Peer-to-Peer Netw. Appl.
DOI 10.1007/s12083-012-0179-x

Exposing mobile malware from the inside (or what is your mobile app really doing?)

Dimitrios Damopoulos · Georgios Kambourakis · Stefanos Gritzalis · Sang Oh Park

Received: 19 July 2012 / Accepted: 29 October 2012

Abstract It is without a doubt that malware especially designed for modern mobile platforms is rapidly becoming a serious threat. The problem is further multiplexed by the growing convergence of wired, wireless and cellular networks, since virus writers can now develop sophisticated malicious software that is able to migrate across network domains. This is done in an effort to exploit vulnerabilities and services specific to each network. So far, research in dealing with this risk has concentrated on the Android platform and mainly considered static solutions rather than dynamic ones. Compelled by this fact, in this paper, we contribute a fully-fledged tool able to dynamically analyze any iOS software in terms of method invocation (i.e., which API methods the application invokes and under what order), and produce exploitable results that can be used to manually or automatically trace software's behavior to decide if it contains malicious code or not. By employing real life malware we assessed our tool both manually, as well as, via heuristic techniques and the results we obtained seem highly accurate in detecting malicious code.

D. Damopoulos (✉) · G. Kambourakis · S. Gritzalis
Department of Information and Communication Systems Engineering, University of the Aegean, Samos, Greece
e-mail: ddamop@aegean.gr

G. Kambourakis
e-mail: gkamb@aegean.gr

S. Gritzalis
e-mail: sgritz@aegean.gr

S. O. Park
Global Science experimental Data hub Center, Korea Institute of Science and Technology Information, Daejeon, Korea
e-mail: sopark3@gmail.com

Keywords Malware · iOS · Dynamic analysis · Behavior-based detection · Smartphone

1 Introduction

Smartphones have experienced a great population growth over the last few years. Nowadays, such devices are an important part of our everyday routine since they are capable of performing a variety of intensive computation tasks and offer different communication interfaces that enable us to access a plethora of ubiquitous services [1]. As expected, this evolution renders smartphones an attractive target to attackers as well. The migration from legacy to converged networks and then to converged communications has also complicated this situation as providers must now deal with millions of devices outside of their reach and the huge growth in converged wireline and wireless network traffic. Even worse, many of these new devices do not yet have adequate security management capabilities, and they add up complexity with the immense variety of applications available from their respective official or third-party online application stores. As a result, network and/or service providers must cope with not only the management and provisioning of these devices and the traffic from specific mobile applications traveling over their wired and wireless interfaces, but also the constantly growing mobile malware threat.

This situation led to the increment of both the number and taxonomies of vulnerabilities exploiting services and communication channels offered for such devices. In fact, smartphones now represent a promising target for malware developers that struggle to expose users' sensitive data, compromise the device or manipulate popular services [2–5]. Also, recent experiences show that by blending spyware

Published online: 04 December 2012

as a malicious payload with worms as a delivery mechanism, malicious applications are capable of being exploited for many facets of espionage and identity theft. According to a recent research, Android and iOS are the two most popular smartphone Operating Systems (OS) sharing together a percent of over 70 % [6]. But it is also estimated that almost one million Android smartphones have been affected by some malware only in the first half of 2011, while the 33.9 % of the free iOS applications had some kind of hidden capability with the intent to leak private user information [6, 7].

Malware especially written for mobile platforms capitalizes on traditional social-engineering techniques such as email and P2P file-sharing, as well as attributes unique to mobile devices (e.g., Bluetooth and Short Messaging Service (SMS) messages). For example, the increasing convergence of various messaging platforms has magnified the mobile malware threat, since users can now send and receive instant messages (IM) and SMS from/to their mobile devices via SMS gateways on the Internet. But, given the very large volume of messages traversing the public IM and cellular networks, the potential for damage from fast propagating malicious software augments exponentially. On the other hand, the detection and tackling of malware developed for ultra-mobile devices and smartphones can be proved a highly demanding task, and as explained further down, it is sure to be more effort-demanding for mobile devices than desktop computers. Specifically, despite the variety of static or dynamic analysis techniques and the signature or behavior-based detection ones described in the literature for personal computers so far, related research for smartphones has been limited leaving several problems unsolved. More precisely, smartphones have limited processing and memory resources, different CPU architecture and a variety of miniature OS versions compared to those of a personal computer, making the malware detection a complex task.

Motivated by this fact, in this paper, we focus on dynamic behavior-based malware detection for such devices. By the term *behavior-based* we mean which (class) *methods* the application invokes and in which sequential order. We concentrate on the popular iOS platform and we introduce a multifunctional software tool, namely iDMA, able to dynamically monitor and analyze the behavior of any application running on the device in terms of Application Programming Interface (API) method calls. That is, iDMA produces a log file which contains in chronological order all native or proprietary API methods the application triggers while running. The aforementioned functionality targets at software testers while there is another one specifically designed for the end-user to detect on-the-fly unauthorized access to private information. To demonstrate the effectiveness of our proposal towards detecting malware, we analyze as a case study the normal behavior of the standard iOS *Messages* application and we compare it with that of two iOS malware subroutines. The results acquired allow us to create behavioral profiles which have been cross-evaluated by well-known machine learning classifiers. As far as we are aware of, this is the first attempt to provide a fully-fledged dynamic solution to analyze iOS applications with the intension to detect malware.

The rest of paper is organized as follows. The next section addresses related work. Section 3 provides some basic background on iOS. Details specific to the implementation of iDMA are given in Section 4. Section 5 assesses the effectiveness of the tool to produce exploitable reports that can be used by software testers to scan for malicious code. Section 6 goes one step further by feeding the results obtained by our tool to heuristic (machine-learning) techniques to identify malicious behavior. Finally, Section 7 draws some conclusions and identifies future work.

2 Related work

Ultra-portable devices and especially smartphones are able to store a rich set of personal information and at the same time provide powerful services, such as location-based ones, Internet sharing via tethering, intelligent voice assistants and so forth. Therefore, it comes as no surprise that popular OS for such devices are facing serious threats that are mainly realized in the form of some malware in the wild. In the same trend, security researchers are seeking novel ways to repel these threats, protect the integrity of the system and preserve the privacy of the end-user. As it is discussed in what follows, this is usually done by employing the same malware detection methods used so far in the realm of personal computers.

To design a Malware Detection System (MDS) for mobile devices, one needs to consider and find answers to the following basic problems. First, it is necessary to define a process and create a tool able to automatically analyze a given software sample and gather useful data that can be later reviewed to assess its behavior. Second, a method is needed to analyze the acquired data and transform them into useful information that may lead to automatically detect malicious behavior or/and private information leaks.

Considering the first issue, static and dynamic analysis are the two types of software analysis methods, which have been mostly used and presented in the literature for personal computers [8–11]. In static analysis, the software of interest is analyzed without executing it. This means that static analysis can be directly employed either on the source code of the software sample or the corresponding binary file and use reverse-engineering techniques to extract a graph overview of what API methods might be invoked

from the code [10]. One of the main problems in static analysis appears in cases where the source code of the sample is not available (this is the common case) or obfuscated and the analysts need to retrieve information by reverse-engineering the binary file which is generally considered a difficult task, especially when the file is encrypted. Also, by using static analysis it is infeasible to determine values that can only be created or calculated when the program is executed. Of course, attackers having the aforementioned knowledge about static analysis drawbacks are able to build their malware using techniques that prevent their code from being statically analyzed.

In dynamic analysis on the other hand, the software sample is analyzed while it is executed by the OS on the host device [10]. In practice, two approaches mainly exist to analyze a software sample dynamically; the first one is by monitoring calls to API methods, while the second is by monitoring the input passed and returned from method invocation(s). Generally, a method contains code that performs a specific task, that is a simple mathematical calculation or more sophisticated operations like creating a Graphical User Interface (GUI), accessing the Internet, loading a file or even making a call or sending an SMS. A method can be either created by a developer for their own application or can be a set of ready to use functions provided by the OS (known as APIs). API methods can be used to cooperate altogether and perform a common task when they get called in a specific hierarchy (sequence). A typical example may be the following: check Internet connectivity (m1), communicate via Internet (m2), load a website (m3). Due to the fact that methods need to be called in a specific order to complete a task, it is possible to create a behavior profile diagram or flowchart of the executed application. The process of intercepting a method call is commonly known as *hooking*. Also, to log a method call into a file it is necessary to create a tool that temporarily intercepts the method(s) being called and inject into them the necessary code that enable the system to record the specific transaction.

As we already mentioned, dynamic analysis gives the ability to track both the actual values that are passed to a method when it is called and those that are going to be returned from the called method. Under this prism and in contrast to static analysis, analysts can extract much more useful information about the data created during the execution of the software. On the negative side, dynamic analysis requires executing the software sample on the device increasing this way the risk that the device may confront. After a log file containing the sequence of method calls has been created, it can be analyzed manually or automatically to construct a behavioral profile which in turn may be translated to normal behavior or malicious one in case the software sample is part of a malware. Of course, automated analysis is most of the time welcomed (especially for the end-user) and in this respect an automated intelligent analysis and detection tool needs to be build.

Two are the main detection methods presented in the literature over the last few years; signature and behavior-based detection techniques. Signature-based techniques, commonly used by antivirus software and Intrusion Detection Systems (IDS), utilize some pre-defined set of signatures which have been created via static or dynamic analysis and represent malicious code. If the same set of signatures gets detected into the application being analyzed, then the IDS indicates that a malware is found. This technique has high rate of malware detection, although a low rate of false positive is maintained [12]. The main problem is that the detection system is unable to identify malicious code until a signature set is created. The second method, also known as heuristic, may employ machine learning algorithms to learn from behavioral profiles in a way that potentially unauthorized actions and malicious profile patterns can be detected. Over the last few years a considerable number of solutions have been presented in the literature trying to detect malware in smartphone platforms.

It is to be noted that in this section we only consider API-oriented malware detection proposals for the Android and iOS platforms. Thus, approaches that deal with detecting illegitimate use of services by a potential malware as a way to detect anomaly behavior [13] and others that examine the hardware performance metrics of a smartphone aiming to detect malware like those in [14, 15] have been intentionally neglected.

In 2009 Schmidt et al. [16] proposed one of the first MDS for the Android platform. Their system performed static analysis of executables extracting this way their list of method calls and comparing it with pre-considered malware methods using the PART classifier. The authors reported an average of 96 % malware detection rate with a 12 % average false alarm rate in their simulation scenarios. A year later, Bläsing et al. [17], presented a sandbox "application" for the Android emulator able to both statically and dynamically analyze suspicious applications. Their system could scan for malicious patterns either inside an application without installing it, or while the application is executing in an isolated environment inside the Android emulator. According to the authors, the proposed system is also able to act as a cloud service. In the same year, Shabtai et al. [18] presented a system able to detect unknown malware instances from static features they extracted from Android games and other kind of applications. To do so they applied a variety of machine learning techniques. More specifically, they managed to achieve a highest detection rate in the vicinity of 92 % (false alarm 19 %) when using the Boosted BayesNet classifier.

In 2011, some more malware detection and analysis mechanisms have been proposed for Android. Luo [19]

tried to identify Android applications that leak sensitive user information by implementing a static method, i.e., a translator that transformed Dalvik bytecode to Java bytecode. Burguera et al. [20] in their work presented an initial approach for dynamically analyzing the behavior of Android applications. They used a crowd-sourcing system to obtain the traces of application's behavior while it was running. Additionally, they proposed a framework to detect malware using client-server architecture. Their experimental results indicated a 100 % detection rate, using the k-means clustering algorithm. Another static mechanism proposed by Batyuk et al. [21] has been able to analyze Android applications and create readable reports for the end-users. Using an automated reverse-engineering technique the authors also proposed a method capable of disabling malicious code residing inside an application without affecting its core functionality.

So far, only two works about malware analysis for iOS devices have been introduced. The first one [10], proposes a static analysis system able to detect privacy leaks caused by iOS applications. By using this system the authors were able to analyze 1,407 iOS application binaries trying to detect those that potentially leak sensitive information such as Unique Device Identifier (UDID), address book records, GPS coordinates etc. The second [22] focused on the possible challenges and open problems that one needs to overcome in the path of creating a dynamic analysis system for iOS. Also, the authors presented some initial ideas and prototypes as a first step towards dynamic analysis. Another interesting work for iOS but in the field of intrusion detection is given in [13]. The authors propose and evaluate a behavior-based IDS for iPhone devices using different machine learning classifiers. They report a True Positive Rate (TPR) of 99.8 %.

From the discussion above it is obvious that so far only eight works have been devoted to the issue of malware detection in both of these popular mobile platforms. All but two introduce static solutions and only one presents a working solution for iOS. This is however anticipated as iOS is considered a closed OS. In fact, both the aforementioned OS restrict access to their internal functions. However, in contrast to iOS, the Android source code is freely available for download and tinkering. Given the aforementioned observations, this work contributes the first dynamic software analysis system for iOS.

3 Background

The primary aim of any dynamic MDS for smartphones is to log every call to API methods, either native or proprietary, and feed these calls to a mechanism able to detect illegitimate behavior. To do so, the MDS needs to fulfill two fundamental requirements. First of all, it needs to gain root permissions in order to be able to hook and override class methods. Second, it needs to run continuously in the background of the OS and constantly track and collect calls to API methods. Although this paper assumes a minimum level of familiarization with iOS programming, this section is necessary for reasons of completeness. Thus, in the following we provide some basic background information on iOS frameworks, Jailbreaking, and hooking of class methods.

iOS is a Unix-like Operating System by nature. Soon after its release, Apple introduced the first SDK for iOS allowing developers to create third-party native applications. To protect its platform from possible malicious applications the vendor has set a variety of restrictions and limitations both inside the iOS and in the way a third-party application is installed in the device. To create custom applications the developers have access only to a limited package of (API) frameworks. Actually, there exist two kinds of frameworks in iOS: public and private ones. The first are those that are endorsed by Apple and are allowed to be used for building App Store applications. Private frameworks on the other hand are intended to be used only by original applications providing high-level programming features. Unfortunately, private frameworks are neither available by the iOS SDK nor documented. Moreover, only inspected, encrypted and digitally signed by Apple's certification authority applications can be released. Each time an application is launched, the iOS recalculates the application signature to verify the integrity of the binary, and if true, it decrypts part of the binary in order to execute it. Also, all third-party applications execute under specifically defined user privileges making this way impossible for them to access the entire iOS file system or to handle low-level processes.

Actually, the only way to run non-certified software is by gaining root permissions on the device using an exploitable vulnerability. This process is generally referred to as *Jailbreak* [23]. Upon jailbreaking, the entire iPhone file system becomes open for use. Also, by jailbreaking the device it is possible to retrieve a private framework. After that, by using the class-dump utility to examine Objective-C runtime information stored in Mach-O files, one is able to generate the (still undocumented) class header file(s) [24]. Apart from the aforementioned limitations, Apple does not offer any frameworks that override iOS class methods. To fill this particular gap, J. Freeman has created the MobileSubstrate extension, a framework that allows developers to deliver runtime patches to system functions using Objective-C dynamic libraries (dylib) [25]. Also, D. L. Howett has contributed Theos, a cross-platform suite of development tools for managing, developing, and deploying jailbreak-oriented iOS development [26]. By creating a dylib and linking it with the MobileSubstrate extension, developers are

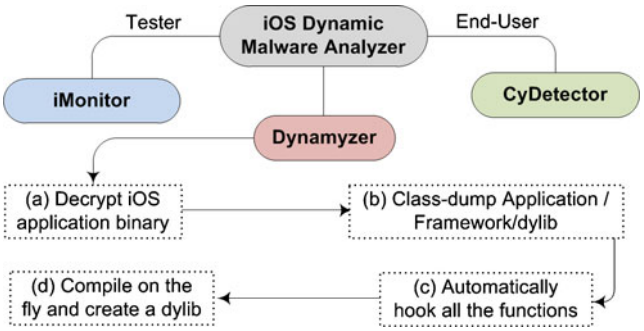

Fig. 1 iDMA overall architecture

able to build applications capable of hooking internal system methods. In addition to all the foregoing, Rastignac [27] created a script namely poedCrackMod that gives developers the ability to dump the executable's encrypted memory.

4 Design and implementation

Considering all the above, we are designing and implementing iOS Dynamic Malware Analyzer (iDMA), an automated malware analyzer and detector for the iOS platform. iDMA consists of a main daemon written in Objective-C and is combined with a proper launch plist (activated at device boot time) and three subroutines written as Objective-C functions, dylibs or bash scripts. iDMA has been implemented using daemons and dylibs for backgrounding, the public and private frameworks and the MobileSubstrate framework with the substrate.h header that overrides iOS API methods. Driven by a menu, the iDMA main daemon depicted in Fig. 1 is responsible for managing its three subroutines which are in charge of the automatic software analysis. That is, the creation of new dylibs to be used to monitor the already analyzed software at a later time (Dynamyzer), the dynamic monitoring of all iOS native and non-native frameworks (iMonitor), and a module able to detect on-the-fly possible unauthorized access to user's private data (CyDetector). It is to be noted that the first two subroutines can be used by researchers or software testers to dynamically analyze iOS software, while the third one by the end-user as a real-time alerting mechanism for detecting possible privacy leaks. iDMA has been compiled using Theos for iOS ARM CPU and tested to run on iOS ver. 5.

In more detail, Dynamyzer is responsible for analyzing new Objective-C frameworks, applications or dylibs stored on the device, i.e., any kind of pre-installed or newly installed software. The reason why we decided Dynamyzer to analyze both applications and dylibs is twofold; first because in the recent literature many third-party applications have been responsible for exposing sensitive user information, and secondly, because hazardous iOS malwares use dynamic libraries for performing their malicious tasks. Moreover, Dynamyzer is able to create a proper dylib for hooking and monitoring the API class methods derived from the analyzed software. Figure 1(a–b) depicts the Dynamyzer discrete processes. In fact, Dynamyzer consists of a combination of scripts able to decrypt the binary file in case of an encrypted application (a) and class-dump it to generate the header file(s) (b). Then, it can automatically hook all the class methods defined in the retrieved header file(s), injecting at the same time the proper source code which allow monitoring the method soon after its invocation (c). Next, using Theos, Dynamyzer compiles the methods being hooked into a dylib which run continuously in the background of the device (d). Bear in mind that in order to decrypt an application, we need to create a modified version of the poedCrackMod script, able to firstly check if the type of the input file is an application, and secondly if the application is in encrypted form or not. Figure 2 depicts a snapshot of a dylib that hooks the method responsible for launching any iOS proprietary application.

iMonitor is responsible for tracking in real-time all or some selection of the currently existing 51 public and 121 private iOS frameworks provided by Apple. Tracking of third-party frameworks (or generally any kind of API) is also possible after analyzing it through the Dynamyzer module. Additionally, the same subroutine is able to monitor 81 standard C frameworks for Portable Operating System Interface (POSIX) systems, an API that provides low-level compatibility between Unix OSs and is also frequently used by malwares [28].

iMonitor consists of a combination of dylibs, one for each header, created by the Dynamyzer. Note that every

Fig. 2 Enabling monitoring on a method

```
%hook SBApplicationIcon                  # hooking the SBApplicationIcon header
-(void)launch{                           # override the function
        WritingAtPath: @"/User/Library/Logs/FunctionBehave.txt"
        start synchronizeFile            # only one function is able to write at the log file
        seekToEndOfFile                  # finds the end of file
        writeData: displayName \n        # appends the name of the function and the returned
        end synchronizeFile
        closeFile
}
%end                                     # end of hooking
```

Table 1 Methods being monitored by CyDetector

Resources	Monitored functions	iOS class
Address book	-(void)ABAddressBookRef ABAddressBookCreate	ABAddressBook
Photo album	-(void)imagePickerController: (UIImagePickerController *)picker	UIImagePickerController
Unique Device ID	-(UIDevice *)currentDevice	UIDevice
GPS coordinates	-(void)locationUpdate: (CLLocation *)location	CoreLocation
Siri authentication	-(void)setSessionValidationData: (NSData *) _data	SACreateAssistant
WiFi connection	-(SCNetworkReachabilityRef) SCNetworkReachabilityCreateWithAddress (CFAllocatorRef allocator, const struct sockaddr *address)	SCNetworkReachability

framework consists of a number of headers, where each one of them includes a collection of class methods. Under this context, iMonitor can be used by software testers for monitoring and logging the behavior of the running application(s) when interacting with native or non-native iOS frameworks. As we show in the next sections the log files produced by this routine can be used to identify malevolent incidents (e.g., by just inspecting the logged events list) or by an IDS as an input sensor with the intention to identify malicious behavior.

CyDetector can be classified as a dynamic signature-based detection tool able to detect only specific system calls commonly used by third-party applications to secretly acquire access to user's private data. Note that for creating this module we took into account the most important calls to iOS native API methods that have been highlighted in the recent literature to be responsible for leaking sensitive user's information. Table 1 summarizes all the selected methods to be monitored by CyDetector. Putting it another way, CyDetector is able to detect if the running iOS application tries to (i) retrieve the unique identifier of the device, (ii) acquire the current GPS coordinates, (iii, iv) access the address book or the photo album, (v) steal Siri authentication keys from the device and (vi) send data over the Wi-Fi interface. In the event one of these methods gets invoked by an application, the user is automatically notified via an alert message about a possible privacy leak attempt. Also, CyDetector provides the user with the choice to decide if they will allow the method that generated the alert to be executed or not.

5 A real case scenario

To test iDMA and assess its effectiveness in detecting malware based on their behavior, we conducted a real use case which is described further down. As already mentioned in Section 2, by performing dynamic analysis, it is possible to detect suspicious method invocation. That is, once an application calls such a method there is substantial possibility for a privacy breach or malicious actions to happen. Naturally, this information can be used into a signature-based mechanism to detect and block these actions. In several cases however, a call to invoke a suspicious method (see Table 1) does not mean necessarily that the application is behaving maliciously. For example, consider the case where a buddy-finder application requires the user's location. In this case, CyDetector, which as already pointed out is a signature-based tool, will inform the user about the invocation of a suspicious method and provide the end-user with the option to decide if they agree to proceed. Based on the aforementioned example, we can presume that signature-based detection is not always enough to detect a malicious task, but it can only be used as an indication and as a first protection measure into user's arsenal of defense tools.

Hence, to create a behavior-based tool able to detect malicious code in an efficient way, it is necessary firstly to understand the basic properties of mobile malware; how it is created, what tasks it executes and in which order, what differentiates it from a legitimate application, and so forth. Thus, for the purpose of this study, it was necessary to analyze both malicious and legitimate applications. Specifically, we selected the well-known iKee-B [28] and the lately introduced rootkit iSAM [2]. On the other hand, the legitimate application we analyzed was 'Messages', a native iOS application provided by Apple for enabling communication via SMS, MMS or iMessaging. We concentrated only on one specific task for each application. Specifically, we examine the infection task from iKee-B, the SMSBomber from iSAM and the opening, writing and sending of SMSs in the Messages application. To monitor all three applications we use iMonitor which is capable of hooking all the necessary iOS and POSIX frameworks. Figure 3 depicts the most important method calls for each application as logged by iMonitor.

The methods being invoked by the 'Messages' application are given in the left side of the figure. More specifically, the user accesses the application directly from the icon displayed in the device's home-screen. After the user touches the icon, the application launches and triggers GUI methods needed to formulate the interface of the application (1). Once the user touches a text-field used to hold the

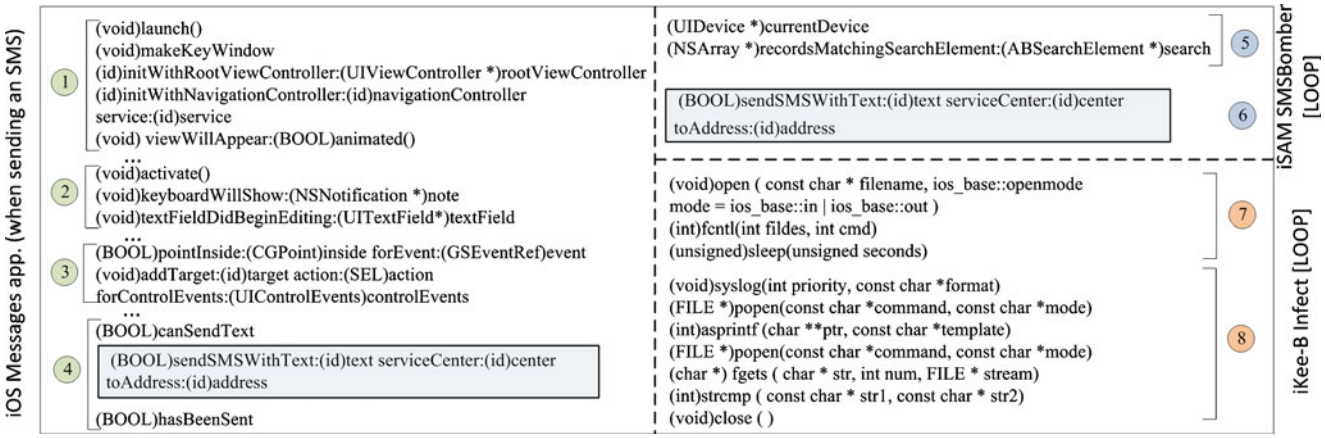

Fig. 3 iMonitor results: Messages, iKee B, iSAM

SMS message or telephone number of the address, the virtual keyboard gets activated. Every time the user types a key, the text or the number inside the text field gets updated (2). As soon as the user presses the Send button (3), the application checks if it is possible for the message to be sent (e.g., if a network connection is available) and if true it completes the task by invoking the SMSWithText:serviceCenter:toAddress method. As a last step, the application checks if the SMS has been sent (4). We then easily observe that for the 'Messages' application to provide a useful GUI and execute all the aforementioned tasks, a large number of private and public iOS frameworks (i.e., ChatKit, CoreTelophony) have been used in combination and in specific order.

All the methods being called by the malicious application are given in the right side of Fig. 3. In the upper right side of the figure, a part of the log file created for the iSAM subroutine namely SMSBomber is depicted. This subroutine is responsible for retrieving the UDID of the device, searching the address book (5) and sending 1,000 SMS to specific targets (6). By observing the log files created by iMonitor, both Messages and SMSBomber (see the two grey rectangles) use the same native, private method, namely sendSMSWithText to send an SMS. Creating a signature of this method to be filtered by CyDetector would be an easy way to alert the user about the possibility of malware. Nevertheless, in the general case, detecting only one method call does not provide enough evidence if the application is malicious or not. This is obvious because the same methods used to perform malicious actions may be invoked by legitimate applications as well.

iKee-B on the other hand is a special case of malware since it is not an Objective-C program (the native iOS programming language). However, as it has been compiled via the GNU Compiler Collection (GCC) it was able to run on iOS. Also, iKee-B employs the POSIX framework which is available to all Unix-based OS, instead of the native iOS frameworks to interact with the device and execute the malevolent task. This fact indirectly makes possible the analysis of iKee-B by Dynamyzer. Every time iKee-B is executed, it checks if the device is already infected and suspends all its functionalities for some minutes (7). Upon wake up it tries to infect other targets (8).

At a first glance, the log files derived for iKee-B do not contain any POSIX method invocation that may be characterized as a possible threat for the device. What we should consider is that every few minutes this application will suspend all its functionalities for some specific time interval and after that it will invoke specific standard C functions that in turn will invoke other POSIX API methods that may be characterized as malicious. It is stressed that the log file does not contain any calls to standard C functions since they cannot be hooked by the Dynamyzer and thus monitored by iMonitor (recall that standard C functions are not part of some object class).

When inspecting the data created by iMonitor, we can assume that every application malicious or not has a specific behavior which can be used to detect malicious code. Although every application performs specific tasks by invoking certain methods in a specific order, in most cases it is quite easy to recognize the legitimate application from the malicious one, just by observing the methods being called. For example, once a legitimate application is launched, it creates a GUI environment making noticeable its presence in contrast to a (typical) malicious one. Also, malicious applications use only basic methods to execute their tasks, being for example uninterested in getting informed if the task is completed or if it has already been completed. This variation is well depicted in Fig. 3 when comparing the records generated for 'Messages' (1–4) and SMSBomber (5 and 6). In addition to all the foregoing, one should highlight that iKee-B uses only low-level POSIX functions and not high-level fully functional Objective-C functions, which are also formed as C functions. This is also an indication

that this portion of code may be malicious. Overall, we can safely argue that the data created by iMonitor, can be proved very helpful for a malware analyst to perceive if an application behaves maliciously or not. This can also greatly contribute in identifying zero-day mobile malware.

6 Employing machine learning

One way to capitalize on the results acquired by iMonitor aiming to create an automatic detection tool (can be also considered as part of an IDS) is to employ machine learning techniques. For the needs of this paper various machine learning classifiers have been utilized. Specifically, as discussed in the following, the analysis procedure takes into account and evaluates four supervised machine learning algorithms, i.e., Bayesian Networks, Radial Basis function (RBF), K-Nearest Neighbor (KNN) and Random Forest. The analysis of the data has been performed on a laptop machine with a 2.53 GHz Intel Core 2 Duo T7200 CPU and 4 GB of RAM. The OS of this machine is OS X Leopard Snow. The data analysis was carried out using the well known machine learning software package, namely Waikato Environment for Knowledge Analysis (Weka).

For using the classifiers it is necessary to provide both training and testing sets. The first is used to train the classifier about the normal and malicious behavior of an application, while the second to test the detection rate and classify the software of interest as malicious or not. Thus, in this case, every record of the training and test files is composed of collected features consisting by the following dyad: *Method, Malicious/Legit*. Where *Method* refers to the method being invoked by the application and *Malicious/Legit* is the binary representation of the two nominal classes (no = legitimate, yes = malicious). An example of such a record is given by the following dyad *(void)launch(), no*.

In our case study, the training set consists of the collected by iMonitor methods (see Fig. 3), characterized according to their type (legit or malicious) and in sequential order. Specifically, the method invocations derived from the iOS Messages application have been represented with the nominal class (no), while the methods from iKee-B and SMSBomber with the nominal class (yes).

Due to the fact that malware for iOS to be used as a test set is for the time being scarce, it was necessary to create new malware instances (i.e., by the exploitation of unofficial iOS frameworks) and/or use some previously non-considered malicious subroutines contained in existing malware. So, for the needs of our experiments, the test set consists of the log files created by iMonitor after running the following applications: (i) a modified version of SMSBomber [2], which retrieves the validation key used by the Siri service for authenticating the iOS device and sends it via SMS to a specific number, and (ii) the iKee-B subroutine responsible for checking the iOS device about bearing the SSH vulnerability. All the methods invoked by the two aforementioned malwares when running on an iOS device (as outputted by iDMA) are summarized in Table 2. These results have been used for testing the classifiers as described further down.

Table 2 Malicious methods used for testing the classifiers

Method invocations	
iKee-B methods used for testing SSH vulnerability	Modified version of SMSBomber
(void)syslog(int priority, const char *format)	(UIDevice *)currentDevice
(int)asprintf (char **ptr, const char *template)	(void)setSessionValidationData: (NSData *) _data
(FILE *)popen(const char *command, const char *mode)	(BOOL)sendSMSWithText: (id)textserviceCenter: (id)centertoAddress:(id)address
(char *) fgets (char * str, int num, FILE * stream)	
(int)strcmp (const char * str1, const char * str2)	

For estimating the effectiveness of the detection process we rely on the well-known True Positive Rate (TPR) and False Positive Rate (FPR) metrics. The results we acquired per classifier are summarized in Table 3. We note that Random Forest is the most promising method showing optimal results of 100 % TPR and 0 % FPR. Bayesian Networks and KNN also obtained very encouraging results, showing a TPR and FPR of 100 %/4 % and 100 %/7 % respectively. It is also of interest that although RBF had the minimum TPR (92 %), it did not produce any false alarm (FPR of 0 %).

As a general remark, all the experiments present highly accurate results, thus providing strong evidence that behavior-based classification in terms of API method invocation may be a very precise way of detecting new types of malware or variations of existing ones.

However, more research is needed to better assess this potentiality. For example, iDMA can be used to construct

Table 3 Malware detection results per algorithm

Bayesian network		RBF		KNN		Random forest	
TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
100 %	4 %	92 %	0 %	100 %	7 %	100 %	0 %

such an API method call oriented behavioral profile by considering a large number of legitimate applications and then feed it to a properly designed IDS to safeguard the device in real time.

7 Conclusions

In the era of converged ubiquitous networks, where nearly everyone owns at least one mobile device, the issue of safeguarding data stored and exchanged between such devices and through popular services for use by mobile users, becomes prominent. In this context, any malware developed specifically to run on modern mobile platforms poses a serious threat. All studies conducted during the last few years show the dynamic evolution of this kind of threat which not only hungers for disrupting network services, but also aims in harvesting keystrokes, passwords, address book records, credit card information, and so on.

Motivated by this fact, we created a tool namely iDMA which can be to the advantage of both software testers as well as end-users. The tool is specifically designed for the proprietary iOS platform and is able to generate exploitable information about the behavior of the application of interest in terms of which method is invoked and in what sequential order. Such a report enables the human actor to make a rather safe decision about the actual (hidden) behavior of the application being examined and if it contains malicious code or not. It also gives the end-user the option to block malicious activity on-the-fly. Our empirical tests with real-life malware reveal highly accurate results in identifying malicious code.

As a statement of direction, we are currently working on using iDMA to create the (invoked) method profile of a large number of widely-used iOS applications. This will allow us to create a rich and generic dataset that represents a non-infected iOS device. This dataset can be used to better assess the efficiency of machine learning techniques in detecting undocumented malware heuristically. Also, it would be interesting to use iDMA in the context of a secure peer-to-peer based architecture (as in [29]) for timely dissemination of information regarding malware. This would greatly contribute against worm propagation and towards minimizing the so-called window of vulnerability.

References

1. Luo H, Shyu ML (2011) Quality of service provision in mobile multimedia—a survey. Human-centric Computing and Information Sciences 1:5. doi:10.1186/2192-1962-1-5
2. Damopoulos D, Kambourakis G, Gritzalis S (2011) iSAM: an iPhone stealth airborne malware. In: Proceedings of the IFIPSec 2011, vol 354(2011). Springer, New York, pp 17–28
3. Damopoulos D, Kambourakis G, Anagnostopoulos M, Gritzalis S, Park JH (2012) User-privacy and modern smartphones: a Siri(ous) dilemma. In: Proceedings of the FTRA AIM 2012
4. La Polla M, Martinelli F, Sqandurra D (2012) A survey on security for mobile devices. In: IEEE communication surveys & tutorials. IEEE Press, New York
5. Teraoka T (2012) Organization and exploration of heterogeneous personal data collected in daily life. Human-centric Computing and Information Sciences 2:1. doi:10.1186/2192-1962-2-1
6. Lookout Mobile Security (2012) Mobile threat report. https://www.mylookout.com/mobile-threat-report. Accessed 20 July 2012
7. Dafir Ech-Cherif El Kettani M, En-Nasry B (2011) MIdM: an open architecture for mobile identity management. JoC 2(2):25–32
8. Egele M, Scholte T, Kirda E, Kruegel C (2012) A survey on automated dynamic malware analysis techniques and tools. ACM Comput Surv 44(2):6:1–6:42. doi:10.1145/2089125.2089126
9. Rieck K, Trinius P, Willems C, Holz T (2011) Automatic analysis of malware behavior using machine learning. J Comput Secur 19(4):639–668
10. Egele M, Kruegel C, Kirda E, Vigna G (2011) PiOS: detecting privacy leaks in iOS applications. In: 18th annual network and distributed system security symposium (NDSS), ISOC
11. Li T, Yu F, Lin Y, Kong X, Yu Y (2011) Trusted computing dynamic attestation using a static analysis based behaviour model. JoC 2(2):61–68
12. Blount JJ, Tauritz DR, Mulder SA (2011) Adaptive rule-based malware detection employing learning classifier systems: a proof of concept. In: Proceedings of the 35th IEEE computer software and applications conf. workshops. IEEE Computer Society Press, Los Alamitos
13. Damopoulos D, Menesidou SA, Kambourakis G, Papadaki M, Clarke N, Gritzalis S (2012) Evaluation of anomaly-based IDS for mobile devices using machine learning classifiers. Secur Commun Netw 5(1):3–14
14. Hahnsang K, Kang GS, Padmanabhan P (2010) MODELZ: monitoring, detection, and analysis of energy-greedy anomalies in mobile handsets. IEEE Trans Mob Comput 10(7):968–981
15. Bickford J, Lagar-Cavilla HA, Varshavsky A, Ganapathy V, Iftode L (2011) Security versus energy tradeoffs in host-based mobile malware detection. In: Proceedings of the MobiSys '11 proceedings of the 9th international conference on mobile systems, applications, and services. ACM Press, New York
16. Schmidt AD, Bye R, Schmidt HG, Clausen J, Kiraz O, Yuksel KA, Camtepe SA, Albayrak S (2009) Static analysis of executables for collaborative malware detection on android. In: Proceedings of the 9th IEEE int'l. conference on communications. IEEE Press, New York
17. Bläsing T, Batyuk L, Schmidt AD, Camtepe SA, Albayrak SAS (2011) An android application sandbox system for suspicious software detection. In: Proceedings of the 6th int'l. conf. on malicious and unwanted software. IEEE Press, New York
18. Shabtai A, Fledel Y, Elovici Y (2010) Automated static code analysis for classifying android applications using machine learning. In: Proceedings of the 2010 int'l. conf. on computational intelligence and security. IEEE CS Press, Los Alamitos
19. Luo K (2011) Using static analysis on Android applications to identify private information leaks. RPE Report, Dept. of Computing and Information Sciences, Kansas State University
20. Burguera I, Zurutuza U, Nadjim-Tehrani S (2011) Crowdroid: behavior-based malware detection system for Android. In: Proceedings of the 1st ACM workshop on security and privacy in smartphones and mobile devices. ACM Press, New York

21. Batyuk L, Herpich M, Camtepe SA, Raddatz K, Schmidt AD, Albayrak S (2011) Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within android applications. In: Proceedings of the 6th int'l. conf. on malicious and unwanted software. IEEE Press, New York
22. Szydlowski M, Egele M, Kruegel C, Vigna G (2011) Challenges for dynamic analysis of iOS applications. In: Proceedings of the workshop on open research problems in network security. Springer, New York
23. Miller C (2012) Breaking iOS code signing. In: Proceedings of the symposium on security for Asia network (SyScan)
24. Nygard S (2012) Class dump. http://www.codethecode.com/projects/class-dump. Accessed 20 July 2012
25. The iPhone Wiki (2012) MobileSubstrate. http://iphonedevwiki.net/index.php/MobileSubstrate. Accessed 20 July 2012
26. The iPhone Wiki (2012) Theos. http://iphonedevwiki.net/index.php/Theos. Accessed 20 July 2012
27. Rastignac (2012) poedCrackMod. http://hackulo.us/wiki/PoedCrackMod. Accessed 20 July 2012
28. Porras P, Saidi H, Yegneswara V (2009) An analysis of the Ikee-B (Duh) iPhone botnet. Technical Report, SRI International Computer Science Laboratory
29. Rahman MS, Yan G, Madhyastha H, Faloutsos M, Eidenbenz S, Fisk M (2012) iDispatcher: a unified platform for secure planet-scale information dissemination. Peer-to-Peer Netw Appl. doi:10.1007/s12083-012-0128-8

Dimitrios Damopoulos is currently a Ph.D Student at the Dept. of Information and Communication Systems Engineering, University of the Aegean. He holds a MSc in Information & Communication Systems Security from the Dept. of Information and Communication Systems Engineering, University of the Aegean, Greece and a BSc in Industrial Informatics from the Technological Educational Institute of Kavala, Greece. His Master Thesis was titled: "Anomaly-Based Intrusion Detection Systems for Mobile Devices", while his Diploma Thesis was titled: "Analysis and implementation of policies for Network Security at the example of the Germany Fachhochschule: Oldenburg/Ostfriesland/Wilhelmshaven". His research interest include Smartphone Security, Mobile Device Intrusion Detection Systems, Mobile Applications and Services.

Georgios Kambourakis was born in Samos, Greece, in 1970. He received the Diploma in Applied Informatics from the Athens University of Economics and Business (AUEB) in 1993 and the Ph.D. in information and communication systems engineering from the department of Information and Communications Systems Engineering of the University of Aegean (UoA). He also holds a M.Ed. from the Hellenic Open University. Currently Dr. Kambourakis is an Assistant Professor at the Department of Information and Communication Systems Engineering of the University of the Aegean, Greece. His research interests are in the fields of Mobile and ad-hoc networks security, VoIP security, security protocols, Public Key Infrastructure and mLearning and he has more than 85 publications in the above areas. He has been involved in several national and EU funded R&D projects in the areas of Information and Communication Systems Security. He is a reviewer of several IEEE and other international journals and has served as a technical program committee member in numerous conferences.

Stefanos Gritzalis is a Professor at the Department of Information and Communication Systems Engineering, University of the Aegean, Greece and the Director of the Laboratory of Information and Communication Systems Security (Info-Sec-Lab). He holds a BSc in Physics, an MSc in Electronic Automation, and a PhD in Information and Communications Security from the Dept. of Informatics and Telecommunications, University of Athens, Greece. He has been involved in several national and EU funded R&D projects. His published scientific work includes 30 books or book chapters, 100 journals and 130 international refereed conference and workshop papers. The focus of these publications is on Information and Communications Security and Privacy. His most highly cited papers have more than 1.600 citations (h-index=20). He has acted as Guest Editor in 30 journal special issues, and has been involved in more than 30 international conferences and workshops as General Chair or Program Committee Chair. He has served on more than 300 Program Committees of international conferences and workshops. He is an Editor-in-Chief or Editor or Editorial Board member for 20 journals and a Reviewer for more than 50 journals. He has supervised 10 PhD dissertations. He was an elected Member of the Board (Secretary General, Treasurer) of the Greek Computer Society. His professional experience includes senior consulting and researcher positions in a number of private and public institutions. He is a Member of the ACM, the IEEE, and the IEEE Communications Society "Communications and Information Security Technical Committee".

Sang Oh Park received the B.S., M.S. and Ph.D. degrees from the School of Computer Science and Engineering at Chung-Ang University, in 2005, 2007 and 2010, respectively. He has been serving as a Senior Researcher of Global Science experimental Data hub Center at Korea Institute of Science and Technology Information since 2012. He served as a Research Professor of the School of Computer Science and Engineering at Chung-Ang University from 2011 to 2012. His research interests include archiving system, embedded system, cyber physical system, home network, and Linux system.