Universitat Oberta de Catalunya

Doctoral Thesis Report

Forensic Imaging and Analysis of Apple iOS Devices

PhD candidate:
Luis Gómez Miralles

Supervisor:
Dr. Joan Arnedo Moreno

A thesis submitted in fulfilment of the requirements
for the degree of Doctoral Programme on Network and Information Technologies

in the

IT, Multimedia and Telecommunications Department

May 2016

Universitat Oberta de Catalunya

Declaration of Authorship

I, Luis Gómez Miralles, declare that this thesis titled, 'Forensic Imaging and Analysis of Apple iOS Devices' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

"It's a cat-and-mouse game. We try to stay ahead. People will try to break in, and it's our job to stop them breaking in."

Steven Paul Jobs, 1955-2011.

UNIVERSITAT OBERTA DE CATALUNYA

Abstract

IT, Multimedia and Telecommunications Department

Doctoral Programme on Network and Information Technologies

Forensic Imaging and Analysis of Apple iOS Devices

by Luis Gómez Miralles

The concept of digital forensics is nearly as old as the field of information technology. A variety of reasons often make it necessary to analyze the data contained in digital devices as part of more complex investigations. A common scenario is technical troubleshooting – for instance, studying a plate reader that fails to scan the plates of a specific car model. Another example are criminal investigations, in which the data in digital devices are used to infer information about one person, usually the device owner; common cases include obtaining and reading a suspect's communications with other people, or determining his location and actions based on the activity logs of her devices.

A set of anti-forensic tools and techniques exist in many disciplines, aimed at reducing the quantity and quality of forensic evidence. The use of anti-forensics does not necessarily cover "bad" actions: there are well-known cases of totalitarian regimes and intelligence agencies abusing forensic tools to conduct surveillance on innocent citizens such as journalists and system administrators. Forensic investigators will try to detect and overcome the effect of anti-forensic processes – giving birth to the concept of anti-anti-forensics. It is obviously an endless fight, and a field worth exploring since it may significantly impact the forensic process.

As many other technical disciplines, the field of digital forensics has experienced a significant boost in the last years due to the proliferation of personal computers and, later on, mobiles devices such as smartphones and tablets – devices that hold significant amounts of information about our lifes: conversations, notes, list of visited places (both on and offline), history of phone calls, and a long etcetera.

In this thesis we present our research on digital forensics in the iOS platform, used by every iPhone and iPad. Our work is structured in three areas, covering different fields of digital forensics:

1. Forensic imaging: the process of extracting the data from the device, minimizing its alteration, in a format as raw as possible.

2. Forensic analysis: examination of the acquired data in order to obtain information that may be useful from a forensic standpoint.

3. Anti-forensic techniques: study of the processes aimed at reducing the quantity and/or quality of evidence left in the device.

In the field of forensic imaging, we concluded that the iPad does support complex USB devices. Using an adapter intended for a different purpose, we were able to connect external hard drives directly to the iPad. This way, it becomes possible to copy the data in the iPad to an attached hard drive, resulting in a x30 speed boost, compared to transferring the data wirelessly, which was the standard at the moment.

In terms of forensic analysis, we found that the use of the AirPrint feature, the iOS protocol for connecting to printers wirelessly, leaves a number of traces in the device, including complete copies of the documents being printed. By modifying an open-source software tool, we were able to recover the contents of those documents that have been printed. We showed that this is possible even when iOS data protection, a hardware-based encryption of the whole filesystem, is present.

Finally, in the area of anti-forensic techniques, we created a proof-of-concept software tool that disables a number of system services used by existing forensic tools to retrieve data. The tool also applies other hardening measures aimed at preventing the abuse of the services that remain activated, if any.

List of research contributions

The relevance of this thesis is backed by a number of research contributions in several journals and international conferences. The doctoral candidate is the main author of all the publications listed below, which have been published or accepted for publication during his work. These publications include:

- two journal papers indexed in ISI-JCR (Q1 and Q3);
- three conference papers published in IEEE Computer Society proceedings, all of them indexed in SCOPUS;
- and one poster presented in an international conference.

In the field of forensic imaging of iOS devices:

- L. Gómez-Miralles, J. Arnedo-Moreno. Universal, fast method for iPad forensics imaging via USB adapter [1]. Proceedings of the 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011, pp. 200-207. DOI: 10.1109/IMIS.2011.41. **IMIS-2011 Best Paper Award**. Indexed in SCOPUS.

  This paper shows that an iPad can control complex USB devices such as hard drives using a cheap adapter. This was leveraged to obtain forensic dumps of the iPad directly into attached USB hard drives, as opposed to transferring the data over the air via Wi-Fi to an external computer which was the standard practice at the moment.

- L. Gómez-Miralles, J. Arnedo-Moreno. Versatile iPad forensic acquisition using the Apple Camera Connection Kit [2]. Computers And Mathematics With Applications, Volume 63, Issue 2, 2012, pp.544-553. Elsevier. DOI: 10.1016/J.CAMWA.2011.09.053. IF: 2.069, Q1: 23/99. Category: COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS (JCR-2012 SE).

  This paper explores the different parameters that affect the process of obtaining a complete dump of the iPad to a hard drive attached via USB. The optimal values for those parameters were identified, resulting in a throughput 29 times faster than that of the wireless methods used at the time.

In the field of forensic analysis of the extracted data:

- L. Gómez-Miralles, J. Arnedo-Moreno. Analysis of the forensic traces left by AirPrint in Apple iOS devices [3]. Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2013, pp. 703-708. DOI: 10.1109/WAINA.2013.40. Indexed in SCOPUS.

  This paper shows that using AirPrint to print documents wirelessly leaves a number of traces in the iOS device, including a temporary copy of the document(s) being printed.

- L. Gómez-Miralles, J. Arnedo-Moreno. AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices [4]. Mobile Information Systems, Volume 2015, Article ID 916262, 10 pages, 2015. Hindawi. DOI: 10.1155/2015/916262. IF: 0.949, Q3: 76/133. Category: COMPUTER SCIENCE, INFORMATION SYSTEMS (JCR-2014 SE).

  This paper presents a method that allows the recovery of documents that have been printed using the AirPrint feature – even in the modern iOS versions which enforce data encryption.

And finally, in the field of iOS security and anti-forensics:

- L. Gómez-Miralles, J. Arnedo-Moreno. Hardening iOS by selectively disabling *Lockdown* services. Poster presented at the 2nd Digital Forensics Research Conference Europe (DFRWS EU), Dublin (Ireland), 2015.

  This poster (attached in Appendix G) introduces a hardening model based on disabling unneeded services; and a proof-of-concept tool, *Lockup*, that implements this model.

- L. Gómez-Miralles, J. Arnedo-Moreno. *Lockup*: A software tool to harden iOS by disabling default *Lockdown* services [5]. Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015, pp. 718-723. DOI: 10.1109/3PGCIC.2015.57. Indexed in SCOPUS.

  This paper presents the details of the *Lockup* tool, which currently works on iOS versions 7 and is freely available as open-source software.

- L. Gómez-Miralles, J. Arnedo-Moreno. Disabling sensitive iOS *Lockdown* services: anti-forensic implications and anti-anti-forensic possibilities. In: Security and Privacy in Intelligent Systems and Communication Networks. Book series: Intelligent Data-Centric Systems. Morgan Kaufmann. Submitted 2016.

This is a book chapter that we were asked to develop as an extended version of [5]. It expands on the proof-of-concept tool and analyzes the anti-forensic consequences of its use, as well as the possible anti-anti-forensic strategies available when dealing with devices protected by such tools.

Acknowledgements

This work would never have happened without the following people, to which I sincerely owe my deepest gratitude.

In order of appearance:

My parents, Marisa and Virgilio. I've known them since long ago and they are very cool! They lent me their laptop in time of need to write this thesis. Oh, and they also raised me!

My thesis advisor, Dr. Joan Arnedo. I had the opportunity to work with Joan while I lived in Barcelona back in 2003 and, apart from being a nice person and valuable colleague, he encouraged me to embark on this doctoral programme. This poor guy has had to chase me for about five years. Thanks for not giving up.

Cynthia, my wife. For the countless hours of care, support, understanding, and nice good coffee early in the morning.

My daughters Carla and Nerea. All the future is belongs to you. Go get it.

Last, but not least, I would like to thank the awesome 1337 persons with whom I have been lucky to work over all these years in the infosec area. Together we have grown both personal and professionally. It is a true pleasure to share this passion and this career with you, my friends.

Contents

List of Figures

List of Tables

Al Rey de Extremadura

Chapter 1

Introduction

In this chapter we outline the motivation for the research presented in this thesis and discuss our contributions to the field of iOS forensics and security.

First we introduce this thesis' object of research in section 1.1. Section 1.2 reviews the state of the art in the field of mobile device forensics and specifically iOS forensics. In section 1.3 we define the main questions and objectives to be targeted by this thesis' research. Section 1.4 presents the main aspects of the research methodology used. Finally, section 1.5 outlines the organization of the rest of this thesis report.

1.1 Statement of the problem

The Oxford dictionary defines *forensic* as *'of, relating to, or denoting the application of scientific methods and techniques to the investigation of a crime'*; or *'of or relating to courts of law'*; the first definition being also referred to as *forensic science.*

The first texts on the topic of forensic science correspond to the field of forensic medicine and date back to the late 18th century [6, 7]. As time passed and science progressed, the term *forensic* has been applied to many scientific fields whenever these fields provided tools and techniques that were useful in the resolution of criminal investigations.

Thus, we have forensic medicine –undoubtly one of the most known fields, probably due to TV shows–, but also forensic accounting: these are the people that inspect a company whenever big sums of money disappear; there are forensic chemists who assist in investigations related to drugs or explosives; experts on forensic dactyloscopy that study fingerprints... and many other disciplines such as art forensics, forensic anthropology, forensic linguistics, forensic psychiatry, etc.

Unsurprisingly, the popularization of computer systems and other electronic devices has resulted in the field of *digital forensics* (or *computer forensics*), defined as *'The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations'* [8].

Digital forensics has become a well-known discipline to law practitioners, judges, and law enforcement bodies all around the world, and nowadays the use of e-mails, SMS messages or other digital files as evidence in court happens constantly. In most countries, law enforcement bodies have created special groups focused on investitaitons in digital and online environments; examples include the FBI's Cyber Crime Division, or in Spain the Policia Nacional's BIT (*Brigada de Investigación Tecnológica*) and the Guardia Civil's GDT (*Grupo de Delitos Telemáticos*).

In the last decade our society has established a tighter relationship with communication technologies thanks to the raise of smartphones. As user requirements grew beyond what mobile phones could realistically offer, companies developed a new family of communication devices – smartphones: full-fledged computers, always on and always online, portable enough to accompany the user at all times, holding all kinds of personal information: phone calls, personal and corporate email, chats, calendars, address books, to-do lists, history of visited places, photographs, videos... many of which are usually synced across a number of online services (Facebook, Twitter, Dropbox, etc.).

As a result of the entanglement between our digital devices and our personal data, in the last years smartphones and other mobile devices such as tablet computers have become increasingly relevant as sources of evidence from a computer forensic standpoint. Data extracted from mobile devices is commonly used nowadays as forensic evidence in criminal investigations - a notorious example being the investigation of Michael Jackson's death [9].

The practice of digital forensics has needed to adapt quickly to the emerging mobile technologies. Where we once had a homogeneous personal computer market, mainly dominated by a few different Windows versions, with minor representations of Mac OS or Unix-based systems, now we find that the most personal devices, the ones that always accompany their users and are more prone to contain sensitive information, run software environments which simply didn't exist a few years ago - namely Google's Android and Apple's iOS. Furthermore, because of the competitive nature of the market, with each new version of these systems new functionalities are added in order to appeal to a greater set of users and thus become their device of choice: biometric authentication, wireless printing (AirPrint)... In fact, some of these new features may manage personal user data and are worth analysing from a forensic investigation standpoint.

Android is a Linux-based initiative developed by Google and the Open Handset Alliance. A beta version was released in late 2007 and version 1.0 hit the streets in September 2008, in what many consider to be Google's answer to Apple's introduction in the smartphone market with the iPhone. From a user perspective, Android and iOS have offered similar features over the years.

Android is used by many different vendors in their smartphones and tablets –usually adding a layer of customization– including Acer, Asus, Bq, HTC, Huawei, LG, Motorola, Samsung, Sony, and others. Applications are distributed through the Google's Play Store, where they can be published without being reviewed; this makes it relatively easy to create and distribute malicious apps for these devices, something that rarely occurs on iOS. Users can also obtain apps from alternate sources in the Internet.

This work focuses on the iOS platform, which as we will explain presents a solid security model that makes it complicated to gain full control over the device – be it with malicious purposes or for research.

The first version of iOS was released together with the first iPhone and iPod Touch models in 2007, although by that time its name was *iPhone Operating System* – it was rebranded as *iOS* when the iPad was released in 2010. Apart from a web browser never seen before in a smartphone, the applications it offered where not very different from those of other devices: phone, messages, contacts, calendar, notes, email, camera... What marked the difference, however, was the user interface and the multi-touch screen to be used with no stylus – typical consequences of Apple's tight integration between carefully chosen hardware and software written for that precise hardware.

Major iOS versions have been released on a yearly basis adding important features such as support for third-party applications through the App Store (after being reviewed by Apple), multitasking, voice control, integration with desktop computer, better integration in corporate environments with support for mobile device management solutions, remote deployment of software and configuration profiles, support for Microsoft Exchange, etc.

Security controls have also been improved over the years in both the software versions and the new hardware releases. The current iOS releases enforce a complex security model [10] supported on a number of security mechanisms including privilege separation, mandatory code signing, address space layout randomization (ASLR) and application sandboxing.

In Q2 2015 the iPhone and the iPad accounted for more than 10% of the global mobile device market share – a number that grew beyond 60% if we focus on big environments (both private and public sectors) [11]. This is partly due to the iOS security model mentioned above, which in turn presents some interesting obstacles to the researcher: the same security mechanisms built into the hardware and software to keep each process and its data isolated from the rest of the system, prevent the researcher from having full control over the device.

Due to its high penetration rate in critical sectors (big companies, public sector) and the challenges posed by its security architecture, we decided to focus our research on the iOS platform.

Aiming to cover as many aspects of the forensics field as possible, we decided to vertebrate our research along these three areas:

1. Forensic imaging or acquisition: data extraction in a format as raw as possible.
2. Forensic analysis: inference of useful information derived from the raw data obtained in the previous step.
3. Anti-forensic techniques: study of processes that reduce the quantity and/or quality of forensic evidence left in the device.

1.2 Background

The areas of mobile devices and information security have received significant and growing attention in recent years. Under that general topic, forensics has also been subject of research and, although the number of publications so far might not be high, grows year after year.

Figures 1.1 and 1.2 show the number of publications indexed in Scopus and ISI Web of Science for the search terms *Android forensics* and *(iOS — iPhone) forensics* in the last six years (2010 to 2015). The Android platform seems to have been more widely explored, possibly due to its openness: Android can run in a wide range of devices, whereas iOS will only work on Apple devices; and generally speaking, a lot of the Android codebase (both at the kernel and applications level) is derived from Linux, which many people are already familiar with and is prone to research given its open source philosophy.

In our case, we have decided to concentrate on the iOS platform given that it poses a number of interesting challenges due to the vendor's restriction on the device usage. iOS enforces a number of security layers –Address Space Layout Randomization (ASLR),

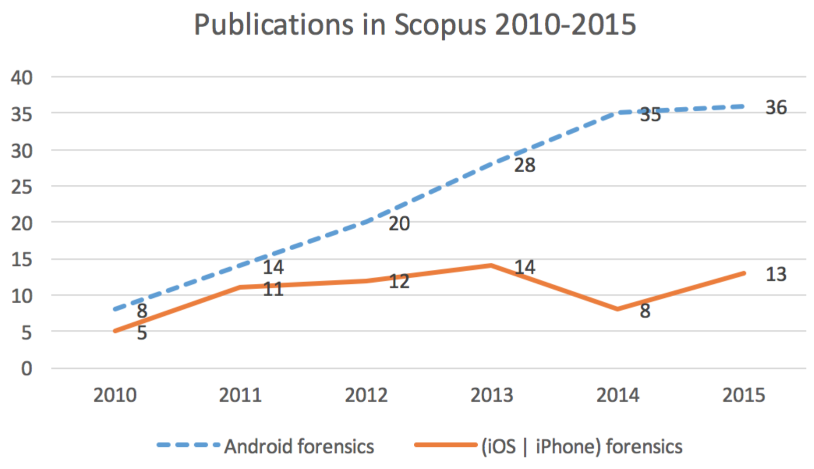


Figure 1.1: Publications in Scopus from 2010 to 2014.

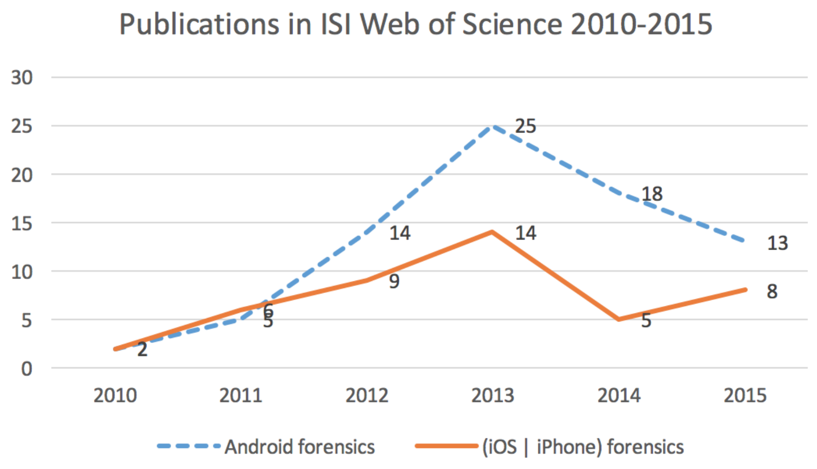


Figure 1.2: Publications in ISI Web of Science from 2010 to 2014.

Data Execution Prevention (DEP), application sandboxing, privilege separation, mandatory code-signing, etc. – that restrict devices to run only applications allowed by Apple and prevent the owner from having full control over the device. This, apart from being the base of the interesting iOS security model, is an obstacle for researchers who wish to inspect the device internals. In order to overcome these restrictions, researchers and other users have to resort to the *jailbreak* technique. In fact, a number of forensic tools and techniques rely on the use of jailbreak to gain access to the device's internals.

1.2.1 Jailbreak

On July 6th 2007, just one week after the iPhone was launched, the existence of a method to get a full, interactive shell was announced by George Hotz [12]. This was the first step towards bypassing Apple's restrictions on their devices, making it possible to execute any program and not only those approved by Apple; a process that has been named **jailbreak**. In other mobile platforms, such as those running Google's Android operating system, a similar process is known as *rooting*. Vendors usually dislike this technique, although in most countries it is legal or at least not definitely illegal.

Once a device has been *jailbroken*, the user has access to the *Cydia* package manager, through which additional software can be installed. This way, the user has access to UNIX shells and standard tools (the part which is relevant for our research) together with all kinds of third-party apps and tweaks.

Different jailbreak techniques and tools exist for most iOS versions, and all of them achieve its purpose by exploiting software bugs in those versions. Due to the restrictions imposed by the iOS security model, usually more than one exploit is required to achieve full control over the device (for instance: one exploit to escape the sandbox, and another one to elevate privileges). Shortly after such tools are released for the most recent iOS versions, Apple studies these tools to learn about the bugs being exploited and patches them in a new iOS release.

In 2008 Moenner pinpointed *"the latency in coverage of newly available phone models by forensic tools"* as one of the problems for forensic specialists working with mobile devices. We consider very realistic that jailbreak updates will usually be available before other software products may be upgraded: vendors may fail to release an update in time to support newer iOS versions; or they may even not release it at all if the product is discontinued. Consequently, we find it valuable to have forensic tools and mechanisms available to use by means of the jailbreak technique.

We have observed that, on average and focusing on modern iOS versions (iOS 7 and 8), it takes three weeks for researchers to release functional jailbreaks, and –again, on average– afterwards it takes Apple two weeks to analyze the jailbreak tools, find the iOS bugs they exploit, and release new iOS versions fixing those bugs. This, and the fact that every iOS release so far has been jailbroken, further supports our statement.

Tethered versus untethered jailbreak

Depending on the nature of the bugs being exploited in a particular iOS version, a jailbreak can be *untethered*, meaning that the effects of the process are permanent and the device will remain at a jailbroken state across system reboots, unless a full system restoration is performed; or *tethered*, meaning that the jailbreak process is unable to survive across reboots, and if the device is rebooted or turned off it will require to be connected each time to an external computer in order to achieve the jailbroken state again.

Although users tend to prefer an untethered jailbreak, considering the simplicity and convenience, having the option of using a tethered jailbreak is very interesting from the research point of view, as it allows the researchers to gain temporary root access to the device and afterwards reboot it into a non-jailbroken state.

Most of the public jailbreak tools released for iOS versions 7 and later have been of the untethered type.

The `redsn0w` tool

As an example of the inner workings of a jailbreak tool, we will now present an overview of the processes followed by `redsn0w`, a popular tool developed by the iPhone Dev Team for jailbreaking iOS versions 4 and 5 in devices with hardware up to the Apple A4 processor; that is: up to the iPhone 4 and the first iPad.

For a more thorough description of each step, refer to [13].

`redsn0w` runs in a computer under OS X or Windows, and the iOS device is connected to the computer through a USB cable. After instructing the user to put the device in DFU mode (*Device Firmware Upgrade*), the tool exploits a well-known heap overflow in the USB DFU stack of the bootrom using the so called `limera1n` exploit, and patches the bootrom to disable the signature verification code.

Having done this, texttttredsn0w builds a custom ramdisk with a patched kernal that will allow execution of unsigned code, and send this ramdisk to the device, which will boot it and run a *jailbreak* process contained within. This process, in turn, will perform the following steps in the device:

1. Remount the system partition as read-write, and make this behavior permanent across reboots. In non-jailbroken devices this partition is always mounted in read-only mode.
2. Install the untethering exploit by adding or modifying files in the system partition.
3. Install a Lockdown service called `com.apple.afc2`. This is done for the convenience of certain jailbreak tools, but it introduces a huge security risk as it exposes the whole filesystem with read-write permission to any trusted device over a USB connection. We addressed this risk in our paper [5] and presented a tool to disable this and other potentially dangerous services.
4. Install basic UNIX tools.

5. Application stashing: free space in the system partition by moving non-critical folders to the user data partition and creating appropriate symbolic links.
6. Installation of application bundles. This normally means installing Cydia, a tool for software package management that will allow the user to add or remove apps.
7. Post-installation process: call `sync()` to ensure the changes are written to disk; unmount the disks; and reboot the device.

A similar approach is followed by other tools in more recent devices. However, the lack of a bootrom exploit in those devices makes it unfeasible to upload and boot a custom ramdisk, and instead userland attack must be used. This usually involves one exploit against a particular app to escape the sandbox, and a second kernel exploit to elevate privileges.

1.2.2 Forensic data acquisition

In the field of forensic science in general, and in digital forensics in particular, it is globally accepted that the initial acquisition of data –and its proper preservation, establishing an adequate chain of custody– is a crucial stage [14] on which relies the credibility of all the evidence that will be derived from the analysis of said data. Under the legal systems of most western countries, if an adversary can cast a reasonable doubt on the process through which the initial data was acquired, it would result in the dismissal of any evidence derived from its analysis.

The term *chain of custody* exists in the whole forensic science field, and it refers to the documentation that reflects the seizure, custody, control, transfer and disposition of evidence (be it physical or digital). The term is also used in other fields, with different meanings: for instance, in the wood industry it refers to the process for ensuring that wood products are originated from forests managed in a sustainable way.

In the field of digital forensics, no matter the technological platform or data type(s) being analyzed, any data –be it a single file, a compressed directory, a dump of a whole storage device, etc.– translates into a stream of bytes. Thus, the chain of custody is generally established by means of a well-known cryptographic tool: hashes.

A hash function receives an input (stream of bytes) of any given size and returns an output of a fixed size. Any small variation in the input produces a completely different output, and it is not technically feasible to generate collisions (different input strings that produce the same output) [15]. Examples of hash functions are: MD6 (returns an output of 512 bits), RIPEMD-320 (320 bits) or SHA-256 (256 bits). Other well-known examples such as MD5 or SHA-1 (with outputs of 128 and 160 bits respectively) are currently considered insecure given the growing ability to find collisions in these functions [16, 17] due to advances in both cryptanalysis and computing power.

Hashes play an important role in digital forensics. The output of a hash is a manageable text string –for instance, the 256 bits of a SHA-256 function can be printed in 64 hexadecimal characters–, and thus for a number of sources of information (say, dumps of storage devices seized during a search operation) it is easy to calculate hashes, even on-the-fly as devices are copied, and print them in paper, ideally transferring them to an affidavit or official minutes. If at any point in the future –say, when the suspect is presented with the results of the investigation– any party questions the integrity of the source data being analyzed, this can be easily verified by just recalculating the corresponding hashes and matching them against the ones recorded during the seizure.

Every existing forensic imaging tool, be it hardware or software-based, offers the user the ability to calculate hashes for the acquired information. Hardware forensic cloners, such as those sold by Logicube, Tableau, Voom or WiebeTech to name a few, display the hash on screen and also store it, together with metadata such as date and time, in a separate file along with the acquired image. And virtually every software tool for forensic cloning (X-Ways, OSFClone, FTK Imager, Encase Forensic Imager... again, to name a few) likewise calculates hashes and stores them, either in a separate file, or together with the image in a container file, as Encase does with its E01 file format. Even with a diverse set of files, there are trivial ways to either calculate hashes for all of them, or put them together in any kind of container (a ZIP file could suffice depending on the case) and calculate the hash of just the container file.

Although our research does not particularly address the topic of the chain of custody, its importance must be noted. If the investigator resorts to the acquisition technique that we presented and discussed in [1] and [2], it is easy to calculate hashes on the fly or as soon as the image has been transferred.

Note that jailbreaking a device is a process that modifies the information stored in it. Consequently, if this is done during the forensic acquisition stage, the investigator must carefully document the exact software or technique used to perform the jailbreak. If needed, in the future this would allow any interested party to study the same process and verify what are the exact modifications that take place in the device. Nevertheless, it is hard to imagine any scenario in which a party may argue that incriminating evidence –such as a particular e-mail message, a chat log, or a GPS bookmark– has been introduced by jailbreaking the device during acquisition; and in any case it would be trivial to dismiss such an accusation.

Moving on to the more technical aspects of acquisitions, a very basic approach to acquiring user data in the iOS platform is the so called *logical acquisition*: connecting the device via the standard USB cable to a computer running iTunes, Apple's multimedia player which is in charge of synchronizing content to the device. Using its AFC protocol (*Apple File Connect*), iTunes syncs existing information (contacts, calendar, email accounts, apps, etc.) and can even retrieve a complete backup of the device; however there are two important caveats in this process:

1. The device needs to be correctly paired with the iTunes software in order to sync. If the device is protected by a passcode (which is the most probable case in all devices introduced since 2013 with the TouchID biometric technology), the investigator cannot unlock the device to authorize trusting (and start syncing data to) this new iTunes installation. A workaround for this is presented in [18]: impersonating a device known (and trusted) by the iOS device, such as the owner's computer, by retrieving from it a set of files known as *escrow keybags*.
2. A dump obtained this way will miss logs and system files that could be of interest in certain scenarios, as well as all the unallocated space, from which deleted files could be recovered.

This logical acquisition is the process followed by most iOS forensic tools such as Lantern or Oxygen, since their first versions.

After the first iOS jailbreak was available, Zdziarski [19] proposed a basic method for obtaining a forensic image of the iPhone with a *physical acquisition* approach, by jailbreaking the device and using SSH access and the `dd` and `netcat` standard UNIX tools, which by that time had already been ported as a part of the growing iPhone jailbreak community (Similar methods are explored by [20] against a Microsoft Xbox device); the data transfer process was done through the device's Wi-Fi interface. In this kind of physical acquisition, the whole storage area is dumped; this includes the unallocated space, from which deleted files could be recovered.

One particular vendor, iXAM [21], developed a 'zero-footprint' solution that relied in the same bugs and exploits used by jailbreak tools. Instead of completing the jailbreak and installing the *Cydia* package manager as usual, their software uploads a tiny, small-footprint software agent which takes control of the system, dumps the solid state storage, and then reboot the device back into its normal state. The problem with these methods is the need for continuous support and upgrades as new iOS versions become available; in fact, according to iXAM website their product works only in the iPhone 4 (introduced in 2010) and older devices.

A similar process was described by [22], although the publication of that paper was supposed to be accompanied by the release of a tool that never saw the light. And other authors have explored the use of similar techniques in other platforms such as Android [23] or Windows Mobile [24]. Another paper in 2013 presented the design and implementation of an iOS forensic tool [25] aimed at simplifying the forensic acquisition of devices running iOS 6, which had been released one year before. However, it seems that the tool itself was not released.

With the release of iOS 4 in 2010, Apple introduced hardware-based encryption, branded as *iOS data protection*). Bédrune and Sigwald analyzed [26] the underlying technology and released [27] a set of open source tools capable of decrypting disk images and even undeleting certain file types; and in fact we used their tools for our publication *AirPrint Forensics: Recovering the Contents and Metadata of Printed Documents from iOS Devices.*

Over time Apple has improved iOS' *data protection* (encryption) implementation at both the hardware and the software levels. At the hardware level, it is important to note that recent devices (introduced 20111 onwards) are shipped with a new bootrom that fixes the bugs exploited by tools such as [27] in jailbroken iOS devices to decrypt and undelete files. And unfortunately, a similar bug has not been found in modern iOS devices – or at least, not publicly announced. As a consequence, so far it is not possible to recover deleted files from modern iOS devices, nor to perform physical acquisition.

Having lost one of the main benefits of jailbreak –the ability to defeat iOS data protection mechanisms and decrypt files, even undelete them–, commercial tools have returned to the *logical acquisition* method [28] which does not require to jailbreak the device. The tools themselves can behave –and be seen by the iOS device as– the iTunes software, and can only acquire whatever information the device is willing to expose – or sync to iTunes. These tools can also operate on iTunes backups extracted from a computer, without access to the original device.

1.2.3 Forensic analysis

Once a forensic investigator has obtained, through a proper acquisition process, a set of data, the next step consists in analysing the data in order to infer relevant information - generally information related to the device usage and the various activities that users may have performed using the device.

In the case of the iOS platform, shortly after the first iPhone model was released in 2007, Zdziarski was the first person to research what mechanisms could be used to infer user activity from the device [19]. By simply targetting two basic data types on which iOS relies heavily (SQLite databases and Property List files), he was able to extract a lot of information about virtually every application existing in the device: address book, calendar events, phone logs, voicemail messages, transcript of sent and received text messages, bookmarked map locations, web bookmarks, email messages, etc.

After the App Store was introduced in iPhone OS 2.0 and 3rd-party applications started to appear, researchers observed that the same techniques (targeting Property List files and SQLite databases) were highly effective. The Property List format is widely used by Apple itself across OS X and iOS, and Xcode –Apple's development environment, mandatory to write iOS applications– generates and embeds files of this type inside the applications. As for SQLite, the format has been used since iPhone OS 1.0 and is well supported by the OS, as well as simple for developers to use.

In 2010, Morrissey [29] carried out an extensive study of the traces left, and how to recover them, by the use of the following iOS 4 components and applications:

- Data from iOS applications: address book, call history, maps (favorites, recent destinations), notes, SMS, media gallery, voicemails...
- Internal iOS components: caches, configuration profiles, history of typed words, logs, preferences, etc.
- Social networking applications: Facebook, AOL AIM, LinkedIn, Twitter, MySpace.
- VoIP applications: Skype, Google Voice.
- Other applications: Craigslist, Google Mobile, Opera, Bing, iDisk.

Over time, commercial tools such as [30, 31] have added the ability to find and present to the investigator the data pertaining to many popular third-party applications such as the ones mentioned above. Interestingly, many of these tools have also added a *timeline* view which aggregates, sorted chronologically, pieces of information extracted across all the applications that the software can interpret.

Subsequently some authors have updated and explored in higher detail some applications from the above branches, how their use affects the device and what files can be found. The logical acquisition method is normally used, given that it works in any iOS version and does not need to jailbreak the device. This means, however, that their analysis focuses solely on files that can be ”seen” by the device –and by the user– at the time of acquisition, ruling out the recovery of any deleted files.

For instance Sgaras [32] focused on the analysis of chat and Voice-over-IP (VoIP) applications. Analyzing iOS devices running Skype and WhatsApp, he was able to extract several items including: lists of contacts with profile pictures, conversations, media files sent from and to the device in each conversation; etc. Iqbal [33] achieved comparable results when focusing on the ChatON Instant Messaging application: he was able to extract contact lists, conversation history, and sent and received files from both iOS and Android devices. Other authors [34, 35] have targeted social networking applications (Facebook, LinkedIn) with similar results.

The Apple iTunes software, available for OS X and Windows, can be used to synchronize contents between the computer and iOS devices, and to store backups of the devices in the computer's hard drive. Carpene [36] explored what information can be inferred about an iOS device by analyzing the backup stored in the iTunes software of a Mac or PC computer. This method has the additional benefits of not risking compromising the original device; however the author enumerates two caveats: that *the evidence extracted is not a raw image of the device, and rather a logical set of data, which isn't as desirable*; and that iTunes offers users the option to encrypt their backups, which would make this method fail.

Cheema published a paper [37] presenting an open source toolkit for iOS forensics called 'iPhone Digital Forensic Analysis Toolkit', however the tool itself apparently was never made public. This could be due to the fact that the tool was designed to support iOS 3, which had been released in 2009 and discontinued in 2010, whereas the paper was published in 2014.

A few works, notably [38] and [39], have focused on the low-level nature of the NAND chips that conform the storage area in iOS devices. In particular, Qiu observes [39] that the probability of recovering deleted files in these devices is much higher than in SSD disks (which also rely on NAND chips) due to the particular garbage collection strategy employed by iOS.

Finally, in 2013 Ariffin [40] developed a method for recovering deleted images from devices running iOS 6. It uses mechanisms similar to those used by [27] to defeat the iOS encryption layer, resulting in a complex method –though highly skilled and meritory– which made it hard to port it to future iOS versions. As far as we can tell, no further research was conducted on this line and the method was never adapted to the newer iOS releases.

We can summarize that researchers have mostly targeted the forensic traces left by many third-party apps, however certain core features have not received as much attention; that is the case of the AirPrint wireless printing system, which we address in our research. And in the particular field of recovering deleted files, there have been interesting initiatives although the improvements to the iOS data protection system at both the hardware and software level in each yearly release cycle make it impossible for any method to stand for long.

1.2.4 Anti-forensic techniques

The concept of *anti-forensics* has been around at least since the beginning of the century, applied to the field of digital forensics, and has received significant attention in the last few years. No unified definition exists for the term, although several authors have attempted to provide it. For instance, Grugq [41] refers to it as *reducing the quantity and quality of forensic evidence*, and Harris [42] as *any attempts to compromise the availability or usefulness of evidence to the forensic process.* Liu [43] identified four primary goals for anti-forensics:

1. Avoiding detection that some kind of event has taken place.
2. Disrupting the collection of information.
3. Increasing the time that an examiner needs to spend on a case.
4. Casting doubt on a forensic report or testimony.

To a certain degree, it is almost always possible to produce some kind of anti-forensic effect in any modern environment by means of more or less advanced tools and techniques. In 2007 Kessler [44] classified the existing anti-forensic tools and methods in four groups: data hiding, artifact wiping, trail obfuscation, and attacks on the forensic tools themselves.

Garfinkel [45] analyzed the existing anti-forensic techniques and came out with a very similar classification: cryptography and steganography (for data hiding); overwriting data and metadata (wiping); minimizing the attacker's footprint; and directly attack computer forensic tools. His analysis also included basic countermeasures such as: sending logs to an external location out of the attacker's reach such as a 'log host' or CD-R media; or making the existing forensic tools more resilient by addressing common flaws in components such as file identification heuristics or decompression algorithms. A wider approach is provided by Harris [42], which again classifies techniques into four groups: destroying evidence; hiding evidence; eliminating evidence sources; and counterfeiting evidence.

There is some research on anti-forensic methods on mobile platforms. For instance, in the Android platform, [46] presented an overview of different techniques that can be used to achieve anti-forensic effects of the four types defined by Harris. The techniques presented include: destroying evidence by deleting specific database records; hiding evidence by using Android's *private* folders; or tampering system files in order to keep certain content from being indexed by the system and make it invisible to an unaware investigator. Albano [47] also suggested a technique for selectively delete evidence on the Android platform, by copying all data from the device internal storage to an external SD card, altering the data there, and moving it back to the device.

As far as our point of interest is concerned, in the iOS platform D'Orazio presented [48] a series of methods for hiding, deleting and counterfeiting evidence on iOS by jailbreaking the device and altering some of the per-file encryption keys used by the operating system's *data protection* feature. His research was conducted on iOS version 6 and it is not clear whether his tools could be ported to current iOS versions.

As far as the author knows, the fourth group of the Harris classification (eliminating evidence sources) has not been subject to research in the iOS platform. This is something that we have addressed in our research.

Recent publications strongly suggest [18] that a number of iOS core system services that present user data to forensic software tools are apparently being abused by certain high-profile attackers [49] to infiltrate telcos and other high-profile companies [50]. It would be interesting to explore the possibility of disabling those system services which the user does not need – which would cover the fourth group of Harris anti-forensics classification: eliminating evidence sources.

1.3 Questions and objectives

Within the context of forensic research of iOS devices, this thesis defines the following objectives, one for each of the three areas: forensic imaging, analysis of the extracted data and anti-forensic techniques.

- O1. To improve the iOS forensic acquisition process.
  - O1.1. To explore the capability of iOS to support the USB mass storage device class protocol.
  - O1.2. To define an iOS forensic acquisition method through an attached USB device.

- O2. To analyze new iOS features from a forensic standpoint.
  - O2.1. To analyze the traces left by the use of AirPrint.
  - O2.2. To create a methodology to extract the traces of AirPrint.

- O3. To integrate anti-forensic tools for iOS.
  - O3.1. To explore the possibility of disabling unneeded system services in order to harden the device security while keeping its features usable.
  - O3.2. To define a model for hardening iOS by disabling exploitable services, and to implement it in a proof-of-concept software tool.

1.4 Research methodology

The work presented in this thesis is based on experimentation with real devices and, where needed, testing with different tools and methods available to evaluate their throughput and features.

In terms of forensic imaging our methods follow the lines of the physical acquisition process described by Zdziarski [19] and Varsalone [51].

In particular, those methods and ours share many characteristics: they all require to jailbreak the iOS device; all of them make use of the `dd` command; and in all cases only the data partition (*/dev/rdisk0s2*) is imaged, omitting the system partition and the initial disk structures such as the partition table. This makes sense given that iOS is designed to store user-generated information in the data partition as shown in figure 1.3, and some of the tools that can deal with iOS images expect to receive the image of this partition and not that of the whole disk; nevertheless, it would be theoretically possible for a power user to customize a jailbroken device and have user data stored in the system partition, potentially preventing the information from being acquired if these methods are used.

However our method differs from the others in two main points.

First, instead of transferring the data over a Wi-Fi network, we will try to attach a physical hard drive to the iOS device and image the device internal storage to the attached hard drive. In order to do this we intend to leverage the *Camera Connection Kit*, an accessory sold by Apple consisting of two adapters that can be connected to the iOS device. One of these adapters provides a USB port and, although it is supposed to be used only to import pictures from attached digital cameras (something that could be done with the relatively simple USB PTP protocol [52]), it is possible that Apple has implemented support for the whole USB mass storage device class protocol.

And secondly, while both Zdziarski and Varsalone transfer data in chunks of 4 KB. In our case we will experiment with greater block sizes which should provide a significant improvement in the throughput.

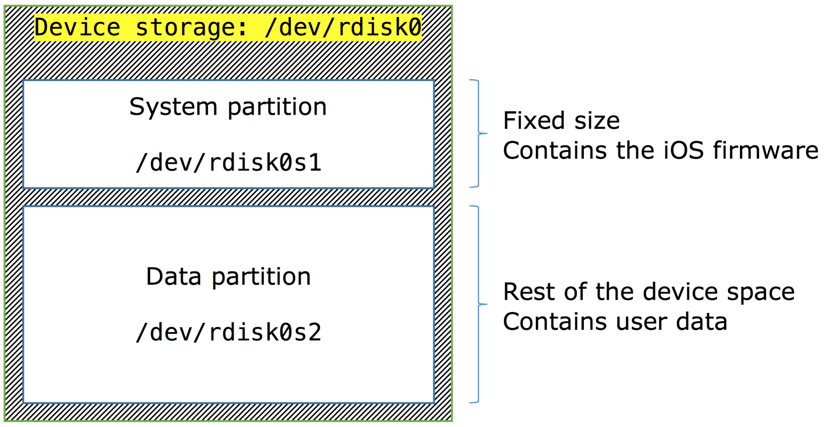


FIGURE 1.3: Partition scheme of every iOS device.

In terms of forensic analysis we use file carving techniques as described in [19] and [53]. This data recovery mechanism is based on targeting specific file types by linearly analyzing the disk stream and identifying specific *signatures* (predefined file headers and footers); while it may generate many false positives if imprecise signatures are used, it has the advantage of being filesystem-agnostic, being a very useful approach when no other tools exist for the filesystem in question.

We decided to analyze the forensic traces generated by the use of the AirPrint subsystem given that this was a new feature at the moment and there was no research at all on this topic. Even at the time of finishing this thesis report, there semes to be no additional research on this particular field.

During this part of our research we have leveraged the work of Bedrune and Sigwald [26], who developed an open-source tool [27] capable of defeating iOS' *data protection* (the hardware-backed encryption features introduced by Apple in iOS 4) by using a known exploit against the BootROM of certain iOS devices (up to the iPhone 4).

Finally, **in terms of anti-forensic techniques** our work has been based on experimentation on the interaction of the iOS device with real devices (a computer, a stereo system, a hands-free device) to study how the modification of core iOS files affected that interaction.

Across the years we have used these devices in our experiments:

- iPhone models: 3G, 3GS, 4, 4S, 5 and 6.
- iPad models: 1st, 2nd and 4th generation.

We also used a number of software and hardware tools:

- In the iOS devices themselves: iOS versions 3 through 8; standard UNIX commands to obtain information about files (`lsof`, `ptrace`, `strace`), processes (`ps`), network (`netstat`), etc.; as well as custom binaries and shell scripts, and common iOS applications.
- In external computers: the `nmap` port scanner, `wireshark` network sniffer, Katana Forensics' `Lantern`, the open-source data recovery tool `photorec`, tools from the `iphone-dataprotection` toolkit, custom tools, and the Apple Xcode development framework (which includes the iOS SDK).
- For certain experiments we resorted to the iPad Camera Connection Kit, an accessory sold by Apple.

In general terms, our experiments have been based on the monitoring of a number of elements such as:

1. Alteration of log files.
2. Creation of new files and directories.
3. Creation of special filesystem elements (symbolic links, sockets, FIFOs).
4. Creation of network sockets.
5. Creation of new system processes.
6. Information streams being transferred over the network.

Due to the privilege level needed to properly monitor those indicators, we conduct our research on jailbroken devices.

1.5 Structure of the thesis

The rest of this thesis report is structured as follows. Chapter 2 contains the main contributions of this thesis' work. Chapter 3 presents the main conclusions and the achievements of our research, and provides possible research directions for the future.

Finally, Appendix A contains the Best Paper Award received for our publication [1]; Appendixes B through F contain the publications that constitute the contributions of this thesis; and Appendix G reproduces a related poster that we presented in an international conference.

Chapter 2

List of publications of the thesis

This chapter lists the publications that conform this doctoral thesis. For each publication, the bibliographical reference and the abstract are provided, as well as a reference to the corresponding annex of this thesis containing the full paper.

In sum, this chapter contains:

- two journal papers indexed in ISI-JCR (Q1 and Q3),
- and three conference papers published in IEEE Computer Society proceedings, all of them indexed in SCOPUS.

2.1 Conference paper: Universal, fast method for iPad forensics imaging via USB adapter

L. Gómez-Miralles, J. Arnedo-Moreno. Universal, fast method for iPad forensics imaging via USB adapter. Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011, pp. 200-207. DOI: 10.1109/IMIS.2011.41. **IMIS-2011 Best Paper Award** (attached in Appendix A).

The full text of this paper is available in Appendix B.

Abstract

The Apple iPad is a popular tablet device presented by Apple in early 2010. The idiosyncracies of this new portable device and the kind of data it may store open new opportunities in the field of computer forensics. Given that its design, both internal and external, is very similar to the iPhone, the current easiest way to obtain a forensic image is to install an ssh server and some tools, dump its internal storage and transfer it to a remote host via wireless networking. This approach may require up to 20 hours. In this paper, we present a novel approach that takes advantage of an undocumented feature so it is possible to use a cheap iPad accessory, the Camera Connection Kit, to image the disk to an external hard drive attached via USB connection, greatly reducing the required time.

2.2 Journal paper: Versatile iPad forensic acquisition using the Apple Camera Connection Kit

L. Gómez-Miralles, J. Arnedo-Moreno. Versatile iPad forensic acquisition using the Apple Camera Connection Kit. Computers And Mathematics With Applications, Volume 63, Issue 2, 2012, pp.544-553. DOI: 10.1016/J.CAMWA.2011.09.053. IF: 2.069, Q1: 23/99. Category: COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS (JCR-2012 SE).

The full text of this paper is available in Appendix C.

Abstract

The Apple iPad is a popular tablet device presented by Apple in early 2010. The idiosyncrasies of this new portable device and the kind of data it may store open new opportunities in the field of computer forensics. Given that its design, both internal and external, is very similar to the iPhone, the current easiest way to obtain a forensic image is to install an SSH server and some tools, dump its internal storage and transfer it to a remote host via wireless networking. This approach may require up to 20 hours. In this paper, we present a novel approach that takes advantage of an undocumented feature so it is possible to use a cheap iPad accessory, the Camera Connection Kit, to image the disk to an external hard drive attached via USB connection, greatly reducing the required time.

2.3 Conference paper: Analysis of the forensic traces left by AirPrint in Apple iOS devices

L. Gómez-Miralles, J. Arnedo-Moreno. Analysis of the forensic traces left by AirPrint in Apple iOS devices. Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2013, pp. 703-708. DOI: 10.1109/WAINA.2013.40.

The full text of this paper is available in Appendix D.

Abstract

Since its presentation by Apple, both the iPhone and iPad devices have achieved a great success and gained widespread popularity. This fact, added to the given idiosyncrasies of these new portable devices and the kind of data they may store open new opportunities in the field of computer forensics. In 2010, version 4 of their operating system (iOS) introduced AirPrint, a simple and driverless wireless printing functionality supported by some network printers. This paper presents an analysis of the traces left by AirPrint and assesses whether it is feasible to recover them in the context of a forensic investigation.

2.4 Journal paper: AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices

L. Gómez-Miralles, J. Arnedo-Moreno. **AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices**. Mobile Information Systems, Volume 2015, Article ID 916262, 10 pages, 2015. DOI: 10.1155/2015/916262. IF: 0.949, Q3: 76/133. Category: COMPUTER SCIENCE, INFORMATION SYSTEMS (JCR-2014 SE).

The full text of this paper is available in Appendix E.

Abstract

Since its presentation by Apple, both the iPhone and iPad devices have achieved great success and gained widespread popularity. This fact, added to the given idiosyncrasies of these new portable devices and the kind of data they may store, opens new opportunities in the field of computer forensics. In 2010, version 4 of the iOS operating system introduced AirPrint, a simple and driverless wireless printing functionality supported by hundreds of printer models from all major vendors. This paper describes the traces left in the iOS device when AirPrint is used, and presents a method for recovering content and metadata of documents that have been printed.

2.5 Conference paper: ***Lockup***: A software tool to harden iOS by disabling default ***Lockdown*** services

L. Gómez-Miralles, J. Arnedo-Moreno. **Lockup: A software tool to harden iOS by disabling default *Lockdown* services**. Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015, pp. 718 - 723. DOI: 10.1109/3PGCIC.2015.57.

The full text of this paper is available in Appendix F.

Abstract

Smartphones and mobile devices nowadays accompany each of us in our pockets, holding vast amounts of personal data. The iOS platform has gained popularity in the last years, in particular in enterprise deployments, due to its supposed higher level of security. Recent research has pinpointed a number of mechanisms that are being abused today in order to compromise the security of iOS devices. In this paper, we present *Lockup*, a proof of concept tool that applies various mitigation measures in order to protect iOS devices against those attacks.

Chapter 3

Conclusions and future work

Throughout this thesis we have presented an overview of the field of mobile device forensics in iOS, covering forensic imaging as well as forensic analysis. In addition, we have addressed the topic of anti-forensic techniques and tools.

In each of the three areas we have presented a significant contribution. In the area of forensic imaging, we were able to use a standard adapter sold by Apple to demonstrate that the iPad has full support for attached storage devices and can actually write to them, greatly improving the thee acquisition process. In terms of forensic analysis, we showed that the use of the AirPrint technology for wireless printing leaves a number of traces in the device including a complete copy of the contents being printed. And in the field of anti-forensics, we proposed a model that allows the user to disable a number of system services which are normally used by investigators to retrieve information from the device – a technique that is also being used as an attack in these days.

Throughout this chapter we will now present how the main objectives of this thesis were achieved through the results presented in the corresponding contributions, and we will propose future directions of research.

3.1 Thesis achievements

This section details how the results presented in the different contributions match with the objectives of this thesis.

3.1.1 O1. To improve the iOS forensic acquisition process.

This objective encompasses advances in the field of forensic imaging.

3.1.2 O1.1. To explore the capability of iOS to support the USB mass storage device class protocol.

The lack of USB ports was one of the arguments used by its competitors to criticize the iPad when it was introduced in 2010. However, at that time Apple also introduced an accessory, the iPad Camera Connection Kit, consisting of two adapters that plug into the iPad connector and provide an SD card reader and a USB port.

According to Apple's specifications, the only possible use of the USB adapter is to connect a digital camera using a USB cable in order to import the camera pictures into the iPad. This would require the iPad to support at least USB PTP (*Picture Transfer Protocol*) [52]. We considered, however, that it might make sense if Apple had implemented support for the whole USB MSC (*mass storage device class protocol*), of which PTP is a subset.

In [1] we concluded that iOS, at least in the iPad builds, does implement support for the USB mass storage device class protocol. Gaining full access to the device through the jailbreak technique, and without doing any substantial change, it was possible to mount external hard drives attached via the USB adapter whenever they were formatted as FAT32 or HFS – the native OS X and iOS filesystem.

From the standpoint of forensic data acquisition, this presents the possibility of directly transferring files from the iPad to an external storage unit, eliminating the need for an external computer in the process. It is worth mentioning that, although our first experiments took place under iOS versions 3 and 4, we have verified that this particular finding still works in modern iOS versions.

It is worth mentioning that up until iOS 8 the Camera Connection Kit (and its more recent version *Lightning to USB Camera Adapter*) was an iPad-only accessory, not supposed to work with the iPhone. In our experiments we did not find a way to use it with an iPhone and, in fact, although at least one hack exists that allows for this in jailbroken devices, only basic human-interface devices (keyboards, mice) can be used, but not other more advanced peripherals such as hard drives.

Starting with iOS 9.2, released in December 2015, the iPhone and the iPod Touch gained support for these camera adapters; it is not easy to understand why it took Apple years to make this change, as it should be trivial from a technical point of view and all these devices share similar hardware and capabilities with the iPad. This means that our findings are now applicable in these devices as well.

3.1.3 O1.2. To define an iOS forensic acquisition method through an attached USB device.

At the moment of conducting our research, the existing forensic options for obtaining a whole dump of an iOS device consisted in transferring the information over a Wi-Fi connection to a nearby computer. After testing this with a first-generation iPad, we observed a sustained throughput of 1 MB/s, meaning that it took around 20 hours to transfer a complete dump of our test device (a 64 GB model). As we discused in [2], there were other methods not publicly available which claimed to image 32 GB in 30 minutes, that is 18.6 MB/s.

In [2] we presented a solution that leverages the USB adapter included in the iPad Camera Connection Kit to speed up this process. Our proposal presents the following advantages:

1. It yields a throughput above 29 MB/s. That is 30x faster than the existing wireless solutions such as [19] and nearly twice as fast as other undocumented methods. As detailed in Figure 3.1, we found that reading the iPad storage in 512 KB blocks provided the best results. Using our method, it took 40 minutes to generate complete dumps of our test device, instead of the 20 hours that were needed through Wi-Fi.

2. It does not rely on an external computer acting as forensic workstation, given that the storage media is attached directly to the iPad. Shell-level interaction can be done via SSH using a client installed in the iPad itself or in any other device such a smartphone. It would also be possible to develop a custom iOS application that, running on the iPad, would perform the dump with no further user interaction needed.

3. It will work with any iOS version as long as a jailbreak is available. This requirement, however, applies to virtually any serious attempt to gain control over an iOS device and its internals.

In order to determine the optimal block size we picked a range of possible values and dumped the iPad internal storage (64 GB) three times. As seen in table 3.1, small differences appear even when using the same block size; these are probably caused by background activity, although efforts were made to keep it to a minimum.

In 2012 Apple substituted the traditional 30-pin “dock” connector in iOS devices with the new Lightning connector which is faster, smaller, and reversible; and released the *Lightning to USB camera adapter* to replace the Camera Connection Kit. Later on, a

bs	Pass 1	Pass 2	Pass 3	Avg
32 KB	20,7	20,8	20,8	**20,77**
64 KB	20,7	25,1	25,1	**23,54**
128 KB	29,5	27,6	27,6	**28,22**
256 KB	28,8	28,8	28,8	**28,80**
512 KB	28,8	29,5	29,6	**29,30**
1 MB	29,0	28,9	29,0	**28,97**
2 MB	28,1	28,1	28,1	**28,10**
4 MB	22,9	22,9	23,0	**22,93**
8 MB	18,1	18,1	18,2	**18,13**
16 MB	16,3	16,3	16,3	**16,30**
32 MB	15,5	15,5	15,5	**15,50**

Table 3.1: Test results: throughput (MB/s) obtained with different block sizes.

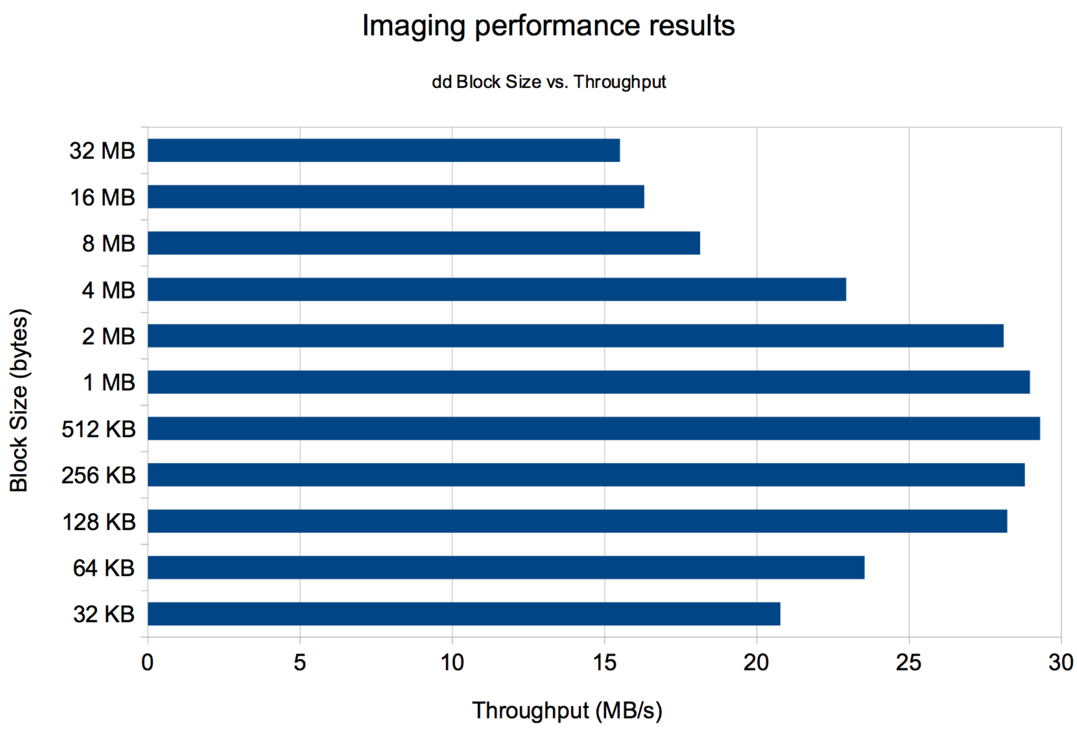


Figure 3.1: How the block size value affects the throughput of our data acquisition method.

new version of the adapter supporting USB 3 was released. Using these new connectors would probably yield even higher transfer rates than those observed in our experiments – something that could also be contributed by the hardware of newer iOS devices.

3.1.4 O2. To analyze new iOS features from a forensic standpoint.

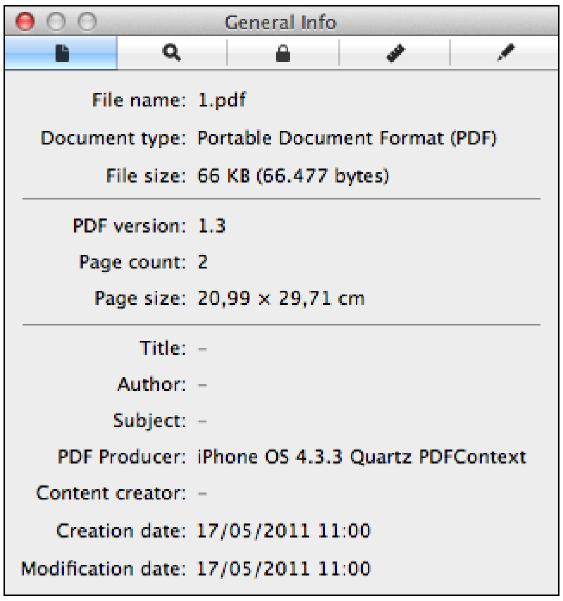
This objective encompasses advances in the field of forensic analysis.

3.1.5 O2.1. To analyze the traces left by the use of AirPrint.

In [3] we experimentally demonstrated that whenever an iOS device sends documents to a printer using the AirPrint protocol, temporary PDF files are generated in the storage area of the iOS device.

We successfully verified that these files are deleted as soon as the printing process finishes, and contain a copy of the whole document being printed (this is usually true even if the user chooses to print only a specified page range) as well as metadata indicating the printing date and time.

In addition, we demonstrated that these files can be recovered in the context of a forensic investigation and can provide valuable evidence to the investigator in cases such as information leaks.



File name:	1.pdf
Document type:	Portable Document Format (PDF)
File size:	66 KB (66.477 bytes)
PDF version:	1.3
Page count:	2
Page size:	20,99 × 29,71 cm
Title:	–
Author:	–
Subject:	–
PDF Producer:	iPhone OS 4.3.3 Quartz PDFContext
Content creator:	–
Creation date:	17/05/2011 11:00
Modification date:	17/05/2011 11:00

FIGURE 3.2: Metadata of a temporary file generated by AirPrint.

After analyzing the metadata of these files we identified that the *PDF Producer* field is common to all of them, which allows us to separate these files from any other PDFs that may exist in the iOS device being analyzed. Figure 3.2 shows an example of the metadata for one of these temporary PDF files.

It is worth noting that a document should leave these traces even if printed from within a secure 'vault' application which may offer additional protections to the user such as data encryption and TouchID authentication: still, whenever the document is sent to the AirPrint subsystem, its contents are mirrored in these temporary PDF files.

3.1.6 O2.2. To create a methodology to extract the traces of AirPrint.

As described in [4], we adapted existing open-source tools for iOS data recovery in order to recover the contents of documents that have been printed through AirPrint.

In particular, after analysing the temporary files generated by AirPrint and determining its type and header, we modified the `iphone-dataprotection` toolkit [27] to make it able to recover deleted PDF documents from iOS devices. In addition we wrote an auxiliary tool that pinpoints those PDFs generated as traces of AirPrint printing, helping the investigator separate these from other regular PDF files that may have existed in the device in the past.

In previous laboratory experiments using devices with no support for hardware-based encryption we printed several batches of ten documents and we constantly obtained success rates above 80%, even reaching 100% in some cases, as described in [3].

In more realistic scenarios with data encryption enabled, our approximation adapting Bédrune and Sigwald's tools was able to recover contents from two documents from a batch of 20; but contrarily to what could be reasonably expected, the contents recovered corresponded to the first two documents in the batch, and not to those at the end of it. We believe this peculiarity may be due to the particular disk allocation strategy employed by the iOS version in use. In any case, it would be interesting to repeat these experiments with other iOS versions to see if the relatively low success rate varies. It is also possible that iOS is actually reusing disk areas as soon as they are freed (and thus overwriting previous temporary files) as part of the data protection strategy: this would make sense as a way of effortlessly wiping previous data for privacy reasons, however the constant reuse of certain storage areas over others might negatively affect the NAND chips that conform it.

3.1.7 O3. To integrate anti-forensic tools for iOS.

This objective encompasses advances in the field of anti-forensic techniques.

3.1.8 O3.1. To explore the possibility of disabling unneeded system services in order to harden the device security while keeping its features usable.

Our work in this area was inspired by Zdziarski [18], who exposed a number of iOS network services –exposed through the `Lockdown` daemon– that constituted an ideal entry point for high-profile attackers targeting the iOS devices of specific individuals to steal information or deploy apps to conduct surveillance on the device owner.

As we published in [5], it is possible to disable any service offered through the iOS `Lockdown` facility. This can be achieved removing the relevant service entry from the `Services.plist` file in iOS 7.

We additionally verified that it is possible to disable unneeded services, such as the network sniffer with no legitimate purpose or the capabilities for MDM synchronization, while keeping the rest of the device features working. Even in the most restrictive modes in which no `Lockdown` services are present at all, other features such as bluetooth connectivity will keep working, allowing the device to interact with audio systems, handsets, etc.

An important implication of the above is that, should a tethered jailbreak be released for the last iOS 7 versions, it would be possible to use it in order to gain full control over the device, apply the desired modifications (add or remove *Lockdown* services), and reboot the device into its original state. This situation would be possible because the configuration resides in a separate file and thus is not affected by the mandatory code signing policy in iOS – we would not be modifying the *Lockdown* binary itself.

In iOS 8 and later versions the configuration file `Services.plist` disappears. We have recently found that this content is instead located at the end of the `Lockdown` binary itself, and we have successfully verified that it is still possible to add and remove services by modifying this part of the binary, without impacting the regular behavior of the system.

A consequence of this observation is that the theoretical strategy of using a tethered jailbreak in order to add or disable services and return the device to its stock, non-jailbroken state, would no longer be possible in the current iOS versions, given that in this case the configuration lies within the binary and thus we cannot modify it without breaking the code signature. Nevertheless, it should still be possible to use this strategy in order to completely disable iOS' *Lockdown* and all the services associated with it.

3.1.9 O3.2. To define a model for hardening iOS by disabling exploitable services, and to implement it in a proof-of-concept software tool.

We observed that many of the standard system services offered by iOS through its *Lockdown* daemon are unneeded for most users – examples include MDM (mobile device management), used in corporate environments to remotely deploy configuration profiles, and over-the-air app installation through iTunes. These services present an expanded attack surface that makes the device more vulnerable to malicious actors.

In [5] we proposed a security model that preserves the user privacy by reducing the device attack surface in the following ways:

1. The most sensitive services (those with absolutely no legitimate use) are disabled. This includes a network sniffer that could be abused by a malicious, surreptitiously installed application, to intercept network traffic in networks where the user has physical access.

2. A number of profiles are defined based on typical use cases; these profiles are increasingly restrictive, meaning that each profile disables a number of services as well as all the services disabled by profiles with lower numbers. Users can choose between those profiles to pick the one that best fits their needs in order to further restrict the number of available services that a potential attacker could target. For instance, few domestic users need to allow the remote installation of software – something that is common in corporate environments with MDM solutions.

3. The services that remain enabled can be restricted to forbid their invocation over-the-air, additionally reducing the attack surface.

4. Finally, the device *pairing records* (the keys that allow other trusted devices, such as computers, to access device data) are purged in a user-defined frequency, hardening the device against the abuse of trusted pairing records (which can be, for instance, surreptitiously copied from another device of the same user).

We implemented these protections in *Lockup*, a proof of concept software tool that runs in jailbroken iOS 7 devices. Our tool not only allows the user to disable exploitable services: it also applies other mitigating measures that contribute to hardening the security of the iOS device. Figure 3.3 summarizes the different components used by *Lockup*.

3.2 Future work

In this section we propose a number of possible research lines to continue our work. As in the rest of this thesis, the information is structured in three blocks: forensic imaging of iOS devices; analysis of the extracted data; and anti-forensic techniques that can contribute to hardening the security of the iOS environment.

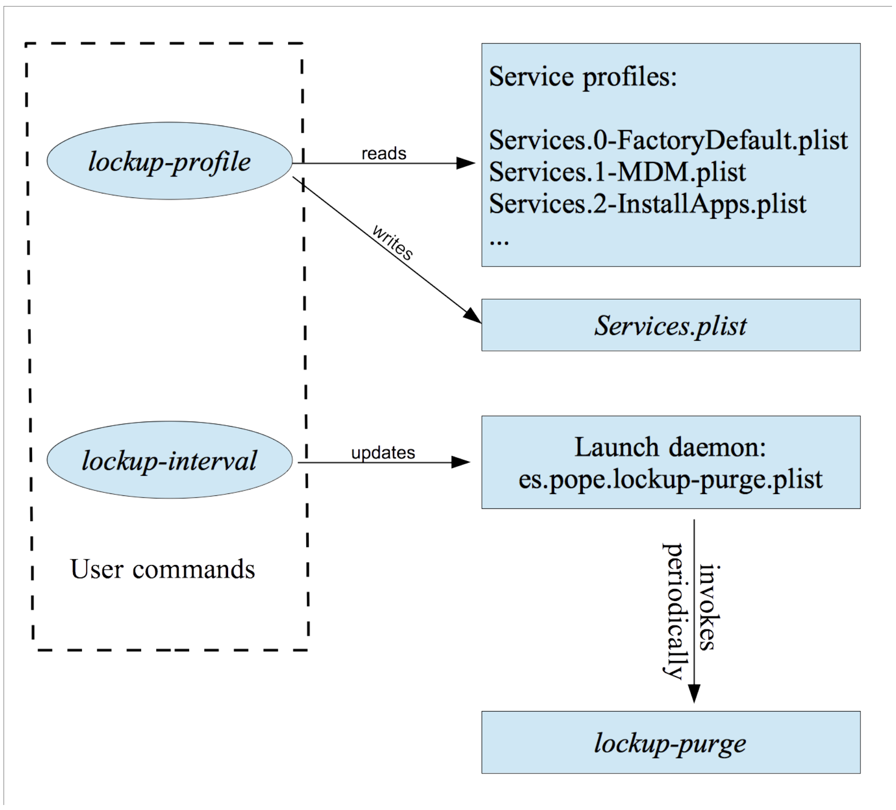


FIGURE 3.3: Basic component architecture of *LockUp*.

In general terms, it would be very positive to get public tethered jailbreaks in modern iOS versions, as it would open the door to new research possibilities.

In the field of forensic imaging, our method based on the Apple Camera Connection Kit could be improved by taking steps to prevent any modifications to the user portion of the iOS storage area, as [19] did in the first versions of iOS by re-mounting the user partition in read-only mode.

It would also be interesting to repeat our experiments with the modern Lightning to USB 3 camera adapter which, together with the hardware of newer iOS devices, is likely to yield even better throughput than that seen in our experiments.

In terms of forensic analysis, we showed that it is possible to recover a number of temporary artifacts, such as those generated by the use of the AirPrint feature. As part of some of our cuurentlly unpublished work, we also found some interesting results about recovering deleted SQLite databases and rollback journals.

Our method works on top of the software tools available at [27], however these tools do not support modern iOS devices due to the lack (so far) of a BootROM exploit, necessary to defeat iOS data encryption and recover deleted files. It would be very desirable to see those tools and exploits upgraded. In early 2016 Apple held a fierce legal battle against the FBI over the court requirement to develop a customized, less-secure iOS version that would allow Law Enforcement agencies to unlock certain devices. This suggests that, as time goes by, it is very unlikely that we see the same kind of tools appear in the future – much less as open source code. Nevertheless, it would be very desirable to see some research published on the matter.

It would also be really interesting to see some works focus on the Apple Pay system and the Secure Enclave Processor (SEP), which holds vital data for Apple Pay and also for the TouchID biometric features.

In terms of anti-forensic techniques we presented *LockUp*, a proof-of-concept software tools that runs on jailbroken devices. It would be ideal to find those mitigation measures integrated into stock iOS versions, but only Apple can do that.

We have been invited to work on an extended version of [5] as a chapter of the book *Security and Privacy in Intelligent Systems and Communication Networks* from the book series *Intelligent Data-Centric Systems*, to be published by Elsevier. At the time of depositing this thesis our work has been submitted and is being evaluated.

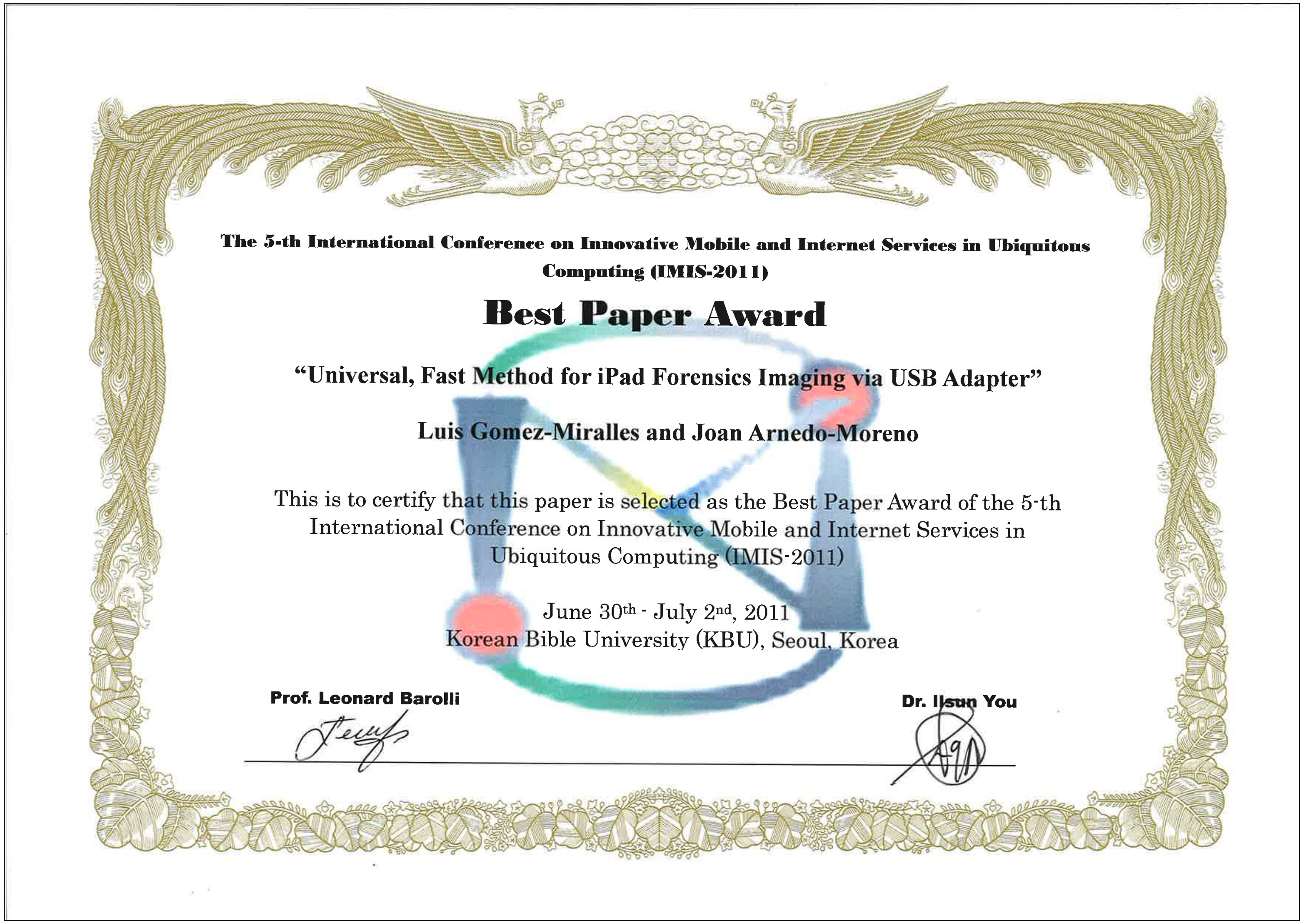
On this same topic, additional research should be undertaken to determine whether it is possible to use a tethered jailbreak in order to temporarily gain full control over the iOS device, add or remove some services, and reboot it into a non-jailbroken state. We believe this should be feasible in iOS 7, and in later versions it should at least be possible to completely remove all services disabling the *Lockdown* daemon by deleting it or rendering its code signature invalid.

To conclude, the discipline of mobile device forensics and even the field of digital forensics as a whole are relatively young and present many interesting and challenging areas for research.

Appendix A

IMIS-2011 Best Paper Award

This appendix contains the Best Paper Award received by the authors at the 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2011) for their paper *Universal, fast method for iPad forensics imaging via USB adapter.*



The 5-th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2011)

Best Paper Award

"Universal, Fast Method for iPad Forensics Imaging via USB Adapter"

Luis Gomez-Miralles and Joan Arnedo-Moreno

This is to certify that this paper is selected as the Best Paper Award of the 5-th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2011)

June 30th - July 2nd, 2011
Korean Bible University (KBU), Seoul, Korea

Prof. Leonard Barolli [illegible]

Dr. Ilsun You [illegible]

Appendix B

Universal, fast method for iPad forensics imaging via USB adapter

L. Gómez-Miralles, J. Arnedo-Moreno. Universal, fast method for iPad forensics imaging via USB adapter. Proceedings of the Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011, pp. 200-207. DOI: 10.1109/IMIS.2011.41. **IMIS-2011 Best Paper Award.**

Universal, fast method for iPad forensics imaging via USB adapter

Luis Gómez-Miralles
Computer forensics and electronic evidence investigator
INCIDE - Investigacion Digital, S.L.
Valencia, Spain
pope@lgomez.es

Joan Arnedo-Moreno
Estudis d'Informàtica, Multimèdia i Telecomunicació
Universitat Oberta de Catalunya
Barcelona, Spain
jarnedo@uoc.edu

Abstract

The Apple iPad is a popular tablet device presented by Apple in early 2010. The idiosyncracies of this new portable device and the kind of data it may store open new opportunities in the field of computer forensics. Given that its design, both internal and external, is very similar to the iPhone, the current easiest way to obtain a forensic image is to install an ssh server and some tools, dump its internal storage and transfer it to a remote host via wireless networking. This approach may require up to 20 hours. In this paper, we present a novel approach that takes advantage of an undocumented feature so it is possible to use a cheap iPad accessory, the Camera Connection Kit, to image the disk to an external hard drive attached via USB connection, greatly reducing the required time.
Keywords: forensics, iPad, cybercrime, digital investigation, Apple.

1. Introduction

Portable devices have become a very important technology in our society, allowing access to computing resources or services in an ubiquitous manner. On that regard, mobile phones have become the clear spearhead, undergoing a great transformation in the last years, slowly becoming small computers that can be conveniently carried in our pockets and managed with one hand. However, as user requirements start including new functionalities beyond those that a mobile phone can realistically offer, advanced portable devices have been developed in order to fulfill them. Such devices try to reach a compromise between a high degree of portability, usability and the ability to provide such advanced functionalities (for example, being able to read or process documents).

The latest contender in the field of embedded portable devices is the Apple iPad, a tablet computer which tries to take advantage of its ancestor's success, the iPhone. It was announced by Apple in January 2010 and launched in the U.S.A. and Europe between April and May 2010. After 80 days in the market, 3 million units had been sold [1]. Given its popularity, it becomes evident that as such devices become widespread, they will also become more common and relevant as sources of evidence from a computer forensics standpoint, providing data about their users. Such data can become very important in cases of crime investigation, where it can be used as evidence in Court or can provide valuable clues to investigators. Since advanced portable devices are usually closed embedded systems with their own idiosyncracies, not actually being fully fledged PCs, forensic data acquisition presents some interesting challenges. That is specially relevant when it is necessary to use non-invasive methods, maintaining the device in the same state (or as similar as possible) as the one it was before the analysis began.

Currently, the easiest method to obtain a forensic image of an iPad device (which can also be basically applied to an iPhone) is to install an ssh server and some tools, retrieve its internal storage contents and transfer the data to a remote host via wireless networking. This approach can take up to 20 hours. In this paper, we present a different approach which relies on a local USB connection with help of a cheap and easily available peripheral, the Camera Connection Kit. This approach greatly reduces the time needed to create a system image. Furthermore, as an additional contribution, the presented keeps a compromise in the amount data which is modified during the acquisition process.

The paper is structured as follows. Section 2 provides an overview of the iPad architecture, focusing on those characteristics specially relevant from a forensic analysis standpoint. In Section 3, a literature review of the current state of iPhone/iPad forensics is presented. The proposed forensic data acquisition method is described in Section 4. Concluding the paper, Section 5 summarizes the paper contributions and outlines further work.

2. iPad architecture overview

From the external point of view, the iPad is basically a big (24x19 cm.) iPhone with a 9.7" screen, providing a resolution of 1024x768. While its internals are very similar to those of its antecesor, the iPad's bigger form factor makes it suitable for longer periods of use, which has motivated the apparition of lots of different applications of every kind. Therefore, the iPad is able to perform tasks perviously reserved to common computers or, up to some point, netbooks.

2.1. Main features

The basic iPad internals are:

- Processor: A custom Apple *A4* ARM processor based on a single-core Cortex-A8, running at a 1 GHz.
- Volatile storage: 256 MB DRAM.
- Non-volatile storage: 16, 32 or 64 GB solid state storage drive.
- Wireless connectivity: 802.11 a/b/g/n and Bluetooth 2.1, the same as every iPhone.
- In addition, the 3G model features an A-GPS (Assisted GPS), and hardware for communicating over UMTS/HSDPA (820, 1900 and 2100 MHz) and GSM/EDGE (850, 900, 1800 and 1900 MHz.

In the process described in this paper, we will use the wireless (802.11) network, and the iPad "Dock" connector, described in the next section.

2.2. Connectors and buttons

The iPad connectors and buttons are very similar to the iPhone's. When placed over the short edge with the round button in the center, we find:

- Top left: a 3.5" jack capable of functioning simultaneously for several audio functionalities.
- Top right: "Lock" button.
- Right edge, near the top: volume and mute controls. In the previous iOS 3 branch, the mute button was a rotation lock switch instead; this function has since been moved to a 'software switch' in the device's graphical interface.
- Bottom center, frontal face: round "Home" button.
- Bottom center, in the edge (below the "Home" button): Apple standard 30-pin "Dock" connector, the same used in every iPhone and most iPods.

.

Figure 1 shows the function of each button. Note that the "Lock" button performs several functions: when the device is off, it will turn it on; when the device is on, a short press will put the device to sleep or wake it form sleep, and a long press will show a dialogue to turn it off. For clarity, in this paper we will keep referring to this button as the "Lock" button. Button configuration is important since, as will be explained in Section 4.1.1, it may be necessary to put the device in *DFU mode* ('Download Firmware Update') in order to setup the device for forensic imaging. When this is needed, installed software usually instructs the user to press a particular combination of these buttons to have the device enter DFU mode.

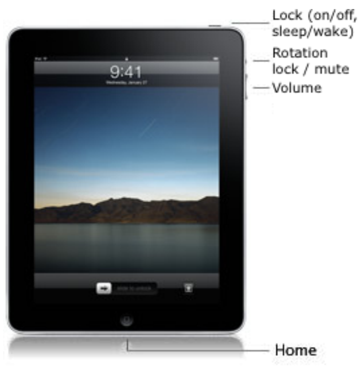


Figure 1. iPad button configuration (iOS 4 and higher).

2.3. Partition scheme

As noted by Zdziarski [2], all devices belonging to the iPhone family contain two partitions:

1) A huge *user data* partition, holding all extra applications installed as well as all the user's data.
2) A small *system* partition containing iOS and the basic applications.

From a forensics standpoint, as far as the user data partition is concerned, some iPad applications which may hold relevant data include enterprise or office software, such as QuickOffice Connect Mobile Suite [3] or Apple's iWork suite [4], [5]. They can all contain text documents or spreadsheets, which are prone to including sensitive or financial information. Although similar applications existed in the iPhone, allowing for direct document editing with no need for an external computer, the iPad's form factor will no doubt boost the existence of documents stored only within the device (and not, for instance, in the suspect's main computer), being edited here and never travelling outside the iPad (with the possible exception of device backups performed by iTunes).

Another possible source of information lies within Apple's *AirPrint* framework released in November 2010 as a feature of iOS 4.2 [6], which provides native printing capabilities to the iPhone and iPad. But long before AirPrint existed, other applications such as PrintCentral [7], already allowed the user to send most document types to a remote printer (connected to a computer with the appropriate server software). These applications' disk caches are likely to hold relevant information such as copies of printed documents.

The system partition contains the base iOS software that comes bundled inside every iOS software update (which explains why they weight hundreds of megabytes). This includes the core operating system and graphical user interface, as well as the standard set of bundled applications such as: Safari, Mail, Calendar, iPod, etc. Note that only the application binaries themselves lie within this partition, whereas the relevant data (for instance, user mail) is stored in the data partition.

3. Current work on IPad forensics

A very basic approach to acquiring user data is connecting the device via the standard USB cable to a computer running iTunes, Apple's multimedia player which is in charge of synchronizing content to the device. Using its AFC protocol (*Apple File Connect*), iTunes syncs existing information (contacts, calendar, email accounts, apps...) and can even retrieve a complete backup of the device; however this presents two problems:

1) The device needs to be correctly paired with the iTunes software in order to sync.
2) Even if the investigator has access to an iTunes backup of the device (say, found in the suspect's main computer), it will not contain unallocated space, from which deleted data can be recovered.

Consequently, more sophisticated methods are required. However, the iPad is distributed as a closed device, meaning that access to its internals is limited and only a those applications approved by Apple may be installed or executed. With these set of restrictions in place, it is extremely difficult to acquire any kind of meaningful forensic data. Fortunately, even though the iPad is a very new device, its internal architecture is very similar to the iPhone's and forensic approaches may be easily ported to the iPad.

On July 6th 2007, just one week after the iPhone was launched, George Hotz announced [8] the existence of a method to get a full, interactive shell. This was the first step towards bypassing Apple's restrictions on their devices, making it possible to execute any program and not only those approved by Apple; a process that has been named **jailbreaking**. In other mobile platforms, such as those running Google's *Android* operating system, a similar process exists which is known as *rooting*. Vendors usually dislike this technique, although in most countries it is legal or at least not definitely illegal. If you ever need to defend this in court, you can do a brief explanation of why jailbreaking the device: to get full access to the system, and thus to the information stored in it, which is crucial in criminal cases which require forensic analysis of these kind of devices.

The jailbreaking process modifies the system partition without alteration of the data partition, which means that it does not alter the user's data, a very important requisite. Even if we assume that some current or future jailbreak methods will modify the user data partition, we can still obtain plenty of useful information, as long as we know what alterations we are responsible for. Ever since their development, the jailbreak tools have been updated to support every new iPhone model and every new iOS version. This method may also be applied to an iPad and, in fact, all the two major forensic approaches in order to recover a complete image from the device are ultimately based on jailbreaking.

The main approach was proposed by Zdziarski [2], who noted that *the iPhone can communicate across several different mediums, including the serial port, 802.11 Wi-Fi, and Bluetooth. Due to the limitations of Bluetooth on the iPhone, the two preferred methods are via the serial port and Wi-Fi*. He proposed a basic method for obtaining a forensic image of the iPhone without tampering the user data partition by jailbreaking the device and using SSH access and the `dd` and `netcat` standard UNIX tools, which by that time had already been ported as a part of the growing iPhone jailbreaking community. Similar methods are explored by Rabaiotti [9] against a Microsoft Xbox. There was not, however, a known, public way to communicate with the device via its serial port, so Zdziarski had to send the forensic image via the device Wi-Fi interface, which is quite slow.

Alternate approaches are provided by some forensics software vendors [10], [11], which have developed solutions that use rather uncommon techniques to get a dump of the solid state storage drive. This is often accomplished by using exploits against more or less known bugs on specific iOS versions in order to execute arbitrary unapproved code, which is actually the same *jailbreakers* do in order to free their devices. However, these vendors do not need to install a complete set of tools in the device. Instead, they tend to upload a tiny, small-footprint software agent which ideally will take control of the system, dump the solid state storage drive through the serial port (dock connector), and will then reboot the device without copying any data to the iPad internal storage.

These methods offers some advantages over the jailbreak approach, being a more straightforward process, simpler to the investigator and leaving little or no footprint on the acquired system. However, it also has some weak points.

First and most important, any propietary method ultimately makes use of an exploit against vulnerabilities of the iOS version of the device, because this is the only way of take such control of the device bypassing every vendor restriction. With every iOS update (usually every few months, downloaded via iTunes), the forensics software must be updated, usually because bugs exploited in previous versions are fixed in the newer version; but even if an exploit still works, exploitation parameters such as memory addresses are very likely to change.

Jansen [12] identified *"the latency in coverage of newly available phone models by forensic tools"* as one of the problems for forensic specialists working with mobile devices. Jailbreaking in the iPad has been moving in a timeframe of barely 1-5 days following iOS updates. We consider very realistic that at some point in the near future, jailbreak updates will be available days or even weeks before some particular forensics software products get the same needed updates.

In addition, many of these proprietary methods are closed and lack any public documentation. Therefore, they are difficult to audit and it cannot be guaranteed that no footprint is actually left on the device. Knowing the process the device is going through, and the precise alterations that this process causes to the device, is a good practice and very important to the forensic investigator.

Therefore, even though some of the proprietary methods may be suitable for the analysis of the device under common circumstances, vendors of such products may fail to release in time an update to support newer iOS versions; they may even not release it at all if, say, the product is discontinued.

Our approach offers what appears to be the best possible throughput, and this is acomplished with a generic UNIX approach via jailbreaking, which is likely to live longer than most iPad forensics software products, thus guaranteeing that it can be applied in future iOS versions.

4. Forensic data acquisition on iPad devices

In this chapter we will describe our method for fast iPad imaging via a USB connection. The method is divided in two general phases: device setup and imaging. Each phase is also divided in several substeps which must be sequentially followed. We will not provide detailed instructions about how to jailbreak an iPad. The description will focus on our technique to recover an image of user data from an already jailbroken iPad.

4.1. Device setup

As mentioned in Section 3, before any forensic analysis may be attempted, a special device setup is required in order to bypass the access restrictions installed by the manufacturer. Once this phase is complete, low level access to the device is actually possible. In addition, it is necessary to install the extra packages needed for our proposed imaging approach.

4.1.1. Jailbreak the device. The actual way to perform the jailbreak varies depending on the iOS version installed on the device. An iPad running iOS 3.2.1 (the initial iOS version preinstalled in most iPads) can be jailbroken by just browsing to `http://www.jailbreakme.com`, a website that exploits a known vulnerability in Safari to take control of the system. The exploits themselves and related documentation can be found at [13]. For a complete, up-to-date chart about jailbreaking tools for each iOS version, refer to [14].

Many jailbreaking tools (`redsn0w`, `PwnageTool`, etc) will require the user to put the device into *DFU mode* with a combination of presses of the "Lock" and "Home" buttons. When this is needed, the software will give the user the necessary instructions.

Should we find a device with a recent iOS version for which no jailbreak procedure exists, it could be acceptable to downgrade to the latest jailbreakable version, although this should be done only as a last resort, and always documenting the steps taken. This would rarely succeed, however, because Apple does not allow to downgrade a device's iOS version after a newer version has been available for some time. There are some workarounds for this but they are not of use in our scenario because they require that we have previously saved some crucial data before installing its present iOS version. Anyway, it is very unlikely that we hit a non-jailbreakable iOS. Take, for instance, iOS 4.2.1 (the first iOS 4 release for the iPad): it was released in November 22 2010, and the appropriate tool for jailbreaking (in that case *redsn0w*) was released the next day [15].

Once a new iOS version has been released, the first jailbreak methods will probably be *tethered*: a tethered jailbreak means that it is only effective as long as the operating system is running. The moment it is rebooted (not when the device is locked), the jailbreak is lost, meaning that two things will happen temporarily until the device is rebooted again into a tethered jailbreak state with the appropriate tool: (1) any jailbreak software installed will not work; and (2) some internal applications (for instance the Safari web browser) may not work, or in the worst case, the whole device might not work at all. We state again that this is only a temporary state, until the device is jailbroken again. It would be acceptable to use a tethered jailbreak for imaging purposes, and in fact part of the tests performed in this paper have taken place over an iPad running iOS 4.2.1, for which tethered jailbreak is the only jailbreak method available at this time.

It is important to note that *jailbreaking* a device does not mean *carrier-unlocking* it. Jailbreak is just a precondition for carrier unlocking. Our proposal needs not perform carrier unlocking, and in fact this is rarely needed in the iPad given that it is usually sold carrier-free.

4.1.2. Charge the battery. It may seem obvious, but it is necessary to have the battery charged to, at least, about 20%. This is because during the imaging process, the iPad's dock connector will be used for USB data transfer, so it will not be possible to plug the device to a power point.

4.1.3. Run Cydia and upgrade available packages. After the device has been jailbroken, a new application labeled *Cydia* [16] will appear in the home screen. This is the software manager that allows installing software not approved by Apple.

When run for the first time, Cydia initializes the device's filesystem and exits. In the next execution, it presents a *Who are you?* prompt, offering three choices; we must choose

'Developer (no filters)', as it offers the widest range of software. Afterwards, if there are available updates to install, it is recommended to perform a 'Complete upgrade'. The device will then restart. We re-open Cydia and, if asked for upgrades, we repeat the process.

4.1.4. Install required software packages. Once Cydia has finished upgrading itself, we use the 'Search' function in Cydia to find and install the following packages:

- *openssh*. This package contains the SSH server that we will use to access the iPad.
- *coreutils*. This package contains the `split` command, which is needed due to reasons that will be exposed later.

The most important tool for this procedure, `dd`, need not be installed, as it is contained inside the essential *coreutils-bin* package, which is installed by default as part of the jailbreaking process.

4.1.5. Network and auto-lock settings. It is necessary to connect the iPad to a wireless network. Another computer in that network will be used to access the iPad via SSH.

Communication between the computer and the iPad will be over SSH, and thus, encrypted. However, we strongly advise to use encryption in the wireless network protocol, and ideally, to use an isolated network for the computer and the iPad only. This is because there is a small window of time in which the device will be accessible with default passwords. There is at least one known worm which penetrates jailbroken iOS devices using these default credentials [17], although nowadays it is nearly impossible to find that code in the wild.

To connect to a wireless network we use the relevant section inside the 'Settings' application. If no wireless network is available, a laptop can be used to create an ad-hoc network and have the iPad join it. The blue button next to the network name reveals the IP address in use (usually acquired via DHCP) and allows the user to manually specify an IP address if needed. The IP address must be noted, as it will be needed later for accessing the iPad from the remote computer.

Still in the 'Settings' application, section 'General', the 'Auto-Lock' option must be set to 'Never'. This will prevent the device from going into sleep mode while the forensic image is being generated, which could interrupt the process. When not in use, the device should be locked (using the Lock button; see section 2) in order to save battery.

We have not tested whether the multitasking capabilities and persistent Wi-Fi in iOS 4 would allow the imaging process to take place while the device is locked. Anyway, given that imaging is a long process that can take more than hour in the biggest devices, we recommend to keep the device awake all the time.

Local access approach. We found at least two ways to apply this method without using a remote computer, although both of them introduce additional complications to the process.

On one hand, it may be possible to install *MobileTerminal* instead of *openssh*, and use the terminal application in the iPad itself to mount the hard drive and image to it. However, at the time of this writing, *MobileTerminal* does not work in iOS versions 4.x, and this software has a history of long delays before being updated to support newer iOS versions.

On the other hand, another approach is to install *openssh* and run an SSH client on the iPad itself. There are many such applications in Apple's App Store, although the fact of keeping this application running during the image generation is likely to alter data and will possibly corrupt the image. Thus, we prefer to use a remote computer and leave the iPad as untouched as possible. Remounting the partition read-only is not a possible solution in this case, as will be explained in Section 4.2.1.

4.2. Device imaging

Once the device is connected to a wireless network, another computer in that same network is used to connect to the iPad via SSH. Using this connection, it is possible to remotely issue commands to the device to initiate the imaging process.

At this point the iPad is accessible via the standard password `alpine`, which works for both the standard `mobile` user as well as for the `root` user, which has full access to the device. The correct way to proceed would be to access the iPad via SSH as the `root` user, and immediately change its password and the password of the `mobile` user account, using the `passwd` command.

4.2.1. Mounting a USB hard drive. In this step we will use Apple's Camera Connection Kit for the iPad [18] in order to access an external USB hard drive. According to Apple, *"the iPad Camera Connection Kit gives you two ways to import photos and videos from a digital camera: using your camera's USB cable or directly from an SD card"* [18]. Thus, it consists of two adapters, one of them being a SD card reader, and the other offering a USB female connector; both of these adapters can plug (one at a time) to the iPad's dock connector, placed in the base, below the "Home" button.

Initial vendor information suggested that the USB adapter only uses the PTP protocol [19] to access the images stored in a camera, and that an actual camera, with its camera-to-USB cable, should be plugged into this connector for the adapter to import the pictures. When this is done, the *Photo* application launches and allows the user to transfer photos and videos from the connected media to the iPad's internal memory.

We have found, however, that the iPad implements the *USB mass storage device class* protocol. Thus, the iPad may mount the disk inserted (regardless of whether it is a hard or solid state storage drive) looking for a `/DCIM` directory as per CIPA DCF standard [20]. If this folder exists, the Photo application will open, allowing the user to import contents; if the folder is not found, the device is unmounted and ignored. We have exploited this undocumented feature to manually mount an external USB hard drive with the appropriate parameters.

As for the filesystems supported, we have been successful in mounting FAT and HFS+ (the standard Macintosh filesystem, which is also the one used for the iPad internal storage). An important issue for Windows users is that their operating system will refuse to format a drive larger than 32 GB as FAT [21], although it can normally mount much bigger FAT partitions and work with them flawlessly. These users will need to use externals tools such as Fat32Format [22]. Mac and Linux users will have no trouble with their standard *Disk Utility* and *mkfs.msdos* tools, respectively.

When we have connected the USB external drive to the iPad (see Figure 2), we can check its presence within the SSH session by running the folowing command:

```
ls /dev/disk1
```

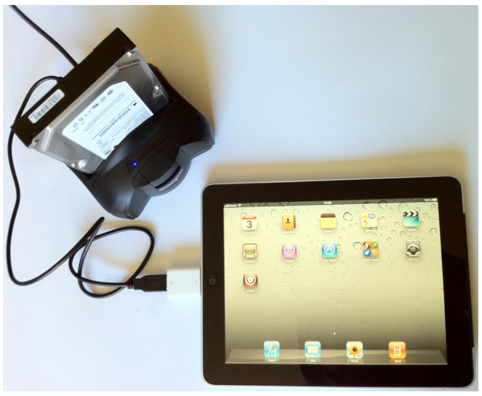


Figure 2. iPad connection to external hard drive via Camera Connection Kit.

The iPad internal storage disk is assigned the node name `/dev/disk0`, so the presence of a `/dev/disk1` implies that the newly connected hard drive has been correctly recognized. If we get an error and there is no `/dev/disk1`, the drive has not been recognized. In our tests, this was usually accompanied by a dialog in the screen complaining that “this device requires too much power”, when trying to connect certain big solid state storage drives and some portable hard drives that take power from USB only. Under iOS version 4 the problem gets bigger because the USB port will no longer emit 100 mA (as it did under iOS 3.x) but only about 20 mA [23]. We found that best results were achieved using a full-size external hard-drive with its own power adapter, or connecting the drive to a powered USB hub.

This command mounts the first partition of the external drive in the `/mnt` directory of the device:

```
mount -t msdos /dev/disk1s1 /mnt
```

We were equally able to mount HFS+ partitions using the `-t hfs` parameter. Due to the Macintosh EFI support, finding the correct partition name for HFS-formatted disks can be tricky. To view the full list of available partitions, we used the command `ls /dev/disk1*`, and we tried to mount all of them until we succeeded.

Zdziarski [2] recommended immediately remounting the data partition in read-only mode (`umount -f /private/var; mount -r /private/var`) prior to beginning the actual imaging. However, in our tests, we found that in both iOS 3 and iOS 4 the system halted if the partition was unmounted; and forcing its remount with `mount -fru` was not supported either.

It must be noted that imaging a mounted partition may alter the integrity of the filesystem contained in the resulting image. In fact we found out that it is possible to end up with images that are unmountable. In order to reduce this risk, no other activity should be taking place in the iPad (neither via the touch screen, nor through the network) while imaging.

Once the disk has been mounted, the command `df -h /mnt` can be used to show its free space and confirm that the drive had been correctly recognized.

4.2.2. Obtaining the forensic image. At this point the working directory was changed to that where the external drive was mounted and the imaging process started with the command:

```
dd if=/dev/rdisk0s2 bs=32M | split -b
4000m - part-
```

The full command can be explained as follows:

- `dd` - The command `dd` is invoked,
- `if=/dev/rdisk0s2` - Taking as Input File (i.e. reading from) the device `rdisk0s2`, which corresponds to the second slice of the iPad’s internal storage, containing the data partition. Due to the partition scheme used in Mac OS and iOS, it is equally acceptable to image `/dev/rdisk0s2s1`.
- `bs=32M` - Using a block size of 32 MB; actually we found that the process works, with similar throughput, for values of 1M and multiples of it.

- `| split` - Instead of writing all these data to a huge file in disk, the data is split.
- `-b 4000m` - Split file size, into smaller files of 4 GB (4000 MB) each.
- `-` This dash means the input content to be split is coming from the previous command, in this case `dd`.
- `part-` - And this is prepended to the name of the output files. The suffix will be two letters, starting with `aa`, as this is the default behavior for the `split` command.

As a result, several 4 GB chunks named `part-aa`, `part-ab`, etc. were generated. Splitting the image in smaller 4 GB files would not be necessary when imaging to a HFS-formatted (Mac) drive.

When finished, the target drive **must** be unmounted before disconnecting it from the iPad. This can be done by either turning the iPad off or unmounting the drive by exiting the `/mnt` folder and running `umount /mnt`.

4.2.3. Reconstructing the image. In order to obtain a full image that can be processed using standard tools, all the fragments must be concatenated. This can be done with a variety of tools in different systems, but a simple command that will probably work in Mac, Linux, and Windows, would be:

```
cat part-* > ipad.dmg
```

The resulting image can then be treated by the methods described in [2] to recover data such as: emails, address book contacts, pictures and videos, Google Maps data, and so on.

As far as image reconstruction is concerned, it must be noted that, starting with iOS version 4, Apple introduced a layer of hardware encryption services [24], which, if activated in the device, will result in partial encryption of the imaged data. Altogether with this, we found a new `protect` option for the `mount` command, which is by default applied to the data partition. We have failed to find any documentation about this parameter, although interestingly enough, the string *protect* also appears inside the Mac OS X `mount` command. Nevertheless, the inclusion of this iOS version into the iPad is still very recent at the time of this writing, so we didn't have much time to experiment with it.

In this scenario, imaging the partition is possible although the resulting image may not be mountable. We think that this is probably due to a layer of encryption, which could be circumvented if the keys are retrieved from the live system after gaining SSH access. Still, carving tools such as Scalpel [25] may be able to recover certain file types.

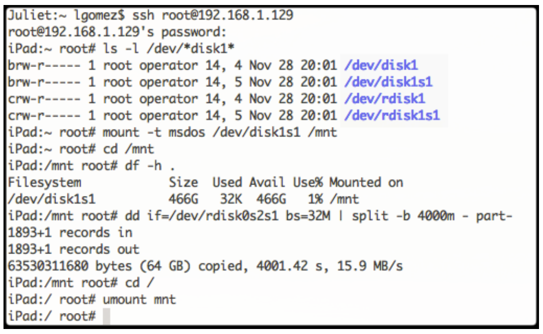
If the device is just passcode-protected, jailbreaking and accessing via SSH is equally possible. The simplest jailbreaking methods working in user-land, such as `jailbreakme.com` will not work given that we are unable to obtain initial access to the device, but other methods (*redsn0w*, *Pwnage Tool*...) could work.

4.3. Performance Results

We performed several experiments measuring the speed of our imaging process proposal via USB connection using the Camera Connection Kit. As can be seen in Figure 3, the process offers a measured throughput of 15.9 MB/s or 0.95 GB/min. This was the highest transfer rate we could achieve, always using common serial-ATA hard drives connected through standard USB-to-SATA adapters. The Figure represents the output of imaging a 64 GB iPad running iOS 4.2.1 to a Seagate ST3500418AS drive.

In comparison, we tested the Zdziarski method over an ad-hoc 802.11n network operating at its maximum theoretical rate (108 Mbps), and we obtained a throughput of barely 1 MB/s, which means that a 16 GB iPad would be imaged in 5 hours and a 64 GB one would require about 20 hours. Our USB approach results in a speed boost of 15x over traditional Wi-Fi imaging.

Forensics software vendors do not seem to release specifications about the imaging times needed by their methods; we could only find that information about Jonathan Zdziarski who states [26] about *"the latest version of the Zdziarski method, which is used in the automated tools available free to law enforcement agencies worldwide"*: *"about 15-30 minutes is all it takes, regardless of whether you're imaging a 4GB iPhone or a 32GB iPhone 3G[s]"*. Assuming he is able to image 32 GB in 'about 30 minutes', we think we have come to the same limit. This is probably the maximum transfer rate of the device's serial port, although it is hard to tell whether this is a physical limit of the port or a software matter that could be improved in future iOS versions.



```
Juliet:~ lgomez$ ssh root@192.168.1.129
root@192.168.1.129's password:
iPad:~ root# ls -l /dev/*disk1*
brw-r----- 1 root operator 14, 4 Nov 28 20:01 /dev/disk1
brw-r----- 1 root operator 14, 5 Nov 28 20:01 /dev/disk1s1
crw-r----- 1 root operator 14, 4 Nov 28 20:01 /dev/rdisk1
crw-r----- 1 root operator 14, 5 Nov 28 20:01 /dev/rdisk1s1
iPad:~ root# mount -t msdos /dev/disk1s1 /mnt
iPad:~ root# cd /mnt
iPad:/mnt root# df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/disk1s1          466G   32K  466G   1% /mnt
iPad:/mnt root# dd if=/dev/rdisk0s2s1 bs=32M | split -b 4000m - part-
1893+1 records in
1893+1 records out
63530311680 bytes (64 GB) copied, 4001.42 s, 15.9 MB/s
iPad:/mnt root# cd /
iPad:/ root# umount mnt
iPad:/ root#
```

Figure 3. Throughput of system imaging a 64 GB iPad.

5. Conclusions and Future Work

In this paper, we have presented a novel approach that takes advantage of a hidden feature in the iPad's USB adapter so it is possible to use a cheap, universally available $30 accessory to image the device directly to a USB drive attached to it. The main contribution of this approach is resulting speed boost to the process, which greatly outpaces existing traditional Wi-Fi approaches, becoming one of the fastest ways to obtain a complete forensic dump of Apple's iPad. In fact, we have apparently reached the speed limit of the iPad's dock connector.

Up to this day, similar transfer rates could only be achieved using commercial tools which are paid and/or restricted to law enforcement agencies, opaque to the scientific community and undocumented. Therefore, it is difficult to assess what is really happening during the imaging process and whether the original data is being somehow altered.

As far as iPad forensics is concerned, a fast imaging method opens some interesting research lines for the future. The ones we find most interesting are live memory dump of the device, study of the iOS 4 encryption system, an autonomous imaging from the iPad to the connected USB drive eliminating the need of a network and a remote computer and analyzing of the forensic artifacts left by the AirPrint subsystem.

Acknowledgments

This work was partially supported by the Spanish MCYT and the FEDER funds under grant TSI2007-65406-C03-03 E-AEGIS and CONSOLIDER CSD2007-00004 "ARES", funded by the Spanish Ministry of Science and Education.

References

[1] InformationWeek, "iPad is top selling tech gadget ever", 2010, http://www.informationweek.com/showArticle.jhtml?articleID=227700347.

[2] Jonathan Zdziarski, *iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*, O'Reilly, 2008.

[3] Quickoffice Inc., "Quickoffice Connect Mobile Suite for iPad on the iTunes App Store", 2010, http://itunes.apple.com/us/app/quickoffice-connect-mobile/id376212724.

[4] Apple Computer Inc, "Pages for iPad on the iTunes App Store", 2010, http://itunes.apple.com/us/app/pages/id361309726.

[5] Apple Computer Inc, "Numbers for iPad on the iTunes App Store", 2010, http://itunes.apple.com/us/app/numbers/id361304891.

[6] Apple Computer Inc., "Apple's AirPrint Wireless Printing for iPad, iPhone and iPod touch Coming to Users in November", 2010, http://www.apple.com/pr/library/2010/09/15airprint.html.

[7] EuroSmartz Ltd., "PrintCentral for iPad on the iTunes App Store", 2010, http://itunes.apple.com/us/app/printcentral-for-ipad/id366020849.

[8] George Hotz, "iPhone serial hacked, full interactive shell", 2007, http://www.hackint0sh.org/f127/1408.htm.

[9] J.R. Rabaiotti and C.J. Hargreaves, "Using a software exploit to image RAM on an embedded system ", *Digital Investigation*, vol. 6, pp. 95–103, 2010.

[10] Katana Forensics, "Lantern", http://katanaforensics.com/use-our-tools/lantern/.

[11] Forensic Telecommunications Services Ltd., "iXAM - Advanced iPhone Forensics Imaging Software", http://www.ixam-forensics.com/.

[12] Moenner L. Jansen W, Delaitre A, "Overcoming impediments to cell phone forensics", in *In Proceedings of the 41st Annual Hawaii International Conference on System Sciences*. 2008, pp. 483 – 483, IEEE CSP.

[13] Comex, "Comex 'star' GIT repository", 2010, http://github.com/comex/star.

[14] "Jailbreak Matrix", 2010, http://www.jailbreakmatrix.com/iPhone-iTouch-Jailbreak.

[15] iPhone Dev Team, "Thanksgiving with Apple", 2010, http://blog.iphone-dev.org/post/1652053923/thanksgiving-with-apple.

[16] Jay Freeman 'Saurik', "Bringing Debian APT to the iPhone", 2008, http://www.saurik.com/id/1.

[17] Sophos Security, "First iPhone worm discovered - ikee changes wallpaper to Rick Astley photo", 2009, http://nakedsecurity.sophos.com/2009/11/08/iphone-worm-discovered-wallpaper-rick-astley-photo/.

[18] Apple Computer Inc, "Apple iPad Camera Connection Kit", 2010, http://store.apple.com/us/product/MC531ZM/A.

[19] International Organization for Standarization (ISO), "ISO 15740:2008 – Electronic still picture imaging – Picture transfer protocol (PTP) for digital still photography devices ", 2008.

[20] Camera & Imaging Products Association, "Design rule for Camera File system: DCF version 2.0", 2010.

[21] Microsoft Corp, "Limitations of the FAT32 File System in Windows XP", 2007, http://support.microsoft.com/kb/314463/.

[22] Ridgecrop Consultants Ltd., "Fat32Format", 2009, http://www.ridgecrop.demon.co.uk/index.htm?fat32format.htm.

[23] 9to5 Mac, "iOS 4.2 emits less USB power on iPad, Camera Connection Kit crippled?", 2010, http://www.9to5mac.com/40091/ios-4-2-emits-less-usb-power-on-ipad-camera-connection-kit-crippled.

[24] Apple Computer Inc, "iOS 4: Understanding data protection", 2010, http://support.apple.com/kb/HT4175.

[25] LLC Digital Forensics Solutions, "Scalpel: A Frugal, High Performance File Carver", 2006, http://www.digitalforensicssolutions.com/Scalpel/.

[26] Jonathan Zdziarski, "iPhone Insecurity", 2010, http://www.iphoneinsecurity.com/.

Appendix C

Versatile iPad forensic acquisition using the Apple Camera Connection Kit

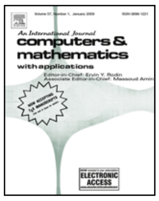
L. Gómez-Miralles, J. Arnedo-Moreno. Versatile iPad forensic acquisition using the Apple Camera Connection Kit. Computers And Mathematics With Applications, Volume 63, Issue 2, 2012, pp.544-553. DOI: 10.1016/J.CAMWA.2011.09.053. IF: 2.069, Q1: 23/99. Category: COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS (JCR-2012 SE).

Computers and Mathematics with Applications 63 (2012) 544–553

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa



Versatile iPad forensic acquisition using the Apple Camera Connection Kit

Luis Gómez-Miralles, Joan Arnedo-Moreno [*]

Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya, Carrer Roc Boronat 117, 08018 Barcelona, Spain

A R T I C L E I N F O

Keywords:
Forensics
IPad
Cybercrime
Digital investigation
Apple

A B S T R A C T

The Apple iPad is a popular tablet device presented by Apple in early 2010. The idiosyncracies of this new portable device and the kind of data it may store open new opportunities in the field of computer forensics. Given that its design, both internal and external, is very similar to the iPhone, the current easiest way to obtain a forensic image is to install an SSH server and some tools, dump its internal storage and transfer it to a remote host via wireless networking. This approach may require up to 20 hours. In this paper, we present a novel approach that takes advantage of an undocumented feature so that it is possible to use a cheap iPad accessory, the Camera Connection Kit, to image the disk to an external hard drive attached via USB connection, greatly reducing the required time.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

Portable devices have become a very important technology in society, allowing access to computing resources or services in an ubiquitous manner. In this regard, mobile phones have become the clear spearhead, undergoing a great transformation in recent years, slowly becoming small computers that can be conveniently carried in our pockets and managed with one hand. However, as user requirements started to include new functionalities beyond those that a mobile phone can realistically offer, advanced portable devices have been developed in order to fulfill them. Such devices try to reach a compromise between a high degree of portability, usability and the ability to provide such advanced functionalities (for example, being able to read or process documents).

One of the top devices in the field of embedded portable devices is the Apple iPad, a tablet computer which tries to take advantage of the success of its ancestor, the iPhone. It was announced by Apple in January 2010 and launched in the USA and Europe between April and May 2010. After 80 days in the market, 3 million units had been sold [1]. Given its popularity, it becomes evident that as such devices become widespread, they will also become more common and relevant as sources of evidence from a computer forensic standpoint, providing data about their users. This kind of data can become very important in cases of criminal investigation, where it can be used as evidence in court or provide valuable clues to investigators. Since advanced portable devices are usually closed embedded systems with their own idiosyncracies, not actually being full-fledged PCs, forensic data acquisition presents some interesting challenges. This is especially relevant when it is necessary to use non-invasive methods, maintaining the device in the same state (or as similar as possible) as the one it was in before the analysis began.

Currently, the easiest method to obtain a forensic image of an iPad device (which can also basically be applied to an iPhone) is to install an SSH server and some tools, retrieve its internal storage contents and transfer the data to a remote host via wireless networking. Unfortunately, this is very slow, and can take up to 20 h. More efficient methods exist, but they

[*] Corresponding author.
E-mail addresses: pope@uoc.edu (L. Gómez-Miralles), jarnedo@uoc.edu (J. Arnedo-Moreno).

0898-1221/$ – see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2011.09.053

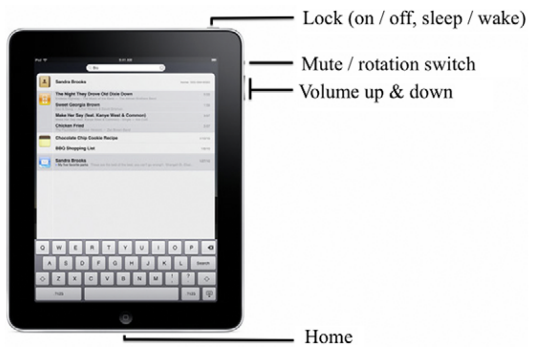


Fig. 1. iPad button configuration (iOS 4 and higher).

are based on closed software, which relies on obscure or undisclosed techniques and is very dependent on the iOS version. As soon as an iOS upgrade becomes available, they can no longer be used until a new version is created.

In this paper, we propose a versatile approach which reaches a compromise between forensic data acquistion speed and an open approach, without the need of vendor specific software or a dependency on a very specific iOS version. Achieved with the help of a cheap and easily available peripheral, the Camera Connection Kit, it is possible to generate a forensic image using an USB connection. Furthermore, this approach still minimizes the amount of data which is modified during the acquisition process.

The paper is structured as follows. Section 2 provides an overview of the iPad architecture, focusing on those characteristics especially relevant from a forensic analysis standpoint. In Section 3, a literature review of the current state of iPhone/iPad forensic imaging techniques is presented. The proposed forensic data acquisition method is described in Section 4. Section 5 provides a brief discussion about which forensic techniques can be applied on the image generated using our method, and, in Section 6, an analysis of its performance is presented. Concluding the paper, Section 7 summarizes the paper's contributions and outlines further work.

2. iPad architecture overview

From the external point of view, the iPad is basically a big (24 × 19 cm.) iPhone with a 9.7″ screen, providing a resolution of 1024 × 768. While its internals are very similar to those of its ancestor, the iPad's larger form factor makes it suitable for longer periods of use, which has motivated the appearance of many different applications of all kinds. Therefore, the iPad is able to perform tasks previously reserved for common computers or, up to some point, netbooks.

2.1. Main features

The basic iPad internals are:

- Processor: A custom Apple *A*4 ARM processor based on a single-core Cortex-A8, running at 1 GHz.
- Volatile storage: 256 MB DRAM.
- Non-volatile storage: 16, 32 or 64 GB solid state storage drive.
- Wireless connectivity: 802.11 a/b/g/n and Bluetooth 2.1, the same as every iPhone.
- In addition, the 3G model features an A-GPS (Assisted GPS), and hardware for communicating over UMTS/HSDPA (820, 1900 and 2100 MHz) and GSM/EDGE (850, 900, 1800 and 1900 MHz).

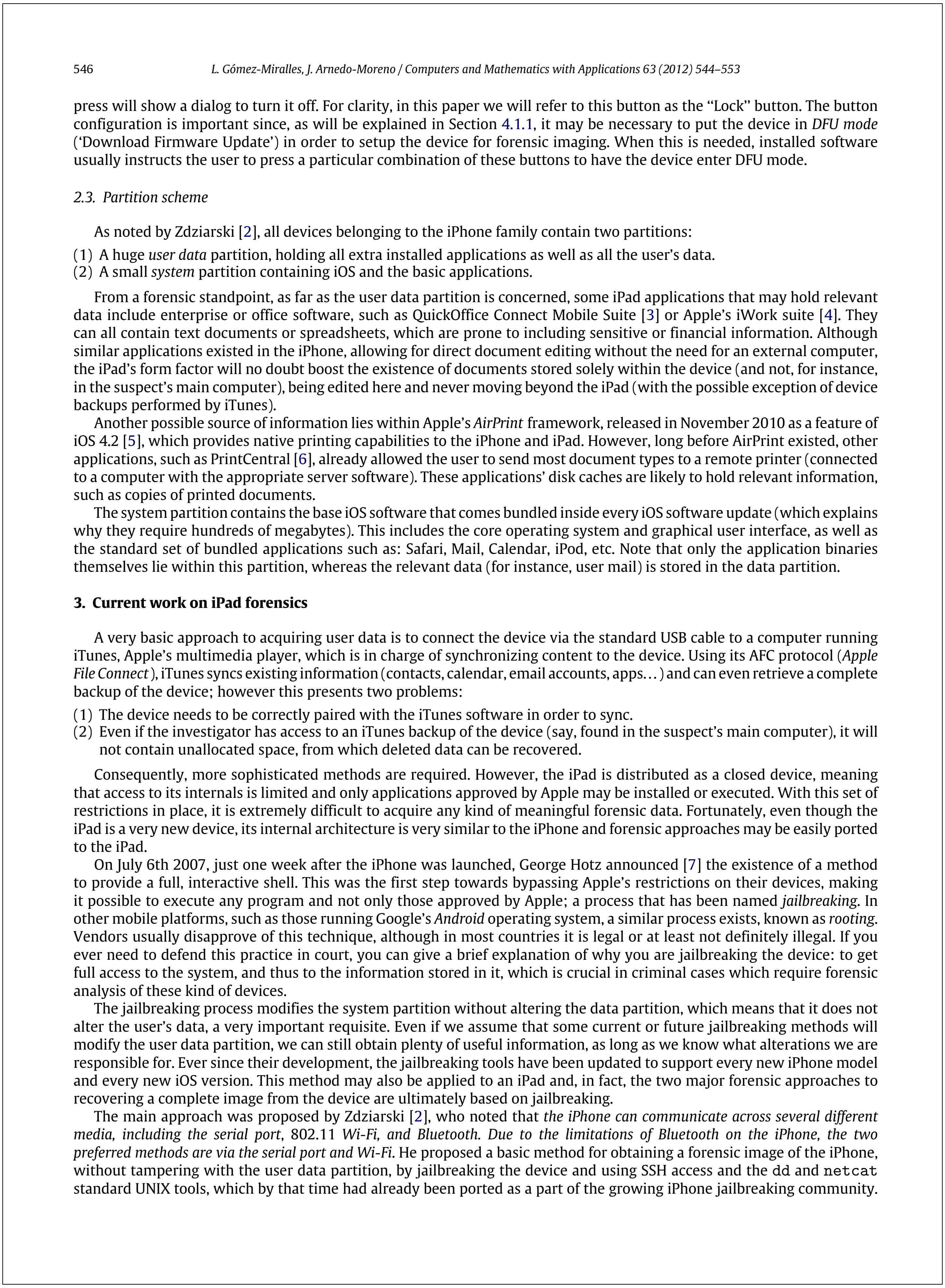
A second generation hit the market in March 2011, featuring a dual-core Apple *A*5 processor and 512 MB of RAM.

2.2. Connectors and buttons

The iPad connectors and buttons are very similar to the iPhone's. When placed over the short edge with the round button in the center, we find:

- Top left: a 3.5″ jack capable of functioning simultaneously for several audio functionalities.
- Top right: "Lock" button.
- Right edge, near the top: volume controls, and one configurable switch which can either mute the device, or lock the screen orientation.
- Bottom center, front face: round "Home" button.
- Bottom center, in the edge (below the "Home" button): Apple standard 30-pin "Dock" connector, the same as used in every iPhone and most iPods. This is also the port used when referring to the iPhone or iPad's 'serial port'.

Fig. 1 shows the function of each button. Note that the "Lock" button performs several functions: when the device is off, it will turn it on; when the device is on, a short press will put the device to sleep or wake it from sleep and a long



press will show a dialog to turn it off. For clarity, in this paper we will refer to this button as the "Lock" button. The button configuration is important since, as will be explained in Section 4.1.1, it may be necessary to put the device in *DFU mode* ('Download Firmware Update') in order to setup the device for forensic imaging. When this is needed, installed software usually instructs the user to press a particular combination of these buttons to have the device enter DFU mode.

2.3. Partition scheme

As noted by Zdziarski [2], all devices belonging to the iPhone family contain two partitions:

(1) A huge *user data* partition, holding all extra installed applications as well as all the user's data.
(2) A small *system* partition containing iOS and the basic applications.

From a forensic standpoint, as far as the user data partition is concerned, some iPad applications that may hold relevant data include enterprise or office software, such as QuickOffice Connect Mobile Suite [3] or Apple's iWork suite [4]. They can all contain text documents or spreadsheets, which are prone to including sensitive or financial information. Although similar applications existed in the iPhone, allowing for direct document editing without the need for an external computer, the iPad's form factor will no doubt boost the existence of documents stored solely within the device (and not, for instance, in the suspect's main computer), being edited here and never moving beyond the iPad (with the possible exception of device backups performed by iTunes).

Another possible source of information lies within Apple's *AirPrint* framework, released in November 2010 as a feature of iOS 4.2 [5], which provides native printing capabilities to the iPhone and iPad. However, long before AirPrint existed, other applications, such as PrintCentral [6], already allowed the user to send most document types to a remote printer (connected to a computer with the appropriate server software). These applications' disk caches are likely to hold relevant information, such as copies of printed documents.

The system partition contains the base iOS software that comes bundled inside every iOS software update (which explains why they require hundreds of megabytes). This includes the core operating system and graphical user interface, as well as the standard set of bundled applications such as: Safari, Mail, Calendar, iPod, etc. Note that only the application binaries themselves lie within this partition, whereas the relevant data (for instance, user mail) is stored in the data partition.

3. Current work on iPad forensics

A very basic approach to acquiring user data is to connect the device via the standard USB cable to a computer running iTunes, Apple's multimedia player, which is in charge of synchronizing content to the device. Using its AFC protocol (*Apple File Connect*), iTunes syncs existing information (contacts, calendar, email accounts, apps...) and can even retrieve a complete backup of the device; however this presents two problems:

(1) The device needs to be correctly paired with the iTunes software in order to sync.
(2) Even if the investigator has access to an iTunes backup of the device (say, found in the suspect's main computer), it will not contain unallocated space, from which deleted data can be recovered.

Consequently, more sophisticated methods are required. However, the iPad is distributed as a closed device, meaning that access to its internals is limited and only applications approved by Apple may be installed or executed. With this set of restrictions in place, it is extremely difficult to acquire any kind of meaningful forensic data. Fortunately, even though the iPad is a very new device, its internal architecture is very similar to the iPhone and forensic approaches may be easily ported to the iPad.

On July 6th 2007, just one week after the iPhone was launched, George Hotz announced [7] the existence of a method to provide a full, interactive shell. This was the first step towards bypassing Apple's restrictions on their devices, making it possible to execute any program and not only those approved by Apple; a process that has been named *jailbreaking*. In other mobile platforms, such as those running Google's *Android* operating system, a similar process exists, known as *rooting*. Vendors usually disapprove of this technique, although in most countries it is legal or at least not definitely illegal. If you ever need to defend this practice in court, you can give a brief explanation of why you are jailbreaking the device: to get full access to the system, and thus to the information stored in it, which is crucial in criminal cases which require forensic analysis of these kind of devices.

The jailbreaking process modifies the system partition without altering the data partition, which means that it does not alter the user's data, a very important requisite. Even if we assume that some current or future jailbreaking methods will modify the user data partition, we can still obtain plenty of useful information, as long as we know what alterations we are responsible for. Ever since their development, the jailbreaking tools have been updated to support every new iPhone model and every new iOS version. This method may also be applied to an iPad and, in fact, the two major forensic approaches to recovering a complete image from the device are ultimately based on jailbreaking.

The main approach was proposed by Zdziarski [2], who noted that *the iPhone can communicate across several different media, including the serial port*, 802.11 *Wi-Fi, and Bluetooth. Due to the limitations of Bluetooth on the iPhone, the two preferred methods are via the serial port and Wi-Fi*. He proposed a basic method for obtaining a forensic image of the iPhone, without tampering with the user data partition, by jailbreaking the device and using SSH access and the `dd` and `netcat` standard UNIX tools, which by that time had already been ported as a part of the growing iPhone jailbreaking community.

Similar methods are explored by Rabaiotti [8] against a Microsoft Xbox. There was not, however, a known, public way to communicate with the device via its serial port, so Zdziarski had to send the forensic image via the device Wi-Fi interface, which is quite slow.

Alternate approaches are provided by some forensics software vendors [9,10], who have developed solutions that use rather uncommon techniques to get a dump of the solid state storage drive. This is often accomplished by using exploits against more or less known bugs in specific iOS versions in order to execute arbitrary unapproved code, which is actually the same as *jailbreakers* do in order to free their devices. However, these vendors do not need to install a complete set of tools in the device. Instead, they tend to upload a tiny, small-footprint software agent which will ideally take control of the system, dump the solid state storage drive through the serial port (dock connector), and then reboot the device without copying any data to the iPad internal storage. This method offers some advantages over the jailbreaking approach, being a more straightforward process, simpler for the investigator and leaving little or no footprint on the acquired system. However, it also has some weak points.

First and foremost, any proprietary method ultimately makes use of an exploit against the vulnerabilities of the iOS version of the device, because this is the only way to take such control of the device, bypassing every vendor restriction. With every iOS update (usually every few months, downloaded via iTunes), the forensics software must be updated, usually because bugs exploited in previous versions are fixed in the newer version; but even if an exploit still works, exploitation parameters such as memory addresses are very likely to change.

Jansen [11] identified *"the latency in coverage of newly available phone models by forensic tools"* as one of the problems for forensic specialists working with mobile devices. Jailbreaking in the iPad has been moving in a timeframe of barely 1–5 days following iOS updates. We consider it very realistic that at some point in the near future, jailbreak updates will be available days or even weeks before some particular forensics software products get the same needed updates. Therefore, even though some of the proprietary methods may be suitable for the analysis of the device under common circumstances, vendors of such products may fail to release an update in time to support newer iOS versions; they may even not release it at all if, say, the product is discontinued.

In addition, many of these proprietary methods are closed and lack any public documentation. Therefore, they are difficult to audit and it cannot be guaranteed that no footprint is actually left on the device. Knowing the process the device is going through, and the precise alterations that this process causes to the device, is good practice and very important to the forensic investigator.

4. Versatile forensic data acquisition

In this section we will describe the proposed method for iPad imaging via the Camera Connection Kit through the USB connection. The method is divided into two general phases: device setup and imaging. Each phase is also divided into several substeps which must be sequentially followed.

4.1. Device setup

This phase encompasses all the necessary steps to leave the device open and ready for forensic image acquisition. As mentioned in Section 3, before any forensic analysis may be attempted, a special device setup is required in order to bypass the access restrictions installed by the manufacturer. Once this phase is complete, low-level access to the device is actually possible. In addition, it is necessary to install the extra packages needed for our proposed imaging approach.

4.1.1. Jailbreak the device

The only requisite in order to apply our method is starting with a jailbroken iPad. We will not provide detailed instructions about how to jailbreak an iPad, just the basic guidelines. Once this prerequisite is met, our method can be applied on any iPad, iOS version notwithstanding. This is in contrast with vendor specific tools, as explained in Section 3.

The actual way to perform jailbreaking varies depending on the iOS version installed on the device. An iPad running iOS 3.2.1 can be jailbroken just by browsing the website http://www.jailbreakme.com, which exploits a known vulnerability in Safari to take control of the system. The exploits themselves and related documentation can be found at [12]. For a complete, up-to-date chart about jailbreaking tools for each iOS version, refer to [13].

Should we find a device with a recent iOS version for which no jailbreak procedure exists, it could be acceptable to downgrade to the latest jailbreakable version, although this should be done only as a last resort, and always documenting the steps taken. This rarely succeeds, however, because Apple does not allow one to downgrade a device's iOS version after a newer version has been available for some time. There are some workarounds for this but they are of no use in our scenario because they require that we have previously saved some crucial data before installing the present iOS version. Anyway, it is very unlikely that we will hit a non-jailbreakable iOS. Take, for instance, iOS 4.2.1 (the first iOS 4 release for the iPad): it was released on November 22 2010, and the appropriate tool for jailbreaking (in that case *redsn0w*) was released the next day [14]. Similar time windows apply for the different 4.3.x iOS versions released during the first months of 2011.

Once a new iOS version has been released, the first jailbreaking methods will probably be *tethered*: a tethered jailbreak means that it is only effective as long as the operating system is running. The moment it is rebooted (not when the device is locked), the jailbreak is lost, meaning that two things will happen temporarily until the device is rebooted again into a tethered jailbreak state with the appropriate tool: (1) any jailbreak software installed will not work; and (2) some internal applications (for instance the Safari web browser) may not work, or in the worst case, the whole device might not work at all. We state again that this is only a temporary state, until the device is jailbroken again. It would be acceptable to use a tethered jailbreak for imaging purposes, and in fact part of the tests performed in this paper have taken place on an iPad with a tethered jailbreak.

Many jailbreaking tools (`redsn0w`, `PwnageTool`, etc.) will require the user to put the device into *DFU mode* with a combination of presses of the "Lock" and "Home" buttons. When this is needed, the software will give the user the necessary instructions.

It also is important to note that *jailbreaking* a device does not mean *carrier unlocking* it. Jailbreak is just a precondition for carrier unlocking. Our proposal need not perform carrier unlocking, and in fact this is rarely needed in the iPad given that it is usually sold carrier-free.

4.1.2. Install required software packages

After the device has been jailbroken, a new application labeled *Cydia* [15] appears in the home screen. In the case that this application already exists, the previous setup is not necessary. This is the software manager that allows the installation of software not approved by Apple.

When run for the first time, Cydia initializes the device's filesystem and exits. In the next execution, it presents a *Who are you*? prompt, offering three choices; 'Developer (no filters)' must be chosen, as it offers the widest range of software. Afterwards, if there are available updates to install, it is recommended to perform a 'Complete upgrade'. The device will then restart. It is necessary to re-open Cydia and, if asked for upgrades, repeat the process.

Once Cydia has finished upgrading itself, using the 'Search' function, the following packages are installed:

- *openssh*. This package contains the SSH server that will be used to access the device.
- *coreutils*. This package contains the `split` command, which is needed due to reasons that will be explained later.

The most important tool for this procedure, `dd`, need not be installed, as it is contained inside the essential *coreutils-bin* package, which is installed by default as part of the jailbreaking process.

4.1.3. Network and auto-lock settings

It is necessary to connect the device to a wireless network, since during the process it will be remotely accessed from another computer. This connection is necessary to issue commands, but not for the actual image data transfer. If no wireless network is available, a laptop can be used to create an ad-hoc network and have the iPad join it.

Communication between the computer and the iPad will be over SSH, and thus, encrypted. However, it is strongly advised to use encryption in the wireless network protocol, and ideally, to use an isolated network for the computer and the iPad only. The main reason for this is the fact that there is a small window of time during which the device will be accessible with default passwords. There is at least one known worm which penetrates jailbroken iOS devices using these default credentials [16], although nowadays it is almost impossible to find that code in the wild.

Once the connection has been established, the iPad 'Settings' application reveals the IP address in use (usually acquired via DHCP) and allows the user to manually specify an IP address if needed. The IP address must be noted down, as it will be needed later for accessing the iPad from the remote computer.

Still in the 'Settings' application, the 'Auto-Lock' option must be set to 'Never'. This will prevent the device from going into sleep mode while the forensic image is being generated, which could interrupt the process. When not in use, the device should be locked (using the Lock button; see Section 2) in order to save battery.

We have not tested whether the multitasking capabilities and persistent Wi-Fi in iOS 4 would allow the imaging process to take place while the device is locked. Anyway, given that imaging is a process that can take a long time in devices with the largest local drives, it is recommended to keep the device awake all the time.

Local access approach

We would like to note that, even though the best way to apply our proposal is by issuing commands via SSH, we experimented and also found at least two ways to apply the imaging method without actually using a remote computer. However, both of them introduce additional complications to the process.

On one hand, it may be possible to install *MobileTerminal* instead of *openssh*, and use the terminal application in the iPad itself to mount the hard drive and image to it. However, at the time of writing, *MobileTerminal* does not work in iOS versions 4.x, and this software has a history of long delays before being updated to support newer iOS versions.

On the other hand, another approach is to install *openssh* and run an SSH client on the iPad itself. There are many such applications in Apple's App Store, although keeping this application running during the image generation is likely to alter data and will possibly corrupt the image. Thus, it is preferable to use a remote computer and leave the iPad as untouched as possible. Setting the user data partition read-only is not a possible solution in this case, as will be explained in Section 4.2.1.

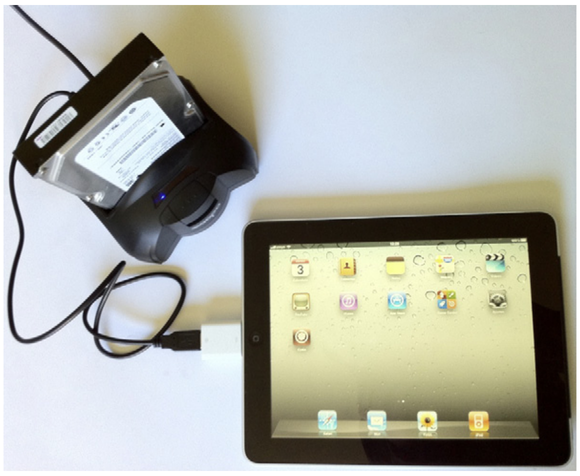


Fig. 2. iPad connection to an external hard drive via Camera Connection Kit.

4.2. Device imaging

Before the process can actually begin, it is necessary to have the battery charged to, at least, about 20%. This is because, during the imaging process, the iPad's dock connector will be used for USB data transfer, so it will not be possible to plug the device to a power point.

With the device connected to a wireless network, another computer is used to establish an SSH session with the iPad. At this point, the device is accessible via the standard password `alpine`, which works for both the standard `mobile` user as well as for the `root` user, which has full access to the device. The correct way to proceed would be to access the iPad via SSH as the `root` user, and immediately change its password and the password of the `mobile` user account, using the `passwd` command.

4.2.1. Mounting a USB hard drive

In this step, the Apple peripheral, Camera Connection Kit [17], is used in order to access an external USB hard drive. According to Apple, "*the iPad Camera Connection Kit gives you two ways to import photos and videos from a digital camera: using your camera's USB cable or directly from an SD card*" [17]. Thus, it consists of two adapters, one of them being an SD card reader and the other offering a USB female connector; both of these adapters can plug (one at a time) into the iPad's dock connector, located in the base, below the "Home" button.

Initial vendor information suggested that the USB adapter only uses the PTP [18] protocol to access the images stored in a camera, and that an actual camera, with its camera-to-USB cable, should be plugged into this connector for the adapter to import the pictures. This is the normal use for this adapter, and is what Apple advertises it for. When this is done, the *Photo* application launches and allows the user to transfer photos and videos from the connected media to the iPad's internal memory.

Our results show that iOS implements the much wider *USB mass storage device class* protocol, of which PTP is a subset. PTP allows basic operations on the items of a video device, usually a photo camera, in which FAT-formatted media store the files under the `/DCIM` directory [19]). Instead, the iPad can mount and make full use of FAT and HFS filesystems.

The device just emulates the behavior of PTP: it automatically mounts FAT drives looking for the `/DCIM`. If this folder exists, the *Photo* application will open, allowing the user to import contents; if it does not exist, the device is unmounted and ignored. In our approach, we exploit this undocumented feature to manually mount an external USB hard drive with the appropriate parameters, and use it to obtain an image of the iPad internal storage.

After connecting the USB external drive to the iPad (see Fig. 2), we can check its presence within the SSH session by looking for the `/dev/disk1` device. The iPad internal storage disk is assigned the device name `/dev/disk0`, so the presence of such a device implies that the newly connected hard drive has been correctly recognized.

Should no `/dev/disk1` device exist, the drive has not been recognized. In our tests, this was usually accompanied by a dialog in the screen complaining that "this device requires too much power", when trying to connect certain large solid state storage drives and some portable hard drives that take power from USB only. Under iOS version 4 it is more problematic, since the USB port will no longer emit 100 mA (as it did under iOS 3.x) but only about 20 mA [20]. Therefore, in our tests, we found that the best results were achieved using a full-size external hard drive with its own power adapter, or connecting the drive to a powered USB hub.

As for the supported filesystems, we have been successful in mounting FAT and HFS + (the standard Macintosh filesystem, which is also the one used for the iPad internal storage). Due to the Macintosh EFI support, finding the correct partition name for disks with EFI partitioning can be tricky. It is not always named `/dev/disk1`. In that scenario, we found it is necessary to list all the extra available disk devices and mount them, one at a time, until one is successful.

An important issue for Windows users is that their operating system will refuse to format a drive larger than 32 GB as FAT [21], although it can normally mount much bigger FAT partitions and work with them flawlessly. These users will need to use externals tools such as Fat32Format [22]. Mac and Linux users will have no trouble with their standard *Disk Utility* and *mkfs.msdos* tools, respectively.

Zdziarski [2] recommended immediately remounting the data partition in read-only mode prior to beginning the actual imaging. However, in our tests, we found that in both iOS 3 and iOS 4 the system halted if the partition was unmounted. Forcing its remount was not supported either. It must also be noted that imaging a mounted partition may alter the integrity of the filesystem contained in the resulting image. In fact we found out that it is possible to end up with images that are unmountable. In order to reduce this risk, no other activity should take place in the iPad (either via the touch screen or through the network) while imaging.

Considering all the above, the best option is to use a drive with traditional MBR partitioning scheme, containing only a HFS + partition. Once the disk has been mounted, it is possible to assess its free space and confirm that the drive has been correctly recognized.

4.2.2. *Obtaining the forensic image*

At this point, the actual imaging process may be started by invoking the Unix low-level copying and raw data conversion `dd` command, as listed in the following line. The resulting data is dumped to the newly mounted filesystem via the USB connection.

```
# dd if=/dev/rdisk0s2s1 of=/mnt/ipad.dd bs=512k
```

If the target drive is FAT32-formatted, the maximum file size it can hold will be 4 GB. This means that images will need to be split into 4 GB blocks. In the recommended scenario (imaging to HFS + drives), this is not necessary.

When finished, the target drive **must** be unmounted before disconnecting it from the iPad. This can be done by either turning the iPad off or unmounting the drive by exiting the `/mnt` folder and running the Unix `umount` command.

If the device is passcode-protected, jailbreaking and accessing via SSH is equally possible. The simplest jailbreaking methods from a user standpoint, such as `jailbreakme.com`, will not work given that we are unable to obtain initial access to the device, but other methods (*redsn0w*, *Pwnage Tool*...) may work.

5. Discussion on image analysis

Using our proposed open method, it is possible to obtain the data stored in the user partition in a manner that can be readily accessed using forensic tools and techniques. This method can be automatically applied to new iOS versions as soon as a jailbreak is available for them, even if it is just a *tethered* jailbreak.

In this section, we provide a brief discussion about how the resulting image can be processed and what kinds of data can be obtained from our image, highlighting some important issues in more recent iOS revisions in that regard.

5.1. *Obtainable data using forensic tools*

Raw images obtained by our method can be treated in many ways. For instance:

- SQLite databases, which can be open with a variety of SQLite clients available for all platforms, will show information such as address book contacts, call history, SMS and e-mail messages, etc. This includes the `consolidated.db` file, which in some iOS 4.x versions keeps a complete log of the user location forever, which can be parsed by iPhoneTracker [23].
- Property list (`.plist`) files containing XML information can be viewed with tools available for all platforms. These files contain other relevant data such as: website bookmarks and cookies, maps history, as well as configuration information for each application installed.
- Images and videos can be viewed. Moreover, each of them is likely to contain embedded GPS coordinates of the place where it was taken. This can be very relevant to certain investigations.
- And finally, as a wise forensic examiner once said, 'when all else fails, we carve'. Raw disk recovery (file carving) tools can be used to recover any file type based on its signature: pictures, voice memos, raw text...This is an extreme resource that implies getting many corrupt files and disclosing valuable data such as names, paths, and timestamps; however it is still a very valuable technique, helpful in many scenarios. Open source tools such as Scalpel [24], as well as most commercial data recovery tools, can be used to extract raw data from the image.

We verified that these and other valuable data can be extracted from an image generated by our method. All these possibilities are widely covered by Zdziarski [2] and Morrissey [25].

5.2. *iOS 4 data encryption*

Starting with iOS version 4, Apple introduced a layer of hardware encryption services called *data protection* [26]. This feature, if activated, will result in partial encryption of the imaged data. Therefore, it is a very important aspect as far as forensic image processing is concerned.

We tested the proposed imaging method on devices with this feature activated and experimented with the encryption layer, observing a new `protect` option for the `mount` command. Under a fresh installation (and subsequent jailbreaking) of iOS 4.2.1 on an iPad, running the `mount` command will show that this option is used for the data partition. The same happens on an iPhone 4. However, on an iPhone 3G (which does not support this feature), the `protect` option does not show up.

In addition, we have observed that iOS 4.2.1 `/sbin/mount` binaries in the iPad and iPhone 4 contain the *protect* string, but not in the iPhone 3G binary. The string is also present in several of the `/sbin/mount_*` binaries under Mac OS X 10.6 Snow Leopard, whereas those same binaries in Linux systems do not contain the *'protect'* string. This leads us to consider that, probably, the `protect` parameter for the `mount` command refers to the new *data protection* (local encryption) iOS feature.

There are two factors that weaken this encryption scheme:

(1) All original iPads sold between March and November 2010 (that is 8–10 million devices) shipped with iOS version 3.x. Even when undergoing a normal upgrade process to iOS 4.x, encryption will not be activated: for this to happen, the device must go through a full restoration, which requires completely restoring its data from the iTunes backup. This takes a notably longer time (one to two hours instead of barely 5–10 min) and will only happen if a fatal error rendered the device unbootable.
(2) Even if data protection is enabled, it could be circumvented as long as the encryption keys are stored within the device itself.

In order to determine whether the encryption keys are stored in the device (i.e. the device can mount the encrypted data automatically, without the need of any user input or network connectivity), we performed the following checks:

(1) The following test was conducted over two devices: a first-generation iPad and an iPhone 4, both of them running iOS version 4.3.1.
(2) The device was jailbroken and an SSH server installed.
(3) Using the iOS 'Settings' app, GSM/3G data connectivity was disabled. In addition, 'flight mode' was enabled.
(4) Wi-Fi connectivity was then enabled again. This causes the device to remain in 'flight mode' (absolutely no GSM/3G activity) while allowing for 802.11 wireless communication.
(5) The device was connected to a Wi-Fi network with no Internet connectivity. Other nearby networks stored in the device's 'preferred networks' list were erased, to ensure that upon reboot the device would join our test network.
(6) The device was turned off, and then on, booting normally.
(7) Once the boot process finished, the device would show its lock screen, asking for the passcode. At this point, the device was left untouched.
(8) From a different computer in the same Wi-Fi network, we initiated a SSH connection to the device, running the 'mount' command and confirming that the data partition (`/dev/disk0s2s1`) was already mounted under `/private/var`.

The encrypted volume is always automatically mounted with no need for passcode entry or Internet connectivity. This leads us to the conclusion that iOS 4 local encryption relies on a key stored within the device itself, which implies that forensic images acquired through our method can be decrypted—even when certain data may be additionally encrypted with the user passcode, which seems to be the case with the Mail application and others.

In fact, in May 2011, ElcomSoft announced [27] the breaking of iOS 4 encryption, with a smart approximation that can extract most encryption keys from the physical device, which confirms that the keys reside within the device itself. However, ElcomSoft restricts the availability of the toolkit to select government entities such as law enforcement and forensic organizations and intelligence agencies.

5.3. Obtaining relevant information from encrypted images

Even though, to our knowledge, no open and easily available mechanism exists to defeat iOS 4 encryption, our experiments have shown that, nevertheless, it is possible to obtain some data from our image. Even though encrypted images initially fail to mount under Mac OS X, we were able to find that, surprisingly, when those images are truncated (i.e. the filesystem loses its final bytes), they mount correctly. These results persisted in all our tests, in which we shortened the images by trailing the final bytes (from 1 KB to 1 GB). We did not run further tests because larger cuts would, in the end, generate problems when reading the images. However, this behavior seems to indicate that the 'encrypted' flag resides in a header at the end of the filesystem and, when this header is missing, OS X does its best to mount the filesystem anyway.

Using this mechanism, the full list of files and directories can be obtained, as well as other details such as last modification time or file length. The file contents themselves are encrypted, but a lot of useful information about the device usage can still be obtained this way, such as the list of installed applications or their last usage date.

In many cases, just the existence of files with certain names and sizes can be relevant evidence. For instance, consider a corporate information leak: if an iOS network client was used to extract the leaked files, or even if just some of the leaked files have been read in an iOS application, the image will show files with:

- Names, sizes, and *m*-time (last Modification time) matching those of the original, leaked files.

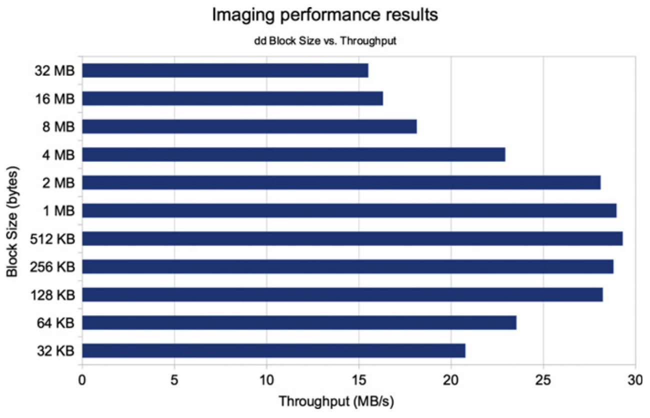


Fig. 3. Throughput analysis with different block sizes.

Table 1
Test results: throughput obtained with different block sizes.

bs	S1 (MB/s)	S2 (MB/s)	S3 (MB/s)	S (MB/s)
32 KB	20.7	20.8	20.8	20.77
64 KB	20.7	25.1	25.1	23.54
128 KB	29.5	27.6	27.6	28.22
256 KB	28.8	28.8	28.8	28.80
512 KB	28.8	29.5	29.6	29.30
1 MB	29.0	28.9	29.0	28.97
2 MB	28.1	28.1	28.1	28.10
4 MB	22.9	22.9	23.0	22.93
8 MB	18.1	18.1	18.2	18.13
16 MB	16.3	16.3	16.3	16.30
32 MB	15.5	15.5	15.5	15.50

- *c*-time (Creation time, or time of the last status Change), indicating when the files first arrived on this device (or, if they were moved to a different directory afterwards, indicating when that happened).
- *a*-time (last Access time), indicating when the files were last read, which can indicate whether the files have been manually reviewed by the user after downloading them.
- Their allocation under the folder of the corresponding iOS application that generated them.

6. Performance analysis

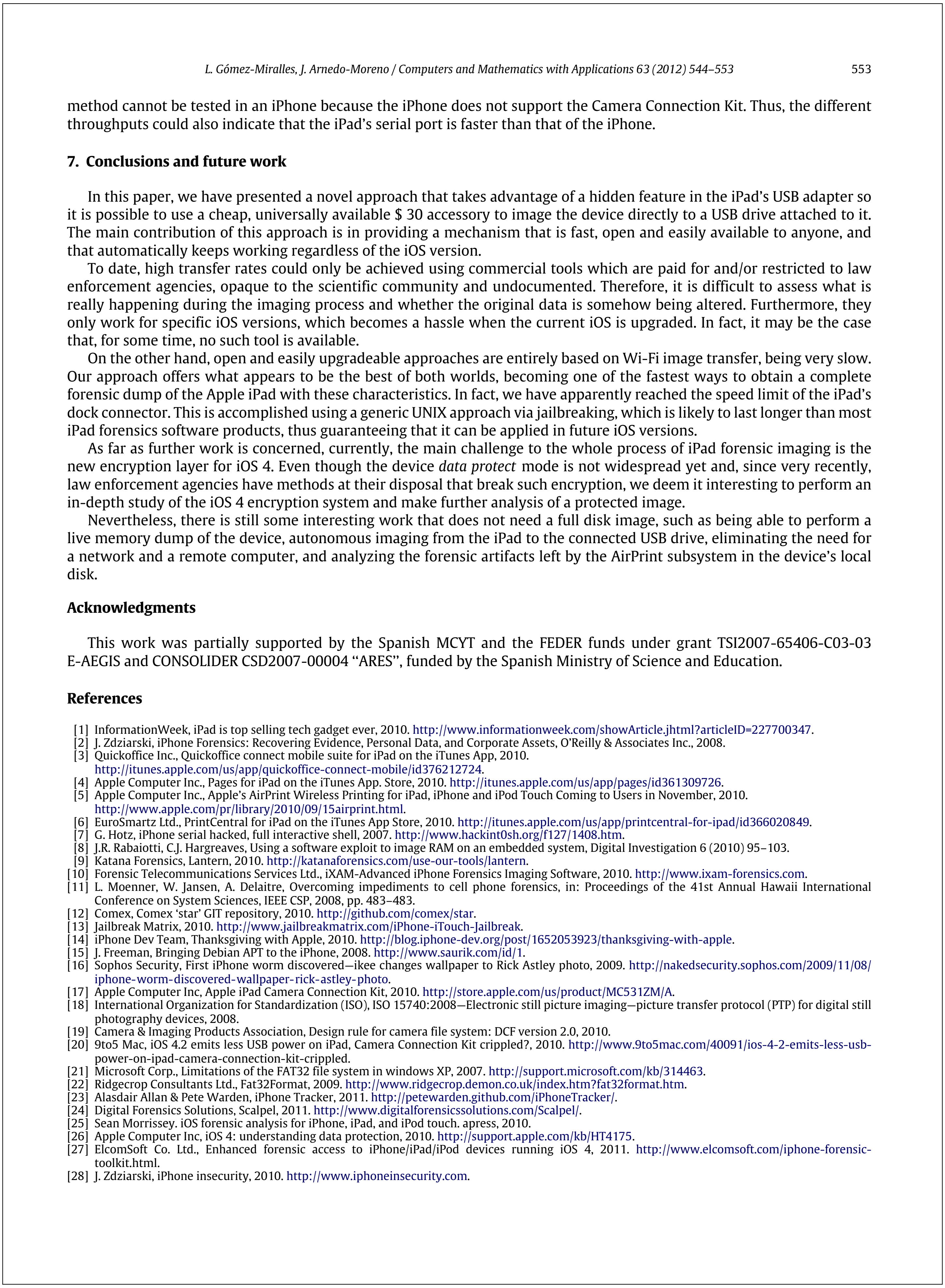
The most important parameter in the efficiency of our proposed method, as far as imaging speed is concerned, is the `bs` (*block size*). In order to obtain an optimal value, we ran a series of tests for each value. External interference was minimized by turning off Bluetooth, 3G data, push notifications, push e-mail, and localization services. Wireless connectivity was needed for our tests, however our wireless network did not have a gateway to the Internet, again to keep the device activity to a minimum. After these settings were applied, the device was rebooted into a clean state.

Under these conditions we ran three complete 63.5 GB dumps for 11 different values for `bs`, amounting to a total of 1.90 terabytes in 33 dumps. The speed obtained in each test (S1, S2, S3) can be seen in detail in the corresponding Table 1 (see Fig. 3 for a quick summary).

We were surprised to find that small values (128 KB–2 MB) provide the best throughput, whereas larger or smaller values result in a notable speed decrease. This is in contrast to what happens with traditional computers (be they desktops or laptops), in which the best results are obtained with block sizes of 16 or 32 MB, usually matching typical hard drive cache sizes.

In comparison, we tested the Zdziarski method over an ad-hoc 802.11n network operating at its maximum theoretical rate (108 Mbps), and we obtained a throughput of barely 1 MB/s, which means that a 16 GB iPad would be imaged in 5 h and a 64 GB one would require about 20 h. Our USB approach results in a speed boost of nearly 30× over traditional Wi-Fi imaging.

Forensics software vendors do not seem to release specifications about the imaging times needed by their methods; we could only find that information for Jonathan Zdziarski, who states [28] "*the latest version of the Zdziarski method, which is used in the automated tools available free to law enforcement agencies worldwide*": "*about* 15–30 min *is all it takes, regardless of whether you're imaging a* 4 GB *iPhone or a* 32 GB *iPhone* 3*G[s]*". Assuming he is able to image 32 GB in 'about 30 min', our method is twice as fast. However, Zdziarski does not mention how much time he needs to image an iPad, and our



method cannot be tested in an iPhone because the iPhone does not support the Camera Connection Kit. Thus, the different throughputs could also indicate that the iPad's serial port is faster than that of the iPhone.

7. Conclusions and future work

In this paper, we have presented a novel approach that takes advantage of a hidden feature in the iPad's USB adapter so it is possible to use a cheap, universally available $ 30 accessory to image the device directly to a USB drive attached to it. The main contribution of this approach is in providing a mechanism that is fast, open and easily available to anyone, and that automatically keeps working regardless of the iOS version.

To date, high transfer rates could only be achieved using commercial tools which are paid for and/or restricted to law enforcement agencies, opaque to the scientific community and undocumented. Therefore, it is difficult to assess what is really happening during the imaging process and whether the original data is somehow being altered. Furthermore, they only work for specific iOS versions, which becomes a hassle when the current iOS is upgraded. In fact, it may be the case that, for some time, no such tool is available.

On the other hand, open and easily upgradeable approaches are entirely based on Wi-Fi image transfer, being very slow. Our approach offers what appears to be the best of both worlds, becoming one of the fastest ways to obtain a complete forensic dump of the Apple iPad with these characteristics. In fact, we have apparently reached the speed limit of the iPad's dock connector. This is accomplished using a generic UNIX approach via jailbreaking, which is likely to last longer than most iPad forensics software products, thus guaranteeing that it can be applied in future iOS versions.

As far as further work is concerned, currently, the main challenge to the whole process of iPad forensic imaging is the new encryption layer for iOS 4. Even though the device *data protect* mode is not widespread yet and, since very recently, law enforcement agencies have methods at their disposal that break such encryption, we deem it interesting to perform an in-depth study of the iOS 4 encryption system and make further analysis of a protected image.

Nevertheless, there is still some interesting work that does not need a full disk image, such as being able to perform a live memory dump of the device, autonomous imaging from the iPad to the connected USB drive, eliminating the need for a network and a remote computer, and analyzing the forensic artifacts left by the AirPrint subsystem in the device's local disk.

Acknowledgments

This work was partially supported by the Spanish MCYT and the FEDER funds under grant TSI2007-65406-C03-03 E-AEGIS and CONSOLIDER CSD2007-00004 "ARES", funded by the Spanish Ministry of Science and Education.

References

[1] InformationWeek, iPad is top selling tech gadget ever, 2010. http://www.informationweek.com/showArticle.jhtml?articleID=227700347.
[2] J. Zdziarski, iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets, O'Reilly & Associates Inc., 2008.
[3] Quickoffice Inc., Quickoffice connect mobile suite for iPad on the iTunes App, 2010. http://itunes.apple.com/us/app/quickoffice-connect-mobile/id376212724.
[4] Apple Computer Inc., Pages for iPad on the iTunes App. Store, 2010. http://itunes.apple.com/us/app/pages/id361309726.
[5] Apple Computer Inc., Apple's AirPrint Wireless Printing for iPad, iPhone and iPod Touch Coming to Users in November, 2010. http://www.apple.com/pr/library/2010/09/15airprint.html.
[6] EuroSmartz Ltd., PrintCentral for iPad on the iTunes App Store, 2010. http://itunes.apple.com/us/app/printcentral-for-ipad/id366020849.
[7] G. Hotz, iPhone serial hacked, full interactive shell, 2007. http://www.hackint0sh.org/f127/1408.htm.
[8] J.R. Rabaiotti, C.J. Hargreaves, Using a software exploit to image RAM on an embedded system, Digital Investigation 6 (2010) 95–103.
[9] Katana Forensics, Lantern, 2010. http://katanaforensics.com/use-our-tools/lantern.
[10] Forensic Telecommunications Services Ltd., iXAM-Advanced iPhone Forensics Imaging Software, 2010. http://www.ixam-forensics.com.
[11] L. Moenner, W. Jansen, A. Delaitre, Overcoming impediments to cell phone forensics, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, IEEE CSP, 2008, pp. 483–483.
[12] Comex, Comex 'star' GIT repository, 2010. http://github.com/comex/star.
[13] Jailbreak Matrix, 2010. http://www.jailbreakmatrix.com/iPhone-iTouch-Jailbreak.
[14] iPhone Dev Team, Thanksgiving with Apple, 2010. http://blog.iphone-dev.org/post/1652053923/thanksgiving-with-apple.
[15] J. Freeman, Bringing Debian APT to the iPhone, 2008. http://www.saurik.com/id/1.
[16] Sophos Security, First iPhone worm discovered—ikee changes wallpaper to Rick Astley photo, 2009. http://nakedsecurity.sophos.com/2009/11/08/iphone-worm-discovered-wallpaper-rick-astley-photo.
[17] Apple Computer Inc, Apple iPad Camera Connection Kit, 2010. http://store.apple.com/us/product/MC531ZM/A.
[18] International Organization for Standardization (ISO), ISO 15740:2008—Electronic still picture imaging—picture transfer protocol (PTP) for digital still photography devices, 2008.
[19] Camera & Imaging Products Association, Design rule for camera file system: DCF version 2.0, 2010.
[20] 9to5 Mac, iOS 4.2 emits less USB power on iPad, Camera Connection Kit crippled?, 2010. http://www.9to5mac.com/40091/ios-4-2-emits-less-usb-power-on-ipad-camera-connection-kit-crippled.
[21] Microsoft Corp., Limitations of the FAT32 file system in windows XP, 2007. http://support.microsoft.com/kb/314463.
[22] Ridgecrop Consultants Ltd., Fat32Format, 2009. http://www.ridgecrop.demon.co.uk/index.htm?fat32format.htm.
[23] Alasdair Allan & Pete Warden, iPhone Tracker, 2011. http://petewarden.github.com/iPhoneTracker/.
[24] Digital Forensics Solutions, Scalpel, 2011. http://www.digitalforensicssolutions.com/Scalpel/.
[25] Sean Morrissey. iOS forensic analysis for iPhone, iPad, and iPod touch. apress, 2010.
[26] Apple Computer Inc, iOS 4: understanding data protection, 2010. http://support.apple.com/kb/HT4175.
[27] ElcomSoft Co. Ltd., Enhanced forensic access to iPhone/iPad/iPod devices running iOS 4, 2011. http://www.elcomsoft.com/iphone-forensic-toolkit.html.
[28] J. Zdziarski, iPhone insecurity, 2010. http://www.iphoneinsecurity.com.

Appendix D

Analysis of the forensic traces left by AirPrint in Apple iOS devices

L. Gómez-Miralles, J. Arnedo-Moreno. Analysis of the forensic traces left by AirPrint in Apple iOS devices. Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA), 2013, pp. 703-708. DOI: 10.1109/WAINA.2013.40.

Analysis of the forensic traces left by AirPrint in Apple iOS devices

Luis Gómez-Miralles, Joan Arnedo-Moreno
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Carrer Roc Boronat, 117 , 7a planta 08018 Barcelona, Spain
pope,jarnedo@uoc.edu

***Abstract*—Since its presentation by Apple, both the iPhone and iPad devices have achieved a great success and gained widespread popularity. This fact, added to the given idiosyncrasies of these new portable devices and the kind of data they may store open new opportunities in the field of computer forensics. In 2010, version 4 of their operating system (iOS) introduced AirPrint, a simple and driverless wireless printing functionality supported by some network printers. This paper presents an analysis of the traces left by AirPrint and assesses whether it is feasible to recover them in the context of a forensic investigation.**

Keywords: Forensics, iPad, iPhone, iOS, Cybercrime, AirPrint, Apple

I. Introduction

Information technologies have grown rapidly in the last decades, changing the way we live, work, and communicate. Portable devices such as smartphones and tablets have evolved from simple phones and agendas into literally full-fledged, always-online computers that fit our pockets, containing huge amounts of valuable data about us: contacts, calendar, e-mails, photographs, as well as a pile of logs: phone calls, chat, geographic positions, etc.

The practice of digital forensics has needed to adapt quickly to the emerging mobile technologies. We once had a homogeneous personal computer market, mainly dominated by a few different Windows versions, with minor representations of Mac OS or Unix-based systems. Now we find that the most personal devices, the ones that always accompany their users and are more prone to contain sensitive information, run software environments which simply didn't exist a few years ago - namely Android and iOS. Furthermore, because of the competitive nature of the market, with each new version of these systems, new functionalities are added in order to appeal to a greater set of users, and thus become their device of choice. However, some of these new features may manage personal user data and are worth analyzing from a forensic investigation standpoint.

This paper focuses on Apple iOS devices (namely, iPhone, iPad and iPod Touch) and, specifically, their capabilities to print wirelessly to compatible printers using the AirPrint technology, presented by Apple in late 2010 [1]. This paper analyzes this relatively recent feature and determines whether using AirPrint to print a document leaves some kind of trace in the iOS device itself which may be open to subsequent forensic analysis, leaving personal user data exposed without their knowledge. Given the popularity and acceptance of iOS based devices [2], any available process which allows to recover user data becomes especially relevant from both a computer forensics and a privacy concern standpoint.

As far as the authors can tell, there has been no research on how AirPrint works behind the scenes and the forensic traces it may leave. Several authors have reviewed the existing, mostly commercial, forensic investigation tools for iOS based devices [3], [4]. However, analysis of AirPrint activity does not seem to be covered by any of the software solutions available for iOS devices. In fact, it may seem odd that something as basic as document printing has been left out of the forensic analysis of mobile devices so far. One must consider, however, that mobile operating systems have lacked common printing frameworks till quite recently. Apple incorporated AirPrint into iOS in late 2010, whereas Android was not provided printing capabilities until early 2011, through Google Cloud Print [5].

This paper is structured as follows. First of all, in Section II, AirPrint and its mode of operation both from a user's and technical standpoint are presented. The results of the analysis of forensic traces left by AirPrint are shown in Section III. Following, in Section IV, the recoverability of such traces and which kind of useful information may be obtained is assessed. Finally, concluding the paper, Section V summarizes the paper contributions and outlines further work.

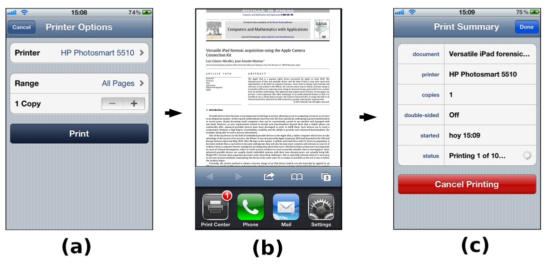
II. Description of AirPrint network printing

Briefly explained, AirPrint is an iOS feature that allows applications to send content to printers using the iOS device's wireless connection. Directly quoting Apple [1]: *'AirPrint automatically finds printers on local networks and can print text, photos and graphics to them wirelessly over Wi-Fi without the need to install drivers or download software'*.

Apple announced AirPrint in September 2010. Two months later, iOS 4.2 was released for the iPhone, iPad and iPod Touch, being the first iOS version to offer this feature to users. Its standard functionality allows printing only to specific, AirPrint-enabled printers. Nevertheless, as of January 2012, there are more than one hundred AirPrint-enabled printers in the market, from five major vendors (Brother, Canon, Epson, HP and Lexmark) [6]. Apple does not support sharing a common printer via the computer it is connected to, even when it was possible with some Mac OS X 10.6.5 beta versions; however, it can be done by using software tweaks such as AirPrintActivator [7].

Long before the introduction of AirPrint, different solutions [8], [9] tried to fill in this gap. Usually, such solutions involved iOS applications capable of opening different file formats and sending them to a desktop computer, running a companion application, which would in turn send the document to the printer itself. Some printer vendors developed specific clients, however, none of these solutions were ever widely spread among users. Currently, with AirPrint working out-of-the-box and embedded into all applications, it is hard to believe that new users will consider using a specific, usually paid, application to handle printing, except maybe in some very particular environments, such as cases where the use of these kind of applications was consolidated before AirPrint was launched, or some advanced capabilities are required by power users.

From a user's standpoint, Figure 1 summarizes the printing process, showing the screens it is actually possible to interact with, as seen on an iPhone.



Printer HP Photosmart 5510
Range All Pages
1 Copy

document Versatile iPad forensic...
printer HP Photosmart 5510
copies 1
double-sided Off
started hoy 15:09
status Printing 1 of 10...

Fig. 1. Step-by-step AirPrint options screen on an iPhone.

In the client side (iOS device), AirPrint-enabled applications contain a "Print" button that, when pressed, will present an extremely simple menu (Figure 1, (a)), with only two or three available options:

1) **Printer:** This option opens a list of all AirPrint-enabled printers found in the local network, showing a "name" and "description" field for each one.
2) **Range:** (optional) Defaulting to "all pages", this option opens a selector which allows the user to choose a range of pages to be printed, rather than all the document.
3) **Copies:** Specifies the number of copies to be printed.
4) Depending on the printer features, additional parameters such as duplex printing can be controlled.
5) **Print:** This button proceeds to send the job to the printer.

The user cannot specify any other kind of information usually available in printing menus, such as paper size or orientation, printing quality, etc. Everything is automatically handled by AirPrint, using some default options. When the user chooses to "Print" the job, the device shows some brief messages (*'Contacting Printer'*; *'Preparing page (...) of (...)'*; *'Sending to Printer'*). However, depending on how long the print job is, these messages may be barely visible or last for several seconds.

After the job has been sent to the printer, the printing menu disappears and the application returns to its previous state. At this point, invoking the list of recently used applications, by double-clicking the device "Home" button, reveals a *Print Center* application (Figure 1, (b)). Unless the user somehow knows this application has started running in background, it may be difficult for him to find it, since no active feedback is provided during the printing process, thus being invisible at casual glance.

Opening the *Print Center* application, the user can see the list of running and pending printing jobs, check their details and cancel them (Figure 1, (c)). When the last job finishes, i.e. the moment the printer ejects the last page, the application closes, and does not appear anymore in the list of recently used applications. As far as it is known, there is no way to open the *Print Center* as a standalone application. It is only executed while there are jobs being printed.

From a technical standpoint, the AirPrint service is known to use the standard IPP protocol at network level for printer management, and Bonjour/Zeroconf [10] for service discovery. A comprehensive description of the printing architecture and its underlaying API in iOS devices can be found in [11].

III. Analysis of Forensic Traces

This section presents the preliminary information that must be considered before more in-depth forensic investigation may proceed. Mainly, assessing which traces are left by the AirPrint subsystem, how they can be discovered, how they behave and which useful information can be obtained from them. All this information was discovered through some basic experiments.

A. Preliminary setup

The following equipment was used for all the analysis and tests throughout this paper:

1) iOS devices: iPhone 3G (8 GB, iOS 4.2.1, multitasking enabled) and iPhone 4 (16 GB, iOS 4.3.3 and 5.0.0), both jailbroken with `redsn0w`.
2) AirPrint-enabled printer: HP Photosmart 5510.
3) Two laptops with 802.11 wireless connectivity.

The iPhone 3G was used so that it would be possible to dump and analyze the filesystem without being affected by the filesystem encryption policy enforced on newer devices. Nevertheless, Bedrune and Sigwald published details about iOS data protection, and shortly after they released the tools and source code capable of breaking this encryption [12]. The device was jailbroken in order to gain root access, and install an SSH server and basic UNIX tools.

Given that iOS enforces the device to run only code signed by Apple (downloaded from the App Store), the process of jailbreaking was used in the tests to bypass that restriction in order to have full access to the devices and be able to run shell commands on them. The jailbreak process is exempted from prosecution under the anti-circumvention section of the

U.S. Digital Millenium Copyright Act [13], and it has been very useful for forensic research in the past [14], [15].

The AirPrint feature depends on the multitasking capabilities of the device. In fact, Apple disabled these feature for some devices in iOS 4 alleging performance issues. However, it is possible to enable them during the jailbreak process using the `redsn0w` tool [16]. By doing so, the chosen device became the perfect testbed for the experiments: a small device (its 8 GB take about 3 hours to transfer via wi-fi), with no encryption, that supports AirPrint. The iPhone 4 was used to confirm that some of the findings still applied on newer devices and under iOS version 5.

For both environmental and budget issues, the best efforts were made to reduce costs and waste. Paper was reused by printing once and again over the same pages, and the life of print cartridges was extended well beyond the limits of readability. Some printers complain on low ink and refuse to continue printing even when they are still giving pretty decent results; luckily, the chosen model kept printing for a long time after the cartridge was empty, and only then it presented a message warning about the warranty issues when continuing printing with empty cartridges. Nevertheless, it is probably obvious that it was necessary to throw away quite a lot of paper during the experiments, as well as quite a few ink cartridges. Anyway, resources were recycled as much as possible.

B. Forensic traces left by AirPrint

Once a root shell is executed, it is possible to invoke several commands before, during and after printing, comparing the results in order to look for remarkable differences. The following commands were executed in the iOS devices:

1) `find / -type (b,c,d,f,l,p,s)` for listing, respectively, all the block special devices, character special devices, directories, regular files, symbolic links, FIFOs, and sockets, in the filesystem.
2) `netstat -an -f inet` for listing any current network connections. This would show active client-server activity, as well as inactive servers awaiting for incoming petitions.
3) `ps aux` for getting information about running processes.

By reviewing the lists of files and directories generated with the `find` commands explained above, it was observed that, when a device prints via AirPrint for the first time, the following folder is created:

```
/var/mobile/Library/com.apple.printd/
```

The moment a document is sent to printing, a new file named `1.pdf` is created under this folder. After running additional tests, it was confirmed that this PDF file exists in disk only while the document is being printed. The moment the printer ejects the last page and considers the job finished, the PDF file is deleted. This is also the moment at which the *Print Center* application disappears from the list of recently used applications.

Printing some documents and copying the resulting temporary PDF files to another location before their deletion, and then examining them, is enough to find that these files are regular PDF files with the same content that is being sent to the printer (no matter whether it was originally in PDF format or not). Hence, an obvious trace is being left in the filesystem, and it reflects exactly what was printed.

It must be noted that, in some of the preliminary tests, before the physical printer was available, a virtual PDF printer was setup in a Mac computer and shared, making it look like an AirPrint printer. It worked as expected, it was possible to print to it from an iPhone, as well as an iPad. However, the printing of the document (in this case, the generation of a file in the hard drive of the Mac) was much shorter than the actual printing of a page through a real printer, with real ink and paper. This greatly reduces the chances of the temporary files being flushed to physical disk from the buffer cache in the iOS device before deletion, and hence their chances of recoverability. Therefore, to obtain accurate results, the experiments need to be performed with a real printer.

C. Properties of the AirPrint temporary files

From the execution of the different tests, the following rules were observed:

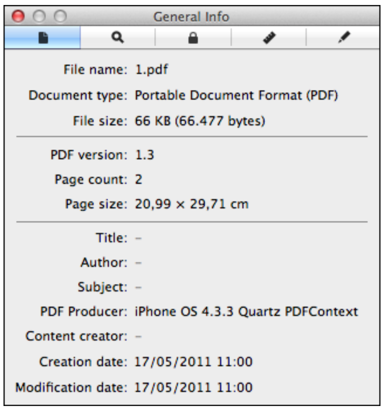
1) For every print job sent via AirPrint, a file with the name of `1.pdf` is created in the directory `/var/mobile/Library/com.apple.printd/`
2) This file is in PDF format, containing the document sent to the printer. This observed behavior is consistent across internal iOS applications (Mail, Safari) as well as third party ones (GoodReader, Papers, Keynote...) The only exception found is the iOS Photos app, which seems not to generate any temporary files on disk when printing, thus not leaving these traces.
3) The file `1.pdf` is deleted as soon as the printing job finishes. The timing observed indicates that this happens not just after finishing the task of submitting the job to the physical printer, but after the document has been completely printed.
4) When one job is being printed, subsequent jobs arriving to the queue generate files named `2.pdf`, `3.pdf`, etc. The behavior observed suggests that in iOS 4 the counter resets as soon as the queue is empty (if a new job arrives later it will be named `1.pdf` again), whereas in iOS 5 the counter seems to keep increasing (each new job gets a higher number even if the queue is empty) until the device reboots.
5) When a job asks for more than one copy of the same document, the temporary PDF file contains only one copy of it. There is probably a command, sent from the device to the printer, indicating the number of copies wanted.
6) When a page range is specified, the temporary PDF file contains only this page range. There is an exception when some applications print files that are themselves PDFs, which is studied later in Section III-D.

D. *PDF metadata of the temporary files*

Using a standard PDF reader, the metadata contents inside different temporary files generated by AirPrint were extracted and compared them. Generally, all the temporary PDF files created by AirPrint can be identified as such by their metadata: they all show the same 'PDF producer' entry (*'iPhone OS x.y.z Quartz PDFContext'*, with *x.y.z* being the iOS version number), and the creation and modification dates both indicate the date and time when the document was sent to the printer (see Figure 2). Therefore, knowing what has been printed from a device looks as simple as recovering deleted PDF files from it and focusing on those with the strings *'iPhone OS x.y.z Quartz PDFContext'*. Moreover, the creation and modification dates contained inside those PDF files in the form of PDF metadata will indicate precisely the printing date and time.

Only one exception was found to this behavior. When printing PDF files, i.e. when the content to be printed is itself in PDF format, some applications behave like described earlier, but others (including iOS built-in applications such as Safari or Mail) actually copy the original PDF file to the `com.apple.printd` directory as `1.pdf`, instead of generating a new PDF file with fresh metadata. In these cases, PDF metadata cannot be used to tell whether one given file is a trace from AirPrint printing: the only peculiar thing about that PDF file is the fact that it resides under such directory.

Considering this, an eventual automated tool aimed at recovering the traces left by AirPrint should go further than just carving for PDF files: it should correctly interpret the internals of the HFSX filesystem and tell whether each recovered file existed within the `com.apple.printd/` directory, or somewhere else.



File name: 1.pdf
Document type: Portable Document Format (PDF)
File size: 66 KB (66.477 bytes)

PDF version: 1.3
Page count: 2
Page size: 20,99 × 29,71 cm

Title: –
Author: –
Subject: –
PDF Producer: iPhone OS 4.3.3 Quartz PDFContext
Content creator: –
Creation date: 17/05/2011 11:00
Modification date: 17/05/2011 11:00

Fig. 2. Metadata of one temporary PDF file generated by AirPrint, as shown by the OS X 'Preview' tool.

IV. Recoverability of AirPrint traces

Given that the temporary PDF files generated by AirPrint only exist in the filesystem for a limited time and are deleted when the document has been printed, the possibility that, depending on disk scheduling and other factors, these files might never be actually flushed to the physical disk must be considered. In that case, they are unrecoverable by a subsequent forensic analysis. This section studies such possibility and assesses the probability that a given trace may be actually recovered at a later time.

The are several factors that increase the chances of the temporary PDF files being flushed to disk, making them potentially recoverable in the future. Some of these factors are:

- Documents that take a long time to be printed because they have many pages or because they contain graphics. Note that, when sending a document to the printer, the user can control very few options: printer, page range, number of copies - nothing else. There is neither an option for printing in 'draft mode', nor one for using only the black cartridge, meaning that everything sent via AirPrint is printed in full color, good quality... and may require quite a lot of time to finish.
- Documents that are sent while there are other printing tasks running.
- Periods of printing interruptions due to the need of human interaction, such as the printer running out of paper or out of ink.

A. *Test scenario*

In order to test the recoverability of AirPrint traces in the form of deleted temporary files, a series of experiments was run. The goals of these experiments were twofold. First of all, to determine whether the temporary files generated by AirPrint are actually written to the physical disk at some point, and thus may be recoverable after deletion. In addition, to asses whether, even if the previous case is true, each of those temporary files would still be recoverable after generating more of them (i.e. what are the chances that the AirPrint temporary files overlap each other in disk, always overwriting the same space and thus making each other unrecoverable).

Some steps in the testing process involved printing documents, while others were just aimed at simulating some casual user activity in the device (mail browsing, software update, reboot). For those tests that involve printing, 10 documents of a given kind are printed in each test. Five of the tests involve printing, which means that after completing all the tests, 50 documents had been printed.

After each of the tests, a dump of the iPhone filesystem was obtained. For those tests that involved printing, the dump process was not started until the printer had finished printing all of the submitted jobs sent.

Given that the chosen device was an iPhone, the disk image was transferred on-the-fly via wi-fi to an external computer, using the method proposed by Zdziarski , based on the `dd` and `netcat` commands [14]. However, in the case of iPad devices, the image could be transferred much faster to USB storage using a method based on the Apple Camera Connection Kit [17].

Note that the set of tests was carried out in a cumulative way, meaning that given any dump $dumpN$, it could contain traces of not only the temporary files created during test $testN$, but of all of the temporary files created during the earlier tests as well. Given that there was a total of 50 documents printed, but 10 of them correspond to the Photos application, which does not generate the AirPrint temporary files (as explained earlier in section III-C), the latest dumps should have a maximum of 40 recoverable artifacts.

A detailed description of the whole testing process follows:

1) Use Safari to print 4 web pages and 6 PDF files. Obtain $dump1$. From this filesystem dump, it should be possible to recover up to 10 temporary PDF files generated by AirPrint.
2) Use GoodReader to print 10 PDF files. Obtain $dump2$. This introduces 10 new temporary files, thus there should be a maximum of 20 recoverable AirPrint artifacts in this dump (10 from the previous test, and 10 more from this test).
3) Use iOS built-in Photos app to print 10 photos. Obtain $dump3$. The number of maximum recoverable AirPrint artifacts remains at 20, given that, as explained earlier, printing from the Photos application does not generate temporary files - an exception not observed in any other application.
4) Use iOS built-in Mail app to print 5 e-mails and 5 attachments. Obtain $dump4$. There should be a maximum of 30 recoverable AirPrint artifacts (20 from earlier tests, plus 10 more created during test).
5) Turn the device off, wait for 30 seconds, turn it on. Obtain $dump5$. This does not generate any new AirPrint temporary files, thus at this point the number of maximum recoverable artifacts remains at 30.
6) Use Safari to print 6 DOC and 4 XLS files. Obtain $dump6$. These 10 new temporary files increase the number of maximum recoverable artifacts to 40.
7) Open the Mail app, download messages and large attachments (totaling 15 MB), turn the device off, wait for one minute, and turn it on. Obtain $dump7$. This does not generate any new temporary files, thus the number of maximum recoverable artifacts remains at 40.
8) Open the App Store. Install available software updates (iBooks and GoodReader). Obtain $dump8$. Again, this does not generate any new artifacts.

B. Recovering the traces

Each filesystem dump was analyzed in order to determine whether the temporary files generated by AirPrint at each stage were recoverable, both immediately after their generation, and at later stages.

In order to recover the traces left by AirPrint, the *file carving* [18] technique was chosen. This is a data recovery method consisting of going through a raw data stream (a filesystem dump in this case) looking for possible 'headers' and 'footers' (beginnings and ends) of known, chosen file types, such as JPEG pictures, MPEG video files, PDF documents, etcetera. The more strict a file type specification is, the easier it is for the carving tool to identify and recover that file type. In addition, some tools perform sanity checks such as establishing a file size limit or checking each recovered item against its format specification, to determine whether it may be corrupt.

One of the benefits of the carving technique is that it can be used, with more or less success, over any kind of data, be it a known or unknown filesystem, a portion of it, or something completely different, such as network captures or RAM memory dumps. Note, however, that many tools implement specific strategies for common filesystems in order to improve overall success rate and extract items only from the unallocated space, skipping the disk space used by normal existing files. This is something very useful when the user just wants to focus on recovering deleted files.

This technique was applied by means of the open source, widely used `photorec` tool [19], which has a long track of usefulness in this kind of scenarios [20] [18], even on mobile devices [21].

Every dump obtained during the tests was carved for PDF files using the default `photorec` parameters. Fine-tuning these parameters would have probably improved the recovery success rate, however, after the tests were completed, it was found unnecessary given that the results achieved were indeed very positive.

The results of trace extraction using the photorec tool were analyzed from two different standpoints:

1) **Recoverability**. Tenths of the temporary PDF files generated by AirPrint were successfully recovered from each filesystem dump, which confirms that those files are indeed flushed to disk.
2) **Persistence**. The artifacts created during our first tests were still recoverable after the last tests. This suggests that, probably due to iOS file allocation strategies, the temporary files generated by AirPrint are not very prone to occupy the disk space previously assigned to other of those files.

Figure 3 summarizes the results. The horizontal axis enumerates each of the test steps previously explained in section IV-A, whereas the blue and orange lines indicate, respectively, how many AirPrint temporary files had been introduced at each step, and how many of them were successfully recovered by `photorec`.

From the results of the experiments, it can be concluded that, immediately after printing one or more documents, there is a very high probability that the file will actually be written to physical disk and, thus, recovering the temporary files generated by this process. For instance: 100% of the temporary artifacts created during $test1$ were successfully recovered from $dump1$. All of them were also successfully recovered from $dump2$ (altogether with the new artifacts introduced during $test2$).

Nevertheless, as expected, the recoverability rate decreases as time goes by and the device continues working. In the tests, this can be seen after $test4$ and $test7$. Considering that the artifacts are stored in unallocated space, it is unavoidable that using the device for long periods of time reduces the chance of recovering such artifacts, as new data stored in the device

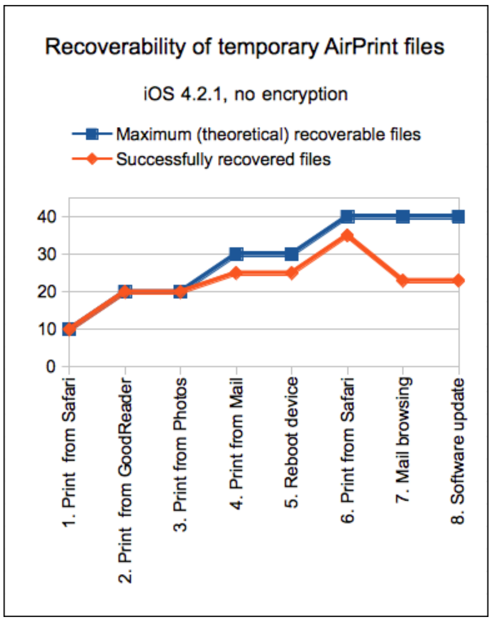


Fig. 3. Recoverability test results

may overwrite that disk space. However, the tests show that the probability does not decrease very quickly, and there is at least a good chance to recover most of the traces.

V. Conclusions and Future Work

This paper analyzes the forensic traces left by usage of the AirPrint functionality in iOS based devices. As a result, it was found that the use of this feature to wirelessly print documents leaves a trace in the device in the form of temporary files, containing a copy of the printed information and indicating precisely when it was printed. The recovery of these AirPrint artifacts can be very valuable from a forensic standpoint in scenarios such as information leaks or inadequate content distribution.

The performed tests show that these traces may be recoverable even after the device continues undergoing activity. Even when the experiments were designed to skip the iOS encryption (optional in iOS 4, enforced in all devices in iOS 5, released in October 2011), these findings could be successfully applied to encrypted devices when combined with the recent undeletion-and-unencryption process implemented by Jerome and Sigwald [12], and hence could be integrated into iOS forensic tools.

In fact, further work includes a more detailed analysis of file recovery in encrypted devices in order to test AirPrint artifact recovery under such scenario. In addition, it would be interesting to assess whether additional information related to the AirPrint capability could be obtained, apart from that directly extracted from the temporary files, such as the specific printer the job was submitted to.

Acknowledgments

This work was partially supported by the Spanish MCYT and the FEDER funds under grant TSI2007-65406-C03-03 E-AEGIS and CONSOLIDER CSD2007-00004 ”ARES”, funded by the Spanish Ministry of Science and Education.

References

[1] Apple Computer Inc., “Apples AirPrint Wireless Printing for iPad, iPhone & iPod touch Coming to Users in November”, 2010, http://www.apple.com/uk/pr/library/2010/09/15airprint.html.
[2] InformationWeek, “iPad is top selling tech gadget ever”, 2010, http://www.informationweek.com/showArticle.jhtml?articleID=227700347.
[3] A. Levinson, B. Stackpole, and D. Johnson, “Third party application forensics on apple mobile devices”, in *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, 2011, pp. 1 –9.
[4] Andrew Hay, Dennis Krill, Benjamin Kuhar, and Gilbert Peterson, “Evaluating digital forensic options for the apple ipad”, in *Advances in Digital Forensics VII*, vol. 361 of *IFIP Advances in Information and Communication Technology*, pp. 257–273. Springer Boston, 2011.
[5] Google Inc., “Google Cloud Print”, 2010, http://www.google.com/cloudprint/learn/.
[6] Apple Computer Inc., “iOS: AirPrint 101”, 2011, http://support.apple.com/kb/ht4356.
[7] Netputing, “AirPrint Activator”, 2011, http://netputing.com/airprintactivator.
[8] EuroSmartz Ltd., “PrintCentral for iPad, iPhone or iPod Touch”, 2012, http://mobile.eurosmartz.com/products/printcentral.html.
[9] Avatron Software Inc., “Print Sharing”, 2011, http://avatron.com/apps/print-sharing.
[10] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*, O’Reilly, 2005.
[11] Apple Inc., “How Printing Works in iOS”, 2011, https://developer.apple.com/library/ios.
[12] J. Sigwald and J.B. Bedrune, “iPhone data protection tools”, 2011, http://code.google.com/p/iphone-dataprotection/.
[13] U.S. Copyright ONce, “Rulemaking on Exemptions from Prohibition on Circumvention of Technological Measures that Control Access to Copyrighted Works”, 2010, http://www.copyright.gov/1201/2010.
[14] Jonathan Zdziarski, *iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*, O’Reilly, 2008.
[15] J.R. Rabaiotti and C.J. Hargreaves, “Using a software exploit to image RAM on an embedded system ”, *Digital Investigation*, vol. 6, pp. 95–103, 2010.
[16] iPhone Dev Team, “redsn0w”, 2011, http://blog.iphone-dev.org/post/5239805497/tic-tac-toe/.
[17] L. Gomez-Miralles and J. Arnedo-Moreno, “Versatile iPad forensic acquisition using the Apple Camera Connection Kit”, *Computers And Mathematics With Applications*, vol. 63, no. 2, pp. 544 – 553, 2012.
[18] M.I. Cohen, “Advanced carving techniques”, *Digital Investigation*, vol. 426, no. 3-4, pp. 119–128, 2007.
[19] CGSecurity, “PhotoRec, digital picture recovery”, 2009, http://www.cgsecurity.org/wiki/PhotoRec.
[20] S. Garfinkel, “Forensic feature extraction and cross-drive analysis”, *Digital Investigation*, vol. 3, no. 1, pp. 71–81, 2006.
[21] I. Pooters, “Full user data acquisition from symbian smart phones”, *Digital Investigation*, vol. 6, no. 3-4, pp. 125–135, 2010.

Appendix E

AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices

L. Gómez-Miralles, J. Arnedo-Moreno. **AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices**. Mobile Information Systems, Volume 2015, Article ID 916262, 10 pages, 2015. DOI: 10.1155/2015/916262. IF: 0.949, Q3: 76/133. Category: COMPUTER SCIENCE, INFORMATION SYSTEMS (JCR-2014 SE).

Hindawi Publishing Corporation
Mobile Information Systems
Volume 2015, Article ID 916262, 10 pages
http://dx.doi.org/10.1155/2015/916262

Research Article

AirPrint Forensics: Recovering the Contents and Metadata of Printed Documents from iOS Devices

Luis Gómez-Miralles and Joan Arnedo-Moreno

Internet Interdisciplinary Institute (IN3), Universitat Oberta de Catalunya, Roc Boronat Street 117, 7th Floor, 08018 Barcelona, Spain

Correspondence should be addressed to Luis Gómez-Miralles; pope@uoc.edu

Received 10 June 2013; Accepted 22 October 2013

Academic Editor: David Taniar

Since its presentation by Apple, both the iPhone and iPad devices have achieved great success and gained widespread popularity. This fact, added to the given idiosyncrasies of these new portable devices and the kind of data they may store, opens new opportunities in the field of computer forensics. In 2010, version 4 of the iOS operating system introduced AirPrint, a simple and driverless wireless printing functionality supported by hundreds of printer models from all major vendors. This paper describes the traces left in the iOS device when AirPrint is used and presents a method for recovering content and metadata of documents that have been printed.

1. Introduction

Information technologies have grown rapidly in the last decades, changing the way we live, work, and communicate. Portable devices such as smartphones and tablets have evolved from simple phones and agendas into literally full-fledged, always-online computers. In this scenario, where mobile devices become ubiquitous, privacy and cybersecurity become a great concern since such devices may contain huge amounts of sensible valuable data about us: contacts, calendar, e-mails, and photographs as well as a pile of logs: phone calls, chat, geographic positions, and so forth.

The practice of digital forensics has needed to adapt quickly to the emerging mobile technologies. We once had a homogeneous personal computer market, mainly dominated by a few different Windows versions, with minor representations of Mac OS or Unix-based systems. Now we find that the most personal devices, the ones that always accompany their users and are more prone to contain sensitive information, run software environments which simply did not exist a few years ago, namely, Android and iOS. Furthermore, because of the competitive nature of the market, with each new version of these systems, new functionalities are added in order to appeal to a greater set of users and thus become their device of choice. However, some of these new features may manage personal user data and are worth analyzing from a forensic investigation standpoint.

This paper focuses on the AirPrint feature of iOS devices (iPhone, iPad, and iPod Touch), which allows them to print wirelessly to compatible printers [1]. In a previous paper [2] we observed that printing a document through AirPrint leaves a trace in the filesystem of the iOS device in the form of a temporary file containing the printed content and with a specific metadata that allows for the identification of this precise kind of files in the filesystem. This paper extends our previous research to analyze if these temporary files can be recovered even in modern iOS versions which use hardware-based data encryption. Considering the rise of mobile devices and applications in general [3, 4] and the hundreds of millions iOS devices in particular [5], any available process which allows to recover user data becomes especially relevant from both a computer forensics and a privacy concern standpoint.

The main contribution of this paper is the exposition of a method to recover from an iOS device the contents and metadata of documents printed through AirPrint, even in modern devices which feature hardware-based data encryption. Analyzing the behavior of AirPrint posed an interesting challenge since iOS is a closed operating system and lacks public documentation about many internal aspects. In addition, even when the mobile threat landscape has been covered by other authors [6], there seems to be no additional research on how AirPrint works behind the scenes and the forensic traces it may leave. Several authors [7, 8] have reviewed the existing (mostly commercial) forensic investigation tools for iOS devices; however, analysis of AirPrint activity does not seem to be covered by any of the existing software solutions.

This paper is structured as follows. First of all, in Section 2, AirPrint and its mode of operation both from a user's and technical standpoints are presented. The analysis of the traces left by AirPrint and the information they contain is shown in Section 3. In Section 4, basic experiments are performed to assess the recoverability of the AirPrint traces in devices where encryption has been purposely disabled. Following, in Section 5, the recoverability is evaluated for the modern devices and OS versions that feature data encryption. Finally, concluding the paper, Section 6 summarizes the paper contributions and outlines further work.

2. Description of AirPrint Network Printing

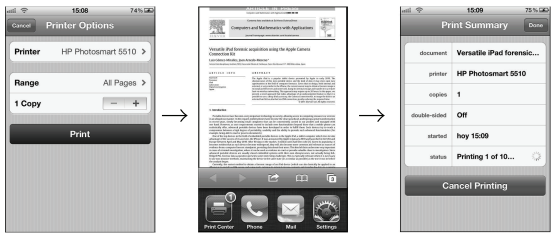
Briefly explained, AirPrint is an iOS feature that allows applications to send content to printers using the iOS device's wireless connection. Apple is directly quoted [1]: "*AirPrint automatically finds printers on local networks and can print text, photos and graphics to them wirelessly over Wi-Fi without the need to install drivers or download software.*"

Apple announced AirPrint in September 2010. Two months later, iOS 4.2 was released for the iPhone, iPad, and iPod Touch, being the first iOS version to offer this feature to users. Its standard functionality allows printing only to specific, AirPrint-enabled printers. Nevertheless, as of July 2013 there were more than seven hundred AirPrint-enabled printer models in the market from sixteen different vendors [9]. Apple does not support sharing a common printer via the computer it is connected to, even when it was possible with some Mac OS X 10.6.5 beta versions; however, it can be done by using software tweaks such as AirPrintActivator [10].

Long before the introduction of AirPrint, different solutions [11, 12] tried to fill in this gap. Usually, such solutions involved iOS applications capable of opening different file formats and sending them to a desktop computer, running a companion application, which would in turn send the document to the printer itself. Some printer vendors developed specific clients; however, none of these solutions were ever widely spread among users. Currently, with AirPrint working out-of-the-box and embedded into all applications, it is hard to believe that new users will consider using a specific, usually paid application to handle printing, except maybe in some very particular environments, such as cases where the use of these kind of applications was consolidated before AirPrint was launched or some advanced capabilities are required by power users.

From a user's standpoint, Figure 1 summarizes the printing process, showing the screens it is actually possible to interact with, as seen on an iPhone.

In the client side (iOS device), AirPrint-enabled applications contain a "Print" button that, when pressed, will present an extremely simple menu (Figure 1(a)), with only two or three available options:



Printer HP Photosmart 5510
Range All Pages
1 Copy

document Versatile iPad forensic...
printer HP Photosmart 5510
copies 1
double-sided Off
started hoy 15:09
status Printing 1 of 10...

(a) (b) (c)

FIGURE 1: Step-by-step AirPrint options screen on an iPhone.

(1) *Printer:* this option opens a list of all AirPrint-enabled printers found in the local network, showing a "name" and "description" field for each one.

(2) *Range:* (optional) defaulting to "all pages", this option opens a selector which allows the user to choose a range of pages to be printed rather than all the document.

(3) *Copies:* this option specifies the number of copies to be printed.

(4) Depending on the printer features, additional parameters such as duplex printing can be controlled.

(5) *Print:* this button proceeds to send the job to the printer.

The user cannot specify any other kind of information usually available in printing menus, such as paper size or orientation and printing quality. Everything is automatically handled by AirPrint, using some default options. When the user chooses to "Print" the job, the device shows some brief messages ("*Contacting Printer*"; "*Preparing page (. . .) of (. . .)*"; "*Sending to Printer*"). However, depending on how long the print job is, these messages may be barely visible or last for several seconds.

After the job has been sent to the printer, the printing menu disappears and the application returns to its previous state. At this point, invoking the list of recently used applications, by double-clicking the device "Home" button, reveals a *Print Center* application (Figure 1(b)). Unless the user somehow knows this application has started running in background, it may be difficult for him to find it, since no active feedback is provided during the printing process, thus being invisible at casual glance.

Opening the *Print Center* application, the user can see the list of running and pending printing jobs, check their details, and cancel them (Figure 1(c)). When the last job finishes, that is, the moment the printer ejects the last page, the application closes and does not appear anymore in the list of recently used applications. As far as it is known, there is no way to open the *Print Center* as a standalone application. It is only executed while there are jobs being printed.

From a technical standpoint, the AirPrint service is known to use the standard IPP protocol at network level for printer management, and Bonjour/Zeroconf [13] for service discovery. A comprehensive description of the printing architecture and its underlaying API in iOS devices can be found in [14].

3. Forensic Traces Left by AirPrint

This section presents the preliminary information that must be considered before more in-depth forensic investigation may proceed. Mainly, it is important to assess whether any traces are left in the device after having printed a document using AirPrint, and if so, how they can be discovered, how they behave, and which useful information can be extracted from them. All this information was discovered through some basic experiments.

3.1. Preliminary Setup. For the experiments described in this section, the following equipment was used:

(i) iOS device #1: Apple iPhone 4, 16 GB (model `A1332`) running iOS 4.3.3 (`8J2`) and 5.0 (`9A334`), both jailbroken using the `redsn0w` software [15].

(ii) iOS device #2: Apple iPhone 3G, 8 GB (model `A1241`) running iOS 4.2.1 (`8C148`), jailbroken using `redsn0w` and enabling multitasking and AirPrint.

(iii) Desktop computer information: Apple MacBook Pro (model `MacBookPro5,1`) running Mac OS X 10.7.2 (`11C74`).

(iv) Printer: HP Photosmart 5510 B111a (model `CQ176B`) running firmware version `EPL2CN1122AR`.

(v) Wireless connectivity: all devices were connected to a wireless 802.11g network to perform the experiments.

(vi) Physical connectivity: all devices were using physical wires for AC power only.

3.2. The "Jailbreak" Process. Given that iOS enforces the device to run only code signed by Apple (downloaded from the App Store), during our experiments we used the "jailbreak" technique to bypass that restriction in order to have full access to the devices and be able to run shell commands on them. The jailbreak process is exempted from prosecution under the anticircumvention section of the U.S. Digital Millenium Copyright Act [16], and it has been very useful for forensic research in the past [17, 18].

Both iOS devices were jailbroken in order to gain full access and install an SSH server and basic UNIX tools.

3.3. Traces Found. Once we were able to execute code in the device, we invoked a series of commands before, during, and after printing, and we compared the results looking for remarkable differences. The most relevant commands used were:

(1) `find / -type (b,c,d,f,l,p,s)` for listing, respectively, all the block special devices, character special devices, directories, regular files, symbolic links, FIFOs, and sockets, in the filesystem.

(2) `netstat -an -f inet` for listing any current network connections. This would show active client-server activity as well as inactive servers awaiting for incoming petitions.

(3) `ps aux` for getting information about running processes.

By reviewing the lists of files and directories generated with the `find` commands explained above, it was observed that when a device prints via AirPrint for the first time the following folder is created:

```
/var/mobile/Library/com.apple.printd/
```

We observed that everytime a document is sent to a printer, a new file named `1.pdf` is created under this folder. With additional tests, it was observed that this PDF file exists in disk only while the document is being printed. The moment the printer ejects the last page and considers the job finished, the PDF file is deleted. This is also the moment at which the *Print Center* application disappears from the list of recently used applications.

By printing some documents and copying the resulting temporary PDF files to another location before their deletion and then examining them we observed that these files are regular PDF files with the same content that is being sent to the printer (no matter whether it was originally in PDF format or not). Hence, an obvious trace is being left in the filesystem, and it reflects exactly what was printed.

It must be noted that, in some of the preliminary tests, before the physical printer was available, we set up a virtual PDF printer in a Mac computer and shared, making it look like an AirPrint printer. It worked as expected and it was possible to print to it from an iPhone; however, the printing of the document (in this case, the generation of a file in the hard drive of the Mac) was much shorter than the actual printing of a page through a real printer with real ink and paper. We observed that this greatly reduces the chances of the temporary files being flushed to physical storage from the buffer cache in the iOS device before deletion and thus their chances of recoverability. Therefore, to obtain accurate results, the experiments need to be performed with a real printer.

3.4. Properties of the AirPrint Temporary Files. From the execution of the different tests we extracted the following conclusions. Unless otherwise specified, every finding applies to all available iOS versions with AirPrint support (versions 4.2 through 6.1 and possibly later versions as well).

(1) For every print job sent via AirPrint, a file with the name of `1.pdf` is created in the directory

`/var/mobile/Library/com.apple.printd/`

(2) This file is in PDF format, containing the document sent to the printer. This observed behavior is consistent across internal iOS applications (Mail, Safari) as well as third party ones (GoodReader, Papers, Keynote, . . .). The only exception found is the iOS Photos app, which seems not to generate

any temporary files on disk when printing, thus not leaving these traces.

(3) The file `1.pdf` is deleted as soon as the printing job finishes. The timing observed indicates that this happens not just after finishing the task of submitting the job to the physical printer, but after the document has been completely printed.

(4) When one job is being printed, subsequent jobs arriving to the queue generate files named `2.pdf`, `3.pdf`, and so forth. The behavior observed suggests that in iOS 4 the counter resets as soon as the queue is empty (if a new job arrives later it will be named `1.pdf` again), whereas starting from iOS version 5 the counter seems to keep increasing (each new job gets a higher number even if the queue is empty) until the device reboots.

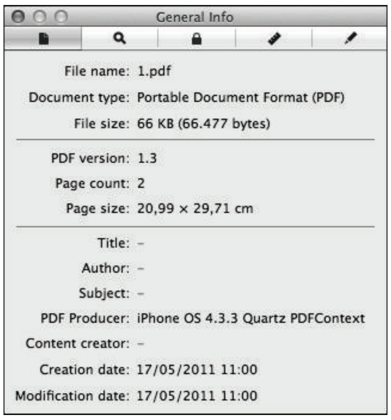
(5) When a job asks for more than one copy of the same document, the temporary PDF file contains only one copy of it. The information on the number of copies to be printed is being sent to the printer in a separate channel (standard PS commands or similar).

(6) When a page range is specified, the temporary PDF file contains only this page range. There is an exception when some applications print files that are themselves PDFs, which is studied later in Section 3.5.

3.5. PDF Metadata of the Temporary Files. Using a standard PDF reader, the metadata contents inside different temporary files generated by AirPrint were extracted and compared. The use of document metadata in forensic investigations has proven useful in different scenarios before [19–22]. A good analysis of the PDF format itself from a forensics point of view, considering its security and privacy aspects, can be found in [23].

Generally, all the temporary PDF files created by AirPrint can be identified as such by their metadata: they all show the same "PDF producer" entry ("*iPhone OS x.y.z Quartz PDFContext*," with *x.y.z* being the iOS version number), and the creation and modification dates both indicate the date and time when the document was sent to the printer (see Figure 2). Therefore, knowing what has been printed from a device looks as simple as recovering deleted PDF files from it and focusing on those with the strings "*iPhone OS x.y.z Quartz PDFContext*." Moreover, the creation and modification dates contained inside those PDF files in the form of PDF metadata will indicate precisely the printing date and time.

Only one exception was found to this behavior. When printing PDF files, that is, when the content to be printed is itself in PDF format, some applications behave like described earlier, but others (including iOS built-in applications such as Safari or Mail) actually copy the original PDF file to the `com.apple.printd` directory as `1.pdf` instead of generating a new PDF file with fresh metadata. In these cases, PDF metadata cannot be used to tell whether one given file is a trace from AirPrint printing: the only peculiar thing about that PDF file is the fact that it resides under such directory.



File name: 1.pdf
Document type: Portable Document Format (PDF)
File size: 66 KB (66.477 bytes)
PDF version: 1.3
Page count: 2
Page size: 20,99 × 29,71 cm
Title: –
Author: –
Subject: –
PDF Producer: iPhone OS 4.3.3 Quartz PDFContext
Content creator: –
Creation date: 17/05/2011 11:00
Modification date: 17/05/2011 11:00

FIGURE 2: Metadata of one temporary PDF file generated by AirPrint, as shown by the OS X "Preview" tool.

Considering this, an eventual automated tool aimed at recovering the traces left by AirPrint should go further than just carving for PDF files: it should correctly interpret the internals of the HFSX filesystem and tell whether each recovered file existed within the `com.apple.printd/` directory or somewhere else.

4. Recoverability of AirPrint Traces without iOS Data Protection

Given that the temporary PDF files generated by AirPrint only exist in the filesystem for a limited time and are deleted when the document has been printed, the possibility that, depending on disk scheduling and other factors, these files might never be actually flushed to the physical disk must be considered. In that case, they are unrecoverable by a subsequent forensic analysis. This section studies such possibility and assesses the probability that a given trace may be actually recovered at a later time.

4.1. Preliminary Setup. For the experiments described in this section, the following equipment was used:

(i) iOS device: Apple iPhone 3G, 8 GB (model `A1241`) running iOS 4.2.1 (`8C148`), jailbroken using `redsn0w` and enabling multitasking and AirPrint.

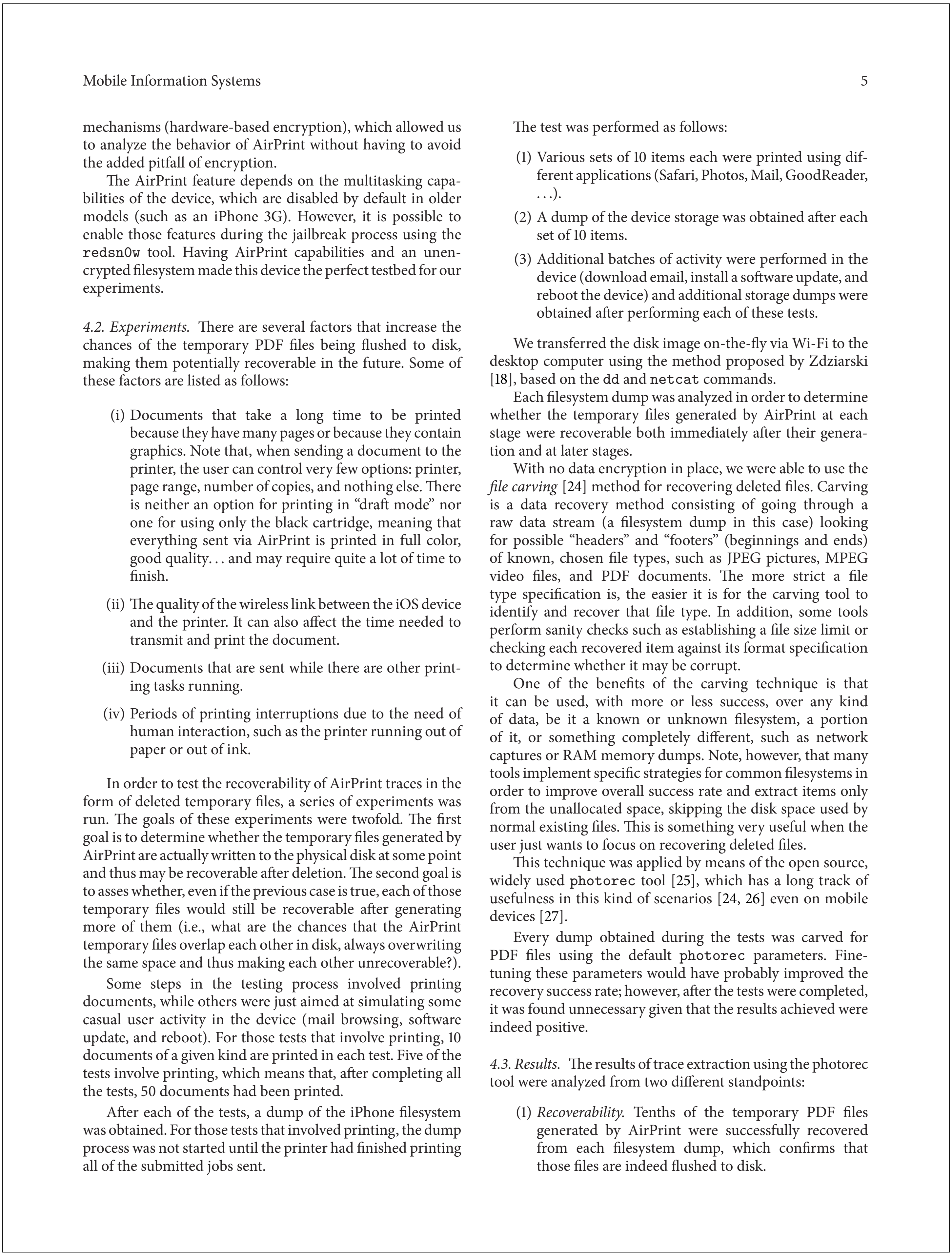
(ii) Desktop computer information: Apple MacBook Pro (model `MacBookPro5,1`) running Mac OS X 10.7.2 (`11C74`).

(iii) Printer: HP Photosmart 5510 B111a (model `CQ176B`) running firmware version `EPL2CN1122AR`.

(iv) Wireless connectivity: all devices were connected to a wireless 802.11g network to perform the experiments.

(v) Physical connectivity: all devices were using physical wires for AC power only.

By using older equipment (an iPhone 3G) in this set of experiments, we had a device without iOS "data protection"



mechanisms (hardware-based encryption), which allowed us to analyze the behavior of AirPrint without having to avoid the added pitfall of encryption.

The AirPrint feature depends on the multitasking capabilities of the device, which are disabled by default in older models (such as an iPhone 3G). However, it is possible to enable those features during the jailbreak process using the `redsn0w` tool. Having AirPrint capabilities and an unencrypted filesystem made this device the perfect testbed for our experiments.

4.2. Experiments. There are several factors that increase the chances of the temporary PDF files being flushed to disk, making them potentially recoverable in the future. Some of these factors are listed as follows:

(i) Documents that take a long time to be printed because they have many pages or because they contain graphics. Note that, when sending a document to the printer, the user can control very few options: printer, page range, number of copies, and nothing else. There is neither an option for printing in "draft mode" nor one for using only the black cartridge, meaning that everything sent via AirPrint is printed in full color, good quality. . . and may require quite a lot of time to finish.

(ii) The quality of the wireless link between the iOS device and the printer. It can also affect the time needed to transmit and print the document.

(iii) Documents that are sent while there are other printing tasks running.

(iv) Periods of printing interruptions due to the need of human interaction, such as the printer running out of paper or out of ink.

In order to test the recoverability of AirPrint traces in the form of deleted temporary files, a series of experiments was run. The goals of these experiments were twofold. The first goal is to determine whether the temporary files generated by AirPrint are actually written to the physical disk at some point and thus may be recoverable after deletion. The second goal is to asses whether, even if the previous case is true, each of those temporary files would still be recoverable after generating more of them (i.e., what are the chances that the AirPrint temporary files overlap each other in disk, always overwriting the same space and thus making each other unrecoverable?).

Some steps in the testing process involved printing documents, while others were just aimed at simulating some casual user activity in the device (mail browsing, software update, and reboot). For those tests that involve printing, 10 documents of a given kind are printed in each test. Five of the tests involve printing, which means that, after completing all the tests, 50 documents had been printed.

After each of the tests, a dump of the iPhone filesystem was obtained. For those tests that involved printing, the dump process was not started until the printer had finished printing all of the submitted jobs sent.

The test was performed as follows:

(1) Various sets of 10 items each were printed using different applications (Safari, Photos, Mail, GoodReader, . . .).

(2) A dump of the device storage was obtained after each set of 10 items.

(3) Additional batches of activity were performed in the device (download email, install a software update, and reboot the device) and additional storage dumps were obtained after performing each of these tests.

We transferred the disk image on-the-fly via Wi-Fi to the desktop computer using the method proposed by Zdziarski [18], based on the `dd` and `netcat` commands.

Each filesystem dump was analyzed in order to determine whether the temporary files generated by AirPrint at each stage were recoverable both immediately after their generation and at later stages.

With no data encryption in place, we were able to use the *file carving* [24] method for recovering deleted files. Carving is a data recovery method consisting of going through a raw data stream (a filesystem dump in this case) looking for possible "headers" and "footers" (beginnings and ends) of known, chosen file types, such as JPEG pictures, MPEG video files, and PDF documents. The more strict a file type specification is, the easier it is for the carving tool to identify and recover that file type. In addition, some tools perform sanity checks such as establishing a file size limit or checking each recovered item against its format specification to determine whether it may be corrupt.

One of the benefits of the carving technique is that it can be used, with more or less success, over any kind of data, be it a known or unknown filesystem, a portion of it, or something completely different, such as network captures or RAM memory dumps. Note, however, that many tools implement specific strategies for common filesystems in order to improve overall success rate and extract items only from the unallocated space, skipping the disk space used by normal existing files. This is something very useful when the user just wants to focus on recovering deleted files.

This technique was applied by means of the open source, widely used `photorec` tool [25], which has a long track of usefulness in this kind of scenarios [24, 26] even on mobile devices [27].

Every dump obtained during the tests was carved for PDF files using the default `photorec` parameters. Fine-tuning these parameters would have probably improved the recovery success rate; however, after the tests were completed, it was found unnecessary given that the results achieved were indeed positive.

4.3. Results. The results of trace extraction using the photorec tool were analyzed from two different standpoints:

(1) *Recoverability.* Tenths of the temporary PDF files generated by AirPrint were successfully recovered from each filesystem dump, which confirms that those files are indeed flushed to disk.

(2) *Persistence.* The artifacts created during our first tests were still recoverable after the last tests. This suggests that, probably due to iOS file allocation strategies, the temporary files generated by AirPrint are not likely to overwrite each other.

The results of these tests, which can be seen in more detail in [2], show that under iOS 4 and with no data encryption in place the temporary files generated by AirPrint are indeed written to disk and are potentially recoverable even after rebooting the device or turning it off.

Considering that the artifacts are stored in unallocated space, it is unavoidable that using the device for long periods of time reduces the chance of recovering such artifacts, as new data stored in the device may overwrite that disk space. However, the tests show that the probability does not decrease very quickly, and there is at least a good chance to recover most of the traces.

5. Impact of iOS Data Protection on the Recoverability of AirPrint Traces

In this section we describe a new set of experiments aimed at assessing whether the traces left by AirPrint in the device's filesystem can be recovered even when modern iOS data encryption mechanisms are in place.

5.1. Preliminary Setup. The experiments described in this section were carried out using the equipment described below:

(i) iOS device: Apple iPhone 3GS, 32 GB (model `A1303`) running iOS 6.1 (10B141), jailbroken using the `evasi0n` software [28].

(ii) Desktop computer: Apple iMac (model `iMac13,2`) running OS X 10.8.4 (`12E55`).

(iii) Printer: HP Photosmart 5510 B111a (model `CQ176B`) running firmware version: `EPL2CN1122AR`.

(iv) Wireless connectivity: all three devices were connected to a wireless 802.11g network to perform the experiments.

(v) Physical connectivity: the printer and desktop computer used physical wires for AC power. In addition, the iOS device was connected most of the time to the desktop computer using the standard Apple USB to 30-pin cable; this powered the device and served as the transmission channel when dumping the device internal storage to the desktop computer.

5.2. Mechanisms to Bypass iOS Data Protection. As we introduced in previous sections, all current iOS devices offer hardware-based encryption, backed with software support at the OS and application level. As noted by Casey et al. [29], "the increasing use of full disk encryption can significantly hamper digital investigations, potentially preventing access to all digital evidence in a case."

The data encryption mechanisms that Apple calls "data protection" were introduced in iOS version 4 (June 2010) and stood publicly unbreakable for nearly one year until, in May 2011, a software firm announced a product capable of bypassing this encryption [30]. This product is restricted to "*established law enforcement, intelligence and forensic organizations as well as select government agencies.*"

At the same time, Bedrune and Sigwald published [31] details about iOS data protection shortly after they released the tools and source code capable of breaking this encryption [32]. Even if the device is locked with a passcode, the tools include a bruteforce script that runs in the iOS device itself and obtains the user-defined passcode, unless it is an alphanumeric code, something rarely seen in these devices, although supported. As of July 2013, their toolkit has been updated and works successfully with supported devices even under iOS version 6.1.

Nowadays most forensic tools support a similar functionality to one extent or another.

5.3. How iOS Data Protection Affects the Recovery of Deleted Files. One of the features of iOS encryption is that it relies on per-file encryption keys, which means that each file in the filesystem is encrypted using a different key. This, in turn, means that recovering a deleted file is more difficult than just retrieving the portion of disk space where the contents of this file reside: one must also recover the necessary filesystem metadata containing the encryption key for that given file.

For this reason, commercial tools, even when able to defeat encryption, do not recover deleted files so far. Instead, these tools usually opt for (a) examining the device backups stored in iTunes in a local computer rather than device itself or (b) just query the device for as much information as possible using standard APIs; for instance, get every sent and received SMS from the Messages application, list recent lookups from the Maps application, or query the Phone application for the recent calls log. Both these approaches, however, overlook any deleted data in the device, skipping a lot of information that could be relevant to the forensic investigator.

Given that the file contents are encrypted, the carving technique cannot be used to look for files (allocated or not). However, a smarter tool looking for deleted file/directory entries in the HFSX filesystem should succeed at recovering these files, even when their contents are encrypted.

Bedrune and Sigwald's `ios_examiner` tool [32] recently incorporated an `undelete` function which applies a novel technique [33] based on using the additional data stored in the filesystem's transaction journal in order to improve the recovery results. This tool is still very recent and under improvement, but it is expected that commercial forensic application developers may include it in later versions of their products or, at least, use similar techniques to allow forensic tools to analyze an encrypted filesystem.

5.4. Adapting the `Iphone-Dataprotection` *Toolkit.* As we started new experiments, we observed that it was certainly possible to recover deleted files from our test devices using the `iphone-dataprotection` toolkit. However, only certain file types were recovered (JPG pictures, SQLite databases, XML files, . . .), whereas no PDF files were recovered at all.

```
magics = ["SQLite", "bplist", "<?xml", "\xFF\xD8\xFF", "\xCE\xFA\xED\xFE",
        "\x89PNG", "\x00\x00\x00\x1CftypM4A", "\x00\x00\x00\x14ftypqt", "\x25PDF-"]
```

Algorithm 1

```
knownExtensions = (".m4a", ".plist", ".sqlite", ".sqlitedb", ".jpeg", ".jpg",
                ".png", ".db", ".json", ".xml", ".sql", ".pdf")
```

Algorithm 2

There are two modifications that must be applied to the software to have it recover PDF files.

The undelete algorithm used by the tool considers that a file is correctly recovered only if the initial bytes of the file match a given set of patterns. The stock list includes a limited set: SQLite databases, XML files, binary property lists, JPEG pictures, Mach-O executable binaries, PNG graphics, and M4A audio files. In order to have the tool recover deleted PDF files, the file signature of the PDF type must be added in `hg/python_scripts/hfs/journal.py` (lines 58-59) as shown in Algorithm 1.

In order to have these files stored in a separate directory, we declare `.pdf` as a known extension by modifying `hg/python_scripts/nand/carver.py` (line 119) as shown in Algorithm 2.

After performing this modification we observed that the `ios_examiner` tool successfully recovered (where technically possible) deleted PDF files. In fact it is even possible to acquire one or more dumps of the device's internal storage using the original (unmodified) tool, apply our described modifications afterwards, and then run the modified tool against the acquired images to recover any deleted PDF files they might contain, some of which can be traces left by the use of AirPrint, whereas others will be regular PDF files that have been deleted or reallocated in disk for whatever reason.

Given a set of recovered PDF files, we wrote a Perl script that outputs a CSV table indicating which of the files correspond indeed to contents printed through AirPrint, and if so, when were the documents printed and under which iOS version as shown in Algorithm 3.

5.5. Experiments. In this series of experiments we wanted to verify whether AirPrint traces were recoverable in scenarios where iOS data protection is enabled and analyze how the amount of free disk space affects the recoverability rate.

A detailed description of the whole testing process follows:

(1) Fill the device with some applications (GoodReader, plus Apple's Podcasts, iBooks, iTunes U, Find My Friends, and Find My iPhone) and multimedia content. Setup an iCloud account for activating the Find My iPhone service and start syncing email, contacts, calendars, reminders, Safari tabs, notes, photos, documents, and data.

(2) After a period of 24 hours for any massive syncing activity to take place, the device storage as reported in "Settings" is 4.1 GB available of a total of 28.3 GB (i.e., 15% free space).

(3) Reboot the device to start from a clean state.

(4) Use the GoodReader application to print a fixed set of 20 documents amounting to a total of 109 paper pages; send each document to the printer only when the previous one has been completely printed.

(5) Turn the device off and acquire forensic image #1.

(6) Boot the device. Remove all optional Apple applications as well as multimedia content (audio, video) and iCloud accounts added in step (1).

(7) After a period of 24 hours for any deletion activity to take place, the device storage as reported in "Settings" is 27.2 GB available of a total of 28.3 GB (i.e., 96% free space).

(8) Reboot the device to start from a clean state.

(9) Repeat step (4).

(10) Turn the device off and acquire forensic image #2.

(11) Recover AirPrint traces from both images and compare the results.

5.6. Results. Table 1 shows the results for each individual file. In each case, we were able to recover between 5% and 10% of the documents printed through AirPrint, extracting the following conclusions:

(i) It is possible to recover the full content of documents printed through AirPrint as well as relevant metadata such as print date and iOS version. In some cases the recovered PDF files may be corrupt but still contain details such as iOS version used and print date.

(ii) The low recoverability rate observed (5–10% in realistic scenarios) could be due to the disk scheduling algorithm used in the iOS operating system (in this particular version at least). This would also explain the fact that the success rate keeps constant regardless of the amount of free disk space.

(iii) At any particular iOS version (existing or future), a change in the disk scheduling subsystem could boost the success rate significantly. Additional work

```
#!/usr/bin/perl
print "Filename,iOS version,Print date\n";
while( $file = shift(@ARGV) ) {
  $ios = ”;
  $date = ”;
  $pdfinfo = ‘pdfinfo $file 2>&1’;
  @metadata = split( /\n/, $pdfinfo );
  if( @metadata[0] =˜ m/iPhone OS.* Quartz PDFContext/ ) {
    ($ios = @metadata[0]) =˜ s/.*iPhone OS ([0-9.]+) Quartz PDFContext/iOS \1/;
    ($date = @metadata[1]) =˜ s/CreationDate: //;
    print "$file,$ios,$date\n";
  } else { print "$file,does not look like an AirPrint temporary file.\n"; }
}
```

Algorithm 3

Table 1: Recoverability of AirPrint temporary artifacts under iOS 6.

#	File	Size (bytes)	Size (pages)	Recovered?
1	000001.DOC	40.960	2	In image #1
2	000002.DOC	57.856	2	In image #2 and partially in #1
3	000003.DOC	55.808	2	No
4	000004.DOC	175.616	4	No
5	000005.DOC	180.736	5	No
6	000006.DOC	67.584	5	No
7	000007.DOC	179.200	30	No
8	000008.PPT	302.592	12	No
9	000009.PDF	39.586	4	No
10	000010.PDF	120.441	4	No
11	000011.PDF	31.367	1	No
12	000012.PDF	22.857	6	No
13	000013.PDF	38.638	2	No
14	000015.PDF	55.964	2	No
15	000016.PDF	150.586	4	No
16	000018.PDF	94.424	9	No
17	000019.PDF	124.152	3	No
18	000020.PDF	4.755	2	No
19	000021.PDF	4.521	2	No
20	000022.PDF	21.235	8	No

is needed to assess whether the results observed are kept consistent across different device models and iOS versions.

The traces of the printing activity from step (9) should be more easily recoverable given that in step (7) we tried to improve the recoverability rate by freeing most disk space (to reduce the probability that some new file, log entry, etc. could overwrite the AirPrint traces once they've been deleted) and by reducing most of the device's background activity (iCloud syncing, e-mail activity, . . .). Hence, the second image should contain a higher number of AirPrint traces than the first one.

In contrast to what could be reasonably expected, we observed that in each case we recovered only one or two AirPrint temporary files out of the 20 possible. It could be thought that only the latest jobs are being recovered; however the traces we found corresponded to the first jobs sent to the printer rather than the last ones.

We performed additional experiments introducing circumstances such as print interruptions due to lack of paper and loss of network link between the device and the printer. In such circumstances, the temporary files remain much longer in the iOS device's filesystem and will persist for an undetermined amount of time (even some time after rebooting the device). Under these conditions we saw the success rate increase to 15%.

6. Conclusions and Future Work

This paper analyzes the forensic traces left by usage of the AirPrint functionality in iOS based devices. We have developed a method which leverages publicly available tools to recover from an iPhone or iPad the contents of documents that have been printed using the standard AirPrint feature. The recovery of these artifacts can be valuable from the point of view of a forensic investigation in scenarios such as information leak or distribution of inadequate content; however it could also pose a privacy risk to the user community.

The traces described could persist even after the original file has been deleted, or if the original file resides inside some "vault-type" application which protects its contents on disk with an additional layer of encryption.

With modern iOS 6 data encryption mechanisms in place, the described method still succeeded in recovering between 5 and 15% of the documents printed through AirPrint. We believe the success rate can depend on factors such as the disk scheduling strategy, and thus different iOS versions could throw different results.

Considering the use case of AirPrint in domestic scenarios, probably home users will not be particularly concerned about this finding, a possible exception to this being explicit graphic content. In this aspect it is interesting to note that Apple's stock Photos application specifically did not generate, in our experiments, the temporary files described in this paper, whereas other 3rd party applications offering added security to store this kind of information are likely to generate the standard AirPrint traces when printing documents. As a general solution, in order to limit the possibility of recovering data from the filesystem, some techniques aimed at performing a secure deletion could be adopted, such as the one presented in [34] for the Android OS.

Further work must be carried out to assess whether it is possible to capture the network traffic generated while printing using techniques similar to [35] and recover the contents of the documents being printed. It would also be interesting to extend the research presented in this paper across a wider range of devices, iOS versions, and 3rd-party applications and to examine if similar issues affect other printing solutions for iOS devices and other mobile devices.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was partly funded by the Spanish Government through projects TIN2011-27076-C03-02 "CO-PRIVACY" and SMARTGLACIS (TIN2014-57364-C2-2-R).

References

[1] Apple, *Apple's AirPrint Wireless Printing for iPad, iPhone & iPod touch Coming to Users in November, 2010*, Apple, 2010, http://www.apple.com/uk/pr/library/2010/09/15airprint.html.

[2] L. Gomez-Miralles and J. Arnedo-Moreno, "Analysis of the forensic traces left by airprint in apple iOS devices," in *Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA '13)*, pp. 703–708, IEEE, Barcelona, Spain, March 2013.

[3] O. Bohl, S. Manouchehri, and U. Winand, "Mobile information systems for the private everyday life," *Mobile Information Systems*, vol. 3, no. 3-4, pp. 135–152, 2007.

[4] S. Caballé, F. Xhafa, and L. Barolli, "Using mobile devices to support online collaborative learning," *Mobile Information Systems*, vol. 6, no. 1, pp. 27–47, 2010.

[5] T. Cook, Apple WorldWide Developers Conference keynote, 2013, http://www.apple.com/apple-events/june-2013/.

[6] A. Castiglione, R. de Prisco, and A. de Santis, "Do you trust your phone?" in *E-Commerce and Web Technologies*, vol. 5692 of *Lecture Notes in Computer Science*, pp. 50–61, Springer, Berlin, Germany, 2009.

[7] A. Hay, D. Krill, B. Kuhar, and G. Peterson, "Evaluating digital forensic options for the apple iPad," in *Advances in Digital Forensics VII*, vol. 361 of *IFIP Advances in Information and Communication Technology*, pp. 257–273, Springer, Boston, Mass, USA, 2011.

[8] A. Levinson, B. Stackpole, and D. Johnson, "Third party application forensics on Apple mobile devices," in *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS '11)*, pp. 1–9, January 2011.

[9] Apple, *iOS: AirPrint 101*, Apple, 2011, http://support.apple.com/kb/ht4356.

[10] Netputing, *AirPrint Activator*, 2011, http://netputing.com/airprintactivator.

[11] Avatron Software Inc, *Print Sharing*, 2011, http://avatron.com/apps/print-sharing/.

[12] EuroSmartz, PrintCentral for iPad, iPhone or iPod Touch, 2012, http://mobile.eurosmartz.com/products/printcentral.html.

[13] D. Steinberg and S. Cheshire, *Zero Configuration Networking: The Definitive Guide*, O'Reilly, 2005.

[14] Apple Inc, How Printing Works in iOS, 2011, https://developer.apple.com/library/ios/.

[15] iPhone Dev Team, redsn0w, 2011, http://blog.iphone-dev.org/post/5239805497/tic-tac-toe/.

[16] US Copyright Office, *Rulemaking on Exemptions from Prohibition on Circumvention of Technological Measures that Control Access to Copyrighted Works*, US Copyright Office, 2010, http://www.copyright.gov/1201/2010/.

[17] J. R. Rabaiotti and C. J. Hargreaves, "Using a software exploit to image RAM on an embedded system," *Digital Investigation*, vol. 6, no. 3-4, pp. 95–103, 2010.

[18] J. Zdziarski, *iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*, O'Reilly Media, 2008.

[19] F. Buchholz and E. Spafford, "On the role of file system metadata in digital forensics," *Digital Investigation*, vol. 1, no. 4, pp. 298–309, 2004.

[20] A. Castiglione, A. De Santis, and C. Soriente, "Taking advantages of a disadvantage: digital forensics and steganography using document metadata," *Journal of Systems and Software*, vol. 80, no. 5, pp. 750–764, 2007.

[21] A. J. Clark, "Document metadata, tracking and tracing," *Network Security*, vol. 2007, no. 7, pp. 4–7, 2007.

[22] M. S. Olivier, "On metadata context in database forensics," *Digital Investigation*, vol. 5, no. 3-4, pp. 115–123, 2009.

[23] A. Castiglione, A. De Santis, and C. Soriente, "Security and privacy issues in the portable document format," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1813–1822, 2010.

[24] M. I. Cohen, "Advanced carving techniques," *Digital Investigation*, vol. 4, no. 3-4, pp. 119–128, 2007.

[25] CGSecurity, *PhotoRec, Digital Picture Recovery*, 2009, http://www.cgsecurity.org/wiki/PhotoRec.

[26] S. L. Garfinkel, "Forensic feature extraction and cross-drive analysis," *Digital Investigation*, vol. 3, pp. 71–81, 2006.

[27] I. Pooters, "Full user data acquisition from Symbian smart phones," *Digital Investigation*, vol. 6, no. 3-4, pp. 125–135, 2010.

[28] Y. D. Wang, N. Bassen et al., evasi0n, 2013, http://evasi0n.com/.

[29] E. Casey, G. Fellows, M. Geiger, and G. Stellatos, "The growing impact of full disk encryption on digital forensics," *Digital Investigation*, vol. 8, no. 2, pp. 129–134, 2011.

[30] ElcomSoft, ElcomSoft investigates iPhone hardware encryption, provides enhanced forensic access to protected, 2011, http://www.elcomsoft.com/PR/eppb110524en.pdf.

[31] J. B. Bedrune and J. Sigwald, *iPhone Data Protection in Depth*, HITB, Amsterdam, The Netherlands, 2011.

[32] J. B. Bedrune and J. Sigwald, iPhone data protection tools, 2011, http://code.google.com/p/iphone-dataprotection/.

[33] A. Burghardt and A. J. Feldman, "Using the HFS+ journal for deleted file recovery," *Digital Investigation*, vol. 5, supplement, pp. S76–S82, 2008.

[34] A. Castiglione, G. Cattaneo, G. De Maio, and A. De Santis, "Automatic, selective and secure deletion of digital evidence," in *Proceedings of the 6th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA '11)*, pp. 392–398, October 2011.

[35] A. Castiglione, G. Cattaneo, G. de Maio, and A. de Santis, "Forensically-sound methods to collect live network evidence," in *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA '13)*, pp. 405–412, Barcelona, Spain, March 2013.

Appendix F

Lockup: A software tool to harden iOS by disabling default *Lockdown* services

L. Gómez-Miralles, J. Arnedo-Moreno. **Lockup: A software tool to harden iOS by disabling default *Lockdown* services**. Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2015, pp. 718 - 723. DOI: 10.1109/3PGCIC.2015.57.

Lockup: A software tool to harden iOS by disabling default *lockdown* services

Luis Gómez-Miralles, Joan Arnedo-Moreno
Internet Interdisciplinary Institute (IN3)
Universitat Oberta de Catalunya
Av. Carl Friedrich Gauss, 5. Parc Mediterrani de la Tecnologia
08860 Castelldefels (Barcelona), Spain
pope,jarnedo@uoc.edu

***Abstract*—Smartphones and mobile devices nowadays accompany each of us in our pockets, holding vast amounts of personal data. The iOS platform has gained popularity in the last years, in particular in enterprise deployments, due to its supposed higher level of security. Recent research has pinpointed a number of mechanisms that are being abused today in order to compromise the security of iOS devices. In this paper, we present *Lockup*, a proof of concept tool that applies various mitigation measures in order to protect iOS devices against those attacks.**

Keywords: Security, Hardening, Privacy, Apple, iOS, iPhone, iPad

I. Introduction

Smartphones have rapidly become ubiquitous in our life. In barely one decade, these small devices have managed to enter our pockets and, nowadays, accompany us at all times, storing all kinds of personal information - often without the user's knowledge: emails and SMS messages, calendars, address books, to-do lists, history of visited places, photographs, voice memos, etc. Moreover, vendors have already started to produce wearable devices which hold an even closer relation with their users, gathering and quantifying diverse data about their life habits - a tendency that will only grow in the future years with devices such as Apple Watch and Google Glass [1].

The rise of mobile technologies has introduced great changes in the information security landscape. Blackberry, the platform that dominated every corporate environment for years thanks to its security features, failed to keep up with its competitors and by Q2 2014 its market share was below 1% [2]. In contrast, 84.7% of the devices sold in that period were Android devices, and 11.7% were iOS devices. However, when it comes to business environments, 67% of new devices activated in a corporate context during the same period were iOS ones [3].

As it tends to happen with every software product, the iOS operating system and the core applications shipped with it have suffered from a number of vulnerabilities in the past, with different degrees of criticality. The most serious ones, for instance, made it possible for remote websites to gain full control over a device browsing them with MobileSafari, the integrated web browser [4]. Fortunately, many of the vulnerabilities uncovered by researchers have been dully patched by Apple in subsequent iOS versions. However, recently, Zdziarski [5] exposed a number of attack vectors against iOS devices through the abuse of certain background services available in all iOS devices. Under certain conditions, these services can leak all kinds of personal data stored in the device, bypassing the optional backup encryption password, and showing no indication at all to the user.

In this paper, we present an analysis of several mitigation techniques that can be used to reduce the attack surface exposed by these services. As a result of this analysis, we introduce *Lockup*, an accompanying software tool that we have created to implement those measures, some of which are novel and, to our knowledge, have not been implemented before. This tool also serves as a proof of concept that such measure can be deployed in an iOS device.

This paper is structured as follows. Section II provides an overview of the iOS security architecture, and presents the problem of potentially dangerous services that can be abused to extract an enormous amount of user data from the device. Section III discusses a number of possible mitigation strategies that can be applied to enhance the device security. Section IV presents *Lockup*, the software tool that we have developed in order to implement those mitigations. Concluding the paper, Section V summarizes the paper contributions and outlines future work.

II. Security and trust in the iOS environment

In this section, we present an overview of the main components of the iOS security architecture and its trust model, and the existing privacy risks. This will allow us to show that, some of the risks originating from certain weaknesses in the iOS trust model have an impact much higher than expected because of a number of iOS background services, named *lockdown*, which have no known legitimate purpose.

A. *Remote access via device trust relationship*

When it comes to sharing information with an external device (be it a desktop computer, an alarm clock that can play music, a car audio system, etc.) the iOS security model works as follows. Whenever the iOS device is connected via cable to a previously unknown computer (or another external device), it presents a dialog on screen prompting the user whether the computer should be trusted, as seen in Figure

1. Upon receiving the user's consent, both devices create and interchange a series of certificates which from that moment will be used to authenticate each other and initiate a secure, encrypted connection. A pairing record consisting of these certificates is stored in well-known filesystem paths in both the computer and the iOS device. However, There is no way for the user of an iOS device to review the list of external devices he has chosen to trust, or to revoke that trust - other than to reinstall the device completely.

As exposed by [5], [14], a computer that has successfully paired with an iOS device can initiate a connection to it and invoke a number of services exposed via the *lockdown* daemon - even wirelessly and without the user receiving any visual indication. The same can be done from any other computer or device, as long as the pairing record is extracted from the trusted computer.

Unfortunately, a number of the *lockdown* services are designed in such a manner that they may leak significant amounts of personal information, even bypassing the user's backup encryption password. Given that any trusted device (alarm clock, car stereo, etc.) gets a pairing record which gives access to all the services, this can be exploited by either placing malicious devices in common areas (airports, coffee shops...) [14] or compromising trusted devices to steal the pairing record stored in it. Then, those pairing records can be used to establish connections to the iOS device, even over the air through either Wi-Fi or cellular connection, in order to perform surreptitious actions such as deploying malicious software or extracting information from the device.

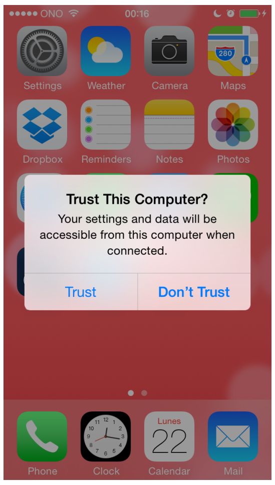
B. *Sensitive iOS device services*

A computer that connects to an iOS device (through a USB cable or Wi-Fi) can invoke a series of services which, from the iOS side, are offered through the *lockdown* daemon [5], [14]. These services have diverse roles, such as allowing iTunes syncing or remote management for MDM purposes, while others have no known purpose and seem to be the perfect backdoors to be exploited by intelligence agencies, forensic products, and malicious actors all alike.

The complete list of services can be explored by checking the file */System/Library/Lockdown/Services.plist*. A fresh installation of iOS 7.1.2 on an iPhone 5 exposes a total of 32 services via *lockdown*, of which Zdziarski [5] identified the following ones as being valuable from a forensic standpoint:

- *com.apple.file_relay.* Can be used to extract huge amounts of information, bypassing the backup encryption setting.
- *com.apple.pcapd.* A network sniffer.
- *com.apple.mobile.MCInstall.* Installs managed configurations, such as the ones used in MDM.
- *com.apple.mobile.diagnostics_relay.* Diagnostics: hardware state, battery level...
- *com.apple.syslog_relay.* Various system logs.
- *com.apple.iosdiagnostics.relay.* Network usage statistics per application.
- *com.apple.mobile.installation_proxy.* Used by iTunes to install applications.
- *com.apple.mobile.house_arrest.* Used by iTunes to transfer documents in and out of applications.
- *com.apple.mobilebackup2.* Used by iTunes to backup the device.
- *com.apple.mobilesync.* Used by iTunes to sync certain data such as Safari bookmarks, notes...
- *com.apple.afc.* Exposes the complete *Media* folder - audio, photographies and videos.
- *com.apple.mobile.heartbeat.* Used to maintain the connection to other services being accessed.

As can be seen, the list includes some potentially dangerous services capable of capturing network traffic, installing applications to the device, or dumping the data stored in the device while bypassing the backup encryption password - all of this without showing any indication to the user.



Trust This Computer?
Your settings and data will be accessible from this computer when connected.
Trust
Don't Trust

Fig. 1. An iPhone prompting the user whether it should trust the connected computer.

III. Mitigation strategies

There are different mitigation measures that can be applied to cope with the weaknesses introduced by the most sensitive iOS services. We try to summarize the most relevant ones.

A. *Delete existing pairing records*

One way to mitigate the problem would be to control the number of trust certificates in the iOS device. This is the approach adopted by the *unTrust* tool [15]: it runs in a computer connected to an iOS device connected via USB and removes all pairing records existing in the device except the one for the computer being used to execute the tool.

One drawback of this approach is that the iOS device still keeps trusting one computer - hence there is still the risk that the pairing record is stolen from the computer and used to connect to the device services. In addition, if the user decides or needs to temporarily trust an external device, away from that computer (such as an audio system), there is no way to revoke that trust or purge the list of trusted devices until the user can get access to the trusted computer and execute *unTrust* again.

B. Limit sensitive services to USB (disable over wireless)

Another approach would be to limit the sensitive services to run only over USB, minimizing the risk for over-the-air attacks. The *lockdown* daemon, responsible for all the sensitive services described in this paper, implements an option (*USBOnlyService*) to limit certain services to USB connections only, disabling the connection to those services over wireless networks. However, this option is only used by one service in iOS 7 (*com.apple.webinspector*, for debugging mobile websites) and one more in iOS 8 (*com.apple.pcapd*, the network sniffer).

C. Disable some services

Finally, it would be ideal to disable the most sensitive services - something that has not been done so far.

The various security measures applied by iOS to any application would prevent us from making these modifications. In order to overcome this, our tool needs to bypass those security measures using the process known as *jailbreak*, as will be explained later.

D. Lock pairing with new devices

Another option worth mentioning is to block pairing with new devices, as implemented by Zdziarski in *pairlock*. This was useful up to iOS 6, given that in those versions external devices would be trusted blindly, without the iOS device presenting any prompt to the user. Since iOS 7 addressed this concern by asking for user permission before trusting new devices, *pairlock* has not been updated to work in iOS 7. Its approach leaves some doors open, as it does not allow the user to revoke existing trust relationships, nor does it address the risk of a pairing record being stolen from a computer or other trusted device.

IV. *Lockup*: IOS HARDENING AND ANTI-FORENSICS

In this paper we present *Lockup*, a software tool that can be installed in devices running iOS versions 7 and 8. As a proof of concept, it *Lockup* hardens the security of the device by addressing the issue of sensitive services using three different approaches:

1) Reducing the attack surface by disabling the most sensitive services. In addition, the user is offered several *profiles*, allowing him to tailor which services are published, and enabling only those needed for the intended use of the device. The rest are eliminated. For instance, a user whose device is not enrolled in MDM systems does not need to allow remote installation of software and configuration profiles.
2) Limiting exploitation opportunities by restricting the rest of services to USB only, eliminating over-the-air threats. This is automatically done in most of the *profiles* mentioned above.
3) Limiting trust relationships by automatically purging all pairing records after a configurable period of time. This constitutes an additional line of defense against attackers capable of stealing a trusted certificate from sources such as the user's computer.

Lockup allows the user to choose between a series of *profiles*, each one increasingly restrictive, depending on what the user needs to do with the device at every moment. In order to define the various *profiles*, we tried a number of configurations, enabling and disabling each service selectively, and attempting various common actions to make an iOS device interact with other external devices. In particular, we tried the following actions:

- Use iTunes in a Mac computer to install applications in the iOS device.
- Use iTunes to transfer files in and out of the applications installed in the iOS device.
- Use iTunes to perform a backup of the data stored in the device.
- Use a bluetooth hands-free device to access the address book of the iOS device and place calls through it.
- Use iPhoto in a Mac computer to import the device's camera roll.
- Use a stereo system to play the audio coming out from the iOS device.

This list illustrates the problems of granting excessive privileges to external devices that access the iOS device's *lockdown* services. If a user does not regularly backup to iTunes, why should those services be exposed when the device is connected to, say, an alarm clock? With *Lockup*, the user can adjust the behavior of the device as needed.

A. Tool capabilities

The main capabilities of *Lockup* can be summarized as follows.

- Controlling the device's trust relationships, by purging the stored pairing records; and
- Disabling certain *lockdown* services and preventing others from being invoked over Wi-Fi connections, in order to disable over-the-air attacks.

One common concern about the potential abuse of iOS services is that iOS lacks a way to see which other devices or computers have been paired with in the past, or to revoke those trust relationships. If the user just hits the wrong button by mistake, the connected device will be trusted forever. This can pose a significant risk, especially considering the possibility of an attacker stealing the pairing record from inside the trusted device and using it to establish remote connections to the iOS device.

In our solution we opted for including a background task that will wipe all trust relationships from the iOS device after a configurable period of time; this applies the mitigation strategy discussed in subsection . Once that happens, connecting to that device will require the user to confirm the trust relationship from the iOS screen. We recommend setting this to low values such as 5-10 minutes, which should suffice for any task involving connection to another device - with the exception of very long synchronization sessions with iTunes, as usually happens when an iOS device is synced with a computer for the first time. During our tests we observed that even values as low as 30 seconds allow normal functioning of standard features such as iTunes syncing; once an iTunes syncing session has started, it will complete successfully even if the pairing record is deleted in the middle of the process.

In addition, as we have introduced earlier in this paper, there are sensitive services potentially very dangerous and with no known purpose (such as *com.apple.mobile.file_relay*, that can be used to extract all kinds of personal information bypassing the backup encryption protection, or *com.apple.pcapd*, which can be used to turn the device into a sniffer that will capture the network traffic it can receive), and it seems obvious to us that these offending services should be removed from every device.

There are also other services that, despite having a legitimate purpose, can also be exploited to leak significant amounts of personal information or inject malicious software into the device. Examples include *com.apple.mobile. installation_proxy* (used by iTunes to install applications in the device), *com.apple.house_arrest* (used by iTunes to copy application files from or to the device) and *com.apple. mobilebackup2* (used by iTunes to backup the data stored in the device).

We propose to define different service levels and keep the device in the most restrictive level that is suitable depending on what the user needs at every moment – a measure that has not been implemented before, to the authors' knowledge. For instance, it is not necessary to keep all the iTunes-related services enabled unless the user wants to connect the device to iTunes, and even then, it is not necessary to expose those services over-the-air if the user uses a cable to sync. Similarly, a lot of users will prefer to disable the MDM-related services, which can be exploited to install software into their devices. This approach applies the mitigation strategies explained in subsections III-B and III-C.

B. Service profiles

Next, we describe the different profiles that we have implemented in *Lockup*, with each profile being increasingly restrictive and consequently more secure. To the knowledge of the authors, a similar solution has not been implemented before, neither in iOS nor in other platforms. In order to decide which services should be disabled in each profile, we have followed two different criteria.

On one hand, the services that we disable first are those that pose a higher privacy risk to the user. These are, for instance: the services that make it possible to bypass the backup encryption password, to capture network traffic, to deploy configuration profiles and applications to the device, etc.

At the same time, the first services that we disable are the ones likely to be needed by a reduced number of users. We first disable the totally unneeded services, afterwards we disable MDM, and then we disable other features that users may need at particular moments (such app installation via iTunes) while we still allow iTunes to obtain backups of the device data.

The following list details which services are disabled in each level. Levels are incremental, meaning that any given level X also applies all the steps performed in previos levels 1, 2, ... X-1.

- Level 1, for MDM - disables:
  - *com.apple.file_relay.*
  - *com.apple.pcapd.*
- Level 2, for synchronizing applications - disables:
  - *com.apple.mobile.MCInstall.*
  - *com.apple.mobile.diagnostics_relay.*
  - *com.apple.syslog_relay.*
  - *com.apple.iosdiagnostics.relay.*
  - Sets all remaining services to *USBOnly*.
- Level 3, for backup - disables:
  - *com.apple.mobile.installation_proxy.*
  - *com.apple.mobile.house_arrest.*
- Level 4, for synchronizing media files - disables:
  - *com.apple.mobilebackup2.*
  - *com.apple.mobilebackup.*
- Level 5, for media sharing - disables:
  - *com.apple.mobilesync.*
- Level 6, no sensitive services. In addition to all the above, disables:
  - *com.apple.afc.*
- Level 7, no *lockdown* services at all.
  - Completely removes every *lockdown* service, including *com.apple.mobile.heartbeat.*

Even in the strictest mode, the device is still capable of interacting with external devices. In particular, in this mode one can successfully connect the iOS device to: a hands-free device (via bluetooth) to import the address book and place phone calls; a Mac computer (through USB cable) to import pictures through iPhoto; and an audio system (again, through USB cable) to play the songs stored in the device.

C. The iOS jailbreak

The term *jailbreak* (also known in other platforms as *rooting*) refers to the act of circumventing vendor's restrictions in order to run code on the device with full privileges. The use of jailbroken devices is very popular among developers and researchers, as it gives them much more control over the device's internals [7]. Although it is hard to find global data about the number of jailbroken devices, a recent report focused in China found that over 30% of iOS devices being used in that country were jailbroken in January 2013 – a number that fell to 13% by December of the same year [16]. Jailbreak has become increasingly popular among users and there are thousands of applications, both free and paid, that can be installed in jailbroken devices - applications that would never make their way into the official distribution channels, given that they infringe the App Store's rules in one way or another.

Examples include software emulators and all kinds of system-wide tweaks that change the device's global aspect [17], alter global elements such as the Control Center or the Notification Center [18], or inject code into other existing applications to change their behavior [19].

In our case, we have leveraged the jailbreak technique to disable specific *lockdown* services and test different connectivity scenarios, and to develop and test the *Lockup* tool. Other users and researchers can install it and benefit from its features, provided that they are using an iOS version for which a jailbreak is available. And, of course, vendors could implement this kind of tool in future OS versions.

Given that jailbreaking a device deactivates some important security features (code signing and sandboxing), it opens the door to a number of threats [20], [21]. Consequently, users must be careful about the origin of the software packages they install in jailbroken devices. We recommend installing only the core software packages needed by the jailbreak process itself, and always changing the passwords of the *root* user as well as the regular *mobile* user.

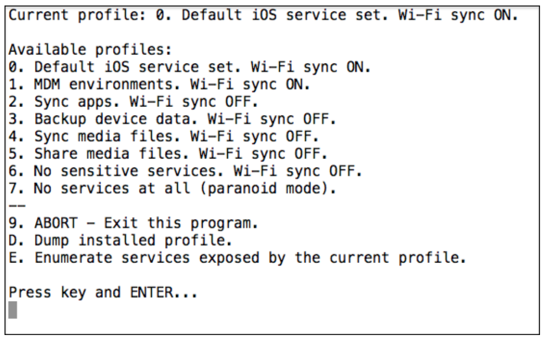
Nevertheless we believe our contribution is useful, not only as a proof of concept implementation, but also as a real tool that can be used in a number of real-world scenarios, for instance in old iOS devices which no longer receive software updates from Apple - and which are usually left with an iOS version for which a jailbreak exists.

D. Implementation details

Lockup is designed to run in jailbroken versions of iOS 7 and 8. It can easily be ported to new major iOS versions as soon as a jailbreak is available for them, which typically happens a few weeks after the official iOS release. In the worst case so far, iOS 7.0 took 95 days until a public jailbreak was available for iOS 7; in contrast, iOS 8 was jailbroken 35 days after its official release. It is also remarkable that many users of jailbreak applications usually stick to an older iOS version until a jailbreak for the new one is available.

The different service profiles are defined by creating multiple copies of the */System/Library/Lockdown/Services. plist* file. In each profile, we disable an increasing number of services. In addition, in most of the profiles, the flag *USBOnlyService* is applied to sensitive services, so that these cannot be abused over the air, either via a Wi-Fi connection, or through the user's cellular connection.

In order to set a profile, the user executes the command *lockup-profile*. This can be done either using a terminal application such as *MobileTerminal* or accessing the device via SSH, if it has been installed. When the command is invoked, the user is presented with a menu as shown in Figure 2. After the user picks a profile, the corresponding service list file is copied over */System/Library/Lockdown/Ser- vices.plist*. For the changes to take immediate effect, a *SIGTERM* signal is sent to the *lockdown* daemon with the *kill* command, which makes it restart and read its new configuration file. Additional options allow the user to enumerate the services exposed by the present profile and dump the whole contents of the *Services.plist* file, which may be specially useful to detect and investigate additional services that may have been installed inadvertently.



```
Current profile: 0. Default iOS service set. Wi-Fi sync ON.

Available profiles:
0. Default iOS service set. Wi-Fi sync ON.
1. MDM environments. Wi-Fi sync ON.
2. Sync apps. Wi-Fi sync OFF.
3. Backup device data. Wi-Fi sync OFF.
4. Sync media files. Wi-Fi sync OFF.
5. Share media files. Wi-Fi sync OFF.
6. No sensitive services. Wi-Fi sync OFF.
7. No services at all (paranoid mode).
--
9. ABORT - Exit this program.
D. Dump installed profile.
E. Enumerate services exposed by the current profile.

Press key and ENTER...
```

Fig. 2. Menu presented by *lockup-profile*.

For the periodic purging of pairing records, *Lockup* uses various files. First, a shell script in charge of deleting the pairing records is installed. Secondly, a *launch daemon*, which will run the previous script periodically, is loaded through */System/Library/LaunchDaemons/es.pope.lockup-purge.plist*. An additional script, *lockup-interval*, can be used to change the interval at which pairing records are deleted (one hour by default). Figure 3 summarizes the main components and the interactions between them.

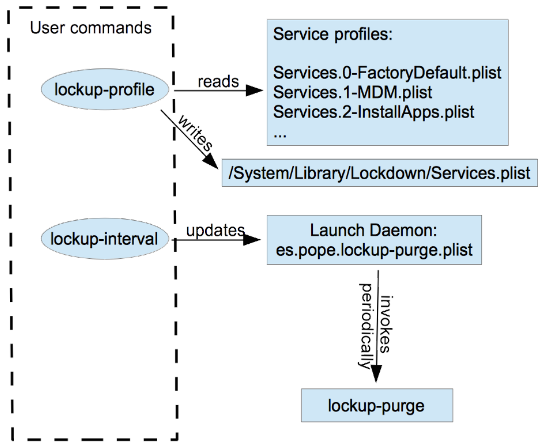


Fig. 3. *Lockup* components and main interactions.

V. Conclusions and Future Work

In this paper we have reviewed the security and privacy risks presented by certain background services that exist in the iOS operating system. We have presented a number of mitigation measures that can be used to reduce those risks. The main contribution of this paper is *Lockup*, a software tool that hardens the security of iOS devices by defining a number of *profiles* which reduce the number of exposed services. In addition, we have discussed the anti-forensic implications of our solution, and the anti-anti-forensics countermeasures that could be used to bypass it. Given the huge amount of personal information that can be extracted by abusing these sensitive services, we believe it is worth exploring this kind of solutions. The expected rise of wearable devices will only increase the need for solutions that enhance the devices' security and privacy levels.

Lockup will be released as free software so that other researchers or developers can adapt it as they find convenient. We also have the intention of continuing working in *Lockup*, maintaining it and adding new features, such as: monitoring and logging connection attempts to *lockdown* services, alerting the user in real time; adding a graphical interface to the software; monitoring the set of available services and alerting the user if new services are added... It would also be possible to integrate it with other solutions such as *activator* [24]. However, from a security standpoint, it would be preferable to keep the software as simple as possible, both in terms of size and in terms of dependencies.

The purpose of the proof-of-concept tool presented in this paper is to fight the security risks presented by a number of iOS unwanted services. It must be kept in mind, however, that our solution will only work in jailbroken devices, and the process of jailbreaking itself implies circumventing and disabling a number of native iOS security mechanisms.

Future research work includes the possibility of creating custom jailbreak tools that after deploying our software return the device to its original state to the best possible extent - this would keep most of the benefits and security features of stock Apple devices, while avoiding exposure through unwanted services.

Acknowledgements

This work was partly funded by the Spanish Government through projects TIN2011-27076-C03-02 CO-PRIVACY and SMARTGLACIS (TIN2014-57364-C2-2-R).

References

[1] T. Starner, Wearable computing: Through the looking glass, in: Proceedings of the 2013 International Symposium on Wearable Computers, ISWC '13, ACM, New York, NY, USA, 2013, pp. 125–126.
[2] International Data Corporation, Worldwide quarterly mobile phone tracker Q2 2014, International Data Group, 2014.
[3] Good Technology, Good Technology mobility index report Q2 2014, http://media.www1.good.com/documents/rpt-mobility-index-q2-2014.pdf (2014).
[4] N. Allegra, J. Freeman, JailbreakMe 3.0, http://jailbreakme.com (2011).
[5] J. Zdziarski., Identifying back doors, attack points, and surveillance mechanisms in ios devices, Digital Investigation 11 (2014) 3–19.
[6] Apple Computer, App Store sales top $10 Billion in 2013, http://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html (2014).
[7] C. Miller, D. Blazakis, D. D. Zovi, S. Esser, V. Iozzo, R. Weinmann, iOS hacker's handbook, Wiley, 2012.
[8] A. Masna, Camera+ pulled from the App Store over hidden feature, http://www.macworld.com/article/1153337/cameraplus_pulled.html (2010).
[9] I. Fried, Emulator runs DOS, Windows on an iPad, http://www.cnet.com/news/emulator-runs-dos-windows-on-an-ipad/ (2010).
[10] M. Rose, Arcade emulator iMAME punted out of App Store, http://www.tuaw.com/2011/12/23/arcade-emulator-imame-punted-out-of-app-store/ (2011).
[11] M. Beasley, Rogue App Store app lets you hide built-in apps and disable iAds, http://9to5mac.com/2013/03/11/rogue-app-store-app-lets-you-hide-built-in-apps-and-disable-iads/ (2013).
[12] Apple Computer, Resources for IT and enterprise developers, https://developer.apple.com/enterprise/ (2014).
[13] Apple Computer, iPhone in business, https://www.apple.com/iphone/business/ios (2014).
[14] B. Lau, Y. Jang, C. Song, T. Wang, P. ho Chung, P. Royal, Mactans: injecting malware into iOS devices via malicious chargers (2013).
[15] Stroz Friedberg, unTRUST, https://github.com/strozfriedberg/unTRUST (2014).
[16] Umeng, Umeng Insight report: China mobile internet 2013 (2014).
[17] J. Freeman, WinterBoard, http://cydia.saurik.com/package/winterboard/ (2014).
[18] D. Lisiansky, CCControls, http://cydia.saurik.com/package/com.danyl.cccontrols/ (2014).
[19] J. Freeman, Cydia Substrate, http://www.cydiasubstrate.com (2014).
[20] A. Apvrille, Inside the iOS/AdThief malware, https://www.virusbtn.com/pdf/magazine/2014/vb201408-AdThief.pdf (2014).
[21] P. Porras, H. Sadi, V. Yegneswaran, An analysis of the ikee.b iphone botnet, in: A. Schmidt, G. Russello, A. Lioy, N. Prasad, S. Lian (Eds.), Security and Privacy in Mobile Information and Communication Systems, Vol. 47 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Springer Berlin Heidelberg, 2010, pp. 141–152.
[22] K. Oestreicher, A forensically robust method for acquisition of icloud data, Digital Investigation 11, Supplement 2 (0) (2014) S106 – S113, fourteenth Annual {DFRWS} Conference.
[23] K. Ruan, J. Carthy, T. Kechadi, I. Baggili, Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results, Digital Investigation 10 (1) (2013) 34 – 43.
[24] R. Petrich, Activator, https://rpetri.ch/cydia/activator/ (2014).

Appendix G

Poster: Hardening iOS by selectively disabling *Lockdown* services

This appendix contains the poster **Hardening iOS by selectively disabling *Lockdown* services**, which was presented at the 2nd Digital Forensics Research Conference Europe (DFRWS EU) in March 2015.

Hardening iOS by selectively disabling *lockdown* services

Luis Gómez-Miralles, Joan Arnedo-Moreno
pope@uoc.edu, jarnedo@uoc.edu

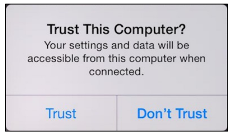
Introduction

Facts:

The U.S.A. N.S.A. has extensive surveillance capabilities over iOS devices [Rosenbach].

Targets include innocent citizens such as sysadmins working for companies where N.S.A. wants to infiltrate [Gallagher].

Surveillance likely happens through default iOS system services and abuse of pairing certificates [Zdziarski].


Trust This Computer?
Your settings and data will be accessible from this computer when connected.

Proposed solution

Many possible mitigations:

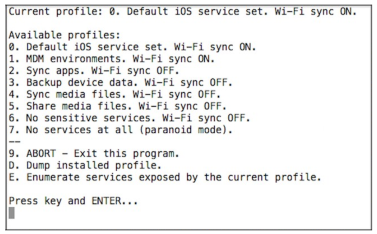
- Disable unwanted services (e.g. sniffer)
- Restrict other sensitive services to *USBOnly*.
- For the rest of services: define a number of *profiles* for typical device uses, such as: remote management through MDM; allow app installation through iTunes; media sync... And choose the desired profile at any moment.
- Periodically delete pairing records (trust certs).

Software implementation

Lockup is an implementation of the proposed mitigations for jailbroken devices running iOS 7.

With one command (*lockup-profile*) the user sets the profile in use at any given time based on the tasks he wants to perform. This disables unnecessary services.

Another command (*lockup-interval*) lets the user define the period at which pairing records are deleted. This sets the maximum lifetime of trust relationships.



```
Current profile: 0. Default iOS service set. Wi-Fi sync ON.

Available profiles:
0. Default iOS service set. Wi-Fi sync ON.
1. MDM environments. Wi-Fi sync ON.
2. Sync apps. Wi-Fi sync OFF.
3. Backup device data. Wi-Fi sync OFF.
4. Sync media files. Wi-Fi sync OFF.
5. Share media files. Wi-Fi sync OFF.
6. No sensitive services. Wi-Fi sync OFF.
7. No services at all (paranoid mode).
--
9. ABORT - Exit this program.
D. Dump installed profile.
E. Enumerate services exposed by the current profile.

Press key and ENTER...
█
```

Fig 2: A screenshot of *lockup-profile*.

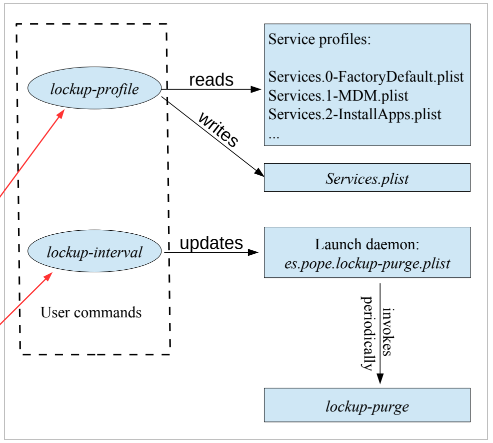


Fig. 1: The different software components of *Lockup*.

Considerations:

- In order to simplify the implementation we defined our profiles as incraesingly restrictive, meaning that if a service is available in profile *n* it will also be available in the lower profiles *n-1*, *n-2*...
- Although the overall security posture is improved, the use of jailbreak disables a number of security protections opening the door to new weaknesses.
- Consequently, users should not install any additional software via Cydia.

Work in progress:

- Port to iOS 8.
- Add a GUI.

Conclusions

This proof of concept has shown that is technically possible to restrict the number of exposed services based on the tasks the user wants to perform in the device at any given time.

Lockup will be released as free software so that other researchers and developers can adapt it as they find convenient.

We intend to continue working in the tool, maintaining it and adding new features such as: monitor the list of available services and detect if new services are installed; log service connection attempts; and offer the user real-time notifications.

The use of jailbreak was unavoidable due to the restrictions inherent to the iOS environment, but it would be trivial for Apple to implement this kind of changes in stock iOS versions. This, however, does not seem likely to happen in any near future.

Using techniques similar to those of *redsn0w* and other jailbreaks, it should be possible to perform only the first steps of the process, modify core system files to disable unwanted services, and return the device to a non-jailbroken state without installing the untethered exploit [Miller]. This would put together the best of both worlds, and is a very promising line of research for the future.

...all your iPhone are belong to us

References and related literature

M. Rosenbach, L. Poitras, H. Stark, iSpy: How the NSA accesses smartphone data, Der Spiegel 37/2013 (2013).
R. Gallagher, P. Maass, Inside the NSA's secret efforts to hunt and hack system administrators, The Intercept (2014).
J. Zdziarski., Identifying back doors, attack points, and surveillance mechanisms in iOS devices, Digital Investigation 11 (2014) 3–19.
C. Miller, D. Blazakis, D. D. Zovi, S. Esser, V. Iozzo, R. Weinmann, iOS hacker's handbook, Wiley (2012).

Acknowledgments

This work was partly funded by the Spanish Government through project TIN2011-27076-C03-02 CO-PRIVACY.

UOC Universitat Oberta de Catalunya

Further information

For related research and additional information, please refer to the authors' website at: www.pope.es

… or just look for this guy

Bibliography

[1] L. Gomez-Miralles and J. Arnedo-Moreno. Universal, fast method for iPad forensics imaging via USB adapter. In *Proceedings of the 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 200 – 207, 2011.

[2] L. Gomez-Miralles and J. Arnedo-Moreno. Versatile iPad forensic acquisition using the Apple Camera Connection Kit. *Computers And Mathematics With Applications*, 63:544 – 553, 2012.

[3] L. Gomez-Miralles and J. Arnedo-Moreno. Analysis of the forensic traces left by AirPrint in Apple iOS devices. In *Proceedings of the 27th International Conference on Advanced Information Networking and Applications Workshops*, pages 703 – 708, 2013.

[4] L. Gomez-Miralles and J. Arnedo-Moreno. AirPrint Forensics: Recovering the contents and metadata of printed documents from iOS devices. *Mobile Information Systems*, 2015.

[5] L. Gomez-Miralles and J. Arnedo-Moreno. Lockup: a software tool to harden iOS by disabling default Lockdown services. In *Proceedings of the 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 718 – 723, 2015.

[6] B. Madea. *Handbook of Forensic Medicine*. Wiley, 2014. ISBN 9781118570630. URL https://books.google.co.in/books?id=QLQZAwAAQBAJ.

[7] M. Lindemann. *Medicine and Society in Early Modern Europe*. New Approaches to European History. Cambridge University Press, 1999. ISBN 9780521423540. URL https://books.google.co.in/books?id=fQxAkrbksTEC.

[8] Gary L. Palmer. A road map for digital forensic research. In *Report From the First Digital Forensic Research Workshop (DFRWS)*, 2001.

[9] J. Hopper J. Avila, K. Folmer. Conrad Murray trial: Michael Jackson audio mumbles about 'lost childhood', 2011. http://abcnews.go.com/US/Conrad_Murray_Trial/conrad-murray-trial-michael-jackson-audio-lost-childhood/story?id=14674700.

[10] Inc. Apple Computer. iOS Security White Paper, 2015. http://images.apple.com/business/docs/iOS_Security_Guide.pdf.

[11] Good Technology. Good Technology mobility index report Q2 2015, 2015. https://media.good.com/documents/mobility-index-report-q2-2015.pdf.

[12] George Hotz. iPhone serial hacked, full interactive shell, 2007. http://www.hackint0sh.org/f127/1408.htm.

[13] C. Miller, D. Blazakis, D. Dai Zovi, S. Esser, V. Iozzo, and R. Weinmann. *iOS hacker's handbook*. Wiley, 2012.

[14] B. Carrier. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of Digital Evidence*, 1, 2003.

[15] B. Preneel. The first 30 years of cryptographic hash functions and the NIST SHA-3 competition. In *Proceedings of the 2010 international conference on Topics in Cryptology CT-RSA'10*, pages 1–14, 2010.

[16] Y. Sasaki, W. Komatsubara, Y. Sakai, L. Wang, M. Iwamoto, K. Sakiyama, and K. Ohta. Meet-in-the-middle preimage attacks revisited: New results on MD5 and HAVAL. In *Proceedings of the 10th International Joint Conference on E-Business and Telecommunications (ICETE 2013) and the 10th International Conference on Security and Cryptography (SECRYPT 2013)*, pages 111–122, 2013.

[17] P. Karpman, T. Peyrin, and M. Stevens. Practical free-start collision attacks on 76-step SHA-1. In *CRYPTO 2015, Lecture Notes in Computer Science*, pages 623–642, 2015.

[18] J. Zdziarski. Identifying back doors, attack points, and surveillance mechanisms in ios devices. *Digital Investigation*, 26:3–19, 2014.

[19] J. Zdziarski. *iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*. O'Reilly, 2008.

[20] J.R. Rabaiotti and C.J. Hargreaves. Using a software exploit to image RAM on an embedded system . *Digital Investigation*, 6:95–103, 2010.

[21] Forensic Telecommunications Services Ltd. iXAM - Advanced iPhone Forensics Imaging Software, 2012. http://www.ixam-forensics.com/.

[22] B. Iqbal, A. Iqbal, and H. A. Obaidli. A novel method of iDevice (iPhone, iPad, iPod) forensics without jailbreaking. In *Proceedings of the 8th International Conference on Innovations in Information Technology*, 2012.

[23] T. Vidas, G. Zhang, and N. Christin. Toward a general collection methodology for Android devices. *Digital Investigation*, 8:s14 – s24, 2011.

[24] G. Grispos, T. Storer, and W. B. Glisson. A comparison of forensic evidence recovery techniques for a windows mobile smart phone. *Digital Investigation*, 8:23 – 36, 2011.

[25] C.-N. Chen, R. Tso, and C.-H. Yang. Design and implementation of digital forensic software for iphone. pages 90–95, 2013. doi: 10.1109/ASIAJCIS.2013.21. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84889037016&partnerID=40&md5=9a54b5ab6f6d51979032250352916899. cited By 0.

[26] J.B. Bedrune and J. Sigwald. iPhone data protection in depth, 2011. HITB Amsterdam.

[27] J.B. Bedrune and J. Sigwald. iPhone data protection tools, 2011. http://code.google.com/p/iphone-dataprotection/.

[28] Y.-C. Tso Y.-T. Chang, K.-C. Teng and S.-J. Wang. Jailbroken iPhone Forensics for the Investigations and Controversy to Digital Evidence. *Journal of Computers*, 11, 2015.

[29] S. Morrissey. *iOS forensic analysis for iPhone, iPad, and iPod touch*. Apress, 2010.

[30] Oxygen Software. Oxygen Forensics, 2014. http://www.oxygen-forensic.com/.

[31] Katana Forensics. Lantern, 2014. http://katanaforensics.com/.

[32] C. Sgaras, M.-T. Kechadi, and N.-A. Le-Khac. Forensics acquisition and analysis of instant messaging and voip applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8915:188–199, 2015. doi: 10.1007/978-3-319-20125-2_16. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84947258502&partnerID=40&md5=96f300261d932142f30670fc41d4bcc7. cited By 0.

[33] A. Iqbal, A. Marrington, and I. Baggili. Forensic artifacts of the chaton instant messaging application. 2014. doi: 10.1109/SADFE.2013.6911538. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84910090280&partnerID=40&md5=6bab4d0d78549c0062dd0768b08cb1a6. cited By 0.

[34] F. Norouzizadeh Dezfouli, A. Dehghantanha, B. Eterovic-Soric, and K.-K.R. Choo. Investigating social networking applications on smartphones detecting facebook, twitter, linkedin and google+ artefacts on android and ios platforms. *Australian Journal of Forensic Sciences*, 2015. doi: 10.1080/00450618.2015.1066854. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84938850824&partnerID=40&md5=06ab967e10d8b404e7020c867da354fb. cited By 0; Article in Press.

[35] S. Zhang and L. Wang. Forensic analysis of social networking application on iOS devices. volume 9067, 2013. doi: 10.1117/12.2051375. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84901345271&partnerID=40&md5=39259bdda6eb0e8c37471673028ec06f. cited By 0.

[36] C. Carpene. Looking to iphone backup files for evidence extraction. pages 16–32, 2011. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84867715994&partnerID=40&md5=18e8dc4d85493458cfba2627774d7b61. cited By 0.

[37] A.R. Cheema, M.M.W. Iqbal, and W. Ali. An open source toolkit for ios filesystem forensics. *IFIP Advances in Information and Communication Technology*, 433: 227–236, 2014. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84911125467&partnerID=40&md5=b6b968f8a49b2d47c4a34714b5bcec35. cited By 0.

[38] J. Park, H. Chung, and S. Lee. Forensic analysis techniques for fragmented flash memory pages in smartphones. *Digital Investigation*, 9(2):109–118, 2012. doi: 10.1016/j.diin.2012.09.003. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84870255379&partnerID=40&md5=0b8ab378b8321fa744fc033208a5ee55. cited By 0.

[39] W.-D. Qiu, Q. Su, B.-Z. Liu, and Y. Li. Ios data recovery using low-level nand images. *IEEE Security and Privacy*, 11(5):49–55, 2013. doi: 10.1109/MSP.2013.50. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84886467513&partnerID=40&md5=1545b51724a940ba38a7cc472fc21911. cited By 0.

[40] A. Ariffin, C. D'Orazio, K.-K.R. Choo, and J. Slay. Ios forensics: How can we recover deleted image files with timestamp in a forensically sound manner? pages 375–382, 2013. doi: 10.1109/ARES.2013.50. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84892395081&partnerID=40&md5=dbc3be2707ae68c0b1e8da508bf13230. cited By 0.

[41] The Grugq. The art of defiling: defeating digital forensics, 2005. http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-grugq.pdf.

[42] Ryan Harris. Arriving at an anti-forensics consensus: Examining how to define and control the anti-forensics problem. *Digital Investigation*, S3:S44–S49, 2006.

[43] Vincent Liu and Francis Brown. Bleeding-edge anti-forensics. *Presentation at InfoSec World*, 2006.

[44] Gary C. Kessler. Anti-forensics and the digital investigator. In *Proceedings of the 2nd Australian Digital Forensics Conference*, 2007.

[45] Simson Garfinkel. Anti-forensics: techniques, detection and countermeasures. In *Proceedings of the 2nd International Conference on Information Warfare and Security*, pages 77 – 84, 2007.

[46] Alessandro Distefano, Gianluigi Me, and Francesco Pace. Android anti-forensics through a local paradigm. *Digital Investigation*, 7:S83–S94, 2010.

[47] P. Albano, A. Castiglione, G. Cattaneo, and A. De Santis. A novel anti-forensics technique for the Android OS. In *Proceedings of the 2011 International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*, pages 380–385, 2011.

[48] Christian D'Orazio, Aswami Ariffin, and Kim-Kwang Raymond Choo. ios anti-forensics: How can we securely conceal, delete and insert data? In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS 2014)*, 2014.

[49] M. Rosenbach, L. Poitras, and H. Stark. iSpy: How the NSA accesses smartphone data, 2013. http://www.spiegel.de/international/world/how-the-nsa-spies-on-smartphones-including-the-blackberry-a-921161.html.

[50] R. Gallagher and P. Maass. Inside the NSA's secret efforts to hunt and hack system administrators, 2014. https://firstlook.org/theintercept/2014/03/20/inside-nsa-secret-efforts-hunt-hack-system-administrators/.

[51] J. Varsalone. *Mac OS X, iPod, and iPhone forensic analysis DVD toolkit*. Syngress, 2009.

[52] International Organization for Standarization (ISO). ISO 15740:2008 – Electronic still picture imaging – Picture transfer protocol (PTP) for digital still photography devices , 2008.

[53] M.I. Cohen. Advanced carving techniques. *Digital Investigation*, 426(3-4):119–128, 2007.