Mobile Forensics:
Comparison of extraction and analyzing methods of iOS and Android

Dmitrijs Abalenkovs, Petro Bondarenko, Vijay Kumarraju Pathapati, André Nordbø, Dmytro Piatkivskyi, John Erik Rekdal, and Pieter Bloemerus Ruthven

Digial Forensics I
Master in Information Security,
Gjøvik University College

December 6, 2012

Abstract

This paper will explore methods for getting access to, extracting and initial analyzing non-volatile stored data from mobile smart devices in the light of the forensic principles like evidence integrity, evidence dynamics, chain of custody and order of volatility. The goal of this paper is to present current work on extraction methods and secondly how to do the initial analysis of the data extracted in order to compare the known methods between the chosen devices.

******

Introduction

Mobile smart devices come in many flavors. They have evolved from different sources like mobile phones, laptops and music players into ”smart” mobile devices of different sizes. Apple’s iPod, iPad and iPhone have a lot in common. They share the iOS operating system, applications and the main difference between them is how they connect to the rest of the world and what additional features they got. Android is a Linux based operating system developed by Google, available on a variety of hardware produced by 3rd party companies. iOS is only available for Apple devices.

Different reasons for trying to get access to data on a mobile device can be motivated by trying to install pirated software or otherwise circumvent restrictions, install malicious code, perform backups, restore incidental deleted files, verifying security features or extracting data for forensics analysis. The requirements for the extraction methods in use depend on the goal.

The Event-based digital forensic investigation framework in [1] describes a phased model of readiness, deployment, crime scene investigation and presentation. The readiness is the pre-crime preparation phase. It is described as setting up logging, developing and testing tools. A detection followed by confirmation leads to a crime scene investigation. The deployment phase has to consider legal requirements, and when it is started, three important steps are performed: preservation, searching for evidence and reconstructing events. Documentation is essential. The last phase is presentation of the findings, often in court.

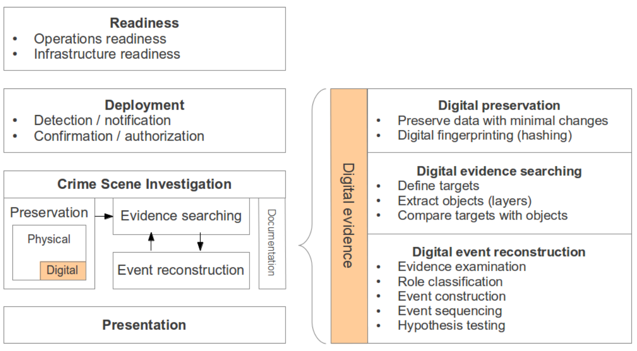


Figure 1: Summary of framework, based on [1]

Digital evidence is said to be a subset of physical evidence, and the same methods apply. One of the main differences between physical and digital evidence is the possibility of making an exact duplicate of the crime scene, and at the same time the danger of transferring analogies from the physical to the digital domain. The authors question the real need for an exact bit for bit copy of all digital objects with this contra analogy:

”Fingerprints are lifted from walls at a crime scene, but the wall is not seized as evidence." [1]

The integrity of preserved data must be assured by applying hashing algorithms. Targets must be identified and searched for, matches must be verified and put in context with other digital and physical evidence. Hypothesis are created and tested against the available digital evidence.

In the following sections we will have a quick look at some background, followed by one sections for each of Android and iOS, both divided in 4 subsections: Relevant security features, accessing and dumping data, initial analysis of extracted data and experimentation. And finally a conclusion.

Background

In traditional digital forensics computers are turned off, storage medium imaged and hashed for off-line analysis. Known operating system like Windows, OS X and Linux have been studied for years and we know a great deal of how to extract data and what information structures are used. Automated tools have been developed to support these tasks. During the last few years, smart mobile devices have hit the market. They assume tasks traditionally being performed on personal computers, they are mobile and always connected. It is assumed they contain a lot of forensically important traces, both because they can be tightly connected to an individual and at the same time can be connected to many of the same services traditionally used on laptops and desktops. Examples may include access to online storage, e-mail accounts, photographs, video and sound recordings, notes and documents, geographic data, Internet usage, chat sessions, social networking services [2] and game statistics.

A mobile smart device, just like personal computers, have volatile memory, processing unit and permanent storage. Permanent storage might include internal storage, and removable memory cards. Examples of communication are via WiFi, 3G/4G/EDGE/GSM etc, bluetooth, infrared (IR), near field communication (NFC) or via a serial bus interface commonly utilizing USB or FireWire. Data could potentially be extracted via many of these, but for practical reasons a high bit rate connection is preferred. WiFi and serial connection are therefore the two most obvious. If the integrity of the device itself is not critical one might also consider opening the device for extraction of the internal memory chip(s). Still one important difference is that mobile devices are typically embedded with "everything on one chip" because of the size requirement. Mounting the storage device to a write blocker and creating an image is not as easy as with a traditional desktop or laptop device.

Using mobile phones in forensics investigations is not a new phenomena. Triangulation of phone position and call logs received from the service provider have been used in court to argue for or against the possibility a suspect could have been involved in a crime scene. Mobile smart phones are rapidly getting equipped with GPS [1], high resolution cameras, motion sensors in addition to powerful computational abilities. There have been news reports claiming permanent storage of GPS logs on these devices [3], and being capable of both audio and video recording stored content could be used to further build a case.

Digital data can be extracted at different abstraction levels. A file dump would be copying files from a file system to another. The main drawbacks are not getting deleted (unlinked) data, slack space data and meta data used by the file system itself. A disk dump, often called a "bit by bit" image would then be copying all data off the medium. At this level, overwritten data is still not accessible and there might be protected areas on the medium not being copied. Recovery and carving for deleted content is possible. The main drawbacks of this method is that it often requires the device to be powered off or not in use to ensure integrity and it requires the same amount of storage as the original. One abstraction level lower at the physical layer even more information might possible be recovered but are seldom used because of the cost. Even reading the medium itself might be difficult in certain situations like when dealing with RAID or uncommon interfaces.

Preserving the integrity of the digital crime scene objects has at least two perspectives. The first is proving to the court no change has been introduced during the investigation. The chain of custody is an integral part of it. The other is avoiding any contamination of possible evidence. Modifying a storage area in order to get access might render deleted data unrecoverable.

The terms "rooting" and "jailbreaking" are often used in the context of bypassing security restrictions on mobile devices. Rooting refers to getting root access in a unix like environment. Jailbreaking "escaping jail" is bypassing the restrictions in an iOS device. These methods require a vulnerability to exploit and can be applied at different abstraction levels, during boot or withing running applications. They can be installed temporary in memory or permanently installed to the storage medium.

[1] Global Positioning System

3.1 Protection mechanisms

3.1.1 PIN, gesture and password

To restrict access to a device several methods can be chosen from. These are PIN, password, and gestures. A PIN is a string of digits ranging from 0-9 and four digits long. The password protections is a variable length alphanumeric pass-phrase including symbols. The gestures has also been widely adopted, and is a pattern between discrete nodes.

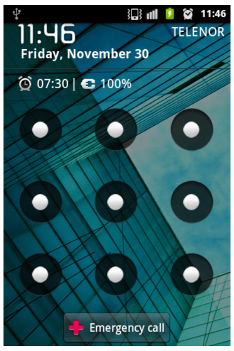


Figure 2: A gesture lock screen on Android OS

3.1.2 Dalvik

To create secure environment to execute applications Android puts every application inside Dalvik Virtual Machines, so that the execution environment of every application is fully isolated. But sometimes there is a need to share information and such mechanisms are provided by Content Providers sharing data using a relational database interface. Each content provider has an associated "authority" describing the content it contains. [4].

Besides DVM, there are basic protection mechanisms in Android that assign unique user and group ID to each application, storing the application data in dedicated place (in /data/data directory) where only this application can access. [5]

3.1.3 Protected/hidden files

Basically, there is no build-in mechanism for protecting or hiding files in Android, but there are few applications that can be used. Example of these can be HideNSeek, File Cover or Top Secret [6].

3.1.4 Encryption

Official support for encryption on Android phones have been available since Android version 3 (Honeycomb). Support for older versions are being supplied through third party applications. Android utilizes the 128 bit AES (Advanced Encryption Standard) algorithm. The key used for this encryption is derived from the PIN code or a password, i.e. gesture lock can not be used with encryption. [7] [8]

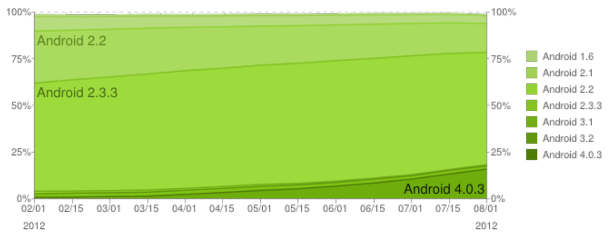


Figure 3: Android version market share, image copied from [9]

A recent market survey indicate that more than 60% of Android phones do not support native encryption in the OS. The new versions of Android support native encryption, but it is not enabled by default.

The most widely used version of Android is 2.3.3 with a share of 60.3%, but it tends to be changed. Android 4.0.3 (now has a share of 15.8%) becomes more popular every day [9].

3.2 Getting access

3.2.1 Access the device

To bypass the gesture mechanism the paper Smudge Attacks on Smart phone Touch Screens ([10]), utilizes a non technical way to bypass the gesture lock by analyzing the touch screen for grease trail left by the users fingers as seen on figure figure 4. Even after using a phone for several minutes after an unlock, as well as after having the phone in a pocket, smudge trails still remain viable. There are applications trying to counteract this. [11]

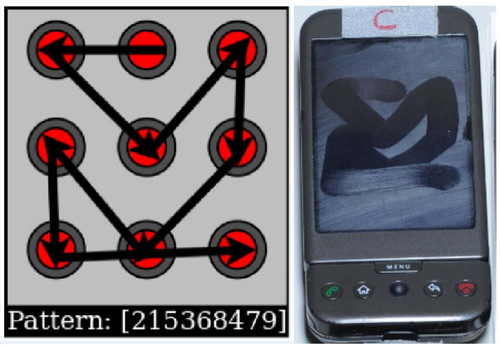


Figure 4: Original and smudge pattern, image copied from [10]

It might be possible to utilize this method on PIN codes too, but it will be hard to use on passwords. To bypass encryption on the phone the analyst would need to either brute force the password or actually get the password from the suspect. The same goes for the PIN, password and gesture if the analyst can not solve that through the method mentioned above. There are a lot of commercial available phone forensics tools that can help with brute forcing the PIN code, password and gesture.

3.2.2 Acquiring the data

Software

The amount of data that one is able to acquire depends if the device has been "rooted" or not. The methods how to root a device varies from model to model. Gaining root access on a device usually involves exploiting a device specific vulnerability in order to allow users to execute the switch user (su) command. This allows privilege escalation. [12]

In article [13] the authors described a method which allows to forge digital alibi in automated way. They claim it is really easy for software to simulate user activity on a device in such a way the modification will be hidden. That makes all evidences which were taken from devices running Android OS questionable. No proper protection method was found. We assume this is a hot topic in digital forensics now, or at least should be.

Logical forensic techniques work only through file system, so the tools can not work with unallocated area. Since using logical techniques does not require root access to the phone, you can apply it for any phone and it is really easy to use such tools. The only thing requirement is putting device in USB Debugging mode. Logical tools have some strict limitations. The most important without having root access only files belonging to current user can be accessed, which means copying most of the forensic relevant data is restricted. Still, a lot of interesting data are owned by the user.

Android debug bridge (ADB) is a powerful tool that allows establishment of a connection between the device or emulated instance. When connected to a device, it builds a virtual network over the USB connection and creates a server on the development machine and a client on the phone. The SDK tool can copy the data from phone to another computer with pull command. Without root privilege only files that can be accessed with normal user access rights can be copied. Even though, such data as unencrypted applications, most of the tmpfs[2] file systems, and directories /proc and /sys can be of interest. [5]

Majority of forensic software implementing logical technique use Content Provider to gather information. In fact, that could be enough to investigate digital crime scene. Examples of Content Providers are call logs, phone contacts, IM messages and SMS's. AFLogical is software implementing a logical technique of data acquisition and freely distributed to law enforcement agencies. It uses Content Providers to get the data and then puts all gathered data into CSV[3] files which can be viewed and analyzed later. One of the files can be of great importance - info.xml which contains information about the device such as the IMEI number. In addition, there are a lot of commercial software such as EnCase Neutrino, viaForensics and viaExtract, but they use the same methods as AFLogical. They have different features and advanced visual representation of data, but skilled user can achieve the same results with AFLogical."

In contrast to the logical techniques, the physical techniques are harder to implement, but the analyst can get access to much more information from the device. Both software and hardware methods can be used. The prerequisites for using the software methods are that one must have root access to the phone. When using physical techniques, investigators usually image the whole device with a utility such as "dd". Since it copies unallocated space, there is a possibility to recover deleted data. Such a methodology poses a high forensic soundness since the data are taken without any modifications and integrity can be assured by using hash functions.

Another method, proposed in [14], uses the boot CD concept to acquire data by using an unlocked recovery console or fastboot mode on the device. This method seems forensically sound as no changes to the operating system is made. The recovery console is used for updating the firmware on the phone. Some manufacturers have released the unlocks for their devices allowing a user to access fastboot mode [15]. After the unlock of the fastboot mode or by using a custom recovery console an analyst can run a custom update from the SD card. This custom update is not an update per se but can contain a custom script that can be used to acquire data from the android device onto the SD card.

Hardware

The Joint Test Action Group (JTAG) was created in 1980's as a means for testing the wiring and interconnects on printed circuit boards. This standard is now widely accepted and today most printed circuit boards (PCBs) have JTAG test access ports (TAPs). The problem with using this method is that it might be difficult to locate the TAPs on the PCBs and trace them to the central processing unit (CPU). The JTAG schematics that can ease the way of doing this, are often considered confidential by the manufacturer. In most cases the analyst would need to check the volt output for each tap.

To get access to the data on the device you would have to physically connect to these tags, this can be done through soldering. Errors in soldering to the JTAG or applying the wrong voltage could possibly disable and or damage the device. However if this is done properly the phone can be reassembled and will function normally with no data loss. All in all this is a difficult operation and should only be done by qualified personnel with sufficient training and experience in soldering and JTAG operations. We consider forensic soundness of JTAG low because there

[2]Common name for temporary file storage facility on unix-like operating systems
[3]Comma Separated Values

is a possibility of changing or even losing the data while soldering. One should use it as a last resort.

There is another method for physical acquisition called chip-off. This is where the chip itself is removed from the PCB and connected it to a special hardware device. This method can be used on destroyed devices where logical acquisition is not a possibility, however the problem here is to get the chip off the PCB without damaging it beyond repair resulting from connectors being destroyed. However, if successful, an image of the NAND flash can be extracted [5]. The integrity of this method is as high as it can get as long as data is not destroyed.

3.3 Data analysis

3.3.1 File System Considerations

Due to the open nature of the Android operating system, manufacturers of devices are able to customize the configuration of the operating system to suit their specific hardware requirements. One such customization is the choice of file system implementation. This choice is dependent on the selected internal memory storage type.

Memory chips can either require RAW access or more conventional BLOCK access. In case of RAW memory, the file system has to understand the underlying memory architecture and perform functions specific to flash memory, such as wear leveling (to prolong device lifespan). BLOCK memory on the other hand can be accessed similarly to a more traditional hard drive and thus allows the use of conventional file systems as the flash specific functions are built into a translation layer in physical memory. [16]

There is thus no single de facto file system being used across all devices. However, the choice is usually limited to YAFFS (Yet Another Flash File System) and YAFFS2 for RAW memory types, and EXT or FAT to BLOCK memory types.

Most Android devices till date have been using either YAFFS or YAFFS2 as their file system of choice. However, newer devices (Android v2.3 and up) have started to more frequently implement BLOCK memory using the EXT4 file system. [17] This shift from YAFFS eases the work for forensic investigators, as it allows the use of more commonly accessible, well established commercial and open source tools.

3.3.2 Data locations

Different types of data might hold value for the purpose of forensic analysis. Artifacts, such as photos and documents can either be stored on the internal memory of the phone, or on the SD card.

The SD card is most often the primary storage location for data stored by the end user. It will typically use the FAT file system in order to allow access from other operating systems. It should be a trivial task to image and analyse it using existing forensic software.

Device memory holds a large amount of system protected data that is not directly accessible to the end user. The data can only be accessed by the operating system and by specific application instances. Application data can be a key source of information to the forensic investigator. Amongst other things, it includes instant messages, emails, browsing history, log files, user accounts, location data and call information.

Android systems are typically partitioned as follows [18]:

- /boot
- /system
- /recovery
- /data
- /cache
- /misc

Application data is stored in the /data partition. The path to application specific database files normally being“ $/data/data/ < applicationname > /databases/$”. This would be a good point for an investigator to start his analysis. Other areas of interest on the device memory could be the /cache partition, which contains temporary application data and /misc, which contains carrier and hardware information. [5]

3.3.3 Data Structures

Android relies on Linux as a base operating system, thus there are a lot of known sources of useful forensic artifacts. In addition to known Linux sources there are a few Android specific areas of interest.

Application data is typically structured in SQLite3 database files. Databases can contain anything from clear text instant messages (e.g. WhatsApp), call logs and SMS messages and is a rich source of forensic artifacts. In addition, the analysis of SQLite databases obtained from physically acquired images can be used to recover deleted items. There are a number of commercial tools that can perform such analysis. In addition to SQLite databases, XML files are also commonly used to store application configuration settings and preferences. [5]

System data is typically structured in plain text format log files, as is usual on a Linux based system. Logs fall into three categories: kernel, system and application logs. Kernel logs are low level logs and mostly provides information regarding communications with hardware, they are of minor interest to an forensic investigator. From system and application logs the analyst can get such information as geographic location, application specific details and timestamps. [5]

When obtaining a physical bit by bit copy of the memory device (internal NAND memory or SD Card) it is possible to retrieve deleted artifacts by analyzing slack space in files as well as unallocated space on the file system. The process used to recover such objects is called carving, whereby a search is done for known file structures within the binary image of the memory. The type of file system used on the device will determine to a great extent how successful such an exercise would be.

FAT and EXT file systems allow for relatively easy analysis of unallocated space since there is a large knowledge base detailing how these file systems work. There are a wide range of tools available for such analysis.

On the other hand, as previously mentioned, the use of YAFFS can prove to be challenging. There is currently limited support amongst forensic analysis tools, with no existing official commercial support. [5] In addition, YAFFS2 utilities a garbage collection function to free up blocks that will eventually permanently delete all obsolete data on the storage area, this could make it difficult for investigators to recover usable data. [16] Luckily there are also some positive points regarding YAFFS. The openness of the file system makes it possible for an investigator to discover its internal workings - however this could be a time consuming process. There are also open source carving tools, such as Scalpel that includes android configuration files in order to carve for known file types.

3.4 Experiment

3.4.1 Setup

The group used a Samsung Gio S5660 device, running Android version 2.3.6 (Gingerbread). The device was restored to factory defaults prior to the experiment and was not "rooted". No encryption was present on the device as there is no native support on this version of Android. A gesture was configured on the device, and for the purpose of the experiment was considered to be unknown to the person attempting to acquire the data.

The following tools were used: Odin 4.42, Clockworkmod 5.0.2.7, Aroma File Manager 1.8, FTK Imager 3.1.1, Windows 7 and 8

The objective of the experiment was to obtain a physical image of the /data partition while limiting any modification to the partition itself.

Challenges faced in acquiring the data were that the device had an unknown gesture pattern. As well as the fact that without modification the user would not have root privileges on the device and would thus not be able to access the entire /data partition directly.

The experiment was mainly based on ideas from [19].

3.4.2 Steps taken

It was decided to install a custom recovery console in order to gain the required privileges to the system. Clockworkmod version 5.0.2.7 (CWM) was chosen as this would allow execution of unsigned update.zip files, which would be crucial for the next stage. In order to install a custom recovery console we had to put the device into "download" mode by powering it off, and then holding volume-down, home and power buttons (this is specific to the device we were testing). No authentication was required to perform these steps, thus circumventing the pattern lock protection. CWM was flashed to the device's recovery console partition through USB using Odin 4.42 running on a Windows 7 machine, no other partitions were altered. Next, Aroma File Manager 1.8 (Aroma) was copied onto the device's SD card through a Windows 8 machine. Aroma is a customized update.zip style package that includes a pre-compiled "update-binary" executable. In a normal instance, the update-binary file is responsible for executing the update.zip script file [20]. However, in the case of Aroma, the binary executes a simple file manager that also gives you access to a terminal window - all operating with recovery mode's elevated privileges. Aroma was then executed on the device by using CWM's functionality to install any unsigned zip file from a SD card location [21].

By mounting the /data partition through CWM and viewing the /proc/mounts file in Aroma, it was determined that the /data partition resided on /dev/block/stl13. In order to calculate a hash value for data integrity assurance, the md5sum command was performed on /dev/block/stl13 and the resulting hash value recorded. Then the command, dd if=/dev/block/stl13 of=/sdcard/data.img, was issued through the terminal in order to do a bit-by-bit copy to the installed SD card. Afterwards an md5 hash value was calculated on data.img and compared to the one obtained earlier and confirmed to be matching.

The resulting image was transferred to a Windows machine running FTK Imager, where it was imported as an image file.

- The file system was identified as FAT16.
- Unallocated space was confirmed to be present and containing some data.
- The file structure could be traversed and SQLite databases extracted.

3.4.3 Some considerations

Ideally a forensic analyst should use a trusted utility, in our example a update.zip binary or script could have been created by the analyst himself and used to automate the acquisition. Another method could be to have a signed update.zip file which could be executed from a standard recovery console - this would require assistance from the device manufacturer but would have a lower impact on the integrity of the device. However, Aroma is an open source project and can be reviewed to understand internal workings. This would require the necessary skills and would be a tedious exercise.

The live state of the device was lost as we had to restart it in order to use CWM, possibly destroying most data that could have been obtained from running memory.

The device used in our experiment was not encrypted, an encrypted device would require additional effort through brute forcing attempts and might not be feasible in each case.

iOS

4.1 Protection mechanisms

The goal of data protection is to keep data safe even if it is compromised by a third party. On iOS devices encryption is bound to the device passcode. One can access the file system only if the device is unlocked. In other words, data protection is enabled after user sets up a passcode. iOS supports both four-digit and alphanumeric passcodes. According to [22] it would take 2.5 years to break a nine-digit passcode using only numbers, and more than 5.5 years to break a six-character alphanumeric passcode with only lowercase letters and numbers.

Figure 5: A lock screen of an iPhone 4S with passcode, image copied from [22].

There are three main pillars of security on iOS devices [23]:

1. a passcode, prevents unauthorized access to the device,
2. a keychain, stores sensitive data such as Wi-Fi passwords, Mail accounts, Safari passwords,
3. and the storage encryption.

The first iPhone was introduced in 2007. Thus, iPhones have already been 5 years on the market. The first iPhone ran on iOS up to the version 3.1.3 and had a crypto processor and two embedded AES keys (GID and UID[24, p. 7]) described later in detail. At that time the lockscreen was the only protection. The passcode (not its hash value) for unlocking the device is stored in the keychain. This security could be easily bypassed by just removing the record from the keychain or by removing the UI setting that asks for the passcode. Moreover, the storage encryption was not possible. The following generation of the iOS devices, such as iPhone 3G, had no real security improvements over the first generation.

But since both the iPhone 3GS and the iOS 3.x were introduced in 2009 every new iOS device such as an iPad or an iPod Touch has a dedicated AES-256 cryptographic engine[4], a so called hardware AES cryptographic accelerator. The task of the AES cryptographic accelerator is to encrypt the file system on the fly in real time.

An AES accelerator has a globally shared key (GID Key) and a unique per-device key (UID Key). Both these keys are not accessible to the CPU and can be used only for encryption and decryption through the aforementioned AES accelerator. The UID Key helps to derive some device-specific AES keys that are later on used to encrypt the file system metadata and files. The GID Key is mainly used to decrypt iOS firmware images given by Apple itself. In other words, it is impossible to extract the GID and UID Keys on all iOS devices. For a normal user it means that if an iOS device is turned off, the copy of the encryption key in the iOS device's accessible memory is deleted. That is why a forensics investigator would have to try all possible keys. And according to NSA this goal is impossible to achieve, even if one would have access to quantum computers [26].

With the iPhone 3GS Apple also introduced storage encryption, where only the user partition was encrpyted. There was no improvement over the passcode or the keychain. In 2010 Apple released the iPhone 4 that is shipped with the iOS 4. There is no notable enhancement in hardware security over iPhone 3GS, but it is a huge leap forward in software security — the passcode was now used to compute a passcode key. This computation is tied to a hardware key. It is important to mention that the same passcode on different devices would produce different passcode keys. That said, off-line brute force is impossible since it needs the hardware key. Brute force

[4]Until today AES is considered as an unbreakable algorithm and was adopted as a U.S. government standard in 2001. Moreover, the National Security Agency (NSA) has even approved AES-256 for storing top-secret data. [25]

attack on the device itself would be very slow. Furthermore, the passcode on iOS 4 has now 3 different types and thus three different security levels

- a four-digit passcode
- a passcode with digits only, longer than four
- an alphanumeric passcode of any length

As earlier though, only the user partition is encrypted. Apple also enabled per file encryption and "too many tries protection" — if a user types in the passcode 10 times incorrectly, the master key will be wiped out and all the data on the device will be unreadable.

In the iPhone 4S there were no enhancements in hardware security over the iPhone 4. However, the next generation of the iOS, the iOS 5 had some security improvements over the iOS 4 — all attributes in the keychain are now encrypted, and not only the password itself. Furthermore, a new LwVM[5] partition scheme was introduced. Now it is possible to encrypt all partitions and not only the user partition. The AES block mode was changed from CBC[6] to GCM[7]

The iOS 6 is claimed to be the most secure version yet. All in all, the iPhone 5 security was not examined thoroughly yet, thus it is not discussed here any further.

4.2 Getting access

4.2.1 Disk image

The paper "Universal, fast method for iPad forensics imaging via USB adapter" [27] explains how to use jailbreak with custom persistent software to image the device, and even suggests downgrading the iOS version as a last resort. If the goal is to extract deleted data, then this method can not be considered to keep the integrity of the data. The authors describe using a USB adapter intended to connect digital cameras to speed up the rate of transfer. They achieve a maximum speed of 15.9 MiB/s on an iPad.

In the paper "A Novel Method of iDevice Forensics without Jailbreaking" [28] dated early 2012 the authors explain that closed source tools like Katana's "Lantern" and Cellebrite's "UFED Physical Analyzer" use exploits against bugs in iOS in order to run unsigned code and by doing this in principle are actually jailbreaking. They then propose their method claiming:

> "Our method for imaging the iDevices does not require jailbreaking it as the required steps for the imaging are done on the RAM of the device hence the device storage is not alternated in any way as nothing is installed on it." [28]

[5]Lightweight Volume Manager

[6]Cipher-Block Chaining

[7]Galois Counter Mode

By definition[24, p. 20] whether data is stored in RAM has nothing to do with the fact that the jail has been broken to put the code in the RAM in the first place. Still forensically speaking, in order to preserve the integrity of the data a method that does not write at all (or very little) to the non-volatile memory is preferred. This example also points to a lack of definition or understanding of the term jailbreak. The mentioned technique requires the device to be rebooted.

Their method ([28]) uses the "Device Firmware Upgrade" mode of the iDevices (a term described as a collection of devices produced by Apple) and exploits a weakness in the booting stages allowing their code to break the chain of trust and be executed to get root access to the specific device. A RAM disk is prepared and loaded in memory with tools necessary to dump the content via a ssh tunnel wrapped in the USB connection. They claim the extraction to consume less than 30 minutes. That measure must depend on the storage size of the device. Time to brute force any password is not described.

There are two aspects not well described, first the authors does not explain explicitly what version of deices they used. They specify the use of iOS version 4.3 in the analysis section and that can be used to look up most likely actual hardware being iPhone 3GS, iPhone 4 or older (including iPad) [29]. The other one is that their tool is mentioned but not available. No verification of their method was possible because of this.

The iPhone-dataprotection tool [30] is an open source set of tool using seemingly exactly the same methods. It's written by Jean Sigwald[24, p. 119] a researcher at Sogeti ESEC Labs. It will be discussed in greater depth in the experiment subsection.

4.2.2 iTunes "logical" backup

This "back door" method of getting the backup files for the iOS device is described in [2]. Available content would be a file by file copy of files from the device meant to support restoration of the device to a previous state. iTunes separate applications, user data and "media content". Applications is separated since they do not contain any user data and the "media content" is simply a copy of files from the iTunes library not considered necessary to back up once more. Getting these files is as simple as finding them and copying them to an external medium. These data can be protected by any kind of full disk or volume encryption mechanisms and by iTunes[24, p. 131] itself.

Even if the device is locked and the key can not be brute forced, if an instance of iTunes has been synchronized with an iOS device iTunes will remember a token that will bypass the passcode in order to be able to perform automatic synchronization and backup. Even changing the pin or passcode on the device does not require a new "handshake" between the device and iTunes. This has been verified on both iPhone 4S and iPad. How to deal with an erased device, either by mistake or by will, is also an interesting question, and not to forget the possibility of backdoors put in place by Apple [31] in order to support regain of "lost access". A related weakness is that display of SMS is allowed by default without entering the passcode. In a setting used as a 2nd factor authentication, this weakens the potential protection.

4.2.3 Closed source tools

A search for closed source tools reviled a huge amount of forensics software targeting iOS. These tools are focused on automating the process of data extraction and develop nice looking analysis functionality. They often also come with cable accessories to connect to supported devices. They are not dedicated to iOS, but are multipurpose tools.

- Oxygen Forensic Suite 2012: Support for iPhones (including version 5), iPad (including version 3) and iPod Touch. A freeware version is available for 6 months [32].
- Micro Systemation XRY: The release notes for version 6.4 claims "logical" support for up to iPhone5 and they mention better support for functionality in iOS6. The "physical" section is empty.
- AccessData MPE+ claims to support up to iPhone4S and iPad3, but the only ones mentioned physically is "iPad" and "iPhone" [33]. They also claim on their product page that "No jail breaking required" [34]
- Elcomsoft iOS Forensics Toolkit discloses a lot of details of their support, but are restricted then it comes to iPhone4S+ devices as they then require the device to be jailbroken. [35]

This is a small sample of available commercial tools, and because they are closed source, it's difficult to determine both what exactly their claimed methods do and whether they are true. It seems like these tools stagger against the same "wall" found in the open source community when it comes to newer iOS devices.

Development of new hardware and new iOS versions makes extraction from iOS devices a moving target, a cat and mouse game between manufactures and all interested in getting access. Periods of waiting for new vulnerabilities to be discovered is to be predicted even in the future [36], jailbreaks development is fast because of high demand. Forensically sound methods tend to lag behind but can benefit from the same vulnerabilities found.

4.3 Data analysis

The social networking applications paper in [2] searched for predefined data in non-encrypted iTunes backup files (version 10.4.0.80) from 32GB iPhone 4 (version 4.3.3 8J2). They rediscovered [37] that each applications user data on the device is backed up to a folder identified by the unit (UDID) and named with a 40 character (160-bit) SHA string of the "domain"-"full path". The applications are backed up to a different folder. The files do not have a file extension, except from a few *.plist and one *.mbdb file. Using the **file** command on every file will reveal JPEG images, ASCII text, XML documents, SQLite databases, binary property lists etc. The tool **plutil** can be used to convert binary plist files to a human readable format. Some files might be base64 encoded. It is not possible to translate the files back to the original path and name directly since SHA is a one way function [38], but the Manifest.mbdb file contains all the information needed to restore the file structure. It can be parsed with a python script found on stackoverflow [39]. An open source tool at http://ipbackupanalyzer.com/ can be used to analyze these files in a graphical user interface.

Two open source tools iPhone backup browser [40] (Windows) and iPhone Backup analyzer [41] can be used to explore these files with built in decoders of the different file formats.

iOS comes with some pre-installed applications, while other applications has to be downloaded from the application store. The exact structure for user downloaded applications are application specific, but the important thing to realize is that a user might be using a different application than the operating. For example, using Opera mini instead of Safari. Bookmarks and browsing history will then not be found in the Safari specific files.

In order to analyze the dumped data support for file system HFSX [28] is necessary. The Sleuth kit supports HFSX [42]. Restoration of deleted data is discussed in the experiment subsection.

4.4 Experiment

Available hardware for experimentation

- iPhone 4S (A5 chip) running iOS 6.0.1
- iPad (A4 chip) running iOS 5.1.1.

4.4.1 Backup files

The non-encrypted backup files on a MAC running MAC OS X v 10.7.5 were found in the

> /Users/user/Library/Application Support/MobileSync/Backup/

folder, one sub folder for each device. The files were transfered to Ubuntu 12.04 and hashed and examined manually using **type** command, **cat** and using **SQLite**. All images (except images inside databases) were automatically previewed independent of the lack of file extensions. **iPhone Backup analyzer** was also tried. Some forensically interesting things:

- /var/mobile/Media/: Photos, books, downloads etc
- A very long list of access WiFi networks (com.apple.wifi.plist)
- SMS logs (sms.db)
- Notes (notes.db)
- Calendar (Calendar.sqlitedb)
- Address Book (AddressBook.sqlitedb)
- Residual thumbnails from deleted images (file hash: 66e9dfd32df4ba531e3e45c2a0b61e8b52926433)
- Call history
- Applications storing passwords in plain text

4.4.2 Physical dumps

iphone-dataprotection [30] on Google code has a readme file describing the steps necessary to exploit the iOS device, transfer a binary copy of the internal drive, get the keybag, and decrypt the files. The first part has to be performed on OSX (10.6+) since it requires Xcode[8] to prepare the bootable RAMdisk image with the tools[9]

This framework is intended on devices with similar architecture as the iPhone 3GS. Devices like the iPad 2 and iPhone 4S has newer hardware (A5 chips), and the current tools does not have an exploit for it, although it is being worked on [44]. iphone-dataprotection can be used on these newer devices if jailbroken and SSH client is downloaded via a alternate application store. In addition to Xcode, the device firmware (IPSW[10]) and the redsn0w exploit kit is used to create a kernelcache and the ramdisk and they are used in the firmare upgrade mode (DFU) to start a SSH server and brute force service on the device. A TCP tunnel over the USB connection is opened and python scripts can be run against it to perform different kinds of dumps. Both a dd[11] of "/dev/rdisk0s1s2" and a NAND dump are supported.

Both dump methods were tried. The "dump_data_partition.sh" dd copy method was performed with a speed of 8MiB/s. The default block size was set to 8192 bit in the file and by changing it to 32768 the speed went up to 9.3MiB/s. The file was hashed and moved to a more powerful Ubuntu machine for analysis together with the keybag from the brute force attack. A script "emf_decrypter.py" for decrypting was used on the image decrypting it in-line the file and then another script "emf_undelete.py" was run to find deleted files. Two folder "junk" and "undelete" were populated with 16 files each. The now plain text dump can be mounted using a loop back device and mounting it via "mount" read only. Since both the iOS file system and Ubuntu use the same file permission system for file access, a normal user will not be able to browse all files unless they happen to have the same user ID. A program **bindfs**[12] can be used to remount the file system with every file "world readable".

The NAND dump had a slower transfer rate downloading 32GiB in 95 minutes or 5.75MiB/s. The SHA1 was shown after the dump. The image were again moved and analyzed for deleted files with the "ios_examiner" script. It requires the package m2crypto available in the Ubuntu repository. Compared to the 32 found files previously this method found 62,429 files, each with a created time stamp. The earliest files were created February 2010. 9,265 of the files were restored in 34 hours. 9,087 of them were claimed probably OK with the difference being files partly recovered. This demonstrates how time consuming the task is. According to system monitor only one core were utilized so there is a possibility of speeding up the method. Because HSFX is journaled keys for individual files with unique keys can be found in the journal for recently deleted files, and many files are not protected individual at all[24, p. 142].

Interesting findings are snapshots of applications[24, p. 116 and 285] [13], old images noted deleted a long time ago, random icons from web surfing and deleted applications, deleted databases (All friends from Facebook was found in a restored sqlite database). Because of the wear leveling when using NAND technology, writes are spread across the medium and it is thus more likely to find old deleted data.

[8]Apple's development kit

[9]These files can be precomputed and tools exist for automating these methods as it will download the necessary precompiled files from the Internet [43]

[10]http://osxdaily.com/2010/10/25/download-iphone-firmware-ipsw/

[11]dd is a common unix tool for "dumb" bitwise copy

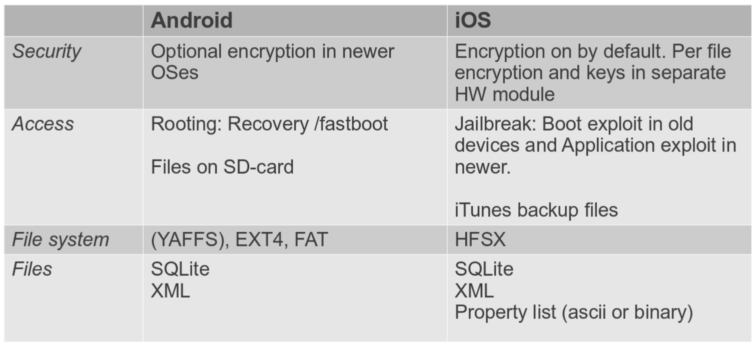
[12]http://code.google.com/p/bindfs/

[13]Screenshots are used to enhance the user experience to make it feel more responsive

Conclusion

It is clear that Apple has an advantage producing both the hardware and the software, and that the security mechanisms contribute greatly to how easy it is to acquire data from a device. Android typically does not have full disk encryption on by default and when it is actually used the encryption is performed in software. For iOS a dedicated chip unavailable to the CPU has stored a version specific and device unique cryptographic key, and as long as there is no known way to extract these keys any brute force attempt must be performed at a slow rate on the device itself. The key hierarchy in later iOS versions has built in some back doors in order for iTunes to access and synchronize without the user having to unlock it. Methods tried for a forensical extraction were boot rootkits/jailbreaks because they leave the smallest footprint. On the tested Android devices a simple dd copy was sufficient. For iOS devices open source tools could only work on iPhone 4 (GSM version), iPad 1 or older iOS devices. New devices has newer hardware where no known vulnerability were found yet. They can still be jailbroken at the application level but that would leave a lot of traces. Even on the old devices a brute force attack must be performed, and files must be decrypted before standard tools such as Sleuth kit can work on them.

Extraction of data, at least from iOS devices is a moving target, a mouse and cat game where the protection mechanisms are getting sufficiently strong, relying on a bit by bit dump is no longer practical. The same phenomena is seen on PCs and laptops with disk encryption enabled, with the major difference being that iOS devices have protection on by default. Referencing back to the forensics framework, maybe it is time to loosen on the strict integrity requirement that every bit has to be preserved and instead focus on improving the chain of custody methodologies? Other related aspects are whether or not the integrity of the device can be trusted at all when a device is rootkitted/ jailbroken and/ or malware is found on it. The understanding of the terms *rooted* and *jailbroken* does not explain fully what they do in terms of how persistent they can be and what they modify on a device. A lot of commercial tools go to a great length to create simple interfaces to present content of databases and XML structures to the user. At least for the open source tools there is still a long way to get closer to the same level of user friendliness and functionality. Digital forensics on mobile devices is getting a more important topic since more and more people are using mobile devices.



	Android	**iOS**
Security	Optional encryption in newer OSes	Encryption on by default. Per file encryption and keys in separate HW module
Access	Rooting: Recovery /fastboot Files on SD-card	Jailbreak: Boot exploit in old devices and Application exploit in newer. iTunes backup files
File system	(YAFFS), EXT4, FAT	HFSX
Files	SQLite XML	SQLite XML Property list (ascii or binary)

Future work

- Integrity considerations given root kit or malicious software
- The continued updating of current tools and automating parsing of application data
- Understanding of the term jailbreaking and rooting

Acknowledgments

Anders O. Flaglien for supervising our group and giving us relevant feedback and interesting articles to read

References

[1] E. H. S. Brian D. Carrier, "An event-based digital forensic investigation framework," *Digital Forensic Research Workshop*, 2004.

[2] A. M. Noora Al Mutawa, Ibrahim Baggili, "Forensic analysis of social networking applications on mobile devices," *Elsevier*, 2012.

[3] J. Cheng, "How apple tracks your location without consent, and why it matters." http://arstechnica.com/apple/2011/04/how-apple-tracks-your-location-without-your-consent-and-why-it-matters/. (Visited November 2012).

[4] P. M. W. Enck, M. Ongtang, "Understanding android security." http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4768655&isnumber=4768640. (DOI: 10.1109/MSP.2009.26).

[5] A. Hoog, "Android forensics: Investigation, analysis and mobile security for google android," 2011.

[6] http://www.technoon.com/how-to-hide-password-protect-multimedia-files-on-your-android-phone.html. (Visited December 2012).

[7] Google. http://source.android.com/tech/encryption/android_crypto_implementation.html.

[8] E. Arvidson, "Encryption on the android." http://www.ehow.com/info_12183909_encryption-android.html.

[9] http://9to5google.com/2012/08/01/android-version-market-share-numbers-come-out-over-60-still-on-gi
(Visited December 2012).

[10] E. M. M. B. Adam J. Aviv, Katherine Gibson and J. M. Smith, "Smudge attacks on smartphone touch screens." http://static.usenix.org/events/woot10/tech/full_papers/Aviv.pdf.

[11] Whispersystems, "Android and data loss protection." http://www.whispersys.com/screenlock.html. Visited December 2012).

[12] Wikipedia, "Android rooting." http://en.wikipedia.org/wiki/Android_rooting. (Visited December 2012).

[13] G. C. G. D. M. A. D. S. P. Albano, A. Castiglione, "On the construction of a false digital alibi on the android os." http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6132892&isnumber=6132770. (DOI: 10.1109/INCoS.2011.129).

[14] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6042741&tag=1.

[15] B. Reid, "Unlock bootloader of any htc android device without voiding your warranty." http://www.redmondpie.com/unlock-nearly-all-htc-bootloaders-while-sparing-your-warranty/. (Visited December 2012).

[16] S. S. F. C. F. Christian Zimmermann, Michael Spreitzenbarth, "Forensic analysis of yaffs2." http://subs.emis.de/LNI/Proceedings/Proceedings195/59.pdf.

[17] crve at h online.com, "Android 2.3 gingerbread to use ext4 file system." http://www.h-online.com/open/news/item/Android-2-3-Gingerbread-to-use-Ext4-file-system-1152775.html.

[18] H. Q. Raja, "Android partitions explained: boot, system, recovery, data, cache and misc." http://www.addictivetips.com/mobile/android-partitions-explained-boot-system-recovery-data-cache-misc/, 2011. (Visited November 2012).

[19] C.-T. L. Sheng-Wen Chen, Chung-Huang Yang, "Design and implementation of live sd acquisition tool in android smart phone," *IEEEXplore*, 2011.

[20] http://wiki.opticaldelusion.org, "Update-binary." http://wiki.opticaldelusion.org/wiki/Update-binary. (Visited December 2012).

[21] XDAdevelopers, "Aroma filemanager 1.8." http://forum.xda-developers.com/showthread.php?t=1646108. (Visited December 2012).

[22] Apple, "iOS Security." http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf, May 2012.

[23] B. Hat, Andrey Belenko, and Dmitry Sklyarov, "Evolution of iOS Data Protection and iPhone Forensics: from iPhone OS to iOS 5." http://media.blackhat.com/bh-ad-11/Belenko/bh-ad-11-Belenko-iOS_Data_Protection.pdf, 2011.

[24] J. Zdziarski, *Hacking and Securing iOS Applications.* O'Reilly Media, first ed., 2012.

[25] NSA, "NSA Suite B Cryptography." `http://www.nsa.gov/ia/programs/suiteb_cryptography/index.shtml`, January 2009.

[26] M. Technology Review, "The iPhone Has Passed a Key Security Threshold." `http://www.technologyreview.com/news/428477/the-iphone-has-passed-a-key-security-threshold/`, August 2012.

[27] L. Gómez-Miralles and J. Arnedo-Moreno, "Universal, fast method for ipad forensics imaging via usb adapter.," pp. 200–207, IEEE, 2011.

[28] H. O. B. Iqbal, A. Iqbal, "A novel method of idevice(iphone,ipad,ipod) forensics without jailbreaking," *IEEEXplore*, 2012.

[29] Wikipedia, "ios version history." `http://en.wikipedia.org/wiki/IOS_version_history`. (Visited November 2012).

[30] J. Sigwald, "Automatic ssh ramdisk creation and loading." `http://code.google.com/p/iphone-dataprotection/wiki/README`. (Visited December 2012).

[31] M. Honan, "Kill the password: Why a string of characters can't protect us anymore." `http://www.wired.com/gadgetlab/2012/11/ff-mat-honan-password-hacker/all/`. `http://www.wired.com/gadgetlab/2012/08/mat-honan-data-recovery/all/`.

[32] oxygen forensic, "Oxygen forensic suite 2012 (standard) freeware." `http://www.oxygen-forensic.com/en/freeware/`. (Visited November 2012).

[33] AccessData, "Mpe+ supported devices." `http://www.accessdata.com/mpe-supported-devices`. (Visited November 2012).

[34] AccessData, "Mpe+ mobile phone examiner plus." `http://www.accessdata.com/products/digital-forensics/mobile-phone-examiner`. (Visited November 2012).

[35] Elcomsoft, "Elcomsoft ios forensic toolkit." `http://www.elcomsoft.com/eift.html`. (Visited November 2012).

[36] D. Madden, "iphone 5 untethered jailbreak: Cross it off your xmas wishlist." `http://www.autoomobile.com/news/iphone-5-untethered-jailbreak-ios-6-3/1008699/`. (Visited November 2012).

[37] A. Crosby, "iphone forensics without the iphone." `http://www.slideshare.net/hrgeeks/iphone-forensics-without-the-iphone#btnNext`, 2010. (Visited November 2012).

[38] S. Bommisetty, "iphone forensics – analysis of ios 5 backups." `http://resources.infosecinstitute.com/ios-5-backups-part-1/`, 2012. (Visited November 2012).

[39] StackOverflow, "How to parse the manifest.mbdb file in an ios 4.0 itunes backup." `http://stackoverflow.com/questions/3085153/how-to-parse-the-manifest-mbdb-file-in-an-ios-4-0-itunes-backup`. (Visited November 2012).

[40] "Browse the files of local iphone/ipod backups." `http://code.google.com/p/iphonebackupbrowser/`. (Visited November 2012).

[41] M. Piccinelli, "iphone backup analyzer." `http://ipbackupanalyzer.com/`. (Visited November 2012).

[42] wiki.sleuthkit.org, "Hfs." `http://wiki.sleuthkit.org/index.php?title=HFS`. (Visited December 2012).

[43] http://msftguy.blogspot.no/, "Automatic ssh ramdisk creation and loading." `http://msftguy.blogspot.no/2012/01/automatic-ssh-ramdisk-creation-and.html`. (Visited December 2012).

[44] J. Sigwald, "Support for a5 devices (iphone 4s, ipad 2)." `http://code.google.com/p/iphone-dataprotection/issues/detail?id=49`. (Visited December 2012).