iSAM: An iPhone Stealth Airborne Malware

Dimitrios Damopoulos, Georgios Kambourakis, and Stefanos Gritzalis

Info-Sec-Lab Laboratory of Information and Communications Systems Security,
University of the Aegean, Samos, Greece
{ddamop,gkamb,sgritz}@aegean.gr
http://www.icsd.aegean.gr/info-sec-lab

Abstract. Modern and powerful mobile devices comprise an attractive target for any potential intruder or malicious code. The usual goal of an attack is to acquire users' sensitive data or compromise the device so as to use it as a stepping stone (or bot) to unleash a number of attacks to other targets. In this paper, we focus on the popular iPhone device. We create a new stealth and airborne malware namely iSAM able to wirelessly infect and self-propagate to iPhone devices. iSAM incorporates six different malware mechanisms, and is able to connect back to the iSAM bot master server to update its programming logic or to obey commands and unleash a synchronized attack. Our analysis unveils the internal mechanics of iSAM and discusses the way all iSAM components contribute towards achieving its goals. Although iSAM has been specifically designed for iPhone it can be easily modified to attack any iOS-based device.

Keywords: Malware, iPhone, iOS, Jailbreak, Stealth, Airborne, Rootkit.

1 Introduction

Mobile devices have evolved and experienced an immense popularity over the last few years. These devices have penetrated the market due to the variety of data services they offer, such as texting, emailing, browsing the Internet, documents editing, listening to music, watching videos and playing games in addition to the traditional voice services. As a result, analysts are expecting a mobile device population of 5 billion by 2015 [1]. Moreover, these devices are capable of performing sophisticated tasks and communicating through various wireless interfaces. As mobile devices hardware functionality and performance get improved, Operating Systems (OS) have similarly evolved. Modern mobile devices run sophisticated OS like Google Android, Apple iOS, Symbian, Palm OS, Blackberry RIM, Windows Mobile 7, that need to confront almost the same risks as desktop computers. It is thus apparent that this growth has exposed mobile devices to an increasing number of security threats. According to Chow and Jones [2], the only difference between desktop computers and mobile devices in terms of security risk is the challenge to understand the inner workings of the OS on different hardware processor architectures.

J. Camenisch et al. (Eds.): SEC 2011, IFIP AICT 354, pp. 17–28, 2011.

Very recently, Kaspersky Lab identified 39 new mobile malware families (SMS trojans, iPhone malware, Android spyware) with 143 variants [3] which try to compromise mobile device security. Also, according to a ScanSafe report, malware volume grew 300% in 2008, and it is noted that several of the legitimate web pages crawling on the Internet maybe infected by different kind of viruses [4]. In the same report it is stated that malicious image files comprised 10% of all Web malware encountered in 2009.

In this paper, we focus on iPhone device security. We create a smart malware namely iSAM to expose possible vulnerabilities of modern mobile devices and OS, and demonstrate that is relative easy to bypass any security control. Towards achieving its goals, iSAM employs a variety of programming techniques (pubic and private frameworks, override OS functions), backgrounding methods (daemons, dynamic libraries), as well as open source iPhone malware resources (e.g. Star exploit, iKee scanner logic). The aim of iSAM is to stealthily execute, six malware mechanisms, self-propagate wirelessly to other iPhone targets and finally connect back to the iSAM bot master server to update its programming logic or to obey commands and unleash a synchronized attack. Although specifically designed for iPhone 2G and iPhone 3G with the 3.1 and 4.0.1 iOS version respectively, iSAM can be easily adapted to attack other Apple iOS devices (iPhone 3GS/4 and all generations of iPod Touch). To the best of our knowledge this is the first rootkit-similar, airborne and stealth multifarious malware that is capable of infecting iPhone devices.

The rest of the paper is structured as follows. The next section presents previous work on the topic. Section 3 provides basic mobile malware design requirements and attributes. Section 4 describes the iSAM overall architecture and presents an analysis of the six proof-of-concept malicious iSAM's subroutines. The last session concludes the paper and gives pointers to future work.

2 Preliminaries and Related Work

Soon after the first iPhone was released, hardware and software modules were developed to bypass root privileges and overcome any restrictions. That is, only software signed by Apple's Certificate Authority is allowed to run on iPhone. This process is generally referred to as *"Jailbreak"*. Upon jailbreaking, the entire iPhone file system becomes open for use. Jailbreaking allows to create and execute third-party software without an official SDK from Apple. The first aim after jailbreaking was to bypass SIM-Lock. Specifically, every iPhone is locked to a particular network provider. Unlocking allows the user to place calls with any GSM/3G carrier by inserting a different SIM into the device.

The *installer* created by the development team RipDev, and *Cydia* created by J. Freeman were the first two package managers that allowed a user to browse and download third-party applications for jailbroken iPhones. The open-source Cydia became very popular after iPhone firmware version 2.0. Since then, every time a hacking team discovers a new iPhone exploit, they publish the corresponding software that jailbreaks the device. Also, the same software installs a version of Cydia, a SSH server, and enables the default root login password *"alpine"*.

In July 2007, T. Ormandy discovered *"libtiff"*, a buffer overflow method that has already been used to attack Sony's PSP device. Hackers inspected Apple's Mobile Safari web browser in order to test and take advantage of the same vulnerability that lay in the Tag Image File Format (TIFF) library, which is used for viewing TIFFs. Finally, they managed to successfully attack iPhone. Capitalizing on this vulnerability they created the web site jailbreakme.com. There, by selecting the "Slide-to-Unlock" button, a malicious TIFF file was simply opened from Mobile Safari leading to injection and execution of an arbitrary code and a straightforward Jailbreak. Once the iPhone has been jailbroken, the exploit patched the libtiff vulnerability in order to avoid future attacks. Apple patched this vulnerability with iOS 1.1.2 firmware. Vaibhav in his Project Report [6], discuses and analyzes the libtiff security breach in detail. Moreover, Chavez in [7] discusses how an intruder can successfully attack a network using a jailbroken iPhone. To perform the attack, she installs and uses a collection of powerful tools (e.g. Metasploit, Nmap, Whois, tcpdump, a terminal, WifiStumbler).

A year later, Apple introduce the new iPhone 3G that incorporates firmware version 2.0. Also, it offered a powerful Software Development Kit (SDK) that gave the opportunity to developers to create and deploy software under certain public frameworks so as to create the AppStore. In the end of July 2008, one of the iPhone third-party games namely *Aurora Feint* was removed from AppStore due to privacy concerns. Actually, the game was uploading to the developers server all contacts stored in the host iPhone. In 2009, serious privacy concerns appeared within the AppStore applications. MogoRoad and Storm8 are only two of the AppStore applications that have been removed after users' complaints about privacy concerns. In July 2009, users have raised serious concerns about their privacy in regard of the behavior of four tracking providers namely Pinch Media, Flurry, Medialets and Mobclix. J. Freeman tried to protect iPhone users by creating PrivaCy, an application for jailbroken iPhones, which blocks AppStore applications from tracking usage information.

The authors in [8] presented a vulnerability in SMS messages, which enables an attacker to inject fuzzed SMS messages into iPhones, Android and Windows Mobile devices. This vulnerability leads to a Denial-of-Service (DoS) attack remaining at the same time invisible to the service provider. This weakness was patched with the new 3.0.1 iOS firmware.

In 2009, researchers were trying to gain access to private information (i.e. contacts, photos, mails, SMS messages, passwords) stored in iPhone devices using various forensics methodologies. J. Zdziarski was the first one who using proper tools was able to retrieve unencrypted the full iPhone disk image. The same year he published a white paper with forensics techniques and tools that could be used to retrieve information from an iPhone device. During the same period, the first iPhone worm namely *Ikee* was released and a wave of worm attacks started. Ikee was simply changing the iPhone's wallpaper. Note that, Ikee was a self-propagating worm attacking only jailbroken iphones using the installed SSH server and the default root password. The same vulnerability was also used by *Dutch 5€ ransom*, a worm that locked the iPhone screen asking 5€ on a PayPal account in order to remove

the worm. *Privacy.A*, was another worm running in stealth mode and be able to steal personal data from the iPhone. In November 2009, a new highly disastrous version of Ikee, namely *iKee.B* appeared in several Europe countries. SRI International analysed iKee.B in [9] and provided technical details about the logic and the internal mechanics of the first iPhone Botnet. Although iKee.B acts similar to Ikee, it includes a Command & Control (C&C) logic to control all infected iPhones via a Lithuanian botnet server. Moreover, it is able to periodically update its malware behaviour. Finally, iKee.B, changes the default SSH password into "ohshit", and collects and sends all SMS messages stored in the device to the bot server. The iKee.B source code is published in [10].

Recently in [11] Seriot, presented some interesting attack scenarios on how a malicious application can use official and public frameworks, provided by Apple, to collect users private information (e.g., phone number, email account setting, keyboard cache entries, Mobile Safari searches and the most recent GPS location) programmatically. This happens without the user's knowledge and without being rejected by the AppStore review.

On July 2010, the United States government and the new Digital Millenium Copyright Act (DMCA) legislation announced that modifications of smartphones, like jailbreak or Unlock are legal as long as they obey the copyright law [12]. Based on the new law, in August 2010, Comex, an iPhone exploit developer, with the help of several other hackers introduced the exploit namely Star or JailbreakMe 2.0. This new exploit can jailbreak all Apple's products which incorporate iOS firmware versions from 3.1.2 to the current 4.0.1. Until then, all previous iOS firmwares have been jailbroken using offline exploits. Star, like JailbreakMe, is a remote browser-based jailbreak that uses two security flaws [13]. The first one, uses a corrupted font embedded in PDF files that crash the Compact Font Format (CFF) to allow arbitrary code execution, while the second one uses a vulnerability in kernel to escalate the code execution to unsandboxed root privileges. Any iOS mobile device that opens a jailbroken PDF file from a website, email, SMS, or Apple's iBook can be automatically jailbroken. A few days after Star was released, Comex published the source code [14].

3 Designing Principles and Requirements for iPhone

The primary aims of a smart malware is to infect the target, self-propagate to other targets and finally connect back to a bot master server. The latter action is highly desirable to update the malwares programming logic by improving already existed features and adding new ones, or to obey commands and unleash a synchronized attack. To achieve the affomentioned goals, the malware needs to fulfill some basic design requirements. First off, it needs to infect the device and gain root permissions. Also, it needs to run continuously in the background of the OS and has smart malware behaviour remaining stealthy to the legitimate user.

The only way to infect an iPhone and gain root permissions is by exploiting a vulnerability on an iOS jailbroken device. In case the target iPhone is already

jailbroken, the malware may attempt to use the SSH vulnerability[1] to wirelessly connect and infect the device. According to Cydia developer, J. Freeman, over 10% of the 50 million iPhones worldwide are jailbroken [15]. Although these devices constitute a large proportion for possible targets, it is necessary to find new ways to infect non-jailbroken iPhones.

To do so, we propose to create a malicious version of Star exploit [14] that is able to work wirelessly. As already mentioned, Star exploit consists of a PDF, which uses two security flaws allowing arbitrary code execution and gaining root privileges, and of a website *"JailbreakMe"* which stores the PDFs caring the exploits (one PDF for each iPhone version and one for each iOS version) [13]. Once the PDF is opened, a dynamic library (dylib) named "installui.dylib" provides graphic interface and downloads from the corresponding website a file named "wad.bin". After that it proceeds to jailbreak the iOS and install Cydia using a second dylib named "install.dylib". The file "wad.bin" is a binary file that contains any type of data; in this case it contains the "install.dylib" and the Cydia package. According to F-secure, any iOS mobile device that opens an exploited PDF file from a website, an email, an Apple's iBook application or accesses a website directly from an SMS message, can be jailbroken [16]. Note that iOS is capable of recognizing automatically hyperlinks sent via SMS.

Once a malicious Star PDF file is opened by an iPhone using our malicious Star version, it is being automatically jailbroaken and installed stealthily malicious software. Also, once an iPhone visits our website or opens the malicious PDF, the exploit procedure begins, stealthily, without providing any graphical interface or any information popups. Furthermore, we inject our malware into the "wad.bin". This means that once the jailbreaking procedure ends, Cydia and our malware will be both installed in the iPhone.

In order to create our malicious version of Star, it was necessary to modify the open source version of Star exploit [14]. Firstly, we decided to pack our malware as a Debian package. Once Cydia is installed in the iPhone, any file with the ".deb" extension stored in the folder "/var/root/Media/Cydia/AutoInstall", will be also automatically installed in the device. To inject our malicious package in the "wad.bin" file, it was necessary to modify the Star source class, named "install" and the python script "wad.py". Also, it was necessary to modify the source file "installui.m" which is used to build the dylib named "installui.dylib". In the source file "installui.m" we deactivated all displayed graphics interfaces making the exploit behave stealthily. Moreover, we edited the domain name from where our malicious "wad.bin" can be downloaded and we recalculated the size of our malicious "wad.bin" file editing the source where it was necessary. Last, after the installation of Cydia we shift our malware package into Cydia's auto-install directory. It is stressed that all these operations are possible because the Safari browsing process has acquired root access using the kernel bug.

The second requirement when designing our malware was the ability to run continuously in the background of the underlying OS. Until iOS version 4,

[1] The SSH vulnerability, allows intruders to remotely access a jailbroken device's file system using the SSH server and the default password "alpine".

multitasking was not officially supported. Unofficially, jailbroken iOS could support applications that run in the background as deamons or use Objective-C dylib. iOS being a Unix-based OS, can provide multitasking using *launchd*, a launch system that supports daemons and per-user agents as background-services. Once an iOS has been jailbroken, any installed application or shell script is able to behave as daemon by creating a launch plist and placing it into the "/Library/LaunchDaemons" iOS directory. Another way to support multitasking is with dylib. When an application is launched, the iOS kernel loads the application's code and data into the address space of a new process. At the same time, the kernel loads the dynamic loader i.e., "/System/MobileSubstrate/DynamicLibraries" into the process and passes control to it. In addition, it is possible to load a dylib at any time through Objective-C functions. Finally, from iOS version 4 and later, Apple provided seven APIs that allow applications to run in the background. Although these APIs are the native way for providing multitasking, it is not the best way to create and launch a malware. A program that uses the native way for backgrounding can be easily spoted by the user from the corresponding menu.

The last requirement is to design a smart malware that will remain stealthy and invisible to the user at all time. These smart malwares need to achieve their purpose stealthily by modifying OS code, functions and/or data. Officially, Apple does not provide any frameworks that override iOS functions. To fill the gap, J. Freeman has created and incorporated into Cydia MobileSubstrate extension, a framework that allows developers to deliver run-time patches to system functions using Objective-C dynamic libraries [17]. By creating a dylib, developers are able to build applications that run in the background and be able to replace internal system functions at the same time.

4 The iSAM Malware

Given the aforementioned requirements and possible solutions, we created iSAM. The iSAM malware has been implemented, using Objective-C source code compiled for iPhone ARM CPU. Also, iSAM was build using the unofficial ways (see Section 3) for backgrounding (daemons and dylibs), the public and private[2] frameworks and the MobileSubstrate framework with the *"substrate.h"* header that overrides iOS functions. This means that certain modules of iSAM can be classified as rookit.

iSAM consists of a main daemon written in Objective-C and combined with a proper launch plist (activated at device boot time) and six subroutines written as Objective-C functions, dylibs or shell scripts. The iSAM main daemon is responsible to manage all subroutines which are in charge of the propagation logic (iSAMScanner), the botnet control logic (iSAMUpdate) and the smart malware behaviour (iCollector, iSMSBomber, iDoSApp, iDosNet). iSAMScanner is activated during the device boot time and runs as a deamon in the background.

[2] Unsupported frameworks, which were retrieved directly from a jailbroken iPhone and have been dumped to get the headers.

iSAMUpdate is activated once per day and only if an Internet connection is available, while the rest four subroutines are activated once per week but at random times. Figure 1 depicts the overall iSAM architecture. Important pseudocode segments of all the iSAM subroutines discussed in this section can be found in [22].

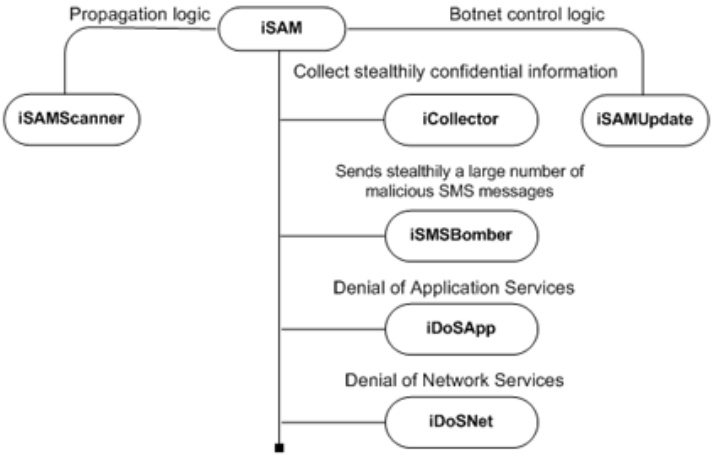

Fig. 1. iSAM architecture

In addition to iSAM, we setup a bot master server namely iSAM Server (iSAMS) having multiple functionality. iSAMS incorporates two basic modules: (a) a repository server where the newer or special customized version of iSAM is stored, and (b) a multithread socket server used to communicate with the infected devices to update iSAM program logic, to collect sensitive information and to control and execute commands directly on the iPhones. Also, iSAMS stores our malicious version of Star exploit namely *mStar*.

4.1 iSAM Infection Methods

As already mentioned iSAM uses two different methods to wirelessly attack and infect iPhone devices. The first method is by using iSAMScanner (see next section) which tries to detect jailbroken iPhones having the SSH vulnerability and infects them directly. Alternatively, we employ mStar, a modified version of the exploit Star, which is able to jailbreak the device and simultaneously infect it with iSAM. A recent report by F-Secure showed that nearly 79.8% of mobile phones infections were as a result of content downloaded from malicious websites or delivered by Bluetooth and SMS messages [18]. Capitalising on these results we use iSMSBomber (see section 4.5) as part of the second infection method to contaminate iPhone devices. iSMSBomber is able to read any telephone numbers stored in the device and send to them stealthly a SMS message with the domain of iSAMS. This is to trick the user into visiting iSAMS. In addition, mStar

can be delivered to an iPhone when visiting our iSAMS via a web link, email attachment or a legal popular AppStore application that uses a website link to redirect to iSAMS. Once 195.251.166.50 (iSAM.samos.icsd.gr) hyperlink is opened via a SMS message, mStar PDF is downloaded from iSAMS and loaded via Mobile Safari. After that, installui.dylib downloads wad.bin and install.dylib jailbreaks the iPhone and installs iSAM.

4.2 iSAMScanner: Scan, Connect, Infect

iSAMScanner is responsible for the propagation logic of iSAM. iSAMScanner driven by iSAM daemon, is activated at iPhone boot time. The iSAMScanner subroutine has three methods: *iScan, iConnect* and *Infector*. iScan is conducting three independent network scans just like iKee.B. Firstly, it scans iPhone's local WiFi network address space, then scans in a random way computer subnetworks on the Internet and finally scans a list of IP address range that belongs to a set of mobile phone companies in Greece (e.g. 195.167.65.0-195.167.65.255, GR, Cosmote) or in other European countries (e.g. 139.7.0.0-139.7.255.255, DE, Vodafone). When a vulnerable iPhone is detected, iConnect connects directly to the SSH Server using the default root password and by using Infector downloads the iSAM.deb package to the directory "/private/var/root/" of the target-device. Finally, Infector installs the package using the command dpkg -i –refuse-downgrade –skip-same-version iSAM.deb. From this step forward, the victim's device is under the control of iSAM.

4.3 iSAMUpdate: Update, Command, Control

iSAMUpdate, is responsible for the botnet control logic of iSAM. It is also used for connecting iSAM back to iSAMS to check whether a newer iSAM version is available. This allows iSAM to be updated e.g., with a new programming logic or follow commands directly from the server in order to unleash an attack. iSAMUpdate is connected back to iSAMS once every day as soon as an Internet connection is detected. Every time iSAMUpdate is activated, it retrieves some useful information from the device and sends them as a textmessage to iSAMS to be stored in the local database. The message is consisted of the iSAM version, the Unique Device Identifier (UDID), which is a unique serial number for each iPhone, the IP address from the e0 interface (WiFi connection on the iPhone) and the GPS coordinates, as long as a GPS is enabled. The following quintuplet gives an example of such a message {version016 ||3bdf7jc607h1j7te441sc02f5h5j6229db66hh63||62.217.70.167||26.700039||37.794186}. In case a newer iSAM version is detected, the server answers back with the name of this version, else it sends back a null message. It is not necessary for the server to respond with the latest version; instead it can answer with a customized response based on the UDID or the georgraphical coordinates if it wants to manipulate the phone in a special way or to attack devices selectively (e.g. attack all devices that roam to a certain area). Once the iSAM client receives the name of the version, it executes a Unix shell script named *"iUpdate.sh"* which is called

with the name of the version as a parameter. The shell script executes two script commands: the "curl -O iSam.samos.icsd.gr/debs/$1.deb", which downloads the newer iSam version directly from iSAMS and the "dpkg –i –refuse –downgrade –skip –same –version $1.deb", that uses the Debian package manager to install the new version. We should note that the name of the iSAM version, which the server has sent, is stored in the variable $1. It is also stressed that once the server has the client's IP address, is able to connect directly to the client's SSH service using the default root password.

An infected iPhone with iSAM is able to search for jailbroken iPhones into three different subnetworks (local subnet, random Internet subnet, mobile provider IP subnet) in order to infected them as well. Moreover, an infected iPhone can be updated or controlled by iSAMS. Lastly, if a non-jaibroken iPhone opens iSAM.samos.icsd.gr hyperlink through a SMS message, will get infected by mStar PDF.

4.4 iCollector: Gathers Private Information from the Device

The purpose of this attack module is to collect stealthily confidential information directly from the device. iPhone stores all user's data in SQLite databases and plist files without providing any encryption mechanism to secure their contents. Once an iPhone has been jailbroken, the iOS sandbox collapses and all databases and plists stored in the path "/var/mobile/Library/" are exposed to the attacker.

iCollector is an iSAM subroutine that collects stealthily sensitive information from iPhone's databases (call, sms, calendar, note) and from Safari's plist files (bookmarks and Web browsing history), storing them into a new database named iCollection.db. After the data collection takes place (see line #1-3 in [22]) and when an Internet connection is detected (line #3-6), iCollector is connected back to the iSAMS using the Client/Server model and TCP sockets in order to send the collected information (line #7-8). iCollector is a dylib written in Objective-C and uses an SQLite library to read, create and write to databases.

4.5 iSMSBomber: Sends Malicious SMS Messages in Stealth Mode

Like all GSM mobile devices, iPhone uses a set of commands, called AT (attention), to dial a number or exchange SMS messages. In addition to AT commands, iPhone employs a high level private framework, named "CoreTelephone" (incorporated to iOS), in order to communicate with the Baseband using Objective-C functions. However, this framework is neither available by the iOS SDK nor documented. The only way to overcome this issue is to retrieve the CoreTelephone framework directly from the files of a jailbroken iPhone and then use class-dump utility. Class-dump examines Objective-C runtime information stored in Mach-O files in order to generate the header files [19]. This procedure is necessary to execute every time a private framework is used. Once the CoreTelephone framework and the header files are available, a direct communication with the Baseband can be placed.

To take advantage of such a powerful framework, we create iSMSBomber. This is an iSAM dylib subroutine that sends silently say 1000 malicious SMS messages

using the private CoreTelephone framework and more specifically the CTMessageCenter header (line #1-3). Firstly, iSMSBomber makes an SQL query to the iPhone's address book database to retrieve telephone numbers from user contacts (line #4-5). In case no contact exists or the contacts are less than 1000, then random numbers are created to reach 1000. Every random number begins with the standard "003069" digit sequence, which represent a mobile phone number in Greece. Then iSMSBomber creates the following message: *"Hello, how are you? I have found an interesting website: 195.251.166.50 - Please send it to all'!'* and by using the sharedMessageCenter function, it sends the message to all the existing (plus random numbers if any) (line #6). Once an iPhone user receives this message and visits the website link, Mobile Safari web browser opens automatically and accesses the site. Recall that this domain is redirected to iSAMS that stores the mSTAR exploit, which in turn contains iSAM. Also note that this message is malicious only for iPhones iOS. Normally, once a SMS message is sent or received, automatically it is stored to the SMS database and a tone rings. iSMSBomber sends stealthily all the 1000 messages without storing them in the SMS database and without playing any tone. The only way to expose its presence is by the end of the month, when the mobile user receives his telephone bill assuming that the user does not usually send a high amount of messages.

4.6 iDoSApp: Denial of Application Services

Modern mobile devices are designed to increase the efficiency and the productivity of mobile users on the go. Therefore, by default, all mobile devices come bundled with some basic pre-installed applications or utilities. iPhone is offered with seventeen pre-installed applications. Additionally, AppStore contains more than $3 * 10^5$ iOS applications [20] offering the user the necessary on the go productivity. One of the main iOS applications is *SpringBoard* that manages the iOS home screen by displaying all icons of the available applications, starts the WindowServer and launches and bootstraps other applications [21]. For example, once a user touches the icon of an application, SpringBoard launches it. The goal of iDoSApp subroutine is to cause DoS in application launching by overriding some system functions required by SpringBoard.

In this context, iDoSApp is a dylib, which is activated at random time frames, is short-term (say 1-minute) and causes real DoS by non-loading an application. To achieve this, it is necessary to replace SpringBoard system functions, by class-dump SpringBoard in order to get the private headers and create a dylib. The headers used by iDoSApp are the *substrate.h* (used for overriding systems functions using the MobileSubstrate framework), the *SpringBoard.h* and the *SBApplicationIcon.h* headers (derived from the class-dump of SpringBoard (line #1-3)). SBApplicationIcon is a system function responsible for the behavior of all icons displayed by the SpringBoard. iDoSApp hooks, modifies and replaces SBApplicationIcon only when the selector is a launch message. A selector in Objective-C language is a message that can be sent to an object or class (4). Normally, every time the user touches on an application icon, a launch message is sent to SBApplicationIcon to load the application. In our case, once the iDoSApp

is activated, it blocks all launch messages that are sent to SBApplicationIcon causing DoS (line #5-7). iDoSApp will not compromise iSAM existence, as some applications can automatically close when an application is written for older or newer iOS versions or when they fail to manage the memory correctly.

4.7 iDoSNet: Denial of Network Services

The aim of iDoSNet subroutine is to cause DoS by deactivating for - say 30 seconds - all communication services (line #3-6). iSAM will activate iDoSNet at random times during a random day of the week. iDoSNet is using a private framework, namely Preferences.framework which can enable/disable the Airplane mode that controls 3G/GSM functions. Furthermore, iDoSNet uses the Apple80211. framework, a private framework that configures all 802.11 network interfaces, to cause DoS (line #1-2). We make the hypothesis that the duration of 30 seconds will not expose the existence of iSAM and the vast majority of users will suppose that it happend due to a temporary interruption to the wireless signal.

5 Conclusion

The evolution of malwares is a continuous race between intruders and defenders. Both use the same programming methods, tools and resources either to create a smart malware or to develop an intelligent malware detection mechanism. Overall, with the increasing risk of mobile malware, designing a highly secure mobile device is still a very challenging task. This paper concentrates on the very popular iPhone device. We design and implement iSAM a new multifunctional malware that is able to wirelessly infect and self-propagate to iPhone devices. iSAM is able to override OS functions and uses a variety of advanced programming methods (public and private frameworks), backgrounding methods (daemons, dynamic libraries), and open source iPhone malware resources (e.g. Star exploit, iKee scanner logic) towards achieving its goals. It is also able to hide its presence, and update its logic via the iSAM bot master server. iSAM incorporates six different malware mechanisms and utilises two different methods to wirelessly infect other devices. The purpose of our study is to highlight iOS weaknesses and offer in-depth information towards combating such threats.

Our future work will concentrate on obtaining detailed experimental results e.g. infection and untraceability rates, collector effectiveness etc as well as into modifying iSAM core so as to be able to automatically infect any iOS-based device.

References

1. Liu, L., Yan, G., Zhang, X., Chen, S.: VirusMeter: Preventing your cellphone from spies. In: Balzarotti, D. (ed.) RAID 2009. LNCS, vol. 5758, pp. 244–264. Springer, Heidelberg (2009)
2. Chow, G.W., Jones, A.: A framework for anomaly detection in OKL4-Linux based smartphones. In: Proceedings of the 6th Australian Information Security Management Conference (2008)

3. Kaspersky lab at mobile world congress 2009 in Barcelona, http://www.securelist.com/en/analysis/204792100/Kaspersky_Security_Bulletin_2009_Malware_Evolution_2009
4. Landesman, M.: The world's largest security analysis of real-world web traffic: annual global threat report, ScanSafe STAT, http://www.scansafe.com/downloads/gtr/2009_AGTR.pdf
5. Apple introduction to security overview, http://developer.apple.com/library/ios/♯documentation/Security/Conceptual/Security_Overview/Introduction/Introduction.html
6. Pandya, V.R.: iPhone security analysis. Project Report, Department of Computer Science, San Jose State University (2008)
7. Chavez, A.: A jailbroken iPhone can be a very powerfull weapon in the hands of an attacker. Project Report, Purdue University, Calumet's CIT Department (2008)
8. Miller, C., Mulliner, C.: Fuzzing the Phone in your Phone. In: BlackHat, USA (2009)
9. An analysis of the Ikee.B (Duh) iPhone botnet, http://mtc.sri.com/iPhone
10. iKee, http://vx.netlux.org/src_view.php?file=ikee.zip
11. Seriot, N.: iPhone Privacy. In: Black Hat, USA (2010)
12. Copyright, http://www.copyright.gov/1201
13. Technical analysis on iPhone jailbreaking, http://community.websense.com/blogs/securitylabs/archive/2010/08/06/technical-analysis-on-iphone-jailbreaking.aspx
14. Comex/Star, https://github.com/comex/star
15. The point of jailbreaking, http://www.saurik.com/id/12
16. How many ways can you remotely exploit an iPhone?, http://www.f-secure.com/weblog/archives/00002003.html
17. Mobilesubstrate, http://cydia.saurik.com/package/mobilesubstrate
18. The state of cell phone malware, http://www.usenix.org/events/sec07/tech/hypponen.pdf
19. Code the Code, http://www.codethecode.com/projects/class-dump
20. iTunes U downloads top 300 million, http://www.apple.com/pr/library/2010/08/24itunes.html
21. SpringBoard, http://www.iphonedevwiki.net/index.php/SpringBoard
22. iSAM: An iPhone Stealth Airborne Malware, Online Material, http://www.icsd.aegean.gr/postgraduates/ddamop/iSAM/iSAM.pdf