A Peek Under the Hood of iOS Malware

Laura García
MLW.RE NPO, Spain
Email: laura@mlw.re

Ricardo J. Rodríguez
Seconda Università di Napoli, Caserta, Italy
University of Zaragoza, Spain
Email: rjrodriguez@ieee.org

***Abstract*—Malicious software specially crafted to proliferate in mobile platforms are becoming a serious threat, as reported by numerous software security vendors during last years. Android and iOS are nowadays the leaders of mobile OS market share. While malware targeting Android are largely studied, few attention is paid to iOS malware. In this paper, we fill this gap by studying and characterizing malware targeting iOS devices. To this regard, we study the features of iOS malware and classify samples of 36 iOS malware families discovered between 2009 and 2015. We also show the methodology for iOS malware analysis and provide a detailed analysis of a malware sample. Our findings evidence that most of them are distributed out of official markets, target jailbroken iOS devices, and very few exploit any vulnerability.**

***Keywords*-iOS, malware, attacks, threats, classification**

I. Introduction

The popularity of smartphones as substitute of personal computer devices – there is an estimated number of 7.22 billion mobile devices in use [1] – makes them attractive to cybercriminals. Malicious software (malware) specifically crafted targeting these devices are becoming a serious threat as numerous security vendors reported [2]–[5].

As stated in Q2 2015 market reports [6], Android dominated the smartphone Operating System (OS) market share with a 82.8%, clearly beating others platforms that had a 13.9% (iOS), 2.6% (Windows Phone), and 0.3% (Blackberry OS). This trend is followed by mobile malware threats: almost 5000 new Android malware files were found every day in 2015 [2]. Similarly, Android malware have been largely studied in the literature [7]–[11] (to name a few), but few attention has been focused on iOS (or other) platforms [11]. This paper tries to fill this gap by grabbing attention into iOS malware.

This lack of attention may be caused by several reasons. For instance, by the own market share: the higher the number of devices, the greater the probability of success of infection – and thus, the greater revenues. Thus, cybercriminals prefer Android instead of iOS as deployment platforms. Furthermore, the differences between Android and iOS security models are noticeable [12], [13]: both follow permission-based approaches (with different granularity) and incorporate platform protection mechanisms to prevent execution of arbitrary code at runtime; but unlike iOS, Android mainly relies on platform protection mechanisms rather than on market protection [11].

The market protection of iOS is based on a review/vetting conducted by Apple for any application to be published on their official app markets. Contrary to popular belief, this vetting process is insufficient to effectively block malware from entering the official markets, as recently happened with XcodeGhost malware family, which infected at least 39 apps published in Apple's official app market during last year. Other families used different attack vectors, such as enterprise/ad-hoc provisioning, abusing private APIs, or compromised iCloud accounts that enable to reach potential targets through official channels.

In this paper, we propose a classification of iOS malware regarding diverse features, such as affected devices, distribution channel, infection, attack goals, and attack vector used, using a similar approach to [11]. In particular, we classify samples of 36 iOS malware families from 2009 to 2015. We also show the methodology followed to iOS malware analysis and a detailed analysis of a real malware sample. Based on our results, we conclude that most of iOS malware are distributed out of official markets, target jailbroken iOS devices, and require user interaction to infect devices.

This paper is organized as follows. Section II reviews the related work. Then, Section III introduces the iOS security model. iOS malware features are described in Section IV. Section V presents the iOS malware families classification and discussion. A case study and methodology for iOS malware analysis are introduced in Section VI. Finally, Section VII concludes the paper.

II. Related Work

Malware targeting mobile platforms have been largely studied, specially focused on Android platforms [7]–[10]. In [7], a smartphone malware detection taxonomy was proposed that focuses on reference behavior, analysis approach, and malware behavior. Android malware behaviors were further studied in [8], where a tool is proposed to dynamically detect malicious activities in Android apps. In [9], a set of 46 mobile malware samples collected from January 2009 to June 2011 was analyzed, focused on current and future incentives. The set included 4 samples of iOS malware families that are also covered in our study. A large set of Android malware was collected in [10], where more than 1200 samples are analyzed and characterized according to attack type and installation method. It is worth also mentioning [14], where a mechanism to bypass Apple vetting based on using private API callings and trampoline functions was shown.

An extensive survey of mobile malware is found in [11]. A classification scheme was proposed based on attack

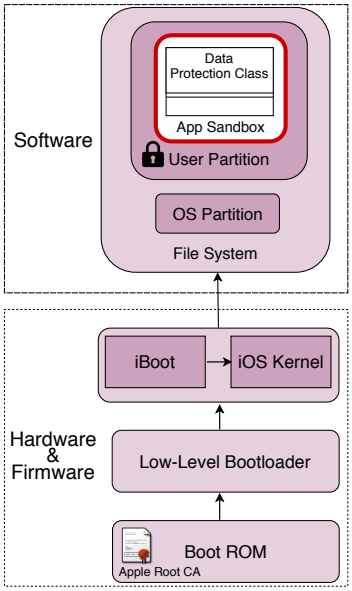

Figure 1. iOS security model.

goals and malware behavior, distribution and infection, and privilege acquisition. They classified 9 mobile malware samples that target iOS devices. In this paper, we follow a similar classification scheme, but with some differences. In particular, distribution and infection are divided into two different features since we distinguish how the malware are spreading and how the infection is performed. We also consider attack vectors used by malware as classification feature. Furthermore, we make a broader study of iOS malware families considering samples of malware families discovered from January 2009 to December 2015.

Regarding tools and methods to prevent attacks in iOS, it is worth mentioning the following. PiOS [15] uses a static analysis approach to detect sensitive information exfiltration. XiOS [16] is an application sandboxing that mitigates attacks such as lazy bindings or abuse of private APIs. iRiS [17] is an application vetting system that uses static and dynamic analysis to detect malicious activities, such as abuse of private APIs. Finally, the abuse of iOS sandboxing using different attack vectors and techniques for identifying apps vulnerable to these threats are proposed in [18].

III. Background

iOS combines software, hardware, and services working together to offer maximum security. These main elements (iOS 9.x onward) are sketched in Figure 1.

The iOS secure boot chain guarantees the lowest levels of software remain unmodified and the execution upon an Apple's valid device. Hence, an iOS device immediately executes when turned on an immutable code (known as *hardware root of trust*, setting up during chip fabrication) from the Boot ROM. This code contains the Apple Root CA public key, used to verify that the Low-Level Bootloader (LLB) is signed by Apple before loading. Next, the LLB verifies and executes the next-stage bootloader, named *iBoot*, which in turn verifies and executes the iOS kernel. The latter finally verifies and executes the full iOS OS, loading the OS partition and the user partition, where all applications are located.

Since hardware and firmware are digitally signed and verified prior execution in iOS devices, applications (apps) are among the most critical elements. Thus, iOS provides different protection layers to ensure that apps are signed, verified, and even isolated to protect user data. In the sequel, we briefly describe these layers.

- **Apple-issued certificate.** iOS enforces to all executable code and third-party apps to be signed using an Apple-issued certificate to ensure that they come from a known and verified source. Hence, iOS developers must verify their identity through the iOS Developer Program to obtain these certificates. Then, they can submit their apps to the official iOS app market, where as final verification they are reviewed to ensure to operate as intended and do not contain obvious flaws or other problems.
- **App sandbox.** Any third-party app is executed in an isolated way, thus preventing an app from gathering or modifying information stored by other app. They are executed as the non-privileged user "mobile" and the entire OS partition is mounted as read-only.
- **Data Protection.** When enabled (by setting up a device passcode), each data file is associated with a specific class that protects data based on when it needs to be accessed, supporting different levels of accessibility.
- **Other security mechanisms.** iOS incorporates well-known security mechanisms in OS level, such as Address Space Layout Randomization to prevent exploitation of memory corruption errors; and in hardware level, such as Data Execution Prevention, which enforces memory pages as writable but non-executable to prevent execution of injected code.

Any app deployed through official Apple app market (known as App Store) must comply with App Review Guidelines. The *Apple review/vetting process* ensures all applications are reliable, perform as expected, and are free of offensive material. This process encompasses a set of over 100 rules separated in different categories, covering diverse aspects such as functionality, meta-data, location, advertising, violence, privacy, and religion, among others.

A submitted app can be rejected by several reasons, such as execution crashing, inclusion of undocumented/hidden features, use of non-public APIs, data read or write outside its container area, or downloading external code. Note that, however, this process does not effectively block malware distribution (e.g., XcodeGhost and Youmi Ad SDK). Weaknesses, such as unofficial redistribution of Apple's official SDK (Xcode), obfuscation of private API calls, or abuse of inter-app interaction services were proved to bypass this vetting process.

IV. Characterization of iOS Malware

This section introduces the characterization of iOS malware carried out in this paper. We collected samples

of iOS malware families after reviewing security-related reports and papers (a subset of them along their description is available in [19]). Table I summarizes their names and discovery date, separated by discovery year. These malware families are categorized by types, depending on *who* are targeting individuals [19]:

- **On-sale malware.** This category includes software available for sale to the public to target individuals. Note that this software is not malicious in nature, but it can be used for malicious purposes. For instance, mSpy is a monitoring software that collects information from the target device (such as text messages, call information, or GPS coordinates). Parents worried by their children safety, or an employee interested in improving their workers productivity may use this software legitimately and with explicit consent.
- **State-sponsored malware.** This type of malware covers tools used by governments (and similar) to target individuals. Well-known examples are Hacking Team tools. State intelligence agencies may use these malware to spy on other states, activists, or journalists, for example.
- **Underground malware.** This category comprises specially crafted software used by cybercriminals to target individuals with malicious intention. As the personal computer malware, they aim at obtaining banking information, at stealing email/social media login credentials, or at profiling the user, among other goals.

We classify each malware family according to different features, as depicted in Figure 2. In the following, we describe these features in detail.

Affected Devices: We distinguished between non-jailbroken (*NJ*) and jailbroken devices (*JD*). iOS jailbreaking consists of removing software restrictions imposed by Apple's OS. Thus, using software exploits, a user can jailbreak any Apple's device. A jailbroken device permits root access to the file system (and hence, to download and install any application regardless its origin) and enables extra features such as SSH service. Let us remark that jailbreaking is a security problem itself, since enables also other applications to dispose of root access to the device. Furthermore, to jailbreak a device removes the intrinsic security mechanisms of Apple, such as code signing or kernel patch protection. Let us also note that iOS vulnerabilities that allow to jailbreak the device, allow malware to coexist. In fact, a malware may secretly jailbreak a non-jailbroken device as part of the infection process.

Distribution: Distribution describes from where a piece of malware is downloaded to the iOS device. We distinguish three sources of distribution: official market (*OM*), when the malware are distributed in App Store; alternative market (*AM*), when they used non-official markets, such as Cydia, AppCake, or Frozen installer; or unknown sources (*US*), when the download is done from untrusted places such as Internet websites.

Malware family name(s)	Discovery date
ON-SALE MALWARE	
Trapsms	Jun 2009
MobileSpy	Jul 2009
OwnSpy	Feb 2010
MobiStealth	Oct 2010
FlexiSpy	Dec 2010
iKeyGuard	April 2011
Copy9	Jul 2011
StealthGenie	Nov 2011
mSpy	Oct 2011
iKeyMonitor	Mar 2012
SpyKey	Apr 2012
Copy10	Aug 2012
InnovaSPY	Sept 2012
1mole	Jan 2013
Spy App	Oct 2014
STATE-SPONSORED MALWARE	
FinSpy Mobile	Aug 2012
Hacking Team tools	Jun 2014
Inception/Cloud Atlas	Dec 2014
XAgent/PawnStorm	Feb 2015
UNDERGROUND MALWARE	
Ikee/Eeki and Duh	Nov 2009
LBTM	Sept 2010
Find and Call	Jul 2012
Nobitazzz (packages)	Aug 2012
AdThief/Spad	Mar 2014
SSLCreds/Unflod Baby Panda	Apr 2014
AppBuyer	Sept 2014
WireLurker	Nov 2014
Xsser mRAT	Dec 2014
Lock Saver Free	Jul 2015
KeyRaider	Aug 2015
XcodeGhost	Sept 2015
YiSpecter	Oct 2015
Muda/AdLord	Oct 2015
Youmi Ad SDK	Oct 2015
TinyV	Oct 2015
SantaAPT	Dec 2015

Table I
SUMMARY OF iOS MALWARE FAMILIES INCLUDED IN THIS PAPER (ORDERED BY DISCOVERY DATE, FROM 2009 TO 2015).

Infection: Malware may infect iOS devices because the user installs (and thus explicitly allows) the malicious app to execute (*AS*) or because they exploit a vulnerability (*EV*). To this regard, we studied the vulnerabilities that affect non-jailbroken devices and surprisingly, although the number of vulnerabilities in iOS from 2007 to 2015 is appreciable (more than 8700), less than a 0.11% were used by the malware considered in this paper. In particular, these vulnerabilities affected to iOS version 9.X (CVE-2015-5837, CVE-2015-5880, CVE-2015-5876, CVE-2015-5839, and CVE-2015-5867), version 8.X (CVE-2015-5770, CVE-2015-3722, CVE-2015-3725, and CVE-2014-4494), and version 7.X (CVE-2014-1276).

Attack Goals: Malware are deployed by cybercriminals to obtain revenue in different ways. As attack goals, we discriminate among the following, according to [11]: spamming (*SM*), when compromised devices receive unwanted messages (via SMS, via dial, or via pop-ups) offering unsolicited services or iOS apps promoted; data theft (*DT*), when the goal is to exfiltrate sensitive data from the device (e.g., bank credentials, iCloud account, or intellectual property); fraud (*FR*), when a compromised device sends premium SMS, calls to premium numbers,

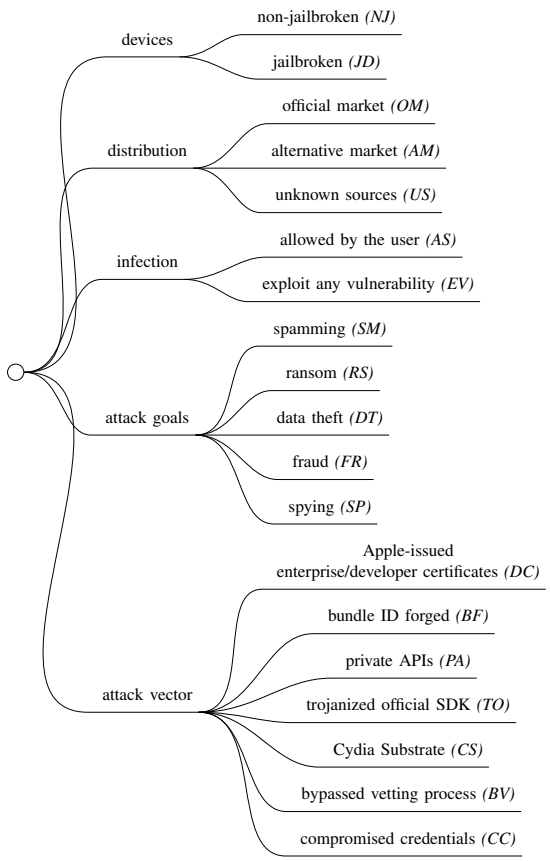

Figure 2. Features of iOS malware considered in this paper.

or even makes unsolicited in-app purchases; and spying (*SP*), when malware are specifically designed to spy on users and steal personal data such as pictures, contacts, or messages, among others.

Although we have not identified malware that hijack iOS devices, it is worth mentioning the Find my Phone exploit case (May 2014) [20]. Featured by Apple to seek missing iOS devices, cybercriminals abused it to block iOS devices remotely and demand money to unblock after hundreds of iCloud accounts were compromised. However, nowadays no malware are found designed to hijack iOS devices.

Attack Vector: Attack vector defines how the malware gain access to a device to perform their malicious outcome. We distinguished seven attack vectors: Apple-issued enterprise/developer certificates (*DC*), when malware misuse legitimate certificates then allowing the installation within non-jailbroken iOS devices out of official market distribution; bundle ID forged (*BF*) (termed as *masque attack* by FireEye, discovered in July, 2014), when malware forge the same bundle identifier as an existing, legitimate app thus replacing and posing it after installing on a compromised device. For instance, WireLurker uses this attack vector by means of a compromised host that connects via iTunes with a device, then allowing to access to shared keychain between the impersonated app and other apps; private APIs (*PA*), when malware abuse undocumented APIs of the iOS framework to provide access to device resources (such as camera, Bluetooth, or

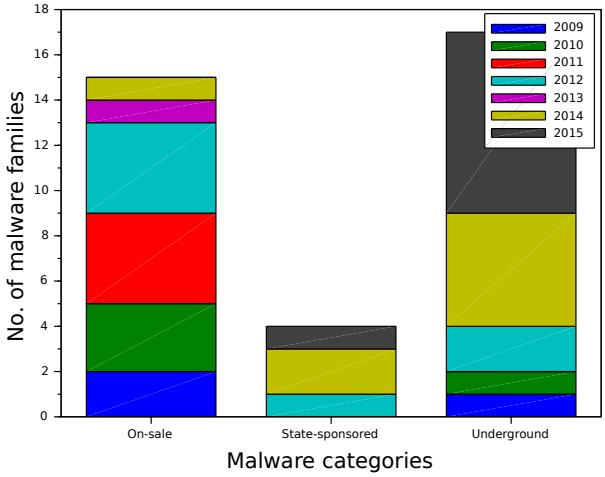

Figure 3. Evolution of malware families per malware category and year.

WiFi connectivity) or to sensitive device information (such as serial number, device ID); trojanized official SDK (*TO*), when malware are built with an unofficial Xcode that is in fact trojanized; Cydia Substrate (*CS*), when malware abuse this developer framework for hooking functions, for loading malicious third-party dynamic libraries, or for performing other tweaks with malicious purposes; bypassed vetting process (*BV*), when malware bypass the App Store vetting process and are deployed in the official market; and compromised credentials (*CC*), when malware use compromised iCloud or SSH service credentials to gain access to a device.

V. Evolution, Classification, and Discussion

In this section, we classified samples of 36 malware families in Table II according to the aforementioned features as well as discuss our findings.

First, we analyze distribution of malware families per category and year, as depicted in Figure 3. On-sale malware are almost equally distributed from 2009 to 2012. In 2013 and up to 2014, few families were discovered. State-sponsored malware emerged in 2012, 2014, and 2015, having few families almost equally distributed. On the contrary, underground malware show an interesting evolution. First families dated from 2009 and 2010. A few families were reported also in 2012. But from 2014, the number of malware family rapidly increased. In particular, the number of malware families related to underground malware in 2015 is greater than any other in all categories. In the sequel, we discuss our findings for each feature considered in this paper.

Devices: The devices affected per malware category are depicted in Figure 4. All malware families target to jailbroken devices, and 30.5% affect non-jailbroken devices. These results clearly indicate that jailbreaking increases the likelihood to be infected by malicious software. Interestingly, only 8.3% of malware exploit iOS vulnerabilities (5.5% in non-jailbroken devices). Thus, as usual, user awareness is highly recommended: to have a non-jailbroken iOS device and to be aware about

Malware family	Devices		Distribution			Infection		Attack goals					Attack vector						
	NJ	JD	OM	AM	US	AS	EV	SM	RS	DT	FR	SP	DC	BF	PA	TO	CS	BV	CC
ON-SALE MALWARE																			
Trapsms	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
MobileSpy	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
OwnSpy	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
MobiStealth	●	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	●
FlexiSpy	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
iKeyGuard	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
Copy9	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
StealthGenie	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
mSpy	●	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	●
iKeyMonitor	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
SpyKey	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
Copy10	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
InnovaSPY	–	●	–	●	–	●	–	–	–	–	–	●	–	–	–	–	●	–	–
1mole	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
Spy App	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
STATE-SPONSORED MALWARE																			
FinSpy Mobile	–	●	–	–	●	●	–	–	–	–	–	●	●	–	–	–	–	–	–
Hacking Team	●	●	–	–	●	●	●	–	–	●	–	●	●	●	–	–	●	–	–
Inception	–	●	–	–	●	●	–	–	–	●	–	●	–	–	–	–	●	–	–
XAgent	●	●	–	–	●	●	–	–	–	–	–	●	●	–	–	–	–	–	–
UNDERGROUND MALWARE																			
Ikee	–	●	–	–	●	–	●	–	–	●	–	–	–	–	–	–	–	–	●
LBTM	●	●	●	–	–	●	–	●	–	–	●	–	–	–	–	–	–	●	–
Find and Call	●	●	●	–	–	●	–	●	–	–	–	–	–	–	–	–	–	●	–
Nobitazzz	–	●	–	●	–	●	–	●	–	–	●	–	–	–	–	–	●	–	–
AdThief	–	●	–	●	–	●	–	●	–	–	●	–	–	–	–	–	●	–	–
SSLCreds	–	●	–	●	–	●	–	–	–	●	–	–	–	–	–	–	●	–	–
AppBuyer	–	●	–	●	–	●	–	–	–	●	●	–	–	–	–	–	●	–	–
WireLurker	●	●	–	●	–	●	●	–	–	●	–	●	●	●	–	–	●	–	–
Xsser mRAT	–	●	–	●	–	●	–	–	–	●	–	●	–	–	–	–	●	–	–
Lock Saver Free	–	●	–	●	–	●	–	●	–	–	●	–	–	–	–	–	●	–	–
KeyRaider	–	●	–	●	–	●	–	–	–	●	●	–	–	–	–	–	●	–	–
XcodeGhost	●	●	●	–	–	●	–	–	–	●	–	–	–	–	–	●	–	●	–
YiSpecter	●	●	–	●	–	●	–	●	●	–	–	●	–	●	–	–	–	–	
Muda/AdLord	–	●	–	●	–	●	–	●	–	–	–	–	–	–	–	–	●	–	–
Youmi Ad SDK	●	●	●	–	–	●	–	●	–	●	–	–	–	–	●	–	–	●	–
TinyV	–	●	–	●	–	●	–	–	–	●	–	–	–	–	●	–	–	–	–
SantaAPT	●	●	●	–	–	●	–	–	–	–	–	●	–	–	–	–	–	●	–

Table II
iOS MALWARE SAMPLES CLASSIFICATION.

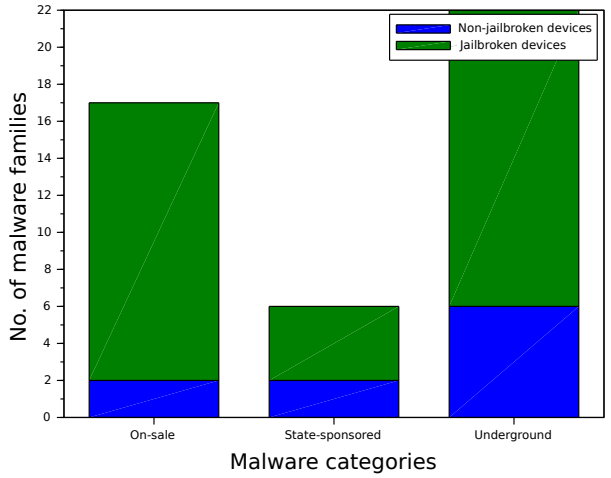

Figure 4. Affected devices per malware category.

apps to install or permissions granted are good defense mechanisms against threats in iOS landscape.

The number of infected devices per malware family is almost unknown. Few industrial security reports reckoned them. For instance, reports on AdThief estimated a total of 75000 devices infected. KeyRaider is known to have successfully stolen over 225000 valid iCloud accounts, as well as thousands of certificates, private keys, and purchasing receipts. As we commented previously, 39 apps were found in the App Store developed with XcodeGhost. Similarly, 256 apps infected with Youmi Ad SDK were found in the App Store, with an estimated number of downloads near to 1 million. SantaAPT, an underground malware family recently discovered, was reported to have infected almost 8000 mobile devices [21].

Distribution: On-sale and state-sponsored malware are distributed evenly over alternative markets and unknown sources, respectively, because the Apple review process rejects any app that transmits user's sensitive data without prior permission. On the contrary, underground malware vary their distribution channels. Namely, 13.9% of malware are distributed in AppStore, i.e., they bypassed the Apple vetting process (see Section III). The rest of underground malware, but Ikee, are distributed over alternative markets (such as Cydia repositories). Note that the presence in these markets is not odd, because no vetting process is followed to prevent malicious apps to appear. Lastly, Ikee has a worm behavior since scans the network seeking jailbroken devices with default SSH password.

Infection: Almost all malware families depend on

the user to perform their malicious activity. Surprisingly, only 8.3% of malware (one family of state-sponsored, two underground) exploit vulnerabilities in iOS devices. These vulnerabilities range from unchanged default password of jailbroken devices to compromised enterprise/ad-hoc certificates and *masque attack*, respectively.

Attack goals: On-sale and state-sponsored malware are focused on spying and data theft, as expected with their intended behaviors. In underground malware, attack goals are sparse: 52.9% perform some kind of data theft; 23.5% spam the compromised devices; 35.3% are fraudsters; and only 17.6% carry out spying activities. Note that malware families normally spread their attack goals; but 35.3% of them present solely a single goal.

Attack vector: Attack vectors are distributed almost evenly over Cydia Substrate in on-sale and state-sponsored malware. These results match with the affected devices, since Cydia is the main market place used by jailbroken devices. Two on-sale malware families use compromised credentials as attack vector to target non-jailbroken devices. State-sponsored malware use also other vectors, such as bundle ID forged (*masque attack*; related vulnerabilities are CVE-2015-3722 and CVE-2015-3725), or misuse of enterprise/developer certificates. The latter is mainly used to target non-jailbroken devices. Underground malware present a major diversity: 52.9% use also CS; 29.4% bypassed Apple review process and thus target non-jailbroken devices through official distribution channels; 11.8% use private APIs; 11.8% misuse enterprise/developer certificates; while compromised credentials, bundle ID forged, and trojanized official SDK are each one used by a single family (5.9%).

Based on our findings and regarding iOS devices, we strongly encourage to not jailbreak them, since they are prone to be successfully attacked. Let us also remark that malware may first jailbreak the device, then continue with process of infection. Thus, we recommend to keep iOS updated to the latest version, to install only applications from trusted sources, and to use native iOS mechanisms to grant or revoke application permissions individually.

Regarding malware families, new families and variants of those considered in this paper will appear before long that target non-jailbroken devices using undiscovered ways to bypass Apple vetting process, as well as new ways to escape from app sandbox and access to other application data.

VI. Case Study: Analyzing a Malware Sample

In this section, we first describe how reverse engineering for iOS app is followed over a set of selected samples, and then we show a detail analysis of a sample. In particular, the samples analyzed are summarized in Table III, grouped by malware family. Note that there exist missing families with respect to the ones we previously categorized. For those, we reviewed and collected security reports from anti-malware vendors since we were unable to retrieve specific samples of these families. Samples considered in this paper are freely available to download[1].

Malware Family	MD5 hash(es)
On-sale malware	
SpyKey	ba91eee0a3cc8c54c69162f37eb0f95a
State-sponsored malware	
Hacking Team	35c4f9f242aae60edbd1fe150bc952d5
Inception	4e037e1e945e9ad4772430272512831c
XAgent	823dcbd2fca465fabae71098bbb81e1e
Underground malware	
KeyRaider	8985ecbc80d257e02c1e30b0268d91e7
YiSpecter	0b98ee74843809493b0661c679a3c90c
	fbf92317ca8a7d5c243ab62624701050
	29e147675af38ece406b6227f3ccd76b
	97210a234417954c7bbe87bfe685eaae
	6e907716dc1aa6b9c490ce58aaae0d53
	62c6f0e3615b0771c0d189d3a7c50477
Muda/AdLord	8b76337397a00337d1cd7104a8b3cae4
TinyV	06036a5ce6927e75c774fc9669259105
	8187fb5f41be95d54931695fba465d7b
	724329f5be3cea4cf5ad51a1c8558638
	ccc9c5207b432cdb60e154a52c796ac1
SantaAPT	790035c9485d8061ae79587cf8d63a64
	a2ea4ebc168384e1d3b2879eeea21421
	5533e8936422649301 00b314014ccbb0
	adc8cd33f6c676797ac949bcd79a9d36
	e8d7eccfa480147bdf588f63accb9319
	884a988a6cdb7584b1d5128e54b53f60

Table III
Specific malware samples analyzed in this paper.

A. Methodology for iOS Malware Analysis

The methodology to analyze an iOS malware sample is shown as an activity diagram in Figure 5. It is comprised of three different stages.

In *Pre-analysis* stage, we aim at obtaining the deciphered code of the malware sample. Note that we need to have the unencrypted binary code to analyze the malicious payload. A malware sample available in the Apple Store can be downloaded as an IPA file, which is ciphered by Apple and thus, we need to decipher it. Otherwise, when the malicious sample is a dylib, package.deb, or an application distributed through Enterprise Provisioning, the binary file is not ciphered and thus we can pass over this stage. The otool tool (with option "-l") is used to verify whether the sample is encrypted, indicated by the cryptid value of the LC_ENCRYPTION_INFO command (a zero value indicates unencrypted). When the sample is encrypted, dumpdecrypted tool is used to obtain the unencrypted code.

Then, *Analysis* stage begins. In this stage, we perform a static and dynamic analysis in a similar way as in desktop malware analysis [22]. In static analysis, disassembler tools such as IDA, Hopper, or radare2 can be used. During dynamic analysis, numerous tools can be used to reveal the behavior of the sample, to trace execution flow, HTTP/HTTPS connections, and monitoring access to the file system. Examples of these tools are Reveal, snoop-it, or introspy, among others. Debugger tools as LLDB are also employed in this stage. Similarly, network sniffing and other network-related tools are used to analyze the communication performed by the sample.

[1]See http://webdiis.unizar.es/~ricardo/software-tools/supplementary-research-material/ios-malware-samples for details.

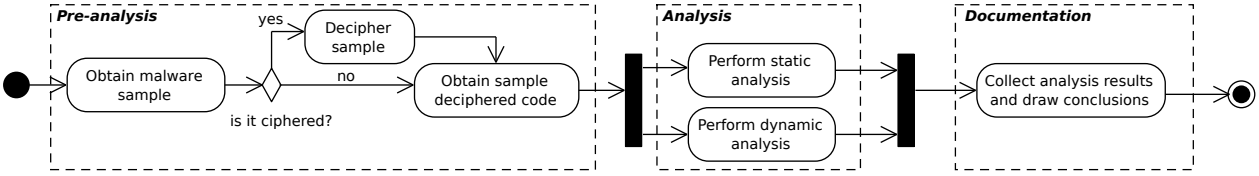

Figure 5. Methodology followed for analyzing an iOS malware sample.

```
000b2c10  db "", 0
000b2c11  db "iPhone5,1", 0
000b2c1b  db "\xE4\xB8\xAD\xE5\x9B\xBD\xE8\x81\x94\xE9\x80\x9A",
000b2c28  db "8.1.2", 0
000b2c2e  db "11A470a", 0
000b2c36  db "GET /data.php?table=other&game=(game) HTTP/1.1\r\n"
000b2c67  db "Host: www.wushidou.cn\r\n\r\n", 0
000b2c81  db "(game)", 0
000b2c88  db "iappstore", 0
000b2c92  db "www.wushidou.cn", 0
000b2ca2  db "name", 0
000b2ca7  db "pass", 0
000b2cac  db "pod", 0
```

Figure 6. Snippet of KeyRaider sample strings.

Finally, in *Documentation* stage all analysis results are collected and analyzed in detail to draw conclusions about the sample under analysis. The following issues shall be addressed: 1) how the sample infects the device; 2) what malicious activity is performed; 3) how it can be removed; and 4) mechanisms to prevent the infection.

B. *Example:* KeyRaider *Analysis*

Herein, we analyze the sample of KeyRaider malware family to illustrate the process followed (MD5 hash is shown in Table III). KeyRaider is a trojan malware presumably developed in China that spread by the Weiphone Cydia repositories targeting jailbroken devices.

We first verify whether the sample is encrypted by analyzing the output of otool. Specifically, it returns the following:

```
Load command 10
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
     cryptoff 16384
    cryptsize 835584
      cryptid 0
```

Thus, the malware sample is uncrypted. We proceed to perform a static analysis using Hopper. First, we review the printable strings of the sample and find interesting strings related to parts of a HTTP GET request, as well as the reference to a domain, www.wushidou.cn, as depicted in Figure 6. Thus, we suspect this domain is the command-and-control (C&C) server used by KeyRaider. According to http://whois.domaintools.com/ server, the domain is registered in China and surprisingly resolves to localhost address (127.0.0.1), which means it is currently unavailable for connecting.

Through Mobile Substrate Framework, this sample hooks *SSLRead* and *SSLWrite* functions in the itunesstored process, which is a system daemon that handles any app download or installation via the official Apple market, and also handles payments and purchases made from the own iOS device. The process of *SSLWrite* hooking is depicted in Figure 7(a). The App Store login information is sent to the App Store server through an SSL encrypted session. When called, the hooking function installed by KeyRaider looks for this login session, using specific patterns to find the Apple account's username, password, and device GUID in the data to transfer. Once retrieved, in the hooking function of *SSLRead* these credentials are encrypted using AES algorithm with an static key present in the binary code and then sent to the KeyRaider C&C server.

In addition, this sample presents binary code able to make forgeable purchases in the Apple Store with accounts retrieved from the own C&C server and to emulate the App Store login protocol. Figure 7 (b,c) shows these code snippets. Since the C&C domain is currently down, we do not perform dynamic analysis as interaction with C&C cannot be further analyzed. Furthermore, static analysis provided enough knowledge about its malicious behavior.

As conclusions, this sample infects jailbroken devices and use the Cydia Substrate to hook the system process in charge of interacting with the Apple official market. Once hooked, this sample accesses to the AppleID account of the compromised user and performs illegal app purchases.

VII. Conclusions and Future Work

Mobile malware are rapidly emerging as a serious threat. Although most of malware target Android platforms, malware targeting other platforms, such as iOS, are starting to appear. iOS incorporates a strong security architecture and a vetting process for deployment of new apps. However, these mechanisms are still ineffective since nowadays malware bypass them by different means.

In this work, we selected samples of 36 iOS malware families from 2009 to 2015 and classified them according to affected devices, distribution channels, infection, attack goals, and attack vector. We found that few of them target non-jailbroken devices or exploit iOS vulnerabilities, while data theft and spying are common goals. We also showed a methodology for iOS malware analysis and analyzed a real sample in detail. We expect to discover before long more malware families targeting non-jailbroken devices, as well as spreading their attack goals.

As future work, we aim at thoroughly analyzing these samples to better identify the underlying attack concepts and thus develop a framework for iOS malware detection.

Acknowledgments

We thank MLW.RE NPO for providing us with different iOS malware samples. The research of Ricardo J. Rodríguez was partially supported by the Spanish MINECO project CyCriSec (TIN2014-58457-R).

```
sub     sp, #0x60
movw    r0, #0x97e4                             ; @"\\n\\n\\n\\n\\n\\n\\n\\n==================
movt    r0, #0xc                                ; @"\\n\\n\\n\\n\\n\\n\\n\\n==================
add     r0, pc                                  ; @"\\n\\n\\n\\n\\n\\n\\n\\n==================
blx     imp___symbolstub1__NSLog
add     r4, sp, #0x20
mov     r0, r4
bl      __Z14getProcessNamev                    ; getProcessName()
movw    r1, #0xf754                             ; "itunesstored", :lower16:(0xb405e - 0x1490a)
mov     r0, r4
movt    r1, #0x9                                ; "itunesstored", :upper16:(0xb405e - 0x1490a)
movs    r2, #0x0
add     r1, pc                                  ; "itunesstored"
-------------------------------------------------------------------------------------------------
movw    r1, #0xf51d                             ; "/System/Library/Frameworks/Security.framewo
movs    r0, #0x4
movt    r1, #0x9                                ; "/System/Library/Frameworks/Security.framewo
str     r0, [sp, #0x78 + var_48]
-------------------------------------------------------------------------------------------------
movw    r1, #0xed9d                             ; :lower16:(0x1378d - 0x149f0)
movs    r0, #0x9
movt    r1, #0xffff                             ; :upper16:(0x1378d - 0x149f0)
movw    r2, #0xf2a6                             ; :lower16:(_o_SSLWrite - 0x149f2)
movt    r2, #0xc                                ; :upper16:(_o_SSLWrite - 0x149f2)
add     r1, pc                                  ; 0x1378d
add     r2, pc                                  ; _o_SSLWrite
str     r0, [sp, #0x78 + var_48]
ldr     r0, [sp, #0x78 + var_74]
blx     imp___symbolstub1__MSHookFunction
```

(a) `SSLWrite` hooking

```
blx     imp___symbolstub1___Unwind_SjLj_Register
movw    r2, #0xba71                             ; "POST /WebObjects/MZFinance.woa/wa/authenticate HTTP/
movt    r2, #0xa                                ; "POST /WebObjects/MZFinance.woa/wa/authenticate HTTP/
add     r2, pc                                  ; "POST /WebObjects/MZFinance.woa/wa/authenticate HTTP/
addw    r0, sp, #0x624
add.w   r1, sp, #0x6d8
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
movw    r2, #0xba8f                             ; "Host: p(pod)-buy.itunes.apple.com\\r\\n", :lower16:(
movs    r0, #0x2
movt    r2, #0xa                                ; "Host: p(pod)-buy.itunes.apple.com\\r\\n", :upper16:(
str.w   r0, [sp, #0x738 + var_48]
add     r2, pc                                  ; "Host: p(pod)-buy.itunes.apple.com\\r\\n"
add.w   r0, sp, #0x630
addw    r1, sp, #0x624
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
movw    r2, #0xba97                             ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
movs    r0, #0x3
movt    r2, #0xa                                ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
str.w   r0, [sp, #0x738 + var_48]
add     r2, pc                                  ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
addw    r0, sp, #0x63c
add.w   r1, sp, #0x630
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
```

(b) Emulation of App Store login

```
blx     imp___symbolstub1___Unwind_SjLj_Register
movw    r2, #0xa470                             ; "POST /WebObjects/MZBuy.woa/wa/buyProduct HTTP/1.1\\r
movt    r2, #0xa                                ; "POST /WebObjects/MZBuy.woa/wa/buyProduct HTTP/1.1\\r
add     r2, pc                                  ; "POST /WebObjects/MZBuy.woa/wa/buyProduct HTTP/1.1\\r
add.w   r0, sp, #0x488
add.w   r1, sp, #0x578
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
movw    r2, #0x9ef5                             ; "Host: p(pod)-buy.itunes.apple.com\\r\\n", :lower16:(
movs    r0, #0x2
movt    r2, #0xa                                ; "Host: p(pod)-buy.itunes.apple.com\\r\\n", :upper16:(
str.w   r0, [sp, #0x5d8 + var_48]
add     r2, pc                                  ; "Host: p(pod)-buy.itunes.apple.com\\r\\n"
addw    r0, sp, #0x494
add.w   r1, sp, #0x488
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
movw    r2, #0x9efd                             ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
movs    r0, #0x3
movt    r2, #0xa                                ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
str.w   r0, [sp, #0x5d8 + var_48]
add     r2, pc                                  ; "User-Agent: AppStore/2.0 iOS/(os) model/(phone) (4;
add.w   r0, sp, #0x4a0
addw    r1, sp, #0x494
blx     imp___symbolstub1___ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EERKS9_PKS
movw    r2, #0x9f1e                             ; "Accept: */*\\r\\n", :lower16:(0xb2e90 - 0x8f72)
movs    r0, #0x4
movt    r2, #0xa                                ; "Accept: */*\\r\\n", :upper16:(0xb2e90 - 0x8f72)
str.w   r0, [sp, #0x5d8 + var_48]
add     r2, pc                                  ; "Accept: */*\\r\\n"
addw    r0, sp, #0x4ac
add.w   r1, sp, #0x4a0
```

(c) Emulation of App Store purchases

Figure 7. Code snippets of interest of `KeyRaider` malware sample analyzed as case study.

REFERENCES

[1] A. M. Memon and A. Anwar, "Colluding Apps: Tomorrow's Mobile Malware Threat," *IEEE Security & Privacy*, vol. 13, no. 6, pp. 77–81, Nov 2015.

[2] D. Emm, M. Garnaeva, A. Ivanov, D. Makrushin, and R. Unuchek, "IT Threat Evolution in Q2 2015," Kaspersky Lab, Tech. Rep., Jul. 2015.

[3] McAfee, "McAfee Labs Threats Report," Tech. Rep., Aug. 2015.

[4] FireEye, "Out of Pocket: A Comprehensive Mobile Threat Assessment of 7 Million iOS and Android Apps," Tech. Rep., Feb. 2015. [Online]. Available: https://www2.fireeye.com/rs/fireye/images/rpt-mobile-threat-assessment.pdf

[5] Blue Coat, "Blue Coat Systems 2015 Mobile Malware Report," Tech. Rep., 2015.

[6] IDC, "Smartphone OS Market Share, 2015 Q2," [Online], Aug. 2015, http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[7] A. Amamra, C. Talhi, and J. Robert, "Smartphone Malware Detection: From a Survey Towards Taxonomy," in *Procs. of the 7th International Conference on Malicious and Unwanted Software (MALWARE)*, 2012, pp. 79–86.

[8] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. van der Veen, and C. Platzer, "ANDRUBIS - 1,000,000 Apps Later: A View on Current Android Malware Behaviors," in *Procs. of the 3rd Int. Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, 2014.

[9] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A Survey of Mobile Malware in the Wild," in *Procs. of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM, 2011, pp. 3–14.

[10] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," in *Procs. of the IEEE Symposium on Security and Privacy*, 2012, pp. 95–109.

[11] G. Suarez-Tangil, J. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, Detection and Analysis of Malware for Smart Devices," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 961–987, 2014.

[12] C. Miller, "Mobile Attacks and Defense," *IEEE Security & Privacy*, vol. 9, no. 4, pp. 68–70, July 2011.

[13] S. Mansfield-Devine, "Android malware and mitigations," *Network Security*, vol. 2012, no. 11, pp. 12–20, 2012.

[14] T. Wang, K. Lu, L. Lu, S. Chung, and W. Lee, "Jekyll on iOS: When Benign Apps Become Evil," in *Procs. USENIX SEC'13*, ser. SEC'13. Berkeley, CA, USA: USENIX Association, 2013, pp. 559–572.

[15] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "PiOS: Detecting Privacy Leaks in iOS Applications," in *Procs. of NDSS 2011*. The Internet Society, 2011.

[16] M. Bucicoiu, L. Davi, R. Deaconescu, and A.-R. Sadeghi, "XiOS: Extended Application Sandboxing on iOS," in *Procs. of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, 2015, pp. 43–54.

[17] Z. Deng, B. Saltaformaggio, X. Zhang, and D. Xu, "iRiS: Vetting Private API Abuse in iOS Applications," in *Procs. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 44–56.

[18] L. Xing, X. Bai, T. Li, X. Wang, K. Chen, X. Liao, S.-M. Hu, and X. Han, "Cracking App Isolation on Apple: Unauthorized Cross-App Resource Access on MAC OS X and iOS," in *Procs. of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015, pp. 31–43.

[19] The iPhone Wiki, "Malware for iOS," [Online], https://www.theiphonewiki.com/wiki/Malware_for_iOS.

[20] T. Warren, "Apple's Find my iPhone feature exploited to hold devices hostage," [Online], May 2014, http://www.theverge.com/2014/5/27/5753726/find-my-iphone-hack-australia-ransom.

[21] CloudSek Info Security Pvt. Ltd., "APT Malware Masquerade as Santa Claus and Christmas Apps," [Online], Dec. 2015, https://www.cloudsek.com/announcements/blog/apt-malware-masquerade-as-christmas-apps-and-santa-claus/.

[22] K. Liu, H. B. K. Tan, and X. Chen, "Binary Code Analysis," *Computer*, vol. 46, no. 8, pp. 60–68, 2013.