An Empirical Study of Pentesting IOS 9 Applications

Sinan Ameen Noman and Haitham Ameen Noman

Abstract— **The mobile has been changed since Apple has released their first iPhone in 2007. Mobile applications have become more powerful than ever, this evolution has created a sufficient range of attacks that can be used on apps and grab a credential data from users who starts making activities on their mobile devices while on the go instead of using it on their own computers. This paper sheds the light on pentesting tools and techniques that can be conducted on iOS applications on a physical iDevice running iOS 9 rather than an emulator.**

Index Term— **Pentesting Tools, Pentesting Techniques, iOS 9**

I. Introduction

iOS is a mobile operating system created by Apple Inc. and assigned exclusively to iDevices (iPhone, iPod, iPad, TV, Apple Watch). iOS in a very simple way is an operating system that runs iPad, iPhone and many other devices that is worshiped by Apple. it's Unix based operating system that is derived from OS X and OS X shares its origin from the Darwin foundation. iOS is not an open source, however parts of the iOS typically taken from open source projects and need to probably be given back due to license constrains such as (CCTools, GDB, etc.) [1]. The only familiarity that iOS shares is when the users use the system; pretty much feel like any other Unix or Linux based system. All iOS devices are powered by processors based on ARM Architecture [2]. The ARM architecture is very different from X86 family which is used on PCs. The ARM architecture has its own assembly language, API's, Functions and arguments. Apple pushes major iOS update to users every year via iTunes or over the air.

The latest update of iOS is 9.2 that has been adopted by 50% of iDevices users while others are still using iOS 8 or older versions. Nowadays, Mobile phones have become a ubiquitous portable device used in our lives and according to Statista [3], there are more than 1.5 million approved apps available in the App Store. App Store provides a large variety of different applications include some free and others are paid apps. IOS Developers use Apple Xcode IDE for developing an application written in Objective-C [4] or Swift [5] programming language for iOS. With Xcode developers can test their application within iOS simulator that is already embedded in Xcode. iOS simulator compiles application into a low level language (local native) which is totally different than android emulator that is used to compile the application into ARM instructions. To test an application on real device, developers have to subscribe to iOS developer program that costs 100$/year because iOS devices is only allowed to run Apple signed applications. Developers need to read and follow Apple Store Guidelines [7] to ensure that their application will successfully submitted into Apple Store. Applications can be submitted into the App Store by using iTunes Connect [8] and developers should wait for Apple App Review "reviewing every app based on app reliability, and performance" that is usual took 8 days according to AppReviewTimes [9]. Jay Freeman (also known as "Saurik") [10] has created a GUI application called Cydia [11] that is used to grant root access [12] into iOS device and allow developers to install any packages, applications, themes, and extensions without being signed/reviewed by Apple. Apple claims that jailbreaking an iOS device will remove security layers that are designed to protect user's personal information and cause instability, battery drain, and other issues [13].

This paper will provide an overview of iOS security mechanism, techniques and tools that can be used to pentest an iOS app on iOS 9. The remainder of this paper is organized as follows. Section 2 describes iOS security measures. Section 3 presents the tools and techniques that the pentester needs in order to explore and modify iOS application. Finally, Section 4 presents our conclusions.

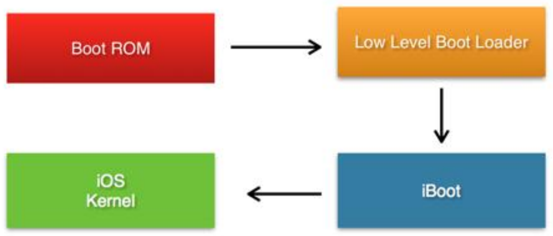
II. iOS Security

The security architecture of iOS is pretty much layered in such a way that is actually very difficult to break through it [1]. There is an encryption happening which starts right from the hardware level extends into the operating system, file system and applications. Also there are things such as data protection using encryption, application sandboxes, code signing and bunch of other things.

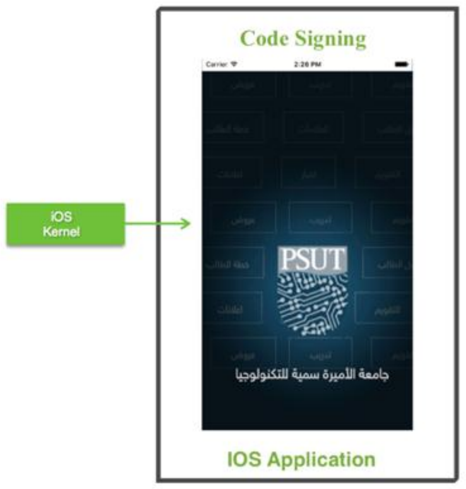
The whole boot process is very critical and important in operating systems. In PC's world a lot of boot levels (Trojan, Rootkit, etc.) have been around for quite some time which compromise the integrity of the operating system. Apple has come up with a solution to avoid PC's boot problems by creating a Secure Boot Chain that is loaded in a protected area inside the memory and cannot be tampered. Secure Boot Chain consists of (Boot Rom, Low Level Boot Loader, iBoot, and iOS kernel).

The Boot Rom contains Apple Root Certificate Authority "CA", the Device Unique ID "UID" key, and Device Group ID key "GID". The Boot Rom will verify the signature of low level boot loader "LLB" before loading and executing it and the LLB verification happens using the Apple Root CA key which is currently in the Boot Rom, and this will ensure that LLB is the one that sent by Apple. Now, the LLB intern boot the second stage of the boot loader called "iBoot" that will verify the iBoot to ensure that is signed by Apple. Finally, the iBoot will boot up the iOS kernel and it will verify again that the iOS kernel has been signed successfully by Apple. Fig.1. illustrates the secure boot chain of iOS devices.

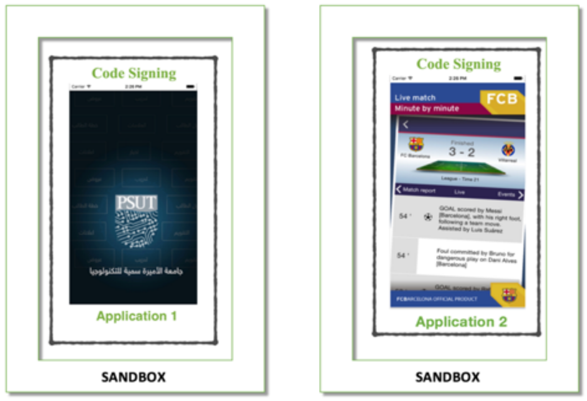
I J E N S



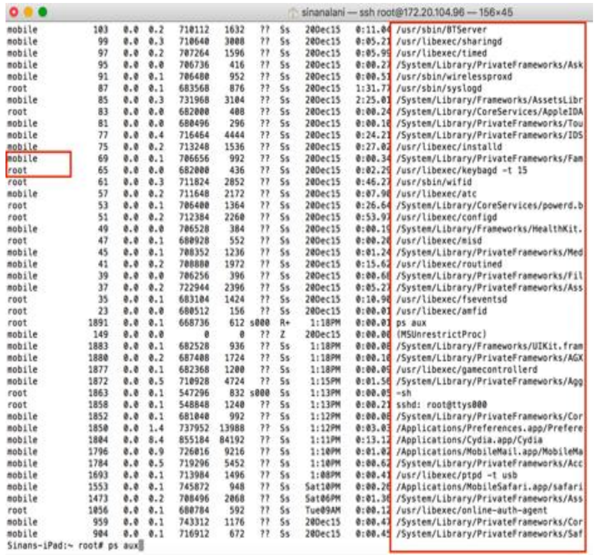
The kernel is loaded into a protected area of memory to thwart it from being overwritten by apps or other parts of the OS and it is used to load up all different operating system's components and processes. The kernel will verify that all applications that are running are signed by Apple and this process called "Code signing" as it shown in Fig.2.



The application level security does not end at code signing; Applications itself are isolated using sandboxes, that means "Application 1"cannot access the data of "Application 2", run time injection of one application into other or probably reading the other application directories is not allowed and not possible. Fig.3. illustrates iOS application sandboxing.

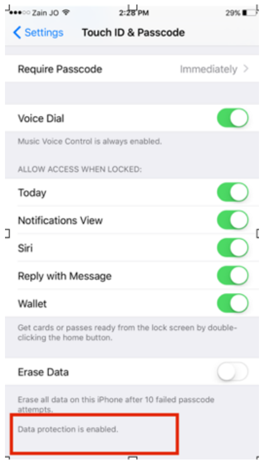


All applications are running under user "Mobile" and all system processes are running under user "Root" as it shown in Fig.4.



```
mobile      103  0.0  0.2  710112  1632  ??  Ss  20Dec15  0:11.04 /usr/sbin/BTServer
mobile       99  0.0  0.3  710640  3008  ??  Ss  20Dec15  0:05.21 /usr/libexec/sharingd
mobile       97  0.0  0.2  707264  1596  ??  Ss  20Dec15  0:05.95 /usr/libexec/timed
mobile       95  0.0  0.0  706736   416  ??  Ss  20Dec15  0:00.27 /System/Library/PrivateFrameworks/Ask
mobile       91  0.0  0.1  706480   952  ??  Ss  20Dec15  0:00.51 /usr/sbin/wirelessproxd
root         87  0.0  0.1  683568   876  ??  Ss  20Dec15  1:31.77 /usr/sbin/syslogd
mobile       85  0.0  0.3  731968  3104  ??  Ss  20Dec15  2:25.01 /System/Library/Frameworks/AssetsLibr
root         83  0.0  0.0  682000   408  ??  Ss  20Dec15  0:00.24 /System/Library/CoreServices/AppleIDA
mobile       81  0.0  0.0  680496   296  ??  Ss  20Dec15  0:00.16 /System/Library/PrivateFrameworks/Tou
mobile       77  0.0  0.4  716464  4444  ??  Ss  20Dec15  0:24.21 /System/Library/PrivateFrameworks/IDS
mobile       75  0.0  0.2  713248  1536  ??  Ss  20Dec15  0:27.02 /usr/libexec/installd
mobile       69  0.0  0.1  706656   992  ??  Ss  20Dec15  0:00.34 /System/Library/PrivateFrameworks/Fam
root         65  0.0  0.0  682000   436  ??  Ss  20Dec15  0:02.29 /usr/libexec/keybagd -t 15
root         61  0.0  0.3  711824  2852  ??  Ss  20Dec15  0:46.27 /usr/sbin/wifid
mobile       57  0.0  0.2  711648  2172  ??  Ss  20Dec15  0:07.90 /usr/libexec/atc
root         53  0.0  0.1  706400  1364  ??  Ss  20Dec15  0:26.64 /System/Library/CoreServices/powerd.b
root         51  0.0  0.2  712384  2260  ??  Ss  20Dec15  0:53.97 /usr/libexec/configd
mobile       49  0.0  0.0  706528   384  ??  Ss  20Dec15  0:00.19 /System/Library/Frameworks/HealthKit.
root         47  0.0  0.1  680928   552  ??  Ss  20Dec15  0:00.20 /usr/libexec/misd
mobile       45  0.0  0.1  708352  1236  ??  Ss  20Dec15  0:01.24 /System/Library/PrivateFrameworks/Med
mobile       41  0.0  0.2  708880  1972  ??  Ss  20Dec15  0:15.62 /usr/libexec/routined
mobile       39  0.0  0.0  706256   396  ??  Ss  20Dec15  0:00.60 /System/Library/PrivateFrameworks/Fil
mobile       37  0.0  0.2  722944  2396  ??  Ss  20Dec15  0:05.27 /System/Library/PrivateFrameworks/Ass
root         35  0.0  0.1  683104  1424  ??  Ss  20Dec15  0:10.90 /usr/libexec/fseventsd
root         23  0.0  0.0  680512   156  ??  Ss  20Dec15  0:00.01 /usr/libexec/amfid
root       1891  0.0  0.1  668736   612 s000  R+   1:18PM  0:00.01 ps aux
mobile      149  0.0  0.0       0     0  ??  Z   20Dec15  0:00.00 (MSUnrestrictProc)
mobile     1883  0.0  0.1  682528   936  ??  Ss   1:18PM  0:00.08 /System/Library/Frameworks/UIKit.fram
mobile     1880  0.0  0.2  687408  1724  ??  Ss   1:18PM  0:00.10 /System/Library/PrivateFrameworks/AGX
mobile     1877  0.0  0.1  682368  1200  ??  Ss   1:18PM  0:00.09 /usr/libexec/gamecontrollerd
mobile     1872  0.0  0.5  710928  4724  ??  Ss   1:15PM  0:01.56 /System/Library/PrivateFrameworks/Agg
root       1863  0.0  0.1  547296   832 s000  Ss   1:13PM  0:00.05 -sh
root       1858  0.0  0.1  548848  1240  ??  Ss   1:13PM  0:00.21 sshd: root@ttys000
mobile     1852  0.0  0.1  681040   992  ??  Ss   1:12PM  0:00.08 /System/Library/PrivateFrameworks/Cor
mobile     1850  0.0  1.4  737952 13988  ??  Ss   1:12PM  0:03.03 /Applications/Preferences.app/Prefere
mobile     1804  0.0  8.4  855184 84192  ??  Ss   1:11PM  0:13.12 /Applications/Cydia.app/Cydia
mobile     1796  0.0  0.9  726016  9216  ??  Ss   1:10PM  0:01.02 /Applications/MobileMail.app/MobileMa
mobile     1784  0.0  0.5  719296  5452  ??  Ss   1:10PM  0:00.62 /System/Library/PrivateFrameworks/Acc
mobile     1693  0.0  0.1  713984  1496  ??  Ss   1:08PM  0:00.41 /usr/libexec/ptpd -t usb
mobile     1553  0.0  0.1  745872   948  ??  Ss  Sat10PM  0:00.26 /Applications/MobileSafari.app/safari
mobile     1473  0.0  0.2  708496  2068  ??  Ss  Sat06PM  0:01.30 /System/Library/PrivateFrameworks/Ass
root       1056  0.0  0.1  680784   592  ??  Ss  Tue09AM  0:00.12 /usr/libexec/online-auth-agent
mobile      959  0.0  0.1  743312  1176  ??  Ss  20Dec15  0:00.47 /System/Library/PrivateFrameworks/Cor
mobile      904  0.0  0.1  716912   672  ??  Ss  20Dec15  0:00.45 /System/Library/PrivateFrameworks/Saf
Sinans-iPad:~ root# ps aux
```

In addition to the system security, Apple offers File Data Protection and encryption to their iOS devices. File data protection is available for devices that offers hardware encryption starting from iPhone3GS or later, iPod touch (3rd generation or later), and all iPad models. The file data protection is used to protect data and files even when the iDevice has been stolen or compromised by generating 256-bit key for every file on the system. This service is enabled by default in iOS and users should enable it by enable a Touch ID which is fingering sensing system that makes secure access to the iOS device in a fast and easy way or Passcode on the iOS device. The passcode/Touch ID are integrated with UID to protect the iOS device from any password attack by using an iteration design that makes as it shown in Fig.5.



Zain JO 2:28 PM 29%

Settings Touch ID & Passcode

Require Passcode Immediately

Voice Dial

Music Voice Control is always enabled.

ALLOW ACCESS WHEN LOCKED:

Today

Notifications View

Siri

Reply with Message

Wallet

Get cards or passes ready from the lock screen by double-clicking the home button.

Erase Data

Erase all data on this iPhone after 10 failed passcode attempts.

Data protection is enabled.

Most of the encryption techniques are depends on UID and GID Keys that is embedded into hardware. Apple claims that they don't know the UID and GID keys nor their suppliers. Every device could end up having a different encryption key which is used for file and data encryption that has different things associated with such as (Keychain, KeyBags, File based encryption).

Apple also provide support for secure data delivery over the network and it supports "SSL, TLS, VPN, Wi-Fi (EAP-TLS, PEAP, TTLS, etc.).

These restrictions have led Jay Freeman to find vulnerabilities in iOS that will lead to install application called Cydia "Jailbreak "that is used to provide freedom from all of these restrictions.

III. Pentration Test

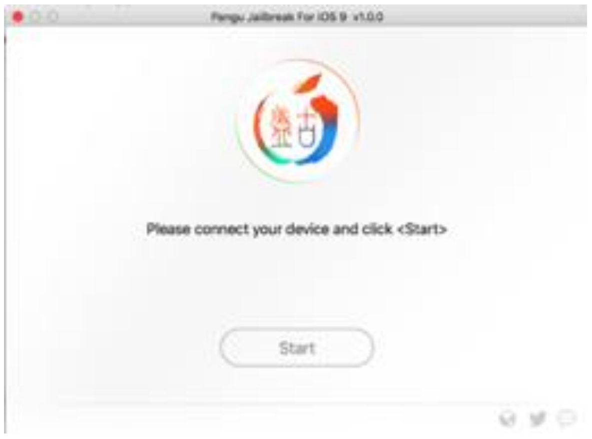
In order to build an iOS pentesting environment, we need to Jailbreak an iOS device to get off Apple security restrictions to achieve reliable results.

This section will focus on iOS application instead of iOS software and we need to install a set of tools that are not signed by Apple and jailbreak an iOS device build our pentesting environment. To perform pentesting we will use iPad Air with iOS 9.0.2.

IV. Jailbreak

Every iOS version has its own jailbreak tool because apple used to patch a security hole that permits application to bypass code signing whenever they release a new update. Jailbreak tools for the iOS devices have always been free and users who are looking for a Jailbreak should be aware from scam websites who claims they offer a Jailbreak tool.

In order to pentest iOS application, we need to perform a jailbreak tool. This paper will use iPad Air running iOS 9.0.2 along with Pangu untethered Jailbreak tool for iOS 9 [14]. An untethered jailbreak tool, meaning that the iDevice does not require to be connected to a computer each time when the device boot. Jailbreaking an iOS device is quite easy; download Pangu tool, install it on computer, connect your iDevice to computer using the cable that comes with the device, and follow the steps that Pangu tool prompt. Fig.6. illustrates a screenshot of Pangu tool running on Mac OS X.


Pangu Jailbreak For iOS 9 v1.0.0
Please connect your device and click <Start>
Start

After installing Cydia, the pentester will see a new app icon on the jailbroken device called Cydia; open the app and choose Developer Mode.

V. Pentration Test Tools

There are many unsigned Cydia apps, tweaks and packages that could help the pentester's job easy and enable them to work in an efficient way to do penetration test. This section will illustrate the pentesting tools were deemed so essential to be used in penetration iOS applications and can be listed hereunder.

OpenSSH: This tool is a console package Cydia tool that is used to secure remote access between an iOS device and a computer. The pentester should change the default root password of the iOS which is "alpine" to prevent the possibility of unsavory people remotely logging into device [15].

IMazing: This tool is the Swiss Army Knife of iOS devices that is used to explore (Apps, Messages, Contacts, Photo, Plist Files, etc.) in a simple and easy way instead of using iTunes. This tool is available on mac OS X and Windows operating systems [16].

TextWrangler: This tool is an open source text editor mac OS X tool that help the pentester to read/modify Plist [17].

Adv-cmds: This tool is a console package Cydia tool that will help the pentester to use set of advanced commands (ps aux, kill, etc.) when they are connected to iOS device via OpenSSH [18].

House Arrest Fix: This Cydia Tweak will allow the pentester to access app containers and explore the iOS device via USB on iOS 8.3 or later [19].

Cyberduck: This is an FTP, SFTP browsing tool that allow the pentester to browse the content of the iOS device easily and copy any type of file inside the iOS with the ability to set permission on files [20].

Keychain Dumper: This tool helps the pentester to dump "Internet Password" or "Generic Passwords" only [21].

VI. Play Around with PLIST Files

Plist is stand for Property List. Every iOS, OS X has its own Plist files that are often used to store user's settings, application configuration and sometimes there are applications that used to store clear text sessions, usernames and passwords. Pentester may escalate the privilege to login into application as an administrator by changing Plist entry "Administrator =0 "to "Administrator=1" during the penetration test and open the app to see that your account has been escalated to administrator privilege.

iOS applications store Plist files inside the app container [Developer/Library/Preferences] and the pentester cannot access these files even if the device was jailbroken on iOS 9 and need to install the following tweak "House Arrest Fix "in order to access these Plist files. The extension of Plist file is "Plist". IMazing is a tool that could help us in exploring and modifying Plist files. Below is an example of how to pentest an iOS application by modify Plist file that is associated with the application itself. We will demonstrate how to pentest an iPad game "2048" that stores the game score in Plist file under applications home directory by using IMazing tool. Fig.7. illustrates a screenshot of "2048" game in the Apple Store.

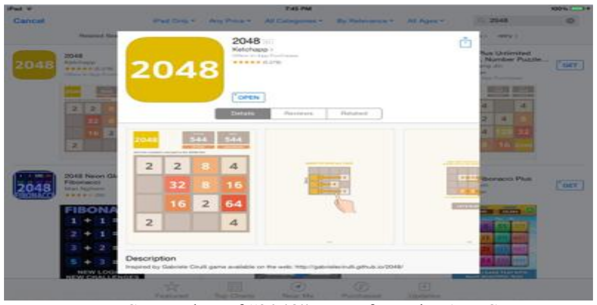
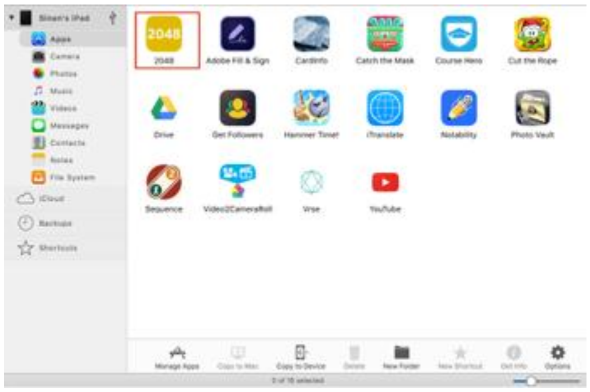


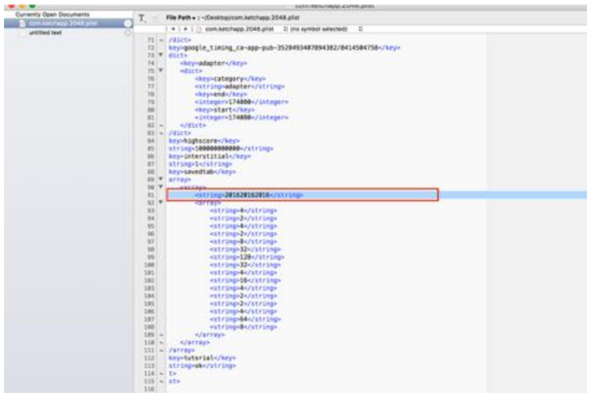
Fig. 7. Screenshot of "2048" game from the App Store

The Pentester should download and install IMazing tool and connect the iOS device to the computer via USB to start explore 2048 app and navigate through 2048 Developer —>Library —> Preferences Folder.

Fig.8. illustrates a screenshot of how apps are looking when the pentester open IMazing app

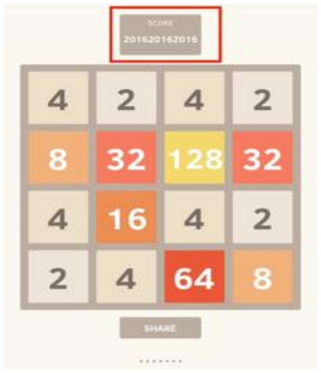


The pentester should get a copy of Plist file into the desktop to see what kind of data can be by using a text editor such as TextWrangler as it shown in Fig.9.



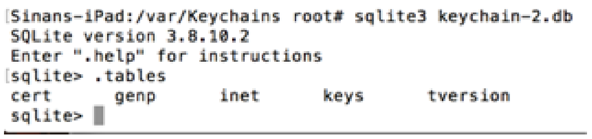
```
/dict>
key>google_timing_ca-app-pub-3528493487894382/8414504758</key>
dict>
    <key>adapter</key>
    <dict>
        <key>category</key>
        <string>adapter</string>
        <key>end</key>
        <integer>174880</integer>
        <key>start</key>
        <integer>174880</integer>
    </dict>
/dict>
key>highscore</key>
string>100000000000</string>
key>interstitial</key>
string>1</string>
key>savedtab</key>
array>
    <array>
        <string>201620162016</string>
        <array>
            <string>4</string>
            <string>2</string>
            <string>4</string>
            <string>2</string>
            <string>8</string>
            <string>32</string>
            <string>128</string>
            <string>32</string>
            <string>4</string>
            <string>16</string>
            <string>4</string>
            <string>2</string>
            <string>2</string>
            <string>4</string>
            <string>64</string>
            <string>8</string>
        </array>
    </array>
/array>
key>tutorial</key>
string>ok</string>
t>
st>
```

In our scenario, we will edit the high score of the game "2048" and then copy it back from desktop to the app container in order to apply the modification that we have made. The result of the modification can be shown in the Fig.10.



VII. Dumping Keychain

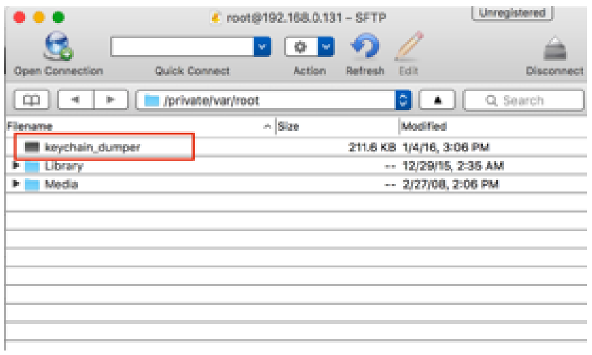
Keychain is a folder that implements SQLite database holds all passwords, identities, encryption keys, private keys, and certificates. This folder is used to be encrypted with a hardware specific key that is unique per device and cannot be extracted from the iOS device itself (AES 128 bit AES Algorithm) and that means that these data cannot be moved into another device. iOS applications store the user's sensitive information in the keychain to issue clear authentication and to not to ask the user every time for login. iOS applications use the keychain API's to write and read data from and to the Keychain. The pentester can access the Keychain database at (/private/var/Keychains). Keychain is a great solution for developers to store sensitive data in an encrypted way instead of storing it as a plaintext in Plist file. All keychain items are saved in 4 tables (Cert, Genp, Inet, keys) as it shown in the Fig.11.



```
Sinans-iPad:/var/Keychains root# sqlite3 keychain-2.db
SQLite version 3.8.10.2
Enter ".help" for instructions
sqlite> .tables
cert       genp       inet       keys       tversion
sqlite> 
```

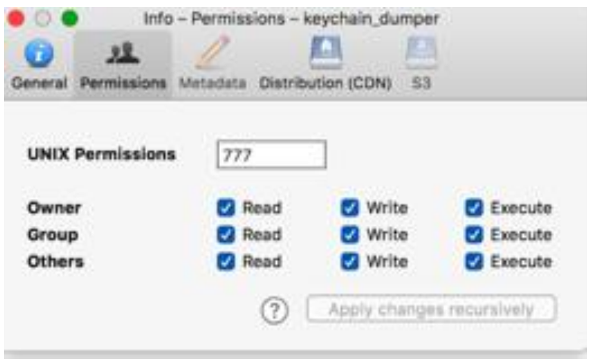
Cert table contains certificates, Genp table contains general passwords, Inet table contains internet passwords and finally keys contains keys and digital identity keychain items.

The pentester will need to use Keychain Dumper tool in order to extract all tables and examining the tables contents. This tool can be added easily into the iOS device by using an easy FTP browsing tool. In our example we are going to use Cyberduck tool that is available on OS X and Windows operating systems to copy the Keychain Dumper tool in iOS device as it shown in Fig.12.



```
Filename          Size       Modified
keychain_dumper   211.6 KB   1/4/16, 3:06 PM
Library           --         12/29/15, 2:35 AM
Media             --         2/27/08, 2:06 PM
```

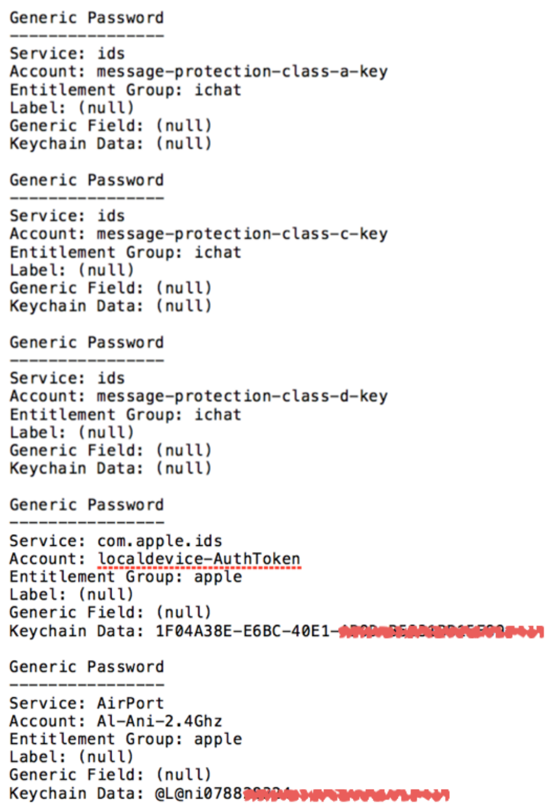
The pentester should now change the file permission of "Keychain_Dumper" file into (777) in order to make the file able to (Read / Write /Modify) as it shown in the Fig.13.


Info – Permissions – keychain_dumper
General
Permissions
Metadata
Distribution (CDN)
S3
Apply changes recursively

UNIX Permissions 777

Owner	✓ Read	✓ Write	✓ Execute
Group	✓ Read	✓ Write	✓ Execute
Others	✓ Read	✓ Write	✓ Execute

After the keychain_dumper has been copied and permission has been set correctly, the pentester should open the Terminal application and run the following command "./keychain_dumper" in order to extract all keychain sensitive data. As we can see from Fig.13., the service "Airport" which belongs to a Wi-Fi password can be shown as a plaintext, while other passwords are encrypted as depicted in Fig.14.



```
Generic Password
----------------
Service: ids
Account: message-protection-class-a-key
Entitlement Group: ichat
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: ids
Account: message-protection-class-c-key
Entitlement Group: ichat
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: ids
Account: message-protection-class-d-key
Entitlement Group: ichat
Label: (null)
Generic Field: (null)
Keychain Data: (null)

Generic Password
----------------
Service: com.apple.ids
Account: localdevice-AuthToken
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: 1F04A38E-E6BC-40E1-[redacted]

Generic Password
----------------
Service: AirPort
Account: Al-Ani-2.4Ghz
Entitlement Group: apple
Label: (null)
Generic Field: (null)
Keychain Data: @L@ni0788[redacted]
```

VIII. CONCLUSION

This paper has shed the light on iOS security measures that are used to protect applications from being modified and these data cannot be explored or modified without jailbreaking the iOS device. Jailbreaking iOS device will remove security layers that are designed to protect user's personal information

REFERENCES

[1] "IOS 9.0 - Source." IOS 9.0 - Source. Accessed January 16, 2016. http://opensource.apple.com/release/ios-90/.
[2] "ARM Processor Architecture." - ARM.Accessed January 16, 2016. http://www.arm.com/products/processors/instruction-set-architectures/
[3] "Number of Apps Available in Leading App Stores 2015 | Statistic." Statista. Accessed January 16, 2016. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
[4] "Programming with Objective-C." About Objective-C. Accessed January 16, 2016. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
[5] "The Swift Programming Language (Swift 2.1): About Swift." The Swift Programming Language (Swift 2.1): About Swift. Accessed January 16,2016.
[6] https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/

[7] “App Store Review Guidelines." - Apple Developer. Accessed January 16, 2016. https://developer.apple.com/app-store/review/guidelines/#legal-requirements

[8] "App Distribution Guide." Uploading Your App to ITunes Connect. Accessed January 16, 2016. https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/UploadingYourApptoiTunesConnect/UploadingYourApptoiTunesConnect.html#//Apple_ref/doc/uid/TP40012582-CH36-SW2

[9] "Average App Store Review Times." Average App Store Review Times. Accessed January 16, 2016. http://appreviewtimes.com

[10] Jay Freeman (Saurik). Accessed January 16, 2016. http://www.saurik.com

[11] "Cydia IOS App." Cydia. Accessed January 16, 2016. https://cydia.saurik.com

[12] What Is Root? -- Definition by The Linux Information Project (LINFO). Accessed January 16, 2016. http://www.linfo.org/root.html.

[13] "Apple." Unauthorized Modification of IOS Can Cause Security Vulnerabilities, Instability, Shortened Battery Life, and Other Issues. Accessed January 16, 2016. https://support.apple.com/en-us/HT201954.

[14] "IOS Security Guide." 2015. Accessed January16, 2016. http://www.apple.com/business/docs/iOS_Security_Guide.pdf

[15] "PanGu IOS 9 Jailbreak Tool." PanGu. Accessed January 16, 2016. http://en.pangu.io

[16] "OpenSSH · Cydia." Accessed January 16, 2016. https://cydia.saurik.com/openssh.html.

[17] "IMazing |iPhone, IPad & IPod Manager." IMazing. Accessed January 16, 2016. https://imazing.com.

[18] “Bare Bones Software | TextWrangler." Bare Bones Software | TextWrangler. Accessed January 16, 2016. http://goo.gl/gA3tuV

[19] “Adv-cmds · Cydia. Accessed January 16, 2016. http://cydia.saurik.com/package/adv-cmds/

[20] House Arrest Fix · Cydia. Accessed January 16, 2016. http://cydia.saurik.com/package/com.npupyshev.mobile.house-arrest/

[21] "Cyberduck |Libre FTP, SFTP Browser." Cyberduck. Accessed January 16, 2016. https://cyberduck.io/?l=en.

[22] "Keychain Dumper." GitHub. Accessed January 16, 2016. https://github.com/ptoomey3/Keychain-Dumper