data/Application/40F3F0A3-B021-4F35-8195-D75A587EF14F/Library/Caches
WebDatabaseDirectory	String	/var/mobile/Containers/Data/Application/40F3F0A3-B021-4F35-8195-D75A587EF14F/Library/Caches
apnappVersion	String	6.0.1 (16080201)
WebKitShrinksStandaloneImagesToFit	Boolean	YES
WebKitOfflineWebApplicationCacheEn...	Boolean	YES
nbg_hasapnregistered	Boolean	YES

Figure 41 (c) - plist file

Moreover, we found several json files that have been used by the app for server calls.

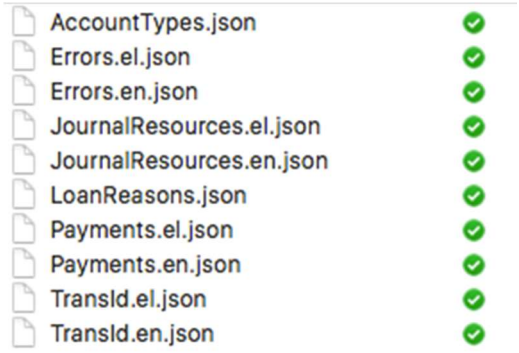


Figure 42 - json files

For example we found the server's response error codes in the Errors.el.json file. This could be important information for a malicious user to manipulate the returning codes.

Errors.el.json (no symbol selected)


```
[
    {
        "code": "HOST.9",
        "message": "Ανύπαρκτη συναλλαγή"
    },
    {
        "code": "HOST.005",
        "message": "Εμφανίζονται μόνο 494 τίτλοι."
    },
    {
        "code": "HOST.006",
        "message": "Το χαρτοφυλάκιο σας έχει περισσότερες από τις 265 παραπάνω εγγραφές.
    },
    {
        "code": "HOST.011",
        "message": "Ανεπαρκές διαθέσιμο υπόλοιπο."
    },
    {
        "code": "HOST.020",
        "message": "Δεν υπάρχει αίτηση ΣΑΤ."
    },
    {
        "code": "HOST.024",
        "message": "Δεν υπάρχει σχέση πελάτη λογαριασμού."
    },
    {
        "code": "HOST.026",
        "message": "Μη επιτρεπτό ποσό"
    },
    {
        "code": "HOST.046",
        "message": "Λάθος κωδικός θεσμικού επενδυτή."
    },
    {
        "code": "HOST.047",
        "message": "Ανύπαρκτη αίτηση."
    },
    {
        "code": "HOST.048",
        "message": "Ανύπαρκτος στο πελατολόγιο."
    },
    {
        "code": "HOST.049",
        "message": "Υπαρκτός αριθμός αίτησης."
```


Figure 43 - error codes

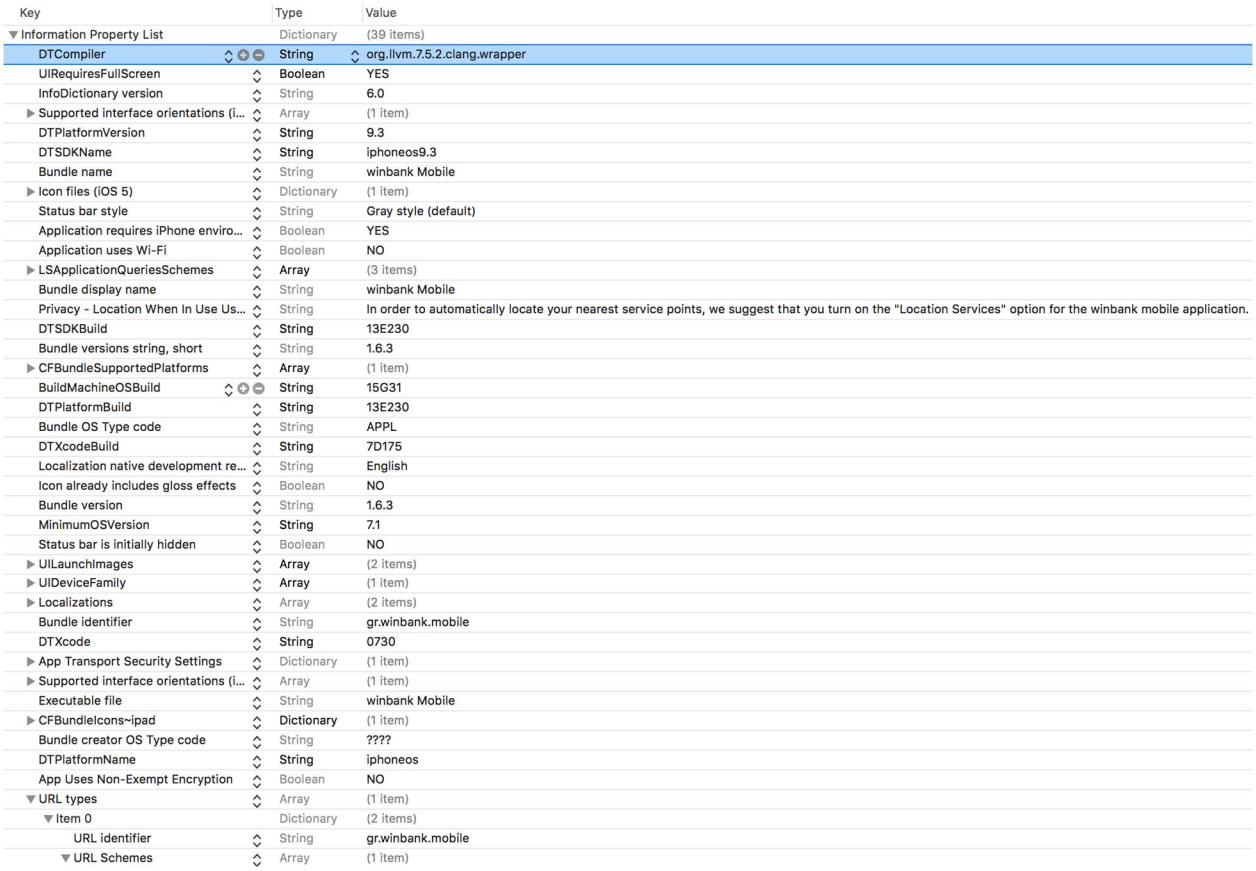
5.5.3 Alpha Bank Mobile Banking App

In this sub-scenario we tried to gather information from Alpha Bank Mobile Banking App. However, we didn't manage to obtain any data including not so sensitive data related to the UI. Moreover, the Clutch tool was not able to decrypt the app.

5.5.4 winbank App

At first glance, the winbank app (Piraeus Bank) was better protected compared to the rest of the m-banking apps, due to not be usable at jailbroken mobile devices. As a result, it was unable to login, so we didn't expect to get sensitive data related to the card. However, after further investigation we collected several data. In Figure 44 (a) apart from the data related to the UI we observe data such as:

- *Minimum iOS version*
- *Location usage*
- *WiFi usage*
- *URL scheme*
- *Xcode version*



Key	Type	Value
▼ Information Property List	Dictionary	(39 items)
DTCompiler	String	org.llvm.7.5.2.clang.wrapper
UIRequiresFullScreen	Boolean	YES
InfoDictionary version	String	6.0
▶ Supported interface orientations (i...	Array	(1 item)
DTPlatformVersion	String	9.3
DTSDKName	String	iphoneos9.3
Bundle name	String	winbank Mobile
▶ Icon files (iOS 5)	Dictionary	(1 item)
Status bar style	String	Gray style (default)
Application requires iPhone enviro...	Boolean	YES
Application uses Wi-Fi	Boolean	NO
▶ LSApplicationQueriesSchemes	Array	(3 items)
Bundle display name	String	winbank Mobile
Privacy - Location When In Use Us...	String	In order to automatically locate your nearest service points, we suggest that you turn on the "Location Services" option for the winbank mobile application.
DTSDKBuild	String	13E230
Bundle versions string, short	String	1.6.3
▶ CFBundleSupportedPlatforms	Array	(1 item)
BuildMachineOSBuild	String	15G31
DTPlatformBuild	String	13E230
Bundle OS Type code	String	APPL
DTXcodeBuild	String	7D175
Localization native development re...	String	English
Icon already includes gloss effects	Boolean	NO
Bundle version	String	1.6.3
MinimumOSVersion	String	7.1
Status bar is initially hidden	Boolean	NO
▶ UILaunchImages	Array	(2 items)
▶ UIDeviceFamily	Array	(1 item)
▶ Localizations	Array	(2 items)
Bundle identifier	String	gr.winbank.mobile
DTXcode	String	0730
▶ App Transport Security Settings	Dictionary	(1 item)
▶ Supported interface orientations (i...	Array	(1 item)
Executable file	String	winbank Mobile
▶ CFBundleIcons~ipad	Dictionary	(1 item)
Bundle creator OS Type code	String	????
DTPlatformName	String	iphoneos
App Uses Non-Exempt Encryption	Boolean	NO
▼ URL types	Array	(1 item)
▼ Item 0	Dictionary	(2 items)
URL identifier	String	gr.winbank.mobile
▼ URL Schemes	Array	(1 item)

Figure 44 (a) - winbank plist file

In Figure 44 (b) we can see the Bundle version of the app and a lot of information in itunes store such as:

- *Purchase date*
- *User's name (first and last)*
- *Apple Id*
- *Vendor ID*

Key	Type	Value
gameCenterEverEnabled	Boolean	NO
gameCenterEnabled	Boolean	NO
bundleDisplayName	String	winbank Mobile
artistId	Number	436.776.671
product-type	String	ios-app
kind	String	software
genreId	Number	6.015
artistName	String	Piraeus Bank S.A.
itemName	String	winbank Mobile
softwareIcon57x57URL	String	http://a894.phobos.apple.com/us/r30/Purple62/v4/81/b1/96/81b196c3-163c-628e-b9e2-ff483bd11c8e/icon114x114.jpeg
bundleVersion	String	1.6.3
▶ softwareVersionExternalIdentifiers	Array	(12 items)
versionRestrictions	Number	16.843.008
▶ UIRequiredDeviceCapabilities	Dictionary	(0 items)
▶ asset-info	Dictionary	(2 items)
▼ com.apple.iTunesStore.downloadInfo	Dictionary	(2 items)
purchaseDate	String	2016-09-05T18:31:31Z
▼ accountInfo	Dictionary	(20 items)
CreditDisplayString	String	
AccountIsNewCustomer	Boolean	NO
AltDSID	String	000774-10-991aae32-4c21-4f22-a86b-e6e79c09118c
AccountSocialEnabled	Boolean	NO
AccountPaidPurchasesPasswordSetting	Number	2
DownloaderID	Number	0
FamilyID	Number	0
DidFallbackToPassword	Boolean	NO
LastName	String	Menesidis
AccountKind	Number	0
AccountURLBagType	String	production
FirstName	String	Michael
AppleID	String	menesidis@gmail.com
AccountAvailableServiceTypes	Number	0
AccountFreeDownloadsPasswordSetting	Number	3
PurchaserID	Number	8.021.156.197
DSPersonID	Number	8.021.156.197
AccountServiceTypes	Number	0
AccountSource	String	device
AccountStoreFront	String	143448-23,29 ab:WJ6jMoo2
fileExtension	String	.app
softwareVersionExternalIdentifier	Number	818.646.135
vendorId	Number	318.972
genre	String	Χρηματοοικονομικά
softwareIconNeedsShine	Boolean	YES

Figure 44 (b) - winbank plist file

Even though we didn't manage to login, the app knows who we are through the itunes store. Last but not least, Figure 45 depicts the non-verified certificate of winbank app which is signed from an unknown authority and such a practice is not suggested.

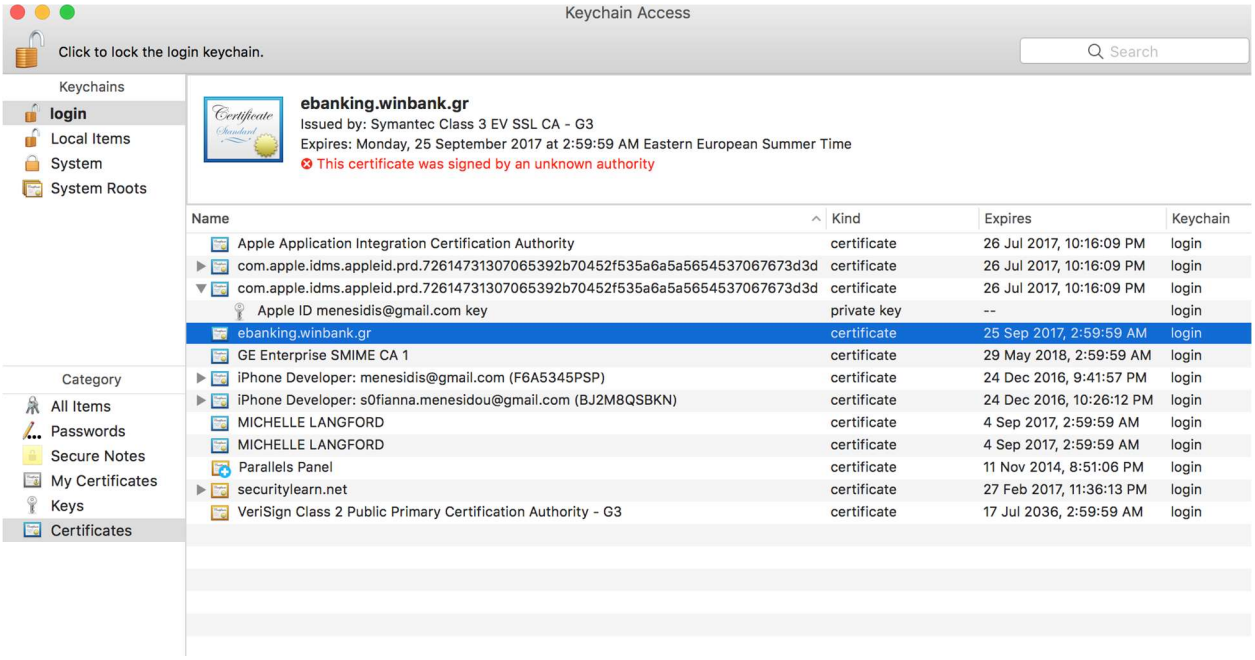


Figure 45 - winbank certificate

According to the aforementioned leaked information, the most secure mobile e-banking app was Alpha Bank Mobile Banking App, because we didn't manage to collect any information. Moreover, the mechanism to forbid jailbroken users to run the app (e.g. winbank app) adds another layer of security in the app and it would be a good practice to be used by all the e-banking apps.

Chapter 6 - Conclusion

This chapter concludes the thesis, while summarises the evaluations and observations of the research. Specifically, the contributions of this thesis are concluded, followed by the limitations of the research.

6.1 Contributions

In this thesis, we address the issues of data and privacy leakage on iOS devices. The goal of the research was to highlight possible ways of gaining sensitive information from a mobile device. By using various forensic tools we achieve to leak information exposure user's privacy. Our main contribution was to feature data leakage through various case studies and evidence. In addition, in terms of future research activities, we plan to develop a tool for automatic leak discovery regardless of iOS version.

6.2 Limitations of the Research

Despite having met the objectives of this thesis, some decisions had to be taken that resulted in limitations imposed on the work. The decisions were caused by practical reasons, or to limit the effort spent in areas where no new insights could be expected. For instance we collected plenty of obfuscated files that we didn't manage to deobfuscated them. Apart from the obvious limitation which is the time of research, a significant restriction is the lack of research works in iOS comparing to android OS due to closed-source iOS platform. Another limitation is the access of an iPhone with one of the latest limitations imposed on the work. More precisely, the iOS version is of paramount importance because different versions could have different restrictions and bugs. These possible differences make necessary distinct research in every iOS version. In addition, in order to install the new iOS version a new jailbreak to our iPhone is needed. Moreover, iOS updates might conclude to security updates that could add extra limitations in our research. For instance, many popular forensic tools don't work at specific iOS version. For that reason, either we have to develop a new tool or patch the existing.

Bibliography

[1] Egele, M., Kruegel, C., Kirda. E. and Vigna, G., (2011),"PiOS: Detecting Privacy Leaks in iOS Applications", In Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, http://www.seclab.tuwien.ac.at/papers/egele-ndss11.pdf

[2] ABI Research, (2016), "45 Million Windows Phone and 20 Million BlackBerry 10 Smartphones in Active Use at year end.", http://www.abiresearch.com/press/45-million-windows-phone-and-20-million-blackberry (accessed 11 September 2016)

[3] Enck, W., Gilbert, P., Chun, B.-G., Cox, L.P., Jung, J, McDaniel, P. and Sheth, A.N., (2010), "TaintDroid: an information-flow tracking system for real time privacy monitoring on smartphones." In Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'10). USENIX Association, Berkeley, CA, USA, 393-407, https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Enck.pdf

[4] Yang, Z., and Yang, M., (2012), "LeakMiner: Detect Information Leakage on Android with Static Taint Analysis." In Proceedings of the 2012 Third World Congress on Software Engineering (WCSE '12). IEEE Computer Society, Washington, DC, USA, 101-104. DOI:http://dx.doi.org/10.1109/WCSE.2012.26

[5] Gibler, C., Crussell, J., Erickson, J., and Chen, H., (2012), "AndroidLeaks: Automatically Detecting Potential Privacy Leaks In Android Applications on a Large Scale", In Proceedings of the 5th international conference on Trust and Trustworthy Computing (TRUST'12). Springer-Verlag Berlin, Heidelberg, 291-307. DOI=http://dx.doi.org/10.1007/978-3-642-30921-2_17 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.298.5095&rep=rep1&type=pdf

[6] J. Han, Q. Yan, D. Gao, J. Zhou, and R. H. Deng., (2013), "Comparing Mobile Privacy Protection through Cross-Platform Applications.", In Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, February 2013, http://www.internetsociety.org/sites/default/files/06_2_0.pdf

[7] Achara, J.P., Lefruit, J.D., Roca, V. and Castelluccia, C., (2014), Detecting Privacy Leaks in the RATP App: how we proceeded and what we found, *Journal of Computer Virology and Hacking Techniques*, 10 (4), 229-238, https://hal.inria.fr/hal-00872967/file/ratp_app_analysis.pdf

[8] Y. Agarwal and M. Hall., (2013), "ProtectMyPrivacy: Detecting and Mitigating Privacy Leaks on iOS Devices Using Crowdsourcing.", In Proceedings of the ACM International Conference on Mobile Systems, Applications and Services (MobiSys), Taipei, June 2013, http://www.synergylabs.org/yuvraj/docs/Agarwal_MobiSys2013_ProtectMyPrivacy.pdf

[9] Werthmann, T., Hund, R., Davi, L., Sadeghi, A.-R. and Holz, T., (2013), "PSiOS: Bring Your Own Privacy & Security to iOS Devices." In 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS), May, 2013, https://www.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/PSiOS.pdf

[10] Chen, T., Ullah, I., Kaafar, M.A., and Boreli, R., (2014), "Information leakage through mobile analytics services." In Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile '14). ACM, New York, NY, USA, , Article 15, 6 pages. DOI:http://dx.doi.org/10.1145/2565585.2565593

[11] Li, L., Bartel, A., Klein, J., Traon, Y., Arzt, S., Rasthofer, S., Bodden, E., Octeau, D., McDaniel, P., (2014), "I know what leaked in your pocket: uncovering privacy leaks on Android Apps with Static Taint Analysis", ISBN: 978-2-87971-129-4, 2014, https://hal.archives-ouvertes.fr/hal-00985490/document

[12] Haris, M., Haddadi, H., and Hui, P., (2014), "Privacy leakage in mobile computing: Tools, methods, and characteristics," arXiv.org e-Print archive, arXiv:1410.4978v1, 2014, http://arxiv.org/pdf/1410.4978.pdf

[13] Apple, (2016), "iOS Security", White Paper, https://www.apple.com/business/docs/iOS_Security_Guide.pdf

[14] Clutch, (2016), https://github.com/KJCracks/Clutch (accessed 11 September 2016).

[15] ibtool, (2014), https://github.com/davidquesada/ibtool (accessed 11 September 2016).

[16] Zdziarski, J., (2012), "Hacking and Securing iOS Applications", O'Reilly Media, Inc., http://www.it-docs.net/ddata/779.pdf

[17] Keychain Dumper, (2015), https://github.com/ptoomey3/Keychain-Dumper (accessed 11 September 2016)

[18] Xing, L., Bai, X., Li, T., Wang, X., Chen, K. and Liao, X., (2015), "Unauthorized Cross-App Resource Access on MAC OS X and iOS", arXiv.org e-Print archive, arXiv:1505.06836, https://reverse.put.as/wp-content/uploads/2015/11/report_unauthorized_app.pdf

[19] Wikipedia, (2016), https://en.wikipedia.org/wiki/.ipa_(file_extension) (accessed 11 September 2016)

[20] Apple, (2015), https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/NibFile.html (accessed 11 September 2016)

[21] Apple, (2016), https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIWindow_Class/#//apple_ref/occ/instp/UIWindow/rootViewController (accessed 11 September 2016)

[22] Apple, (2016), https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDevice_Class/#//apple_ref/occ/instp/UIDevice/identifierForVendor (accessed 11 September 2016)

[23] Apple, (2016), https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/ConfiguringYourApp/ConfiguringYourApp.html (accessed 11 September 2016)

[24] Apple, (2016), https://developer.apple.com/library/mac/documentation/Darwin/Reference/ManPages/man1/plutil.1.html (accessed 11 September 2016)

[25] Apple, (2014), https://itunes.apple.com/us/app/airport-utility/id427276530?mt=8 (accessed 11 September 2016)

[26] Mayer, D.A., (2015), "Blackbox iOS App Assessments Using idb", Whitepaper, https://www.blackhat.com/docs/ldn-15/materials/london-15-Mayer-Blackbox-iOS-Application-Assessments-Using-IDB-wp.pdf