

# Hardening iOS devices against remote forensic investigation

Luis Gómez-Miralles<sup>a \*\*</sup> and Joan Arnedo-Moreno<sup>b</sup>

<sup>a</sup>*Universitat Oberta de Catalunya, E-mail: pope@uoc.edu*

<sup>b</sup>*Universitat Oberta de Catalunya, E-mail: jarnedo@uoc.edu*

## Summary

Smartphones and mobile devices nowadays accompany each of us in our pockets, holding vast amounts of personal data. The iPhone and iPad are between the most important players in this new market, especially in enterprise environments due to the supposed higher level of security of the iOS platform. Recent research has exposed a number of iOS services that are used by forensic tools to extract information from the devices, and are also prone to being abused by malicious parties. We present a proof of concept tool to prevent this abuse by disabling unnecessary services, and analyze its anti-forensic consequences, the privacy implications, and the possible countermeasures (anti-anti-forensics).

*Keywords: Apple; iPhone; iPad; iOS; forensics; surveillance*

## 1 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 a vast amount of personal information - often without the user's knowledge: phonecall logs, emails and SMS messages, calendars, address books, to-do lists, history of visited places, photographs, voice memos... as well as 3rd-party application data such as chat logs from apps like WhatsApp and Telegram. 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 the Apple Watch and other similar devices.

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% ([Corporation 2014](#)). In contrast, 84.7% of the devices sold in that period were Android devices, and 11.7% were iOS devices. When it comes to business environments, 67% of new devices activated in a corporate context during the same period were iOS ones([Technology 2014](#)).

As 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

---

\*\*Corresponding author.

(Allegra and Freeman 2011). Fortunately, many of the vulnerabilities uncovered by researchers have been duly patched by Apple in subsequent iOS versions. However, it is possible that, at a certain point in the past, the same particular vulnerabilities were used by malicious actors to attack devices running specific iOS versions.

During 2013, Edward Snowden’s revelations about the capabilities of the United States National Security Agency (NSA) showed that this body has more robust and persistent surveillance mechanisms over mobile devices (Rosenbach et al. 2013) than could be reasonably expected based on the general security posture of those platforms. As noted by (Zdziarski. 2014) this can be achieved by abusing a series of 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. These mechanisms can be used by forensic software solutions in order to collect information from the device – however, the same mechanisms are likely being exploited to gain unauthorized access to users’ personal and corporate devices. As an example, it has been documented that the NSA habitually targets the personal resources of innocent people working as system administrators, with the purpose of getting into the networks of the companies they work for (Gallagher and Maass 2014).

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 measures can be deployed in an iOS device.

This paper is structured as follows. Section 2 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 3 discusses a number of possible mitigation strategies that can be applied to enhance the device security. Section 4 presents *Lockup*, the software tool that we have developed in order to implement those mitigations. In Section 5, a number of important questions are discussed, including the consequences of the jailbreak process, the anti-forensic implications, and the anti-anti-forensic measures that can be used to bypass our tool. Concluding the paper, Section 6 summarizes the paper contributions and outlines future work.

## 2 Security and trust in the iOS environment

In this section, we present an overview of the main components of the iOS security architecture, its trust model and the existing privacy threats, as well as the different approaches that exist for the forensic collection of data. 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 which have no known legitimate purpose.

Over time, the basic iOS system has incorporated a number of security protections, including application sandboxing, mandatory code signing, DEP (data execution prevention) and ASLR (address space layout randomization). These measures are aimed at reducing the attack surface, thus complicating both bug exploitation and the subsequent privilege escalation.

A thorough analysis of each of these layers can be found in (Miller et al. 2012).

In addition, every major iOS version has incorporated an increasing number of enterprise features ([Apple Computer, Inc. 2014b,a](#)), especially since the release of iOS 4 and the iPad in 2010 - from Exchange and Mobile Device Management (MDM) support to biometric authentication. Some of these features require the device to be remotely manageable somehow. The main method to achieve this would be generating a trust relationship with an external device, which, from then on, will be able remotely access a set of special services made available by the iOS operating system.

However, as we will explore later, this might be a double-edged sword, as those same capabilities can be used by malicious actors to read the data stored in the device or to surreptitiously install applications capable of tasks as dangerous as recording audio and capturing network data.

## 2.1 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.

As exposed by ([Lau et al. 2013](#); [Zdziarski. 2014](#)), 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.

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.

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...) ([Lau et al. 2013](#)) 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.

## 2.2 Sensitive iOS device services

A computer that connects to an iOS device (either via USB cable or through the network in TCP port 62078) can invoke a series of services which, from the iOS side, are offered through the *lockdown* daemon ([Zdziarski. 2014](#); [Lau et al. 2013](#)). 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 (Zdziarski. 2014) identified the following ones as being valuable from a forensic standpoint:

- **com.apple.file\_relay.** There is no known legitimate use for this service. It is designed to obtain huge amounts of information: the user's complete address book, calendar, SMS database, call history, voicemail, notes, photos, list of known Wi-Fi networks, GPS positioning logs, list of email accounts configured in the device, a list of all files existing in the device, with their metadata (size, creation and modification dates)... even a list of every single word typed in the device, together with its word count; as well as and a number of additional system logs. No indication is shown to the user when this is done. Note that, although the user can set a backup encryption password through iTunes, data sent through this service is not encrypted in any manner.
- **com.apple.pcapd.** A network sniffer for which, again, no known legitimate purpose is known. It can be activated remotely and leaves no trace to the user. Apart from seeing the network traffic of other devices near the victim, and possibly performing man-in-the-middle attacks, this could also have other interesting uses for attackers: they could obtain information about the networks available at a certain location (possibly a restricted facility where the victim has access) in order to prepare more advanced attacks in which they could impersonate those networks; another possible use would be confirming or discarding the presence of other people at that location, by examining unique identifiers of the nearby devices, such as Bluetooth MAC addresses.
- **com.apple.mobile.MCInstall.** Installs managed configurations, such as the ones used in MDM deployments. This makes sense in corporate environments, where the company may need to enforce security restrictions on the device, preload applications, or provision encryption certificates; but is very rarely required in the case domestic users, and is a possible entry point for an attacker willing to deploy hidden applications to the victim's device, for instance with the purpose of recording background audio.
- **com.apple.mobile.diagnostics\_relay.** Provides diagnostics information such as hardware state and battery level.
- **com.apple.syslog\_relay.** Exposes various system logs.
- **com.apple.iosdiagnostics.relay.** Presents per-application network usage statistics.
- **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. If the owner has set a backup encryption password through iTunes, the data sent through this service will

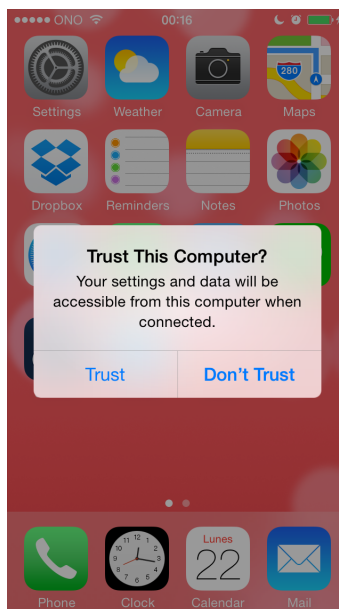
be encrypted with that password – something that does not happen in the case of the *com.apple.file\_relay* service.

- **com.apple.mobilesync.** Used by iCloud and iTunes to sync 3rd-party application data as well as data belonging to core iOS applications: Safari bookmarks, notes, address book entries, etc.
- **com.apple.afc.** Exposes the complete *Media* folder - audio, photographs and videos.
- **com.apple.mobile.heartbeat.** Used to maintain the connection to other services being accessed.

The two first services (*file\_relay* and *pcapd*) are the most dangerous ones. When these were first identified by Zdziarski, Apple responded to news site iMore (Ritchie 2014) stating:

We have designed iOS so that its diagnostic functions do not compromise user privacy and security, but still provides needed information to enterprise IT departments, developers and Apple for troubleshooting technical issues.

However, we agree with Zdziarski (Zdziarski 2014) that there seems to be no realistic scenario in which opening the door to such an enormous amount of user data can be justified for ‘*diagnostic*’ and ‘*troubleshooting*’ purposes.



**Figure .1:** An iPhone prompting the user whether it should trust the connected computer.

## 2.3 Forensic acquisition approaches

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 multi-media 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 (Zdziarski. 2014): 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 (Zdziarski 2008) 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 (Rabaiotti and Hargreaves 2010) 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 (Forensic Telecommunications Services Ltd.), 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 (Iqbal et al. 2012), 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 (Vidas et al. 2011) or Windows Mobile (Grispos et al. 2011). Another paper in 2013 presented the design and implementation of an iOS forensic tool (Chen et al. 2013) 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*). Bedrune and Sigwald analyzed (Bedrune and Sigwald 2011a) the underlying technology and released (Bedrune and Sigwald 2011b) 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 2011 onwards) are shipped with a new bootrom that fixes the bugs exploited by tools such as (Bedrune and Sigwald 2011b) 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 (Y.-T. Chang and Wang 2015) 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. On the other hand, well-resourced attackers probably count on more advanced tools even capable of exploiting bugs which are not publicly known in order to surreptitiously retrieve data from the device.

### 3 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.

#### 3.1 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 (Friedberg 2014): 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.

#### 3.2 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, none of the sensitive services in iOS 7 requires this option. Starting in iOS 8, the option is applied by default to *com.apple.pcapd* (the network sniffer).



### 3.3 Disable some services

Finally, it would be ideal to disable the most sensitive services - something that has not been done so far. This is the approach chosen for our tool, *Lockup*. Given the access level required, it runs only on jailbroken devices, although it would be trivial for Apple to implement these changes in stock iOS versions.

### 3.4 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.

## 4 *Lockup*: iOS hardening and anti-forensics

As a proof of concept, in this paper we present *Lockup*, a software tool that can be installed in devices running iOS versions 7 and 8. *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: *com.apple.file\_relay* (the service that retrieves lots of data bypassing the backup encryption password) and *com.apple.pcapd* (the network sniffer), both with no known purpose and vaguely defended by Apple (Ritchie 2014), are disabled right away. 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, most users are not enrolled in corporate *Mobile Device Management* systems; hence they don't need to allow remote installation of software and configuration profiles – which are indeed very dangerous attacks vectors.
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.

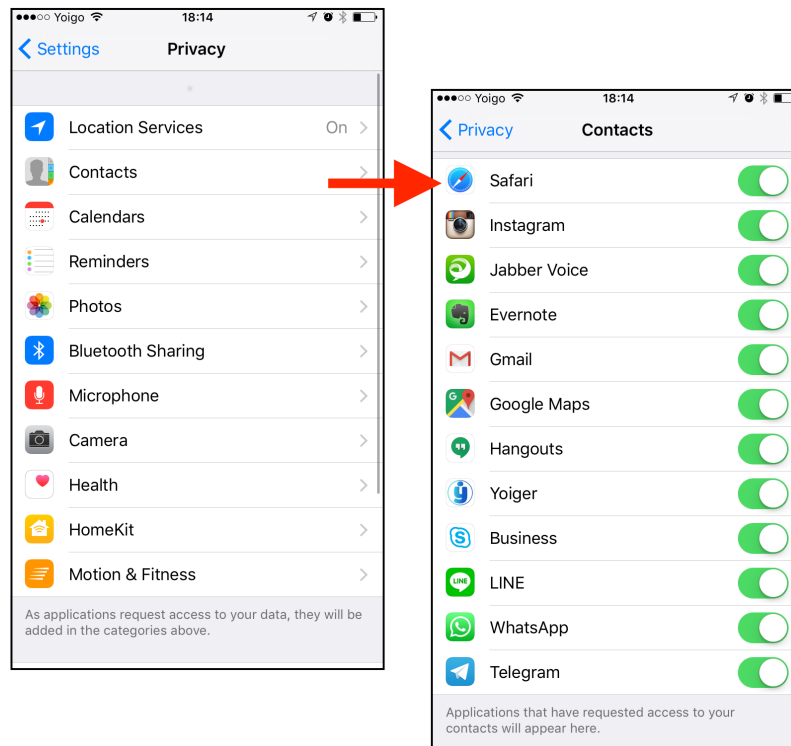
When analyzing the trust model by which an accessory such as a car audio system can have access to all personal data stored in the device, one comes to the conclusion that Apple decided to sacrifice a certain degree of user security in order to simplify the user experience.

To a certain degree, the same problem affected application security up to iOS version 7. Citing (Miller et al. 2012) about the iOS sandbox security feature: *One thing to notice about the iOS sandbox is that every third party app from the App Store has the same sandbox rules.*



That means that if Apple thinks one app should have a certain capability, all apps must have that capability. This differs, for example, from the Android sandbox where every app can have different capabilities assigned to it based on its needs. One of the weaknesses of the iOS model is that it may be too permissive.

Starting in iOS 8, Apple implemented a more granular permissions system for applications (not so for peripherals). A custom prompt is presented to the user the first time that a particular application requests access to certain items (addressbook, camera roll, camera and microphone, location, health data...) and the user can authorize or deny it. Furthermore, a new ‘Privacy’ menu in the Preferences app allows the user to review which applications have access to each restricted set of data, and revoke it if desired, as shown in Figure .2.



**Figure .2:** New Privacy options in iOS 8 and higher.

It would be desirable to be presented a similar prompt whenever external devices require access to at least the most sensitive *lockdown* services – those that expose personal user data. In our case, it did not seem feasible to implement those changes in *lockdown*, as it would only be possible via reverse engineering, and even then, such modifications would be complicated to maintain across future iOS updates. Hence we have resorted to a different strategy. Instead of setting individual permissions for each external device, *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 any given time. We believe it may be worth having the choice to sacrifice a fraction of the simplicity to improve the security of iOS devices.

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.

#### 4.1 Tool capabilities

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

- Controlling the device’s trust relationships, by periodically purging the stored pairing records; and
- Disabling certain *lockdown* services and prevent others from being invoked over Wi-Fi connections, in order to prevent 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 (unless a full restoration of the device is performed, deleting all user data). This poses 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, applying the mitigation strategy discussed in section 3.1. Once that happens, connecting to that device will require the user to confirm the trust relationship from the iOS screen. We observed that even with values as low as 1 minute standard features such as iTunes syncing still work normally (once the user authorizes the pairing by pressing ”Trust” in the device screen). Note that these features keep working even when the pairing record is deleted in the middle of a session – as long as both devices are connected, the iOS device will not need to re-trust the external device.

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 the user needs – 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 prefers to sync using a USB cable. 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 sections 3.2 and 3.3.

## 4.2 Service profiles

Next, we describe the different profiles that we have implemented in *Lockup*, with each profile being increasingly restrictive and consequently more secure. 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.

### Level 1: suitable for MDM

In the first security level that we propose, we disable only those services which have no known purpose and can leak significant amounts of information. In particular, this level disables the following services:

- *com.apple.file\_relay*.
- *com.apple.pcapd*.

Devices configured in this mode will still be fully functional, even for remote management in MDM environments - although many of these environments enforce policies in which no jailbroken devices are allowed, which makes sense from the security point of view given that, as we have already mentioned, jailbreaking a device disables a number of security mechanisms. In any case, this should be the bare minimum level used by any user with a jailbroken device.

## Level 2: suitable for syncing applications

In addition to the restrictions defined in the previous level, this level disables the remote installation of configuration profiles and a number of diagnostics services:

- *com.apple.mobile.MCInstall.*
- *com.apple.mobile.diagnostics\_relay.*
- *com.apple.syslog\_relay.*
- *com.apple.iosdiagnostics.relay.*

Moreover, the rest of the sensitive services (the ones subsequently disabled in the levels described below) are set to USB only, meaning that it is no longer possible for a malicious party to attack them over-the-air, using either Wi-Fi or cellular. Apart from this, the most important implication is that we disable the installation of management configurations, which can be abused by malicious parties in order to track a device via GPS, wipe its data remotely, or deploy additional certificates to the device with effects such as facilitating man-in-the-middle attacks against HTTPS connections. This change may affect the device's ability to enroll in certain MDM environments.

The vast majority of domestic users should be able to use this profile without noticing any adverse effect. An iOS device configured at this level can still sync applications and media with iTunes, and consequently, still exposes services that could be abused to install additional applications, or to retrieve the user content stored in any application installed.

## Level 3: Suitable for backup

In addition to the restrictions previously defined, this level disables the following services:

- *com.apple.mobile.installation\_proxy.*
- *com.apple.mobile.house\_arrest.*

This change disables the remote installation of applications to the device. Automatic application syncing with other iOS devices will still work if enabled (from Settings - iTunes Store and App Store - Automatic Downloads - Applications).

It also keeps iTunes from transferring files in and out of the applications installed in the iOS device. Note that if a particular application offers its own mechanism for uploading files to the device (for instance many applications can activate an integrated web server for this), this will still work properly.

The use of iTunes to manage the device applications and data is becoming less and less common as modern iOS versions, except maybe at the time of acquiring a new device and populating it with a backup of the previous one. Thus, this profile would still be appropriate for most domestic users.

A device configured in this mode will still allow iTunes to perform backups of the user data stored in the device. This service could be abused by an attacker to obtain the information stored in the device; however, if the user has set a backup encryption password, files transferred through this service will be encrypted with that password.

#### Level 4: Suitable for syncing media files

In addition to the restrictions already described, this level disables the following services:

- *com.apple.mobilebackup2*.
- *com.apple.mobilebackup*.

This level disables iTunes capability to backup the data stored in the device.

A device configured at this level will still be able to sync media files with iTunes, and consequently can expose the contents of the *Media* folder (audio, pictures and video), which could potentially be reached by an attacker abusing a trust relationship. However, all the content belonging to 3rd-party apps should be safe – or at least not reachable via *lockdown* services.

If bookmarks, address book and calendar data are not being synchronized through iCloud, it may also be possible to obtain these items from a device configured at this level through the *com.apple.mobilesync* service.

#### Level 5: Suitable for media sharing

Apart from the restrictions previously defined, this level disables:

- *com.apple.mobilesync*.

With this change, iTunes syncing capabilities stop working completely, and the only piece of sensitive information exposed through *lockdown* services is the *Media* folder, containing the pictures and videos stored in the device, as well as voice memos, music and podcasts. This may be necessary for some programs or peripherals to access the media files stored in the device.

This profile is suitable for users as long as they do not rely on iTunes for the backup of their device data – something increasingly common as iOS now allows users to store their backups directly on their iCloud storage.

#### Level 6: No sensitive services

In addition to the changes performed in the previous levels, this level disables the following service:

- *com.apple.afc*.

This level breaks compatibility with programs such as iFunBox, which allow the user to browse files stored in the device. In addition, some peripherals may also rely on this service for accessing the files stored in the device, and consequently will stop working when this profile is applied.

Still, the profile should be appropriate for most users that don't use iTunes to manage nor backup their device.

#### Level 7: No lockdown services at all

This level completely removes every *lockdown* service, including *com.apple.mobile.heartbeat*.

It is worth mentioning that, even in this mode, the device is still capable of interacting with external devices through other mechanisms that do not rely on *lockdown*. In particular, we verified that the following actions work properly even when the device is configured in this mode:

- Connect the iOS device to a Parrot Minikit Smart hands-free device via bluetooth, being able to import the address book and place calls.
- Connect the iOS device to a Mac computer via USB cable and using the iPhoto software in the computer to import the photographs and videos stored in the iOS device.
- Connect the iOS device to a Denon RCD-M39 audio system via USB cable so that the audio played in the iOS device sounds through the Denon audio system.

### Additional considerations

It is worth mentioning that our solution will disable any additional services that may have been installed by the user, with or without his knowledge, at the moment of jailbreaking the device.

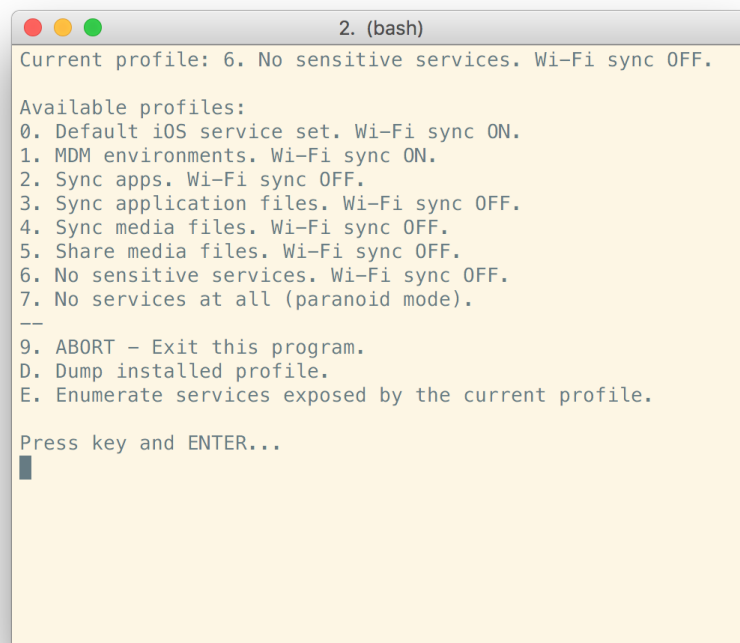
One case worth mentioning is *com.apple.afc2*, a service which is installed by many jailbreak tools and can also be installed separately by users through Cydia. This service exposes the whole filesystem of the iOS device through *lockdown*, making it possible for a trusted external device (or an attacker who has stolen the pairing record from it) to read and write any file in the device either through a USB connection or over the air. Given the privacy risks it represents, this service is always disabled in all profiles. If a user needs remote filesystem access, there are better alternatives from a security standpoint, such as installing OpenSSH and using a companion desktop applications such as *FileZilla* or *PuTTY* – if this is done, it is important to change the default password of the *root* and *mobile* users.

## 4.3 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 .3. After the user picks a profile, the corresponding service list file is copied over */System/Library/Lockdown/Services.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.

A terminal window titled "2. (bash)" with a yellow background. The text inside the window is as follows:

```
Current profile: 6. No sensitive services. Wi-Fi sync OFF.

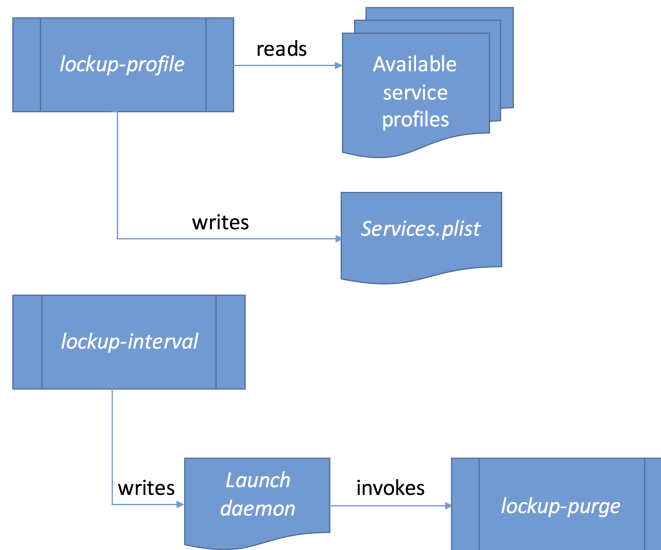
Available profiles:
0. Default iOS service set. Wi-Fi sync ON.
1. MDM environments. Wi-Fi sync ON.
2. Sync apps. Wi-Fi sync OFF.
3. Sync application files. Wi-Fi sync OFF.
4. Sync media files. Wi-Fi sync OFF.
5. Share media files. Wi-Fi sync OFF.
6. No sensitive services. Wi-Fi sync OFF.
7. No services 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...
█
```

**Figure .3:** 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.



**Figure .4:** *Lockup* components and main interactions.

## 5 Discussion

This section analyzes the security and forensics implications of using a tool like *Lockup*.

### 5.1 The jailbreak process

One of iOS' main features is application sandboxing, meaning that an iOS application does not have full system access due to the security barriers put in place by the operating system. This, in turn, keeps us from developing and running custom software. In order to overcome this restriction we resort to the *jailbreak* technique, described here.

When the first iPhone model was released in 2007, the OS did not incorporate most of its current security features: there was no sandboxing, no code-signing enforcement, no DEP and no ASLR. It took barely a week for George Hotz to figure out how to escape the OS limitations and gain root access to the device (Hotz 2007) - a process that is referred to as *jailbreak* in the iOS platform, or *rooting* in the Android platform.

A number of tools have been published over time which allow to jailbreak certain iOS versions; examples of these tools are *redsn0w*<sup>1</sup> (iPhone Dev Team 2011), *greenpois0n* (Team 2010) and

<sup>1</sup>Mind the spelling: that character in the middle of all those names is a *zero*, not a capital 'o'.

*evasion* (Wang et al. 2013). Whenever one such tool is released, it is analyzed by Apple to uncover the iOS bugs being exploited, which are patched in subsequent iOS releases - meaning that jailbreaking each iOS version requires finding new bugs that can be exploited, in what Steve Jobs himself once described as *a cat-and-mouse game* (Soghoian 2007).

The use of jailbroken devices is very popular among developers and researchers, as it gives them much more control over the device’s internals (Miller et al. 2012). 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. By December of the same year, this number had gone down (Umeng 2014) to 13%. This fluctuation can be attributed to the fact that iOS 7, released in September, did not get a jailbreak until late December. In any case, the numbers show that a significant fraction of iOS devices are jailbroken.

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 (Freeman 2014b), alter global elements such as the Control Center or the Notification Center (Lisiansky 2014), or inject code into other existing applications to change their behavior (Freeman 2014a).

It is worth mentioning that a number of important features currently present in official iOS releases existed before as tweaks in the jailbreak community, in some cases influencing Apple’s final implementation. Some examples are the Notification Center, the way the copy and paste feature works, and the ‘cards’ interface for switching between applications.

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.

## 5.2 Effects of jailbreak on the security model

It is often argued, and partly true, that jailbreaking a device introduces a number of security weaknesses (Apvrille 2014; Porras et al. 2010) by disabling important security controls such as application sandboxing and mandatory code signing. Consequently, the use of this technique must always be weighed against the new risks introduced.

In the case of the tool presented in this paper, we consider that the security tradeoffs of jailbreaking the device make sense in at least three possible scenarios:

1. With research purposes, i.e. in order to test the tool and evaluate the adoption of similar approaches in other tools or as an iOS core feature.
2. In the case of obsolete devices which are stuck forever in old iOS versions with known vulnerabilities – think for instance dad’s old iPhone which he used for two years, then handed over to mom who after another two years passed it to their teen child. This is the case of the 1st-gen iPad with iOS 5.1.1, every iPod Touch up to the 4th generation, and every iPhone up to the iPhone 4; the iPhone 4S and several iPad models are likely to remain stuck at iOS

9 when the next version is released in September 2016. Given that these iOS versions contain known vulnerabilities (exploited by the corresponding jailbreak tools), owners of these devices may prefer to jailbreak the devices themselves in order to install this and/or other protections.

3. Even in modern devices and iOS versions, as part of a more complex hardening strategy involving additional tools, possibly developed by power-users themselves.

When jailbreaking a device, users should always change the default passwords for users *mobile* and *root* (which is *alpine* in both cases), and install only the required software packages, always from trusted sources.

### 5.3 Anti-forensic implications

We did not have anti-forensics in mind when we started developing this tool. However, as exposed by (Zdziarski, 2014), most if not all of the forensic tools in the market extract information from iOS devices through the same sensitive services that we have discussed in this paper. Consequently, using *Lockup* to restrict these services would result in those tools not being able to extract data from the device.

Given that our tool offers different service levels or *profiles*, the effectiveness of forensic tools against a particular iOS device with *Lockup* installed will depend on the profile being used at any particular moment. Although we have not been able to pit our tool against commercial forensic software tools –something we would like to do if we get resources–, by definition (according to the services being disabled), when the forensic tools attempts to retrieve data from the device itself (i.e. excluding other analysis sources such as iTunes backups stored in a desktop computer) these should be the anti-forensic effects of the different profiles available:

- *Level 1.* In this level we have disabled *com.apple.file\_relay*, preventing any tool that leverages this service from dumping nearly all data on the device and bypassing the backup encryption password. Although it is not known whether any publicly available forensic tool leverages this service –something we would like to address in future research–, we can assume that it is trivial for an adversary to develop such tool.
- *Level 2.* Starting at this level, all services are forced to work over USB cable, disabling its invocation over Wi-Fi or cellular networks. This should not have any impact on forensic tools given that generally these tools recommend to retrieve the data over USB cable.
- *Level 3.* Having disabled *com.apple.mobile.house\_arrest*, forensic tools’ ability to retrieve 3rd-party app data (in particular, user data stored within apps) may be somehow impacted – although most of those data is still reachable through *com.apple.mobilesync* and possibly also in the form of an iTunes backup (although in this case it may be protected with a backup encryption password, if set).
- *Level 4.* In this level the device will not provide data in the form of iTunes backup, because the services *com.apple.mobilebackup* and *com.apple.mobilebackup2* have been disabled.

- *Level 5.* This level disables *com.apple.mobilesync*, which should prevent forensic tools from retrieving most application data from the device; this would apply to both 3rd-party applications, as well as stock iOS apps (messages, phone logs, calendar events, etc.).
- *Level 6.* By disabling *com.apple.afc*, this level prevents access to the latest bit of personal information that was still being exposed: the *Media* folder containing the camera roll (pictures and videos), voice memos, music and podcasts.
- *Level 7.* This level definitely disables all *Lockdown* services. There is little difference between this, and disabling any profitable service as we have already done by level 6.

In summary, levels 4 and higher should keep most user data away from forensic tools, with the exception of photos and videos (as sharing multimedia materials is still enabled in levels 4 and 5, for those users who prefer it). For absolute isolation, levels 6 or 7 should be used.

It may be argued that using an iOS version for which a jailbreak is available would make it easier for forensic tools to access the data in the device. However, after jailbreaking the device it is possible to patch some or all of the bugs used by the jailbreak process - thus achieving a higher grade of protection.

With additional resources, we would like to conduct a more detailed study on how the existing commercial tools interact with an iOS device protected by *Lockup*.

#### 5.4 Countermeasures: anti-anti-forensics

If *Lockup*, or a similar solution, gets popular, the forensic tools could be adapted to cope with it. Given that *Lockup* can only be installed in iOS versions for which a jailbreak is available, forensic tools could be modified to exploit the flaws that permit the jailbreak in each particular iOS version, in order to regain control over the device and reactivate the services needed to extract information. This, however, would most probably require: physical access to the device; knowing the device passcode (a password set by the user); and manual interaction in order to be able to powercycle the device and possibly press certain buttons during boot in order to enter a special boot mode known as DFU. This means that attacks via malicious devices such as the one described in (Lau et al. 2013) are much less likely to success when *Lockup* is installed and a restrictive profile is applied.

As for attackers trying to abuse the *lockdown* services, their methods would stop working after applying a restrictive profile with *Lockup*. It would still be possible to retrieve user information by finding and exploiting other vulnerabilities in the particular iOS version running in the device, or through other means such as iCloud (Oestreicher 2014; Ruan et al. 2013).

Assuming an attacker gets to execute code on the iOS device, she could modify the *Services.plist* file at will and re-enable any service to exploit it at a later time. The device owner could notice this by running *Lockup*, which in this case shows a warning informing the user that the installed *Service.plist* file does not correspond to any of the known profiles, as shown in figure .5. Future versions of *Lockup* could include periodic checks for this and warnings for the user – something that, again, could be targeted by an adversary.

```
2. (bash)
Current profile: WARNING!! Unknown or corrupt Services.plist file.

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. Sync application files. 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...
e
Enumerating services...

com.apple.crashreportmover
com.apple.mobile.notification_proxy
com.apple.mobile.heartbeat
com.apple.preboardservice
com.apple.misagent
com.apple.mobile.insecure_notification_proxy
com.apple.atc
com.apple.thermalmonitor.thermtgraphrelay
com.apple.mobile.MDMService
com.apple.rasd
com.apple.purpletestr
com.apple.mobile.mobile_image_mounter
com.apple.webinspector
com.apple.afc
com.apple.radios.wirelesstester.root
com.apple.mobile.debug_image_mount
com.apple.mobile.assertion_agent
com.apple.springboardservices
com.apple.crashreportcopymobile
com.apple.radios.wirelesstester.mobile
com.apple.hpd.mobile

bash-3.2$ █
```

**Figure .5:** *Lockup* warning the user about the currently installed *Services.plist* file, and enumerating the services currently enabled.

## 6 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.

As it is usually said, a chain is only as strong as its weakest link. If all it takes is compromising a trusted device such as a desktop computer in order to put an iOS user under severe surveillance, that makes iOS not safer than the average desktop computer – with its unpatched OS, its obsolete Java, its always-vulnerable Flash Player... This level of security is acceptable to keep away casual attackers, but it will not stand a high-profile targeted attack against specific users.

*Lockup* has been released as free software ([Gomez-Miralles and Arnedo-Moreno 2015](#)) so that other researchers or developers can adapt it as they find convenient. We 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* ([Petrich 2014](#)). 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.

An interesting future line of research would be creating custom jailbreak tools that after deploying this 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. Another interesting point would be to create a survey of services used by common commercial accessories; in addition, we would like to obtain licenses for commercial tools –mainly those used by Law Enforcement– in order to test them against our software.

### List of acronyms

- ASLR** Address Space Layout Randomization.
- DEP** Data Execution Prevention.
- DFU** Device Firmware Upgrade.
- GPS** Global Positioning System
- HTTPS** HyperText Transfer Protocol Secure.

**SMS** Short Message Service.

**SIGTERM** SIGnal TERMinate.

**MDM** Mobile Device Management.

**NSA** National Security Agency of the United States of America.

**OS** Operating System.

**USB** Universal Serial Bus.

**TCP** Transmission Control Protocol.

## Glossary of terms

**Address Space Layout Randomization (ASLR).** A measure aimed at difficulting the exploitation of buffer overflow vulnerabilities by randomizing at run-time the position in memory of the data structures used by the application.

**Cydia.** The *de facto* unofficial App Store for jailbroken devices.

**Data Execution Prevention (DEP).** A technique to complicate buffer overflow exploits, by marking the data pages in memory as not executable. This makes it no longer possible for an attacker to store exploit code in the memory space of a variable.

**Device Firmware Upgrade (DFU).** A special mode in which iOS devices can be put by entering a specific combination of button presses. A device in DFU mode will receive a firmware image through USB cable, will write it to the device internal storage and will try to boot the image.

**iCloud.** Apple's cloud storage platform, capable of syncing documents and data across iOS devices and Mac computers.

**iOS.** The operating system used by the iPhone, iPad, and iPod Touch. Before the introduction of the iPad in 2010 it was simply called *iPhone OS*, even when it was used in the iPod Touch as well.

**iPhoto.** Apple's photo management software for desktop computers; normally it is also used to import pictures from external devices such as digital cameras and iOS devices.

**iTunes.** Apple's multimedia software for desktop computers, which is also used to manage and backup the iPhone, iPad and iPod.

**Jailbreak.** A technique to suppress iOS' code execution restrictions, thus being able to run custom software in the iOS device. It is one of the many measures typically used by researchers for examining the iOS internals; a similar technique exists in the Android platform, called *rooting*. The process is also popular among users because it allows them to run all sorts of tweaks that would never be approved by Apple to be published in the official App Store.



**Lockdown.** An iOS system process that presents a number of network services, which can be accessed from a host computer either through USB cable or wirelessly. Its behaviour is somehow comparable to that of the `inetd` daemon present in many UNIX systems.

**MobileSafari.** The stock web browser embedded in every iOS version.

**Mobile Device Management (MDM).** A set of software products designed to control mobile devices. These are typically used in corporate environments, where the company can enforce certain security settings, preload corporate applications, deploy certificates and configuration profiles, etc.

**Network sniffer.** A software program designed to capture network data (wireless data in this case) regardless of whether the device is the intended destination of such traffic. Simply put, a wireless network sniffer can be used to intercept the traffic of other nearby devices.

## Index of terms

- anti-forensics
- forensics
- hardening
- iOS
- iPad
- iPhone
- iTunes
- lockdown
- security
- sniffer

## References

- Allegra, N. and J. Freeman, 2011: JailbreakMe 3.0. <http://jailbreakme.com>.
- Apple Computer, Inc., 2014a: iPhone in business. <https://www.apple.com/iphone/business/ios>.
- Apple Computer, Inc., 2014b: Resources for IT and enterprise developers. <https://developer.apple.com/enterprise/>.
- Aprville, A., 2014: Inside the iOS/AdThief malware. <https://www.virusbtn.com/pdf/magazine/2014/vb201408-AdThief.pdf>.

- Bedrune, J. and J. Sigwald, 2011a: iPhone data protection in depth. HITB Amsterdam, HITB Amsterdam.
- Bedrune, J. and J. Sigwald, 2011b: iPhone data protection tools. <http://code.google.com/p/iphone-dataprotection/>, <http://code.google.com/p/iphone-dataprotection/>.
- Chen, C.-N., R. Tso, and C.-H. Yang, 2013: Design and implementation of digital forensic software for iPhone. 90–95, 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.
- Corporation, I. D., 2014: *Worldwide quarterly mobile phone tracker Q2 2014*. International Data Group.
- Forensic Telecommunications Services Ltd., ????: iXAM - Advanced iPhone Forensics Imaging Software. <http://www.ixam-forensics.com/>.
- Freeman, J., 2014a: Cydia Substrate. <http://www.cydiasubstrate.com>.
- Freeman, J., 2014b: WinterBoard. <http://cydia.saurik.com/package/winterboard/>.
- Friedberg, S., 2014: unTRUST. <https://github.com/strozfriedberg/unTRUST>.
- Gallagher, R. and P. Maass, 2014: Inside the NSA's secret efforts to hunt and hack system administrators. <https://firstlook.org/theintercept/2014/03/20/inside-nsa-secret-efforts-hunt-hack-system-administrators/>.
- Gomez-Miralles, L. and J. Arnedo-Moreno, 2015: Lockup. <http://www.pope.es/lockup>.
- Grispos, G., T. Storer, and W. B. Glisson, 2011: A comparison of forensic evidence recovery techniques for a windows mobile smart phone. *Digital Investigation*, **8**, 23 – 36.
- Hotz, G., 2007: iPhone serial hacked, full interactive shell. <http://www.hackint0sh.org/f127/1408.htm>.
- iPhone Dev Team, 2011: redsn0w. <http://redsn0w.com//>.
- Iqbal, B., A. Iqbal, and H. A. Obaidli, 2012: A novel method of iDevice (iPhone, iPad, iPod) forensics without jailbreaking. *Proceedings of the 8th International Conference on Innovations in Information Technology*.
- Lau, B., Y. Jang, C. Song, T. Wang, P. ho Chung, and P. Royal, 2013: Mactans: injecting malware into iOS devices via malicious chargers. <https://media.blackhat.com/us-13/US-13-Lau-Mactans-Injecting-Malware-into-iOS-Devices-via-Malicious-Chargers-WP.pdf>.
- Lisiansky, D., 2014: CCControls. <http://cydia.saurik.com/package/com.danyl.cccountrols/>.
- Miller, C., D. Blazakis, D. D. Zovi, S. Esser, V. Iozzo, and R. Weinmann, 2012: *iOS hacker's handbook*. Wiley.

- Oestreicher, K., 2014: A forensically robust method for acquisition of icloud data. *Digital Investigation*, **11**, **Supplement 2 (0)**, S106 – S113, fourteenth Annual {DFRWS} Conference.
- Petrich, R., 2014: Activator. <https://rpetri.ch/cydia/activator/>.
- Porras, P., H. Sadi, and V. Yegneswaran, 2010: An analysis of the ikee.b iphone botnet. *Security and Privacy in Mobile Information and Communication Systems*, A. Schmidt, G. Russello, A. Liroy, N. Prasad, and S. Lian, Eds., Springer Berlin Heidelberg, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Vol. 47, 141–152.
- Rabaiotti, J. and C. Hargreaves, 2010: Using a software exploit to image RAM on an embedded system. *Digital Investigation*, **6**, 95–103.
- Ritchie, R., 2014: Apple reaffirms it has never worked with any government agency to create a backdoor in any product or service. <http://www.imore.com/apple-reaffirms-never-worked-any-government-agency-backdoor-product-service>.
- Rosenbach, M., L. Poitras, and H. Stark, 2013: iSpy: How the NSA accesses smartphone data. <http://www.spiegel.de/international/world/how-the-nsa-spies-on-smartphones-including-the-blackberry-a-921161.html>.
- Ruan, K., J. Carthy, T. Kechadi, and I. Baggili, 2013: Cloud forensics definitions and critical criteria for cloud forensic capability: An overview of survey results. *Digital Investigation*, **10 (1)**, 34 – 43.
- Soghoian, C., 2007: A game of cat and mouse: The iPhone, Steve Jobs and an army of blind hackers. <http://www.cnet.com/news/a-game-of-cat-and-mouse-the-iphone-steve-jobs-and-an-army-of-blind-hackers/>.
- Team, C. D., 2010: greenpois0n. <https://github.com/Chronic-Dev/greenpois0n>.
- Technology, G., 2014: Good Technology mobility index report Q2 2014. <http://media.www1.good.com/documents/rpt-mobility-index-q2-2014.pdf>.
- Umeng, 2014: Umeng Insight report: China mobile internet 2013.
- Vidas, T., G. Zhang, and N. Christin, 2011: Toward a general collection methodology for Android devices. *Digital Investigation*, **8**, s14 s24.
- Wang, Y. D., N. Bassen, et al., 2013: evasi0n. <http://evasi0n.com>.
- Y.-T. Chang, Y.-C. T., K.-C. Teng and S.-J. Wang, 2015: Jailbroken iPhone Forensics for the Investigations and Controversy to Digital Evidence. *Journal of Computers*, **11**.
- Zdziarski, J., 2008: *iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets*. O'Reilly.
- Zdziarski, J., 2014: Apple responds, contributes little. <http://www.zdziarski.com/blog/?p=3447appleresponds,contributeslittle>.

Zdziarski., J., 2014: Identifying back doors, attack points, and surveillance mechanisms in ios devices. *Digital Investigation*, **11**, 3–19.