The effectiveness of remote wipe as a valid defense for enterprises implementing a BYOD policy

By

Ali Servet Uz

A thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the M.Sc. degree

In Systems Science

Abstract

In today's work place where corporations allow employees to use their own smart phones to access their company's network and sensitive data, it is essential to ensure the security of said data. When an employee smart phone is compromised, companies will rely on the remote wipe command that attempts to remove sensitive data.

In this thesis, we analyze the effectiveness of remote wipe commands on the Apple iPhone and Android model devices and demonstrate how data can be recovered following a remote wipe procedure. We conduct two experiments on each device to verify whether remote wipe is a viable defense mechanism or not. Furthermore, we touch on the subject of mobile forensics used by law enforcement and review methods and techniques used to recover data for use as evidence in criminal cases.

Acknowledgements

I would like to thank my supervisor, Dr. Carlisle Adams for his unwavering support and guidance throughout the writing of this thesis. Without his meticulous feedback and persistent help this thesis would not have been possible. He has set an example of excellence as a researcher, mentor, instructor and role model.

I would like to thank my friends, colleagues and professors who have all contributed a great deal to this thesis in terms of discussion and suggestions. I am very grateful to all of you.

Finally, I would especially like to thank my amazing family for the love, support, and constant encouragement I have gotten over the years.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

The rising popularity of smart phones is a fact that can be ignored no longer. Long gone are the days where mobile devices were used just to place and receive phone calls, we are in the era of the smart phone. Smart phones, are mobile devices that are built on mobile operating systems with more advanced computing capability and connectivity than a feature phone. They are used to check e-mail, perform financial transactions, browse the internet, place calls, and for other similar activities that a user would perform on a personal computer. The significant increase in these devices' popularity and usage has given rise to the "consumerization of smartphones", in which users, or more precisely, employees, use their personal devices at work to access the enterprise network and data. This trend is specifically referred to as BYOD (Bring Your Own Device) or BYOT (Bring Your Own Technology). Recent research shows that BYOD has a high rate of acceptance among companies and employees and that this rate will only continue to rise due to the fact that BYOD is becoming increasingly more common within corporate environments and will soon become inevitable [1].

The use of personal devices in the enterprise has several advantages that benefit both the company and the employee. Foremost among the positive aspects is convenience. Equipped with a smartphone, employees literally have everything they need in the palm of their hand—contacts, schedules, e-mail, access to corporate data, search engines and applications. The ability to access data wherever work takes them means employees can not only get closer to customers, but do so without the physical barrier of a desk or even a laptop between them. For professionals from salespeople to physicians, this is a distinct plus.

Mobile devices are also more powerful than ever before, thanks to the latest processors. Their performance has made them equivalent to—rather than poor substitutes for—desktop and laptop alternatives.

Because mobile devices are more popular than ever before, both companies and consumers benefit from manufacturers' economies of scale. Mobile devices are now available at a reasonable cost, especially compared to a desktop computer. Gone are the days when special-purpose tablets came highly customized with a price tag to match; mobile devices now use standard parts in standard configurations, making it simpler for IT to configure them for the corporate network. And finally, with technology that's much more intuitive than ever before and increased ease-of-use within applications, the learning curve is lower and the time-to-value for new applications is shorter. User acceptance is higher, which not only improves productivity but also enhances the relationship between IT and employees [79].

Although the benefits of BYOD are plentiful it also comes with significant risks and security challenges.

The fragmented market of smartphones makes device manageability a non-trivial task. With different smartphone models such as BlackBerry, iPhone, Android, and Windows Phone, the different architectures and operating systems each device runs on can pose a challenge to companies that want to centralize device management to a single interface and security issues may arise due to the complication of controlling what application each user installs on his or her phone.

Services such as Dropbox, Google Drive, and other cloud based technologies enable users to copy files into the cloud for later retrieval. Corporate information residing in such services may pose a security risk since they no longer reside in the protected corporate boundaries. Furthermore, smartphones are popular targets for malware: a

recent report from Symantec determined that mobile malware has increased by 58 percent last year compared to the year before. Although this is a significant increase, at this point in time, the infection rate is less than 0.1 percent and relatively small when compared to the number of infected personal computers, but it remains an issue nonetheless [81]. Of all mobile threats, 32 percent attempted to steal information stored in phones [55]. An infected device can leak corporate data, e-mail, financial transactions and much more, causing great damage to a company's finances and reputation while also letting competitors take advantage of the privileged information and use it to their advantage.

Although the security risks mentioned here threaten the implementation of BYOD, the most vital security challenge entails managing lost or stolen devices. Device theft or loss was ranked first as the factor that had the greatest impact on the vulnerability of mobile data, followed by malicious applications downloaded on to the smartphone [35]. While a Good Samaritan may return the lost device back to its owner, a thief may target the device for its contents such as passwords, e-mails, online banking information and sensitive enterprise data. A common thief may steal the device just to sell it on the black market not concerning himself over its contents; however, a corporate spy may actually seek to retrieve information to help a competitor or individual gain an advantage. The cost of a stolen and compromised device is significant. It will result in substantial exposure of enterprise secrets and can cause insurmountable damages to the company's finances and reputation, perhaps resulting in its bankruptcy.

Corporate espionage has been around for decades. In 1811 Francis Cabot Lowell traveled to England and stole the plans for the Cartwright loom, which he memorized while touring a factory. With it, Lowell brought home the blueprints for America's industrial revolution [21]. Corporations greatly value sensitive information and will engage in some form of corporate espionage or intelligence to gain information about

their competitor. Corporate espionage is generally viewed as a covert or illegal practice of investigating competitors, usually to gain valuable information resulting in some form of advantage for the competition. It can occur in several ways, such as using a 'mole' or trusted insider to leak information. Such a person can be induced willingly or through bribes to act as a spy for the competitor [71]. A rising trend for corporate espionage these days, however, is mobile device loss or theft.

Growing concerns over the risks and challenges of implementing BYOD have paved the way for software called Mobile Device Management (MDM). Companies providing MDM solutions have become popular with BYOD implementations by offering services such as securing, monitoring, managing and supporting smartphones from most vendors and models. These services would allow an enterprise to centralize the management of a fragmented smartphone cluster, control the delivery of mobile content and most importantly, secure the mobile device against malicious agents. The primary concern of MDM is to enforce security in a BYOD implementation. MDM simplifies the security by centralizing operations such as malware detection, mandatory PIN/Password locks, restricting applications that violate policy, remote lock, full and selective wipe. The MDM server is generally placed within the enterprise and employees register their devices with the server. MDM encompasses the entire BYOD lifecycle from initial device enrollment to device retirement when the employee leaves the enterprise.

MDM applications have user agent software that is downloaded and becomes the focal point for communications with the MDM server. After the appropriate client agent is downloaded, it's installed onto the device. From there, the agent software examines the phone or device state, reports findings to the MDM control server, and the MDM control server then delivers instructions to the phone. The agent then makes

adjustments to the phone's or device's behavior according to rules/policies dictated by selections made administratively in the MDM application via the received messages.

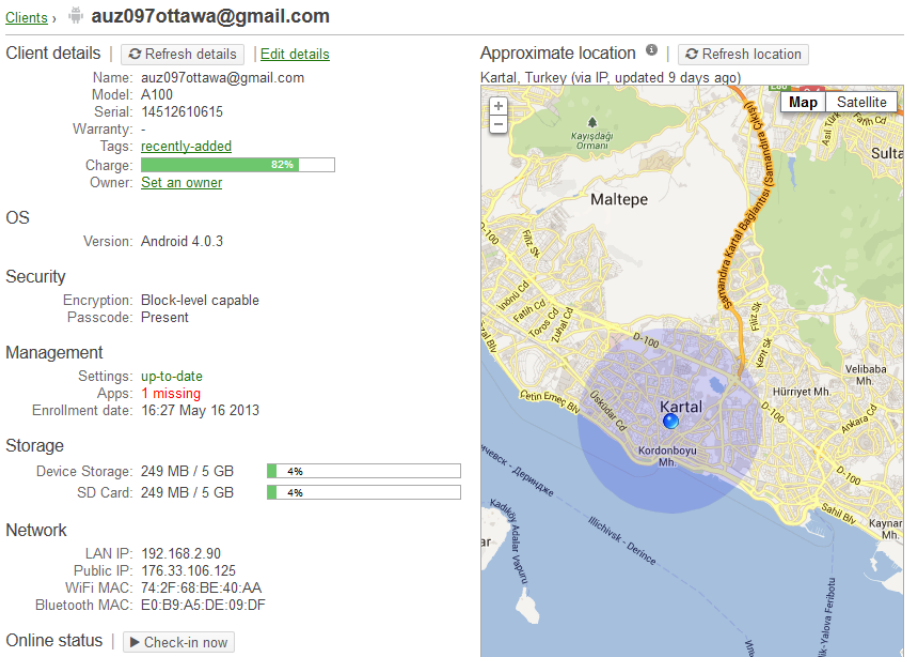

Figure 1. Smartphone state on MDM interface

The smartphone user agent app periodically "phones home" or sends messages via an email or SMS proxy application to communicate the smartphone state. Messages are then pushed (sent to the mobile device), where the resident agent software reacts to them, changing policy, adding or disabling features, perhaps backing up phone data, or reacting to changes. MDM applications must control a range of features and be able to detect numerous smartphone or device states. This becomes complicated for MDM application vendors for many reasons as there are strong differences between platforms, carriers, operating systems, and user options [22].

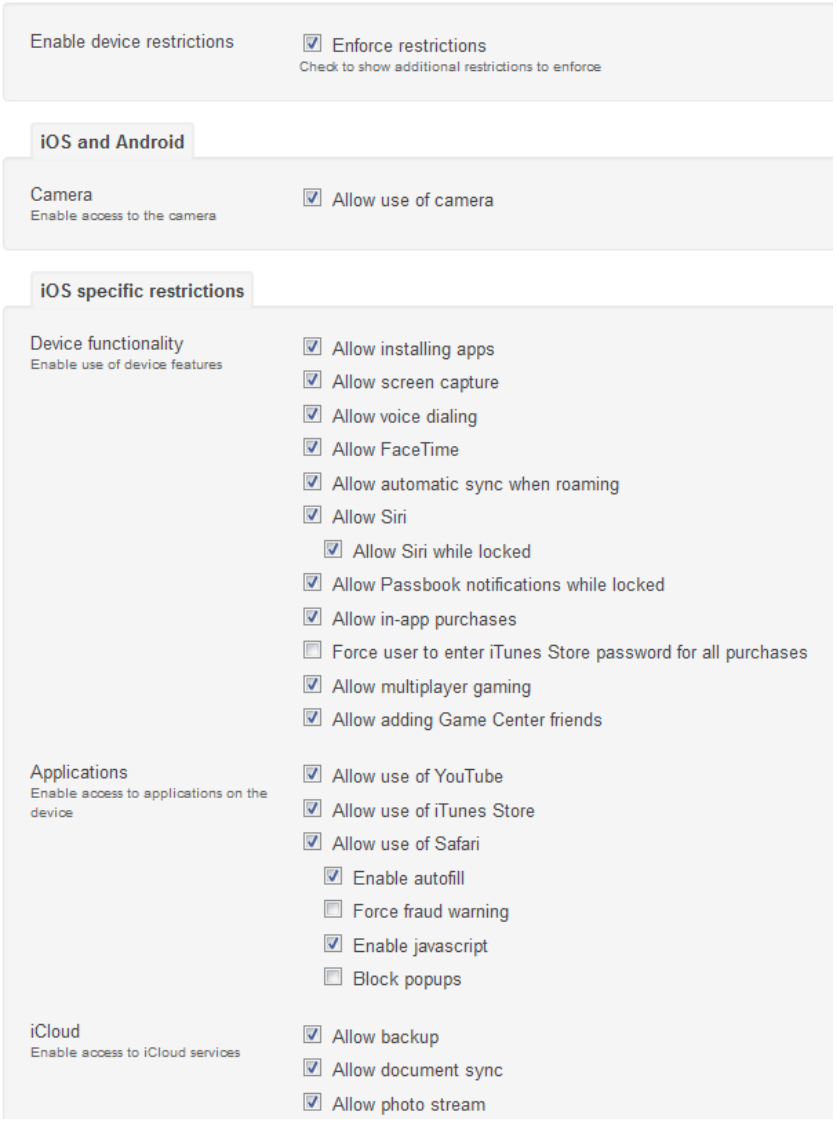

Figure 2. Sample MDM policy list

MDM applications can use containerized solutions or leverage the smartphone's inherent security to protect enterprise data. The latter uses the smartphone's native applications to perform its operations and apply policies. For example, passcode restrictions are enforced at the device level, which means users must enter their passcode even when using their smartphone for personal use. Moreover, enterprises gain powerful application management capabilities, such as blacklisting/whitelisting and silent install and removal of applications. Native MDM solutions must rely on the

mobile OS for its security and any exploit that compromises the OS may also compromise enterprise data.

A simpler native MDM solution like Microsoft Exchange ActiveSync provides employees with enterprise e-mail, calendar and contacts. It allows an enterprise to expose Exchange e-mail to the Internet, which lets smartphones connect from anywhere. But ActiveSync doesn't have all the features of a full-fledged MDM application. It cannot push applications, apply policies, or grant employees access to resources.

The second solution is setting up a secure container on the employee's smartphone which allows total segregation of enterprise data. Usually, the container is protected by a password, which the employee must enter every time he wants to access it. Additionally, data at rest in the container is or should be encrypted, although the implementation may vary with each vendor.

In containerized solutions, the application replicates and replaces native OS capabilities, such as the e-mail client and browser, and forces the user to use non-native applications to access enterprise resources. Third party replications may degrade user experience because of their restricted functionality and dissimilarity with their native counterparts.

When retiring a device, a selective wipe is performed that removes only the container as opposed to fully wiping the device contents. This preserves personal information and privacy by only removing enterprise data.

Both MDM solutions have the capability to remotely lock, wipe, and enforce passcodes on the device. Remotely sending commands is one of the cornerstones of MDM security and is intended to preserve device integrity.

1.2 Motivation

BYOD is becoming increasingly popular and the security risks are very concerning. Smart phones use Flash technology as internal storage, which operates in a significantly different manner than a hard disk drive (HDD). This difference is also attributed to the way file deletion is performed on Flash. In an HDD, when a file is securely deleted by overwriting the physical sectors pertaining to the file's location with a specific pattern, recovery of said file is usually impossible. When performing the same task in Flash, due to the underlying architecture, a file that is deleted and overwritten can still be recovered by analyzing the physical medium. We examine Flash technology in Chapter 3 and present a clear picture of its underlying architecture.

This thesis examines the effectiveness of the remote wipe feature on certain mobile devices. Our motivation for choosing this topic stems from our interest in MDM solutions and the performance of their remote wipe functions. We aim to analyze whether a remote wipe is a valid defense for a compromised device or if an attacker can retrieve any residual data after the device has been wiped.

1.3 Contribution

In this thesis, we analyze the security of using remote wipe as a defense mechanism for a compromised device. We demonstrate on an Android tablet and an iPhone smart phone that while the remote wipe seemingly erases sensitive data on the surface, an attacker can retrieve a portion of the erased data through the use of what we call 'offensive forensics', which describes the use of forensics methods to unearth sensitive information not intended to be seen by unauthorized parties. Budgetary reasons limited us to performing our analysis on an Android Iconia A100 tablet and an iPhone 4G and using an MDM software's 30-day trial as a test client. In order to preserve its privacy, the company has not been named in this thesis.

1.4 Outline

In the next chapter, we survey the literature in MDM and BYOD security and present their findings and their shortcomings. In Chapter 3, we examine the inherent security mechanisms of the Android and iPhone smart phones and the additional security that is contributed by MDM products. In Chapter 4, we introduce our findings through a simulation of scenarios where remote wipe is and isn't successful and the attacker attempts to recover data in each outcome. In Chapter 5, we discuss the topic of mobile forensics and the different methods and challenges of extracting data from a lawfully confiscated device. In Chapter 6 we conclude, discuss the possibilities of securely erasing information, and suggest some directions for future work.

Chapter 2

Literature Review

2.1 Introduction

As BYOD gains popularity so do MDM products marketing their ability to facilitate BYOD integration in companies. While academic journals and papers have yet to address this topic, many security conferences (BlackHat, Defcon, Hack in Paris, etc…) have invited speakers and called for papers on BYOD and MDM security. However, in an academic framework, research is still in its infancy signaling the need for academics to publish more in this field. Due to the aforementioned facts, unfortunately, peer reviewed papers were not available at the time of writing; however, we were able to locate corporate white papers and peer-reviewed conference papers. In this section, we present a survey of related work, pertaining mostly to breaking the containerization structure of MDM due to weak passwords, incorrectly implemented encryption schemes and vulnerabilities to web exploits on the administration interface. We also look at the privacy and legal aspect of BYOD in terms of ownership of information and data residing on an employee's smart phone.

2.2 Literature review

2.2.1 Blanchou

In *Auditing Enterprise Class Applications and Secure Containers on Android* **Marc Blanchou** from iSec Partners provides a deep analysis of container-based MDM solutions [14]. This paper focuses on the security of Android in the enterprise environment under the assumption that the device has been compromised (lost or stolen). Although MDM providers are marketing their solutions by underlining increased mobile security, it turns out that these solutions are not so secure after all and most security features can be bypassed by a knowledgeable attacker.

However, the fault does not solely rest on the MDM providers' shoulders; part of the blame for the ineffective security can also be attributed to the Android OS. Due to the

inherent limitations of the OS, MDM products do not have the architecture to properly enforce their policies.

Android is currently the fastest growing mobile platform with thousands of new users every day. It currently holds the highest market share, passing all other smartphone vendors in number of units sold [15]. Android users are also using their devices for business which also indicates that these devices are beginning to store sensitive information.

The author mentions that the Android security model has been erroneously based on the laptop security model. This is because, unlike the laptop, a mobile device rarely gets shut down, it is always turned on and always connected, making the laptop security model insufficient. The "always-on" connectivity raises new security risks and attack vectors that are not common to personal computers.

In order to facilitate the adoption of smart phones in the enterprise, a 'security container' model is being adopted and provided by the current generation of MDM companies. Most of these companies claim to provide enterprise-grade security for Android devices and the author insists that there is a need to assess such security claims due to the ever increasing rate of mobile device use in the enterprise.

The author picks the two most popular MDM software providers at the time of assessment (2011 - March 2012) - **Good Technologies** and **Mobile Iron** on Android [16 & 17].

Android devices, by design, run each application in its own sandbox based on privilege separation. Each application runs with its own User ID (UID) and Group ID (GID). Similar to Linux, Android applications are prevented from accessing resources from another UID. Furthermore, applications are not allowed to access or modify data that might negatively impact other aspects of the system. In order to do so, an application

must explicitly request permissions that are not inherent within the sandbox model and the user must approve this request upon installation. Such permissions may include accessing or modifying user data such as other application's files, emails, contacts or using network resources. Starting with Android 3.0, nicknamed Honeycomb, additional security measures are provided such as full disk encryption (FDE), however it is still common to encounter devices with Android versions below 3.0 [22].

Android is marketed as a consumer device and has very limited inherent features that would keep it secure in an enterprise environment. To address these limitations MDM companies have developed enterprise class applications that attempt to improve the security and manageability of Android devices. They introduce the ability to perform remote analytics, remote wipe, password management and encryption of corporate data. Secure containers attempt to segregate corporate data through the encryption of enterprise email, contacts, calendars and attachments on the device. These containers rely on a PIN to decrypt the data. This PIN is usually distinct from the unlock PIN on an Android device.

The author explains the threat agents associated with the assessment of these products; these include the individual attacker with reverse engineering skills who is able to analyze the internal workings of the product and bypass the security implementations, the corporate spy, who is after corporate information and seeking ways to compromise the employee's device, and finally government and law enforcement agencies who wish to access information on mobile devices due to legal reasons.

The two MDM products both claim they can ensure the security of enterprise data at rest against the mentioned threats. Quotes from the main page of both vendors' websites indicate that they sell their product through the guise of increased security. Some quotes include:

“Governments have tested the product and approved it for their most sensitive deployments”

“Over-the-air transmission and enterprise data at rest on the devices are secured with industry-leading AES-192 encryption.”

In Good for Enterprise, the author observes that sensitive data is retrievable when the device powered on and locked (PIN-locked). The application stores enterprise data in a SQLite database; the data is then encrypted with a symmetric key. The key is maintained in memory, even when the device is locked and the application is not in use. Being able to access this key on a stolen device compromises the security of all data stored by the application: email, contacts, appointments, and identifiers allowing an attacker to impersonate the user. By dumping the live physical memory of the device, an attacker can analyze the dump to retrieve the key and decrypt the enterprise data.

The symmetric key is derived from the user’s PIN code, usually a 4-digit number and easily brute forced. A second vulnerability, identified by the author, is to retrieve the user password and pass it through the identified algorithms to derive the database encryption key. Verification of the password requires computing a symmetric key’s hash using a Password Based Key Derivation Function (PBKDF2-SHA1 with a number of iterations, see PKCS #5 and PBKDF2 function) and one SHA1 operation. Good for Enterprise uses the PIN and an 8 byte salt. The hash and salt are obfuscated with an XOR between a static string and the hash as well as a great number of bitwise operations. This is then stored in an XML file located in *data/data/com.good.android.gfe/shared_prefs/GoodLockPrefs.xml*

By brute forcing the hash and salt using a GPU cluster, the author was able to reconstruct the password used to derive the master key. Using the GPU to break a PBKDF2 implementation is very effective and has been proven by brute forcing a

wireless network's WPA2 password in which the key is defined as PBKDF2 (HMAC−SHA1, passphrase, SSID, 4096, 256) where HMAC-SHA1 is a pseudorandom function, the passphrase is a master password from which a derived key is generated, SSID is the public name of the wireless network used as a salt, 4096 is the iteration count and 256 is the length of the derived key, which in this case is 256 bits [20]. More than a hundred thousand keys per second can be generated using the already old Radeon HD 5970 [19]. An attacker using a GPU cluster such as a FPGA or Amazon EC2 instances would have considerable computing power for a relatively cheap price.

Further vulnerabilities are identified with the product's jailbreak detection process. Jail breaking an Android device means obtaining root permissions allowing a user to perform actions that would otherwise be restricted such as modifying or deleting system files and so on. MDM products contain jail break detection functions that determine if the user has root permissions on the device or not. The author observes that these functions are performed by looking for a specific list of applications and binaries, and returns true if it finds any one of them. For example, the product tries to become root by running 'su -c ls' and expects it to fail on a non-root phone, it may also look for the "Superuser.apk" application file which is a common application that is installed after jail breaking an Android device. These detections are weak against a root exploit like "Rage Against The Cage". Furthermore, renaming the 'su' variable in 'system/bin/su' on a rooted phone would also prevent detection.

As with Good for Enterprise, Mobile Iron also exhibited similar vulnerabilities. The AES-256 encryption key used is generated using a pseudo random number generator (PRNG) with the android_id (UUID) as its seed. The Android UUID is a constant number for the lifetime of the device and shouldn't be used with encryption algorithms. An attacker can easily retrieve the UUID and derive the key. Furthermore, Mobile Iron's jailbreak detection was also prone to similar weaknesses as with Good for Enterprise.

The author does a thorough analysis on the security of the two MDM products, but does not mention possible attacks that could be executed following a remote wipe. The author also assumes that the Android devices that are compromised have USB debugging already turned ON, allowing the attacker to connect to the device via adb.

2.2.2 Brodie

In *Practical Attacks Against Mobile Device Management* [44] presented at the BlackHat Conference 2013, senior security researcher for Lacoon mobile security **Daniel Brodie** explains how MDM can be tricked through hidden code in applications, which run after installation.

The main topic involves two attack models: *Mass Mobile Malicious Apps* and *Targeted Mobile Attacks*, aka Spy phones.

The first attack, *Mass Mobile Malicious Apps*, is performed when a consumer oriented malicious application finds its way on to the user's smart phone in hopes of financial gain. Such malware may include applications that monetize on premium texts, dialers, SMS spammers, and mobile banking applications. However, this malware does not target enterprise data, and instead targets the user's personal information and identity. Therefore, these types of malware are not significantly detrimental to an organization and its data.

In *Targeted Mobile Attacks*, mobile surveillance software is installed on particular individual devices turning the device into a spy phone. These infected devices are privy to all data on the device, as well as to all communication passed on the device.

The ultimate goal of deploying such attacks is cyber-espionage, so therefore spying applications are tailored for the specific individual that it is meant to spy on. These applications are called mRATs (Mobile Remote Access Trojans) and they are commercially available for sale, Figure 3 shows a web site comparing and rating several

mRATs. As such, the impact of such an attack to an organization is significantly high, from gaining access to corporate emails and extracting memos discussing company roadmaps, to recordings of confidential phone calls and board meetings. The differences between the two attack models and their impact is depicted in Figure 4.

Figure 3. Comparison of mRATs

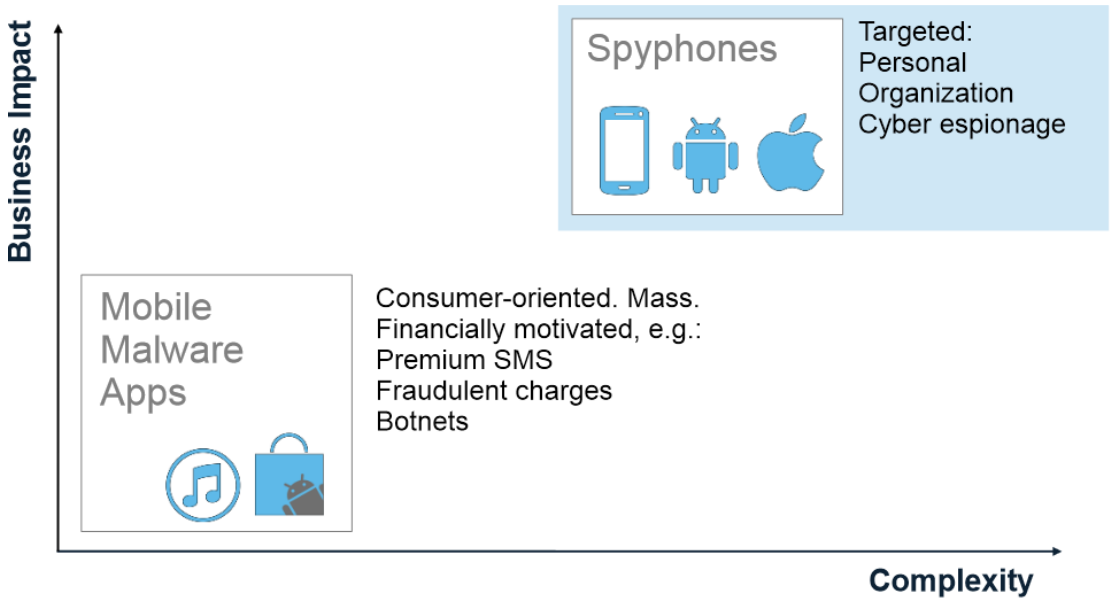

Figure 4. Identifying between Malware and Spy phones

From a sample of 500,000 smartphones the authors' survey revealed that 1 out of 800 devices (as of June 2013) were infected with a spy phone. The following figure shows the infection distribution rate by mobile OS:

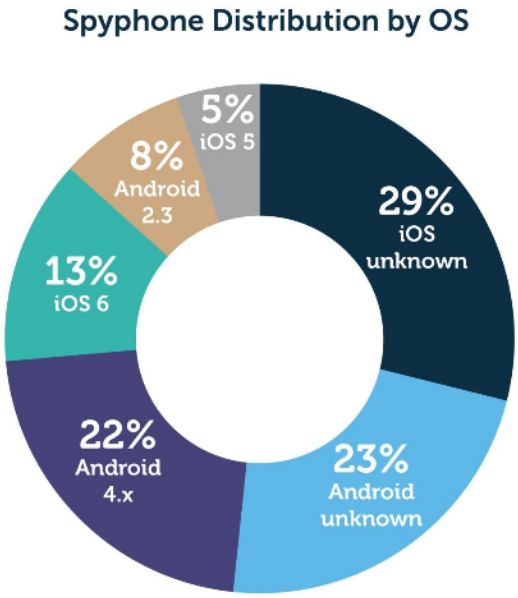

Figure 5. Mobile OS infection rate

The author introduces MDM secure containers, which are designed to protect and segregate enterprise data through means of encryption, jailbreak detection, restricting the installation of hostile applications, and by leveraging the built-in mobile sandbox environment.

A two-stage attack is presented to bypass MDM security restrictions; the attack involves publishing a seemingly innocuous application through the Google Play market and getting the user to install it via phishing attempts or having physical access to the device. The application then downloads the rest of the payload, which is the Trojan used to turn the smart phone into a spy phone. The payload leverages an OS vulnerability to gain root privileges and creates a hidden 'suid' binary file that will be used for specific operations. The payload can then listen to important actions such as the reading of an e-mail. Once an action is recorded, the payload dumps the physical

memory to a location where the attacker can analyze and retrieve it unencrypted. The detailed attacks are presented for both the Android and the iPhone devices:

Android:

1 – The victim installs a seemingly innocent looking application published by the attacker. Once the application is installed, it will commence a download operation of the malicious code to be executed

2 – The malicious application exploits a privilege escalation vulnerability, such as the Exynos chipset exploit for Samsung Galaxy devices.

3 – A hidden 'suid' binary is created for operations requiring root and is hidden in an execute-only directory where MDM jailbreak detection is unlikely to find it.

4 – The device is now transformed into a spy phone and listens to 'adb' logs.

5 – The spy phone will wait for a log event that represents a user reading an email

6 – The spy phone dumps the heap and finds the email structure, extracts it and sends it to the attacker's server. The attacker can then view the contents with a hex editor:

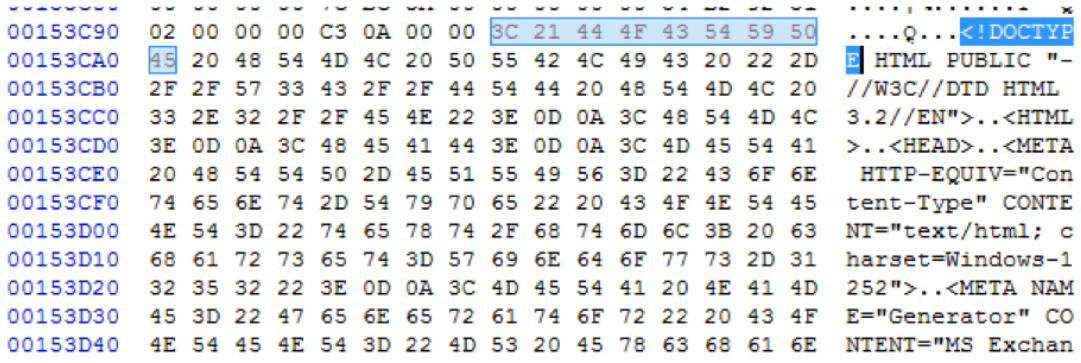

Figure 6. Reading unencrypted e-mail in hex editor

iPhone/iPad:

1 – The attacker, similar to the Android version of the attack, installs a signed application on the targeted device.

2 – The attacker uses a Jailbreak exploit in order to inject code into the secure container.

3 – The attacker removes any trace of the jailbreak.

4 – The spy phone places hooks into the secure container using standard Objective-C hooking mechanisms.

5 – The spy phone is alerted when an email is read.

6 – Finally, the spy phone sends the email to the attacker's server.

While the author presents novel attack vectors against MDM secure containers, these attacks, especially on i-devices, are non-trivial operations. In order to trick a victim to install a repackaged application that secretly downloads and installs malicious code, he or she would need to explicitly accept the installation of the certificate and then of the application itself. Also, based on the attack vectors, the authors have not broken the container so much as extracted clear-text data from the mobile device's volatile memory. While these attacks are quite devastating to the enterprise, it requires a lot of user involvement in order for it to work as intended.

2.2.3 Andrivet

In *Security of MDM (Mobile Device Management)* [40], presented in Hack in Paris 2013, **Sebastien Andrivet** analyzes two MDM products: **Mobile Iron** and **Good Technologies**, in order to test if an administrator or an IT employee can read/steal emails without authorization and without trace. Only some aspects such as the installation/deployment, the enrollment of devices and the management interface are tested due to the extensiveness of the software.

The Mobile Iron application has an administrator interface to manage the configuration of the server, network, email, certificates and so forth. An administrator can also create other local users that can be assigned administrative privileges.

From the Mobile Iron administrator interface, an employee can issue a "magic" request that displays the secure container password of other employees. However, to issue this request, the employee must be authenticated as an administrator. The author, by creating a new user, intercepts the data in transit between the web application and the server and discovers that the local user usernames and passwords are being sent in clear-text. This vulnerability allows one administrator to steal the password of another and impersonate him or her. However, in order to exploit this vulnerability the user must have administrative privileges to begin with.

Figure 7. Intercepted data showing passwords in clear

The author presents further vulnerabilities such as Cross-site scripting (XSS) exploits residing within the administration interface. Mobile Iron was susceptible to XSS in various areas while Good for Technologies was only vulnerable at one point of their interface.

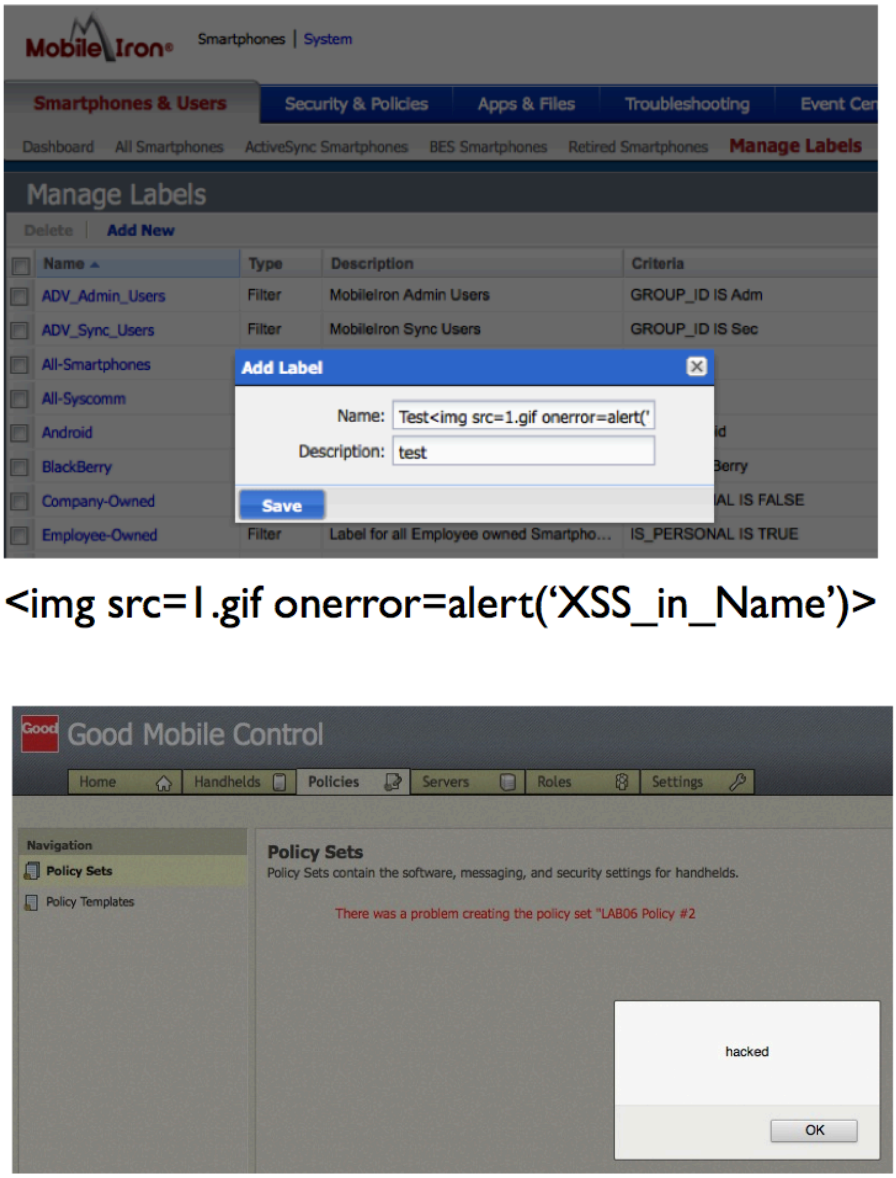

Figure 8. XSS vulnerabilities on both products

Both products were vulnerable to Cross-Site Request Forgery (CSRF) and neither product implemented anti-CSRF measures. Through the use of XSS and CSRF attacks the author was able to remotely enable the passcode lock of any enrolled iPhone device and also assign administrator roles to any user in the database.

Another serious vulnerability concerns the encryption of users' passwords. The author findings show that the application initializes five fixed symmetric keys during installation and chooses one at random to encrypt passwords. When a user enters his or

her password, the verification is done by using each key one by one to decrypt the encrypted password in the database and compare the result to the entered value.

While these attacks were reported and supposedly fixed, the author states that other problems still remain. Unfortunately these problems are not revealed in the public version of the presentation.

2.2.4 Schwarton

In *BYOD - The Privacy and Compliance Risks from Bringing Your Own Mobile Device to Work* [75], presented at Hack in Paris 2013, **Winn Schwartau** talks about the rising popularity of consumer oriented devices in the enterprise. Companies not implementing a BYOD policy observe a decrease in productivity as users want to use their smartphones for personal and work related activities. That being the case however, the author notes that BYOD is also associated with an increased risk to the enterprise. The current BYOD trend is similar to the use of personal computers (PC) in the enterprise during the 1980's, when they were used to connect to the main enterprise mainframe posing a huge security risk to corporate assets. The PCs were implemented with a flawed security and subsequent virus infections resulted in financial damage and theft of data. BYOD is thought of as the same flawed concept where devices, which are not designed for enterprise use, are being used in the enterprise.

Almost 2 billion devices are connected to corporate networks worldwide and while this number is rising steadily, interviewed CTOs are not aware of the specific number of devices connected to their own network, which factors in to the aforementioned risk.

The author criticizes MDM solutions for being marketed as products that convey a false sense of security to the corporations and managers that are implementing them. The truth lies in the fact that the security of the leading MDM products can only be as good as the security of the mobile operating system of the device the product is installed on.

Furthermore, device loss and theft is very common, especially in North America. Statistics show that 1 out 4 employee devices (25%) get lost every year.

Recent research confirms that both the iPhone and Android operating systems have been prone to malware, exploits and further security exploits. The risk of implementing BYOD is mainly associated with storing personal and corporate data on smart phones with an inherently insecure operating system and environment. Nevertheless, these factors are not considered when measuring the risk of implementing BYOD, mainly due to the efforts of legal and marketing departments.

A popular method used by MDM products to illustrate the security of their solution is to implement containerization, which is a sandbox client, implementing its own web browser and e-mail client, used to segregate corporate data. However, MDM companies, who aren't specialized in building and implementing browsers or email clients, implement their own version and thus become susceptible to security vulnerabilities and breaches within their containerized data.

The author comments on the legal aspect of BYOD as well, in which a company may declare, when an employee accepts their BYOD policy, that they legally own the device. This may result in putting the employee's privacy and personal data at risk. A company, by leveraging certain laws and policies, may decide to completely wipe the employee's personal device if it determines that the device may be a threat to the corporate network. This action may cause the employee to pursue legal action against the company due to privacy invasion. On the other hand, the author questions whether a personal device accessing and containing corporate information belongs to the employee or the company.

The current situation of MDM shows a bleak and unconvincing scenario: the amount of risk associated with deploying MDM solutions is significantly high given that

containerization has been broken, the security of MDM has been debunked, and personal/corporate data segregation is flawed at best. The author, at the end of the presentation, recommends the use of 2 devices, one for personal and the other for corporate use, thus preventing most of the security issues discussed previously.

Yet, within the context of BYOD, bringing two devices goes against the current trend of using one device for work. Employees are eager to bring their personal device to work and expect enterprise data to be segregated. While the security of BYOD is under scrutiny, the concept is fairly recent and needs time to develop its security measures for various types of implementations. The author also fails to mention that MDM companies are reactively fixing vulnerabilities published by academic researchers and moving towards building much better products with security as a priority.

2.2.5 Wei et al.

In *Reliably erasing Data from Flash-Based Solid State Drives*, presented at the USENIX FAST '11 conference, **Wei et al.** evaluate traditional hard-disk sanitization methods when used on solid state drives (SSD). The authors empirically evaluate the effectiveness of hard drive and of the SSD's built-in sanitization methods and commands by extracting raw data following the execution of these methods and commands. Flash based storage has a very different internal structure compared to that of a hard drive and it is unclear whether traditional sanitization methods work as effectively on flash based storage as they do for hard drives.

The authors define sanitizing as a process of deleting all or part of a storage device so that the data contained is difficult or impossible to recover. Many government standards and secure erase programs use multiple overwrites to sanitize data on hard drives and therefore many industries rely on these methods to erase sensitive data. As far as hard drive sanitization goes, there has been no demonstration of data recovery following such a sanitization process. SSDs defer from hard drives in almost every

aspect. SSDs use flash memory to store data. Flash memory is divided into blocks, where each block is divided into several pages. Write operations are performed on a per-page basis while erase operations are performed on a per-block basis. Once a page is written to, it cannot be modified before erasing the entire block that the page occupies. The flash translation layer performs the mapping between the logical block addresses and the physical pages via an ATA or SCSI interface.

The mismatch between the write and erase operations prevents the overwriting of a logical sector. When the sector is overwritten, the FTL writes the new data to another page and updates its map so that the new data appears as the target logical sector. The previously written page still remains in physical memory and is referred to as a "digital remnant". Since physically overwriting single pages are not possible on SSDs, most traditional sanitization methods that perform multiple overwrite operations do not perform as effectively as they do on traditional hard drives.

The authors use repeated fingerprint data to identify the recovered data following an erasure method. The methods tested are: internal SSD commands, repeatedly writing over the drive using traditional methods, and degaussing the drive.

Built-in sanitize commands: most modern SSDs have built-in sanitize commands that instruct the drive's firmware to run a sanitization process. The commands and firmware are built by the manufacturer so each drive implements different methods for sanitization. The built-in methods involve the use of ATA security commands named "ERASE UNIT", "ERASE UNIT ENH" and "BLOCK ERASE", all of which perform the same function: erase all user-accessible areas on the drive. The authors tested the internal commands for twelve SSDs and the results showed that only four SSDs out of twelve correctly executed the sanitization process (Table 1).

SSD #	Ctlr # & Type	SECURITY ERASE UNIT	SEC. ERASE UNIT ENH
A	1-MLC	Not Supported	Not Supported
B	2-SLC	Failed∗	Not Supported
C	1-MLC	Failed†	Not Supported
D	3-MLC	Failed†	Not Supported
E	4-MLC	Encrypted‡	Encrypted‡
F	5-MLC	Success	Success
G	6-MLC	Success	Success
H	7-MLC	Success	Success
I	8-MLC	Success	Success
J⋆	9-TLC	Not Supported	Not Supported
K⋆	10-MLC	Not Supported	Not Supported
L⋆	11-MLC	Not Supported	Not Supported

Table 1. Sanitization results

Overwrite techniques: Traditional methods used for hard drives execute multiple overwrite operations on logical block addresses. These standards and tools repeatedly overwrite the drive with between 1 and 35 bit patterns. For example the DoD 5220.22-M method created by the Department of Defense uses a three-pass method. The first pass writes only zeroes, the second writes only ones and the final pass writes a random character to each logical sector. The results obtained were generally poor. Some methods were able to sanitize the disk effectively but only after several passes, which took a significant amount of time and the results proved to be unreliable at best. While some of the drives were properly sanitized others still contained remnant data after multiple overwrites.

Degaussing: Degaussing is an effective way for destroying hard drives by disrupting or eliminating the magnetic domains on the drive. Since traditional hard drives are magnetic storage drives, the degaussing technique proves to be very effective. However, SSDs use Flash memory, which is not magnetic in nature. The authors degaussed individual Flash memory chips written with fingerprint data instead of entire SSDs for the experiment and subsequent analysis revealed that data had remained intact.

In addition to the methods used for entire drive sanitization, the authors also tested for single file sanitization on SSDs. All the methods used for single file sanitization failed to completely remove the file and 4% to 75% of the file's contents remained on the SSD and were recoverable.

These results are significant in an era where Flash memory use in handheld devices and computers is gaining popularity. The authors demonstrate that Flash-based storages function very differently than traditional hard drives and file deletion is much more difficult on an SSD than on a hard drive. However, the authors have not analyzed recent developments in communication between the operating system and Flash storage such as the TRIM command which allows the OS to tag individual pages that are no longer being used and allow for more efficient erase operations. Furthermore, SSDs are used for internal storage in desktop and laptop computers and their Flash controllers are much more advanced than the ones used by smart phone or tablet Flash storage. Handheld device is rising in popularity and corporations are relying on these devices to securely handle sensitive information.

Chapter 3

Smart phone & MDM security

3.1 Introduction

In this chapter we detail the security measures each smart phone model implements along with a brief overview of additional security provided by MDM products for smart phone use in the enterprise. In the next chapter, we perform data recovery experiments on iPhone and Android devices to test if MDM remote wipe securely deletes enterprise data.

3.2 Smart phone security

3.2.1 iPhone security

The iPhone is Apple's smart phone designed for consumer use. Early models of the iPhone, mainly the original iPhone and iPhone 3G, had little, in terms of security and protection, for data at rest. Data was stored on the user partition unencrypted and could be read by jail breaking the device or by booting a custom firmware. The next iPhone model, the iPhone 3GS, featured hardware encryption. This device was shipped with iPhoneOS (iOS) 3.x, which made limited use of the hardware encryption module. The main goal of device encryption was the ability to do a quick device wipe by erasing the encryption key thus rendering the data inaccessible [24]. However, the encryption did not prevent a user from obtaining decrypted data from the device; decryption was completely transparent and the device would decrypt any data an application might try to read [47]. With the introduction of iOS 4, encryption was further improved through individual file encryption. If a disk image were to be obtained, the contents would be unreadable because each individual file would remain encrypted. In summary, the iOS Data Protection feature was designed to protect data "at rest" and to make offline attacks difficult. It achieves this design goal by tying encryption keys used for actual data encryption to the UID encryption key, which is a key embedded into the hardware and is unique to each device. This key cannot be extracted by regular software means.

The encryption key tying is not direct but achieved through a multi-level key hierarchy. Apple iPhones have a hardware-based AES engine built-in with a unique 256-bit AES device key embedded into the CPU (unreadable through software means). Each file on the device data partition is encrypted using a per-file encryption key that is unique to that file (the file key is generated upon creation of that file). The unique file key is then wrapped with a class key which is then stored in the file metadata. The class key is encrypted using the unique device key and stored in the device's physical storage (Flash storage). The file system metadata is stored, in encrypted form, using a key specific to file system metadata. The UID generates two keys: the 0x835 and 0x89B keys. On the Flash storage a specific area is set aside for key storage called the effaceable area (specially designed for secure deletion). Three keys are stored in there: the BAG1 key, which is used to encrypt the Keybag that stores the encryption keys used to encrypted sensitive data such as e-mail, the EMF! Key, which is the metadata key and finally the Dkey, which is the class #4 key. The EMF! Key is combined with the 0x835 key to create the EMF key which is used to encrypt file system metadata. The Dkey is combined with the 0x89B key to create the Class 4 wrapper key, or Dkey. Almost all files are considered Class 4 except sensitive ones such as e-mail and text messages, which are encrypted using other class keys (found in Keybag). The process is depicted in Figure 6.

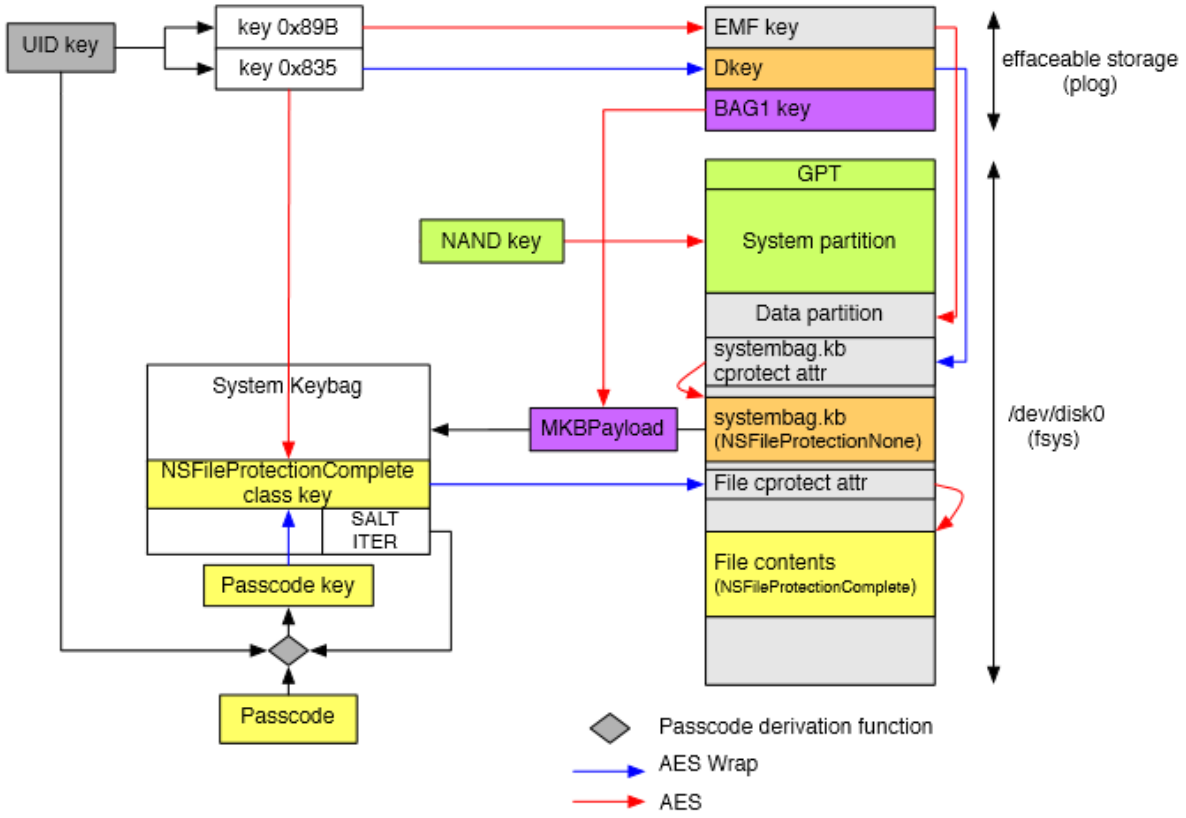

Figure 6. iPhone Data Protection [23]

From the figure above, it can be seen that in order to decrypt each and every file on the iPhone, one would need to acquire the Whitening key and the unlocked System Keybag. The Keybag is a collection of protection class keys.

Protection class keys are master encryption keys used to unlock files based on their access policy. Protection classes are the encryption mechanism used to enforce the access policies of files.

Some files are so important that the operating system should be able to decrypt them only when the device's user interface is unlocked. These files' encryption keys are wrapped with a class key that is available only after the user has entered his passcode. When the device locks again, the key is wiped from memory, making the files unavailable again.

Protection class master keys are stored in an escrow known as a Keybag. The Keybag contains the encrypted protection class master keys, as well as other keys to system files on the device. The system Keybag is encrypted using another encryption key named BAGI, which is also stored in the effaceable storage of the Flash storage. The effaceable storage is a special area in the flash storage that contains the EMF, BAG1 and DKEY keys and is designed to be securely wiped in a forensically sound manner when a user decides to restore the device to factory conditions, rendering the data inaccessible.

Whenever the user authenticates to meet a specific security protection policy, the encrypted keys in the Keybag can be decrypted. Each key is utilized to protect a certain type of file on the device. All protection class keys are tied to the hardware UID key (via key 0x835, which is derived directly from the UID, and some of them are additionally tied to the user-specified pass code. Knowing all protection class keys enables the decryption of all data on the device. The protection classes are shown in the following Figure [23].

ID	Name	Description
1	NSProtectionComplete	File is available only when device is unlocked
4	NSProtectionNone	File is available even when device is locked
6	kSecAttrAccessibleWhenUnlocked	Keychain item is available only when device is unlocked
7	kSecAttrAccessibleAfterFirstUnlock	Keychain item is available only after device has been unlocked
8	kSecAttrAccessibleAlways	Keychain item is available even when device is locked
9	kSecAttrAccessibleWhenUnlockedThisDeviceOnly	Same as kSecAttrAccessibleWhenUnlocked and keychain item is not included in backup
10	kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly	Same as kSecAttrAccessibleAfterFirstUnlock and keychain item is not included in backup
11	kSecAttrAccessibleAlwaysThisDeviceOnly	Same as kSecAttrAccessibleAlways and keychain item is not included in backup

Figure 7. List of protection classes (classes 2, 3, 5 are not used in iOS 4 but are implemented in iOS 5 and up) [23].

The iPhone uses HFSX as its file system, a version of HFS+ built specially for iOS devices. HFSX is a block file system designed to interact with traditional Hard Disk Drives (HDD). The file system communicates with the physical storage through a Flash Translation Layer (FTL).

3.2.2 Android security

The Android operating system is a modified version of the Linux kernel designed for mobile devices [60]. Several manufacturers, aiming to compete in the smart phone market, have adopted the Android OS contributing to the fragmentation of the Android smart phone devices. Android devices come in all shapes and sizes, with vastly different performance levels and screen sizes. Furthermore, there are many different versions of Android that are concurrently active at any one time, adding another level of fragmentation. What this means is that developing apps that work across the whole range of Android devices can be extremely challenging and time-consuming. Despite the problems, fragmentation also has a great number of benefits – for both developers and users. The availability of cheap Android phones (rarely running the most recent version) means that they have a much greater global reach than iOS, so app developers have a wider audience to build for. It may be tricky to do, but the potential reward definitely makes it worthwhile. For consumers, extreme fragmentation means that they can get exactly the phone they want – big or small, cheap or expensive, with any number of different feature combinations [4]. A complex chart of the Android device market can be seen in Figure 8, each shaded area represents the device brand's market share and each small block within each shaded area represents a device model of that brand.

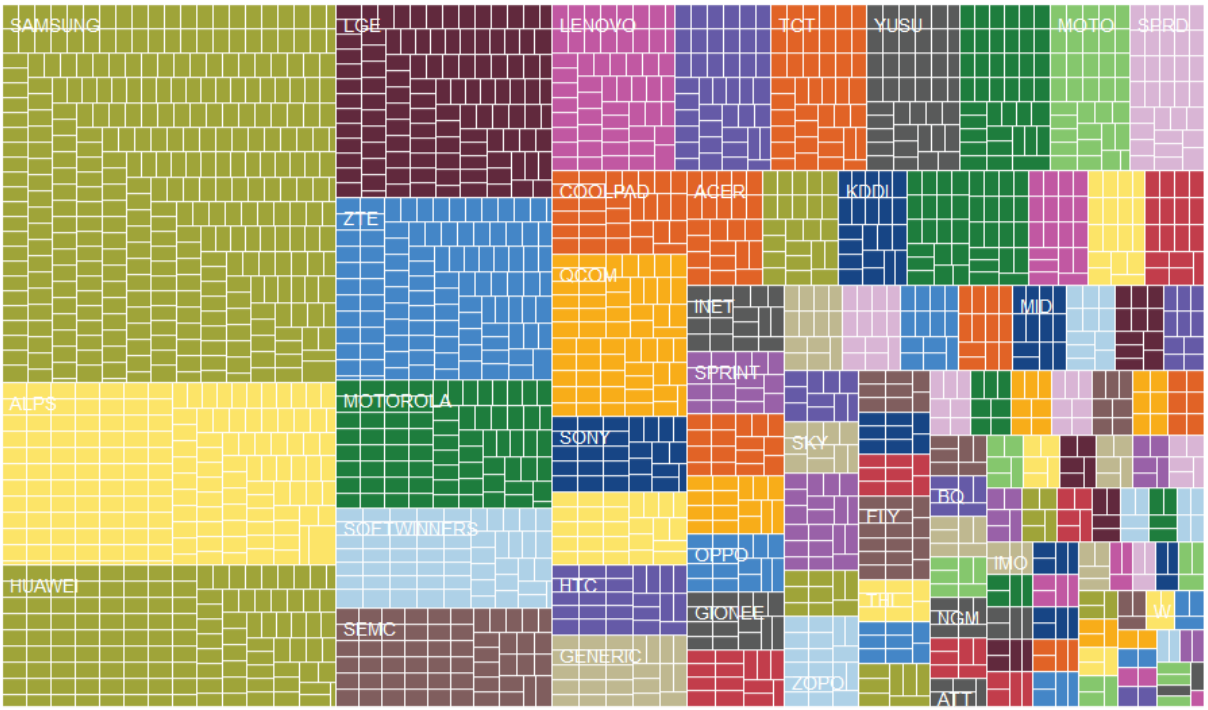

Figure 8. Android device fragmentation [4]

However, the fragmentation is not limited to device manufacturer; it also extends to the operating system version being used. The operating system of Android is also extremely fragmented. Contrary to iOS, several different versions of the Android OS can be in use at any given time. Figure 9 depicts a chart comparing OS fragmentation on both devices.

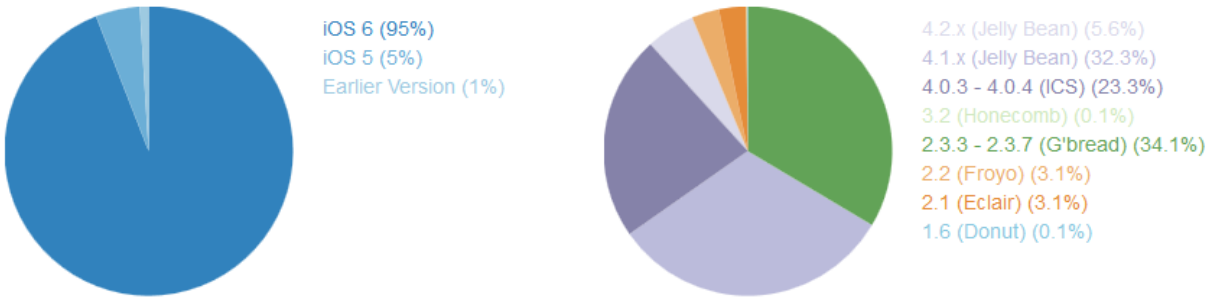

Figure 9. iOS and Android OS fragmentation comparison [4]

Android is mainly a consumer-oriented device and was not been built with enterprise use in mind. However, with the iPhone's escalating popularity with enterprises, Android devices have started to move towards that path as well.

Although late in entering the enterprise market, Android has employed certain security measures to comply with corporate policies. Up until Android OS 3.0, the use of encryption was non-existent. The only security included with earlier versions of Android was a PIN or Pass code or Pattern lock. Starting with version 3.0, the operating system was upgraded with the ability to encrypt the user file system [5].

The encryption mechanism is based on dm-crypt, a software encryption scheme that is a Linux kernel feature and works at the block device layer, e.g. an HDD that has been subdivided into logical partitions or blocks. Android's encryption only encrypts the /data partition, which holds sensitive user data.

The encryption process itself is handled by Android's volume daemon (vold). Once the daemon determines that there are no errors preventing the /data partition's encryption, it sets up a cryptographic mapping process that creates a virtual cryptographic block device that maps onto the real block device, but encrypts each sector as it is written, and decrypts each sector as it is read and finally, vold creates and writes out the crypto footer.

The crypto footer contains details on the type of encryption, and an encrypted copy of the master key to decrypt the file system. The master key is a 128 bit number created by reading from /dev/urandom. It is encrypted with a hash of the user password created with the PBKDF2 function from the SSL library. The footer also contains a random salt (also read from /dev/urandom) used to add entropy to the hash from PBKDF2, and prevent rainbow table attacks on the password. Also, the flag CRYPT_ENCRYPTION_IN_PROGRESS is set in the cryptographic footer to detect

failure to complete the encryption process. The crypto footer is kept in the last 16 Kilobytes of the partition, and the /data file system cannot extend into that part of the partition. When either encryption method has finished successfully, vold clears the flag CRYPT_ENCRYPTION_IN_PROGRESS in the footer, and reboots the system [27].

Additionally, Android devices employ various lock screen measures, such as PIN code lock, Passcode lock, Pattern lock and Facial recognition.

3.2.3 BlackBerry security

The BlackBerry is a line of smart phones and services designed and marketed by BlackBerry. The first BlackBerry device was an e-mail pager released in 1999, and the most recent devices, the BlackBerry Z10, Q10 and Q5 were released in 2013. The user interface varies by model; older devices had featured a physical keyboard, while newer ones make use of a touch screen keyboard [77].

What differentiates BlackBerry smart phones from the others is its default design for enterprise use. While other brands of smart phones were initially designed for consumer use and slowly transitioned into the enterprise market, BlackBerry's design was purposefully made for enterprise (mainly, its ease of integration into an organization's e-mail system through a software package called BlackBerry Enterprise Server (BES) [73]). The primary BES feature is to relay email from a corporate mailbox to a Blackberry device through the use of push technology, because all new emails, contacts, and calendar entries are pushed out to the BlackBerry device immediately, as opposed to the user manually synchronizing the data [30]. The alternative to using BlackBerry Enterprise Server is to use the BlackBerry Internet Service (BIS). BlackBerry Internet Service was developed primarily for the average consumer rather than for the business consumer. The service allows users to access POP3, IMAP, and Outlook Web App (not via Exchange ActiveSync) email accounts without connecting through a BlackBerry Enterprise Server (BES). BlackBerry Internet Service allows up to 10 email

accounts to be accessed, including proprietary as well as public email accounts (such as Gmail, Hotmail, Yahoo and AOL). BlackBerry Internet Service also supports the push capabilities of various other BlackBerry Applications [56].

Blackberry devices are considered primarily for enterprise use and therefore are designed with security in mind. BlackBerry provides network security by using AES to encrypt all communication between the device and the BES. In addition, the device provides additional security by enabling content protection or encryption of data at rest. If content protection is turned on, user data that the BlackBerry device stores is always protected with a 256-bit AES encryption algorithm. Content protection of user data is designed to perform the following:

- Use a 256-bit AES content protection key to encrypt stored data when the BlackBerry device is locked.
- Use an Elliptic-curve cryptography (ECC) public key to encrypt data that the BlackBerry device receives when it is locked.

The user can also determine the "strength" of content protection by choosing between three settings: Strong, Stronger and Strongest where each setting forces the device password to be a certain length (the "Strongest" setting forces a password of 21 characters minimum) [32].

However, the data communicated between the BlackBerry and the BIS is not encrypted and is sent in plain text. For BIS users, only the mobile carrier's standard 3G/2G protection applies. RIM states that: "Email messages sent between the BlackBerry Internet Service and the BlackBerry Internet Service subscriber's BlackBerry smart phone are not encrypted. When transmitted over the wireless network, the email messages are subject to the existing or available network security model(s)." [45]

3.2.4 Windows Phone security

Windows Phone is Microsoft's proprietary smart phone brand. It is primarily aimed at consumer rather than enterprise use. The latest major release is Windows Phone 8, launched in late 2012. The brands (HTC, Huawei, Nokia and Samsung) are currently producing Windows Phone 8 devices [66]. While a late entry into the smart phone market, Windows Phone 8 has managed to grab a 3% market share, surpassing BlackBerry and making it the third most popular smart phone [64].

In terms of security, the Windows Phone provides users with a device lock mechanism and software encryption. The locking mechanism is similar to other platforms such that it allows users to choose between a numeric PIN or an alphanumeric Pass code. Windows Phone uses BitLocker Drive Encryption technology, which is Microsoft's proprietary encryption software. As with other mobile encryption schemes, its objective is to protect sensitive data by encrypting the device's internal storage. A Trusted Platform Module (TPM) protects the encryption key. The TPM is a secure microprocessor embedded in the CPU, which can store cryptographic keys and allows only trusted components of the operating system to access it [38]. According to Microsoft, with both PIN-lock and BitLocker enabled, the combination of data encryption and device lock would make it very difficult for an attacker to recover sensitive information from a device.

3.3 MDM Security

MDM Software provides additional security for consumer-oriented smart phones to enable use in the enterprise. Their main objective is to segregate and protect sensitive corporate information from attackers and provide a quick and simple solution to remove said data when the user leaves the company or the device gets stolen or lost. The methods employed by MDM are explained in the following paragraphs.

3.3.1 Containerization

MDM Containers are considered sandbox applications that attempt to limit corporate data within its boundaries. These applications provide the employee with proprietary implementations of web-browsers and email clients [67]. Depending on the software provider, some containers require a separate PIN or Pass code to access the application, while others rely on the mobile operating system's native encryption scheme to protect corporate data. In summary, the main objective of a container is to prevent sensitive data from leaking into the device's personal user space.

3.3.2 Remote Wipe

MDM Software provides enterprises with a remote wipe feature for lost or stolen devices. Depending on corporate policy this function may perform either a full remote or selective wipe. Both functions are aimed at securely removing sensitive data from the device, preventing any sort of recovery [18]. The difference is the amount of data that is deleted. In a full remote wipe, the entire device's data, personal and corporate, is deleted. Essentially, this function factory resets the employee's device.

A selective wipe, on the other hand, removes only corporate data from the device. MDM providers that employ containerized solutions often use this version of remote wipe, which removes the container application and all of the data residing in it.

Both functions remove sensitive data from the device; however, after a full remote wipe, if the employee manages to retrieve the lost or stolen device, all of his personal data will have disappeared as well. Following a selective wipe, the employee can simply re-enroll his device with the MDM server restoring corporate data and email on the device.

3.4 Smartphone Storage

Modern day smart phones use Flash as their internal storage, which differs significantly from traditional magnetic Hard-disk drives (HDD). Although the operational

differences between both storage mediums are many, we primarily focus our attention on the difference in file deletion.

We present, in the next chapter, experiments performed on both iPhone and Android devices demonstrating that simply issuing a remote wipe to the device does not, physically, destroy all data. However, we feel it necessary to write a brief introduction on Flash technology in order to solidify the main differences it has from an HDD.

3.4.1 Flash

Flash storage is currently dominating the smart phone market. The fast read/write speeds and the compactness of the hardware make it an attractive option for smart phone manufacturers.

Flash is a block structured storage system that is increasingly being used to supplement or replace traditional magnetic storage in many applications. Flash storage media comes with more durability, faster read/write speeds, and less power consumption. It is also lighter in weight than traditional hard disks. These advantages are the main reason such a shift in storage technology is occurring on portable media.

The main component of Flash media is Flash memory. Flash memory devices are non-volatile, which means the memory retains the data after the power has been removed. The two most common Flash technologies used for building memory devices are NOR and NAND. NOR-based Flash memory is the older technology, which supports high read performance at a smaller capacity range. On the other hand, NAND-based Flash memory supports higher capacities with significantly higher read and write performance [6].

NAND Flash is organized as an array of *erasable* blocks. These memories are accessed much like block devices, such as hard disks or memory cards. Each block consists of a number of pages. The pages are typically 512, 2,048 or 4,096 bytes in size. Block sizes

can range from 16KB to 512KB. Reading and programming is performed on a page basis, however erasure can only be performed on a block basis [78].

Flash cells can be Single Level Cells (SLC) that hold one bit per cell or Multi Level Cells (MLC) that can store more than one bit per cell. Typically, MLC may represent four different states [8]. The tradeoff in more capacity of the MLC flash is increased read/write latency and a shorter lifetime [9]. The write latency is due to a more sophisticated programming technique to store the precise charge needed to achieve the threshold voltage distribution. Similarly, the read operation is longer because it takes longer to distinguish between the four possible charges stored in the cell. The lifetime of MLC devices decreases due to the inability to distinguish between the four different states, compared to only two in SLC [10].

One important challenge with Flash memory is that a page can be programmed only a limited number of times; for consumer-grade Flash storage this number is usually within the range of 10,000 to 100,000 erase cycles. This limit is also referred to as write endurance. SLC flash typically has a higher write endurance than MLC flash.

With flash media, not all blocks get used at the same rate; some blocks may get written to more frequently than others. This may cause certain blocks to wear-out sooner than others and become unusable. In response to this limitation, manufacturers may implement an algorithm known as wear leveling. This algorithm makes sure that write operations are spread out evenly across all blocks ensuring that no block gets written to more frequently than others. The wear-leveling algorithm is part of the Flash Translation Layer (FTL), which is an additional software layer between the file system and the NAND Flash memory. The FTL allows operating systems to read and write to NAND Flash memory devices in the same way as disk drives and provides the

translation from virtual to physical addresses. Wear leveling can also be implemented on the file system directly (Figure 10).

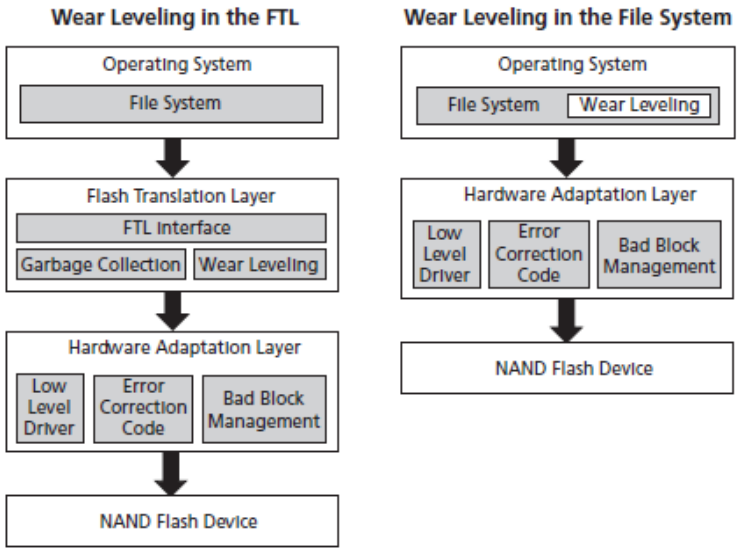

Figure 10. Different modes of wear-leveling implementations

The FTL allows any standard file system to utilize a flash memory device by emulating a block device. The FTL stands between the host operating system and the flash device and translates the standard block commands from the host. It will present memory from the flash in sectors and blocks, much like the standards used by a hard disk drive interacting with a FAT file system, while hiding the intricacies of the flash device, such as block erasing and wear leveling. It does this by mapping the block and sector addresses of the standard file system to physical addresses. So, as data is physically moved to different locations in order to implement wear leveling, the FTL presents the data to the host operating system as if the data is written to a static location. The FTL may be implemented on a hardware controller that exists on the chip itself working as the intermediary between a host OS (utilizing a standard file system such as FAT) and the hardware adaptation layer, or it may exist as part of the operating system.

The second approach is to design or use a file system designed for Flash memory that implements a wear-leveling algorithm. Such file systems include the open source

YAFFS (Yet Another Flash File System) and JFFS (Journaling Flash File System) [39, 25]. Instead of using traditional block device file systems, these systems create their own data structures on the flash device. These data structures are loaded into the host computer's RAM when the flash file system is mounted. They are then used by the file system to implement wear leveling, bad block management, and block reclamation during block erasure. Typically, the structure will utilize status flags that indicate the state of the physical pages and blocks, and erase counts to even wear. A key distinction between FTLs and Flash file systems is that the Flash file system gives the OS direct access to the physical Flash storage whereas through an FTL the OS communicates with the physical storage through the Flash controller. Flash file systems are typically used in embedded applications where the flash storage is not removable, such as the YAFFS implementation in the Android operating system for mobile phones [11]. Devices using an FTL implement block-level file systems such as ext or FAT primarily designed for HDDs. OS commands to the file system get remapped to the physical storage via the FTL (Figure 11).

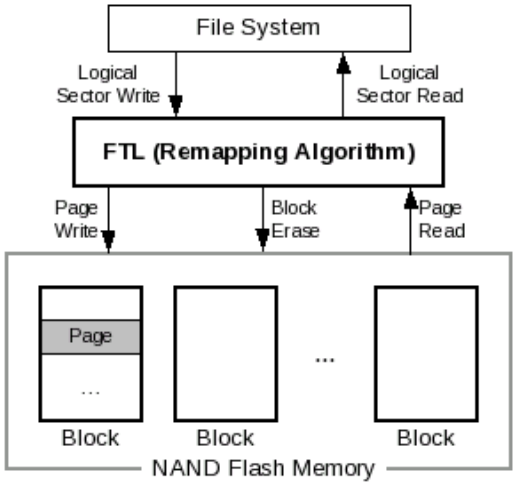

Figure 11. FTL Remapping of file system writes/reads

NAND Flash can be implemented either in raw or managed form. In raw form, the entire management of the NAND Flash memory is left to the host processor; usually a

file system developed specifically for Flash is used in these cases, such as the previously mentioned YAFFS2 or JFFS2 file systems.

Managed NAND is a form of Flash memory that comes coupled with a controller that manages data stored on the NAND packaged into a single chip, more commonly known as an embedded multi-media card (eMMC) device comprised of both Flash memory and the Flash controller. eMMC devices help simplify mass storage designs for tablets and smart phones.

Based on the latest JEDEC eMMC specification, eMMC memory is produced in extremely compact sizes - freeing up space for other components [28]. The difference between NAND Flash and Managed NAND Flash can be seen in Figure 12.

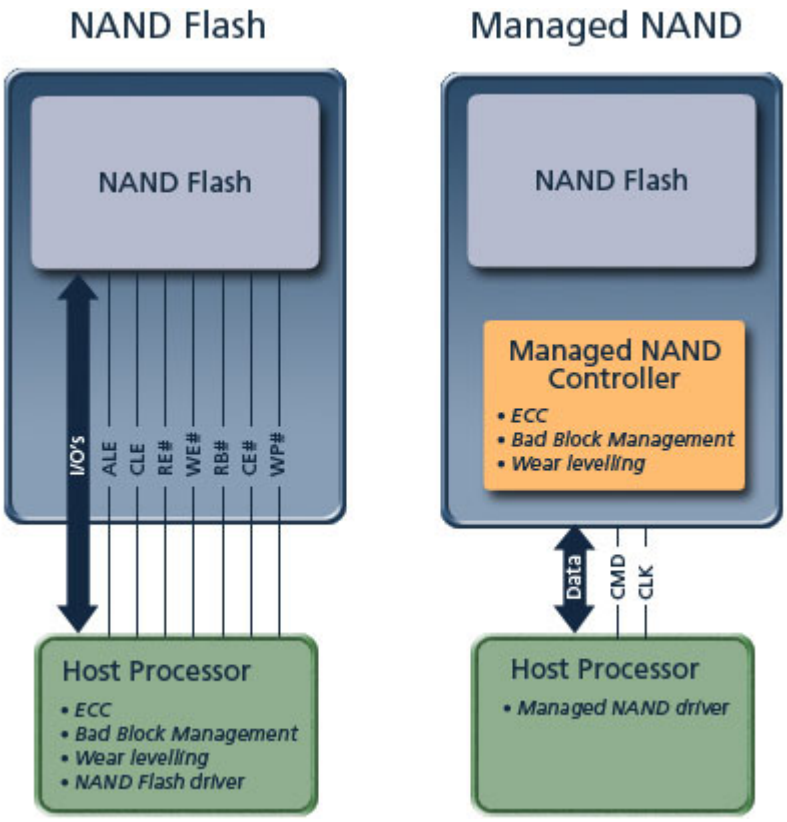

Figure 12. NAND Flash vs. Managed NAND

A flash page may be individually programmed and read (bits changed from one to zero), but an individual page cannot be erased (set back to one). For example, if we were to program a byte on a page from 11111111 to 11111100, further allowed changes could be 11101000 or 10011100, but 11111110 would not be possible since the second right-most bit is changed from a zero back to a one. In order for the last change to happen, the entire block would need to be erased, which would effectively reset all the pages within that block back to 11111111.

If the data in some of the pages of the block are no longer needed (referred to as stale pages), then the pages with valid data in that block are read and re-written into another previously erased empty block [76]. The stale pages are then erased through a block-erase operation making them writeable (programmable). This is a process called garbage collection (GC) [31].

However, a limitation of the garbage collection process is that it cannot identify which data is invalid since it is incapable of directly communicating with the file system. Deleted pages will be considered invalid on the OS-level, but will keep getting re-written to new blocks by the garbage collection process since the physical medium does not know these pages are marked as invalid. Finally, when the OS writes a new file to the deleted file's location, the pages of the new file are written elsewhere and the pages of the deleted file are marked for GC. Figure 13 illustrates this concept.

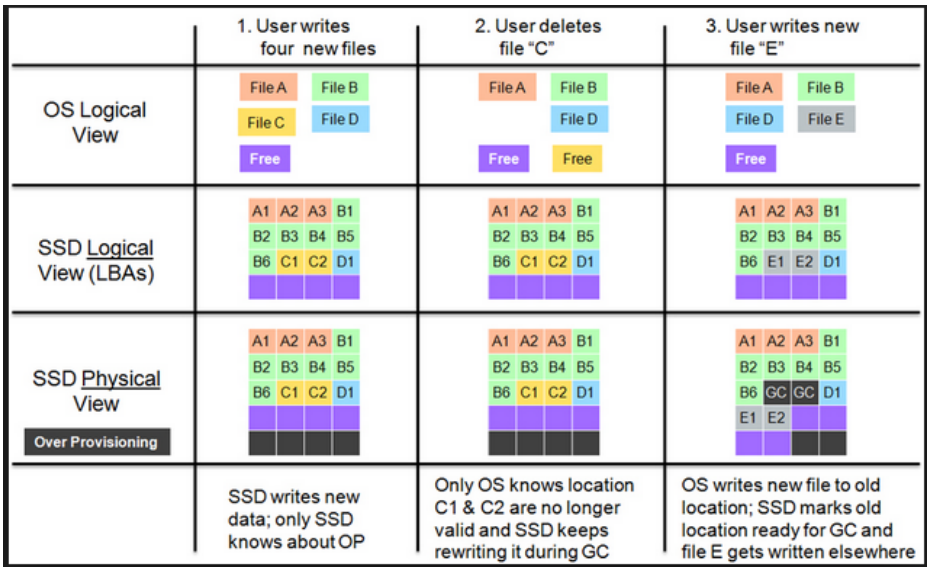

Figure 13. Difference between OS-level and Physical-level

In order to deal with this limitation, a command named TRIM was developed. This command allows the OS to communicate with the Flash storage and note the pages that are invalid ensuring that when a block is set for erasure, the invalid pages are not copied to a new block. This command's success varies depending on the controller managing the flash storage and its firmware. As the implementation of the firmware is closed source, the effectiveness of the TRIM command would be correlated with how well the firmware and the controller are implemented. Figure 14 illustrates what happens when the OS issues a TRIM command.

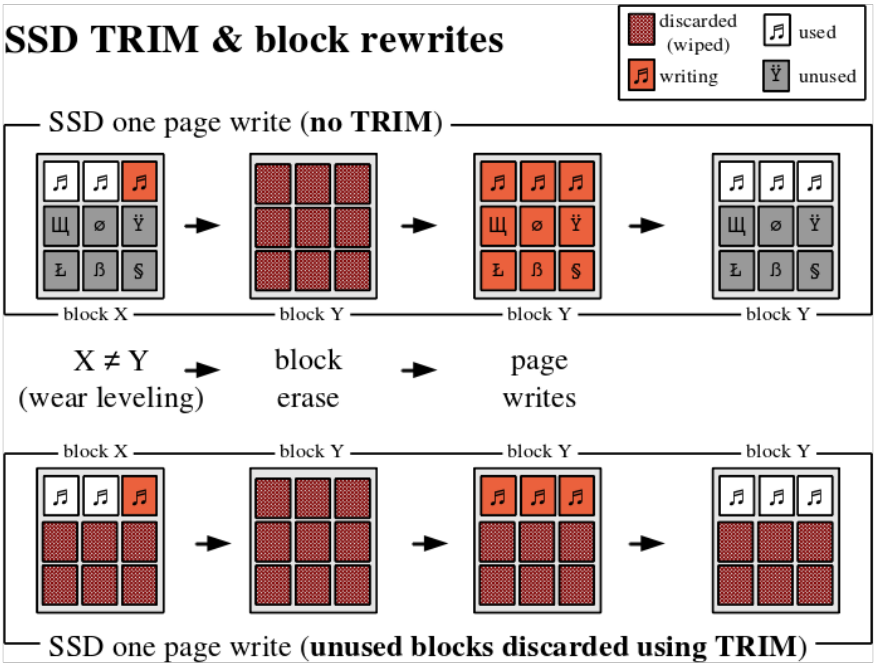

Figure 14. Comparing block erasure with TRIM disabled and TRIM enabled

3.5 Conclusion

In conclusion, deleting data from Flash memory is much more difficult than deleting from an HDD. The block-level erasure structure and wear-leveling algorithm implemented by Flash memory allows for data to persist on the physical Flash medium long after it has been marked for deletion by the OS.

Chapter 4

Experiments and Results

4.1 Introduction

The goal of the research in this chapter is to determine the effectiveness of data recovery from mobile devices following a remote wipe. In our experiments, the tested devices were factory-reset then loaded with applications, e-mails and an MDM client with sensitive data. They were then screen-locked and subjected to testing. We aimed at replicating a device state as similar as one would be in a corporate BYOD environment. Our testing is aimed at recovering data in two outcomes: when a remote wipe function is circumvented but the device remains in a locked state, and when a remote wipe function has been successful in deleting the data on the device.

Our experiments, due to budgetary limitations, were only done on an Acer Iconia A100 Android tablet and an iPhone 4 running iOS 4.3. While we did not have access to advanced technologies in order to access the Flash chips of the devices directly, we made use of open source forensics tools and existing exploits to demonstrate that data can be recovered from a compromised device.

We considered the following scenarios for both devices:

- A screen-locked Android device that is compromised and remote wipe has been circumvented.
- A screen-locked Android device that is compromised and remote wipe has been successfully performed.
- A screen-locked iPhone 4 device that is compromised and remote wipe has been circumvented.
- A screen-locked iPhone 4 device that is compromised and remote wipe has been successfully performed.

Prior to the discussion of our results, we find it useful to mention the methods used to circumvent the remote wipe command as the same methods were used for both devices.

4.2 Remote wipe circumvention methods

The MDM software sends the remote wipe command to the device from a central administration server, however there is no function on the client module that sends a success message upon execution of the wipe.

In order for this command to be received, the device needs to be connected to a network. A wireless (Wi-Fi) or 3G connection would work in this case. We present three methods used to prevent device network communication and potentially circumvent a remote wipe function:

1) SIM Card. A smart phone device that isn't already paired with a Wi-Fi network connection can communicate via 3G. By removing the SIM Card, the smart phone is no longer paired with a service provider and cannot connect to a 3G network, effectively rendering the device unable to communicate with the outside network. Although this method is practical in nature, it does not prevent the device from connecting to a public Wi-Fi network at a later time and resuming the remote wipe process.

2) Putting the device in 'Airplane Mode'. Putting the smart phone in Airplane Mode suspends all device communication and signal transmission functions. As the name implies, it allows the user to use the device while on board an aircraft and not interfere with the aviation equipment, mainly the ability to communicate with the air control center. While this method does prevent the device from transmitting or receiving any data, it is only accessible once the correct lock-screen password has been entered.

3) Faraday bag. Invented by Michael Faraday in 1836, it is an enclosure that blocks external static or non-static electric fields. It acts like an isolation cage and blocks the radio waves from reaching the device keeping the device's integrity intact. The Faraday

bag is equipped with an extension cable that allows the device to remain in the bag and connect to an outside source (Figure 15). This method does not require any previous knowledge about the device and would allow an attacker to operate on it in a communication free environment.

Figure 15. Device connecting to computer from a Faraday Bag

All three methods aim to accomplish the same goal: to prevent the device from communicating with any outside network. However, the first two methods do not guarantee complete device isolation. If the device's communication, from the time it has been compromised, is not quickly disabled, the remote wipe function will execute successfully. However, we demonstrate that even with a successful remote wipe, permanent data removal is not guaranteed.

4.3 Experiments

4.3.1 Android experiment – Remote Wipe Unsuccessful

The first experiment involves testing a screen-locked Android device in which the user has no root privileges and remote wipe function has been circumvented. As there are multiple screen lock options, we tested them separately by attempting to bypass it and gain access to the data. Each experiment is performed by resetting the Android back to its initial state (i.e. Screen-lock and removing root privileges).

The screen lock options are: Pattern-lock, PIN lock and Pass code lock.

- Pattern lock: A mix of gestures done on the phone by connecting points on a matrix in order to unlock the screen (Figure 16).

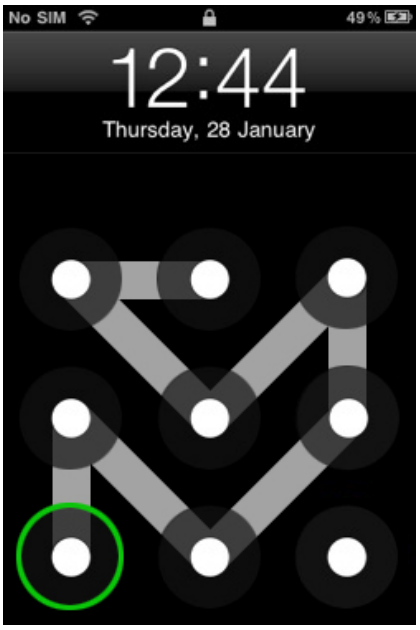

Figure 16. Android Pattern lock

- PIN Code lock: A number composed of 4 or 5 digits.
- Pass code lock: An alphanumeric password of a certain length.

Testing the pattern lock, we were able to bypass it using the Smudge Attack [41]. Under specific lighting, the user's fingers leave an oily trail that discerns the pattern used to unlock the screen. By following the trail, an attacker may unlock the device using trial and error. Before attempting to unlock the device, the attacker would need to map out all possible pattern combinations that can be formed from the recovered trail. (Figure 17).

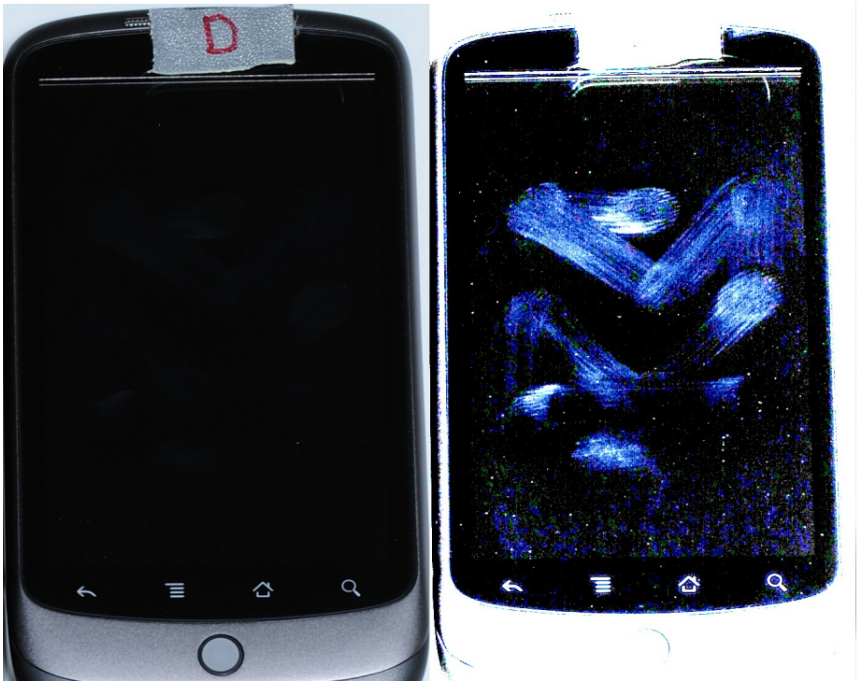

Figure 17. Smudge attack to reveal the Pattern lock

However, research shows that devices used in enterprise are more likely to use the PIN or Pass code lock options for stronger security.

For the PIN and Pass code options, a manual brute force option would take a significant amount of time (a 4-digit PIN code leads to 10,000 possibilities) so we bypassed the lock itself through the use of Android's debugging bridge (ADB) [2].

ADB is command-line utility that is provided with the Google Android Software Development Kit; it allows developers to connect to an Android device through a USB connection and issue shell commands. By default it is disabled, but it can be enabled for development purposes or advanced customization of the device. The option to enable ADB is located under Settings -> Developer Options.

Although ADB's main objective is to allow developers to install and modify their applications, it can also be leveraged to access restricted areas on the device.

Android stores a salted SHA1-Hashsum and MD5-Hashsum of the PIN or Pass code in a special file called password.key located in /data/system/ and the salt is stored in an

SQLite database called settings.db located in
/data/data/com/android.providers.settings/databases.
However, files located in the system folder are only accessible to the root user. By default, the root user is disabled on factory-issued Android smart phones preventing users from attaining privileged control on the device. The process of enabling root access is called 'rooting' the smart phone, mainly used for circumventing limitations imposed on the device by certain carriers and hardware manufacturers and performing administrator-level operations that would be otherwise unavailable to a normal Android user. An entire Internet community is dedicated to pursuing rooting efforts for all models and brands of Android smart phones [37]. By using one of the many methods available, we rooted the device in order to read the password.key and settings.db files.
Through ADB, we were able to copy the files and run a brute forcing process to retrieve the screen-lock PIN or Password with the aid of Hashcat [20], an open-source password-recovery tool that supports a list of hashing algorithms.
Organizations are aware of the dangers of leaving ADB enabled, so they employ policies in the MDM software that administratively disable the ADB option from turning on. This, in turn, prevents any access to the device from a USB connection.
An Android device that is screen-locked with ADB disabled is assumed to be more secure against brute force attacks when compromised.
In our next test, we tried to access an un-rooted device with ADB disabled by leveraging Android's device initialization protocol called Fastboot.
Fastboot is a special diagnostic protocol that the device can boot into. While in Fastboot, the smart phone's flash file system or operating system can be modified. Such modification requires running unsigned code on the device, which, by default, is disabled by a locked boot loader. A boot loader is the first bit of code that is executed before the operating system starts. A locked boot loader only allows running software

signed by the device vendor and prevents installation of custom software that it does not recognize.

As with its rooting efforts, the same community is also dedicated to providing several methods for unlocking the boot loader on an Android device. By unlocking the boot loader a user is able to install custom software (i.e. modified operating system or recovery) on to the device.

Through Fastboot we were able to unlock the boot loader of the device; for the Acer Iconia A100, we had to boot into Fastboot by holding down the Volume Up and Power button at the same time and once the device was powered on we flipped the lock switch on and off to enter Fastboot mode. Through Fastboot, we unlocked the bootloader by using a script and then installed a custom recovery named TWRP [33], a custom recovery for Android that allows advanced recovery, restoration and installation options that aren't available by default. We then booted the device into recovery by holding the Power button and Volume Down buttons and connected to it through USB. The TWRP recovery allowed us to connect to the device via ADB, and from there we were able to reproduce the previous methods to bypass the screen-lock and access the device.

As an added security layer Android devices can also be encrypted. The encryption process was described in Chapter 3. It can be seen from the source-code for Android's encryption scheme that the master-key is stored, in encrypted form, in the last 16 Kilobytes of the /data partition [34]. On a rooted Android device, device partitions can be copied locally for further analysis. For this method we used the 'dd' command, a Linux command that is specifically designed to copy files bit-by-bit. The /data partition was located by using the 'mount' command and found in /dev/block/mmcblk0p8. We started copying the partition with dd and transferred it locally by using netcat, a networking service for reading from network connections.

In order to break the encryption we needed to locate the magic number in the footer of the partition that is put to identify the footer structure (Figure 18).

```
// crypto footer structure format
struct crypt_mnt_ftr {
  __le32 magic;   /* crypto footer magic number */
  __le16 major_version;
  __le16 minor_version;
  __le32 ftr_size;  /* footer size in bytes (not including key) */
  __le32 flags;   /* flags for Android OS */
  __le32 keysize; /* keylength in bytes */
  __le32 spare1;  /* ignored */
  __le64 fs_size; /* Size of the encrypted fs, in 512 byte sectors */
  __le32 failed_decrypt_count;
  unsigned char crypto_type_name[64]; /* cipher, mode, IV method, etc. */
};
```

Figure 18. Android encrypted footer structure

The magic number is programmed as the hexadecimal value 0xD0B5B1C4, which was found in our hex editor stored in Little-Endian; the most significant byte is stored last (Figure 19).

```
3C4608FB 27DBBCA6  D¸ ∫Ù²D^ 0dòi-g¿ç:ŒÙµ®öLÜM ÿ ¢ø¶ §2È[N≥ctAK ^ág‹<F °'€º¶
C4B1B5D0 01000000  &íÙ\ nÅ˜  bÒ/gòÆRΔfl { /û…ßj?¯2Î AØπmà ˜Ô√∏æ6ãô‡uƒ±µ-
00000000 00000000  h              ‡ ±          aes-cbc-essiv:sha256
C2DE7567 2BA7FE45                                      Mê,ù%˜  ¬fiug+ß¸E
00000000 00000000                                   ûRfi&ÜV@§¶†∂Ê´Üfl¢
```

Figure 19. Encryption Magic Number – Highlighted in gray

After identifying the magic number, the entire contents of the encryption footer can be identified as seen in Figure 20.

```
Magic          : 0xD0B5B1C4
Major Version  : 1
Minor Version  : 0
Footer Size    : 104 bytes
Flags          : 0x00000000
Key Size       : 128 bits
Failed Decrypts: 0
Crypto Type    : aes-cbc-essiv:sha256
Encrypted Key  : 0x82AF933B1AF0968D835239CE69526C60
Salt           : 0x31D720E6F7F78A23D793E125378E5F49
```

Figure 20. Encryption footer contents

By locating this number we can navigate within the footer to locate the salt and encrypted master key. By running a password guess process through a PBKDF2 with the salt, we were able to retrieve the key used to encrypt the master key. By decrypting

the master key, we were finally able to decrypt the user data located in the /data partition we copied and access the user data. The decryption process itself depends on the strength of the password used. For a 4-digit PIN, our process took approximately 5 minutes while a 5-digit PIN took approximately 50 minutes. A strong alphanumeric password with special characters and a length longer than 12 would make it very difficult to crack, however it is important to note that GPU cracking methods are becoming more popular and can process a greater number of password combinations per second than a CPU [52].

In our first scenario, based on our tests, if an Android device's remote wipe command is circumvented, an attacker can gain access to device data even if it is screen-locked and encrypted.

4.3.2 Android experiment – Remote Wipe Successful

In the second scenario of our experiments, we tested an Android device that has been remote wiped successfully, meaning the command from the MDM server was sent and successfully executed on the device before the attacker had time to circumvent it. In this experiment, we demonstrate that a remote wipe by itself is not sufficient to permanently remove data from the Android device and through the use of open-source tools data can be retrieved by performing a series of operations on the device.

As mentioned in the previous chapter, the Android smart phone uses Flash as its storage technology and a block-level file system as its file system that communicates with the physical storage medium through a Flash Translation Layer. When a file is deleted, there are two things that happen: first, the file system removes its reference to the file indicating to the operating system that the file no longer exists. Second, the file is marked for deletion on the physical storage. However, files marked for deletion on Flash do not get deleted immediately because of Flash's block-level erase.

To test the security of the MDM software's remote wipe function, we filled the Android device with a set amount of data.

We set up two e-mail accounts, a Gmail and a separate account with the MDM application's e-mail account. We sent and received messages from both accounts. We setup Facebook, Twitter and Dropbox accounts. We sent and received messages from the Facebook and Twitter accounts and uploaded files on to the Dropbox application. We uploaded one picture and one document to the MDM container's secure document storage application named Backpack. We also took three photos with the device's camera.

In this scenario we assume that the employee has the ability to e-mail company information to himself through his personal e-mail account, or upload files on to Cloud applications such as Dropbox. Communication through social media sites such as Facebook and Twitter are also very common in corporate environments.

We then issued a remote wipe command from the MDM administration interface. The device was reset to factory conditions following the successful remote wipe. Browsing the device file system revealed no trace of any accounts, or applications that were installed before the remote wipe occurred.

In order to test the remote wipe's success, we first located the /data partition by issuing a mount command on the device. Its location was revealed under /dev/block/mmcblk0p8. We then acquired the image of the /data partition, through the use of the dd command to a file named mmcblk0p8_afterwipe.dd for offline analysis.

Once acquired, we used an open-source forensics tool called The Sleuth Kit (TSK) [36] to test data recovery. The Sleuth Kit is an open source digital investigation tool (a.k.a. digital forensic tool) that runs on Windows, Linux, OS X, and Unix systems. It can be used to analyze disk images and perform in-depth analysis of file systems (such as NTFS, FAT, HFS+, Ext4, and UFS) and several volume system types.

TSK analyzes the image by extracting data in the image's unallocated space. As the dd command images the partition bit-by-bit, we acquire an exact copy of the partition as it

resides on the device. The unallocated space of an image, or free space for the OS, is the space on the file system that can be written to. For example, the /data partition we acquired was 5 Gigabytes in size with 1 Gigabyte filled with the device operating system and applications. This leaves 4 Gigabytes of unallocated space after the remote wipe function has executed.

In a secure deletion, the unallocated space will be filled with all zeroes leaving no data to be recovered, but in our analysis, after uploading the image into TSK, the unallocated space was found to contain data.

With TSK we were able to convert the unallocated space to another image file and analyze its contents. We loaded the new image file and performed a keyword search on it searching for all occurrences of ASCII and Unicode strings in the image file and extracting them into an output file, which was then processed.

Two files were produced: mmcblk0p8_afterwipe.dd-0-0-ext.unalloc-blkls.asc and mmcblk0p8_afterwipe.dd-0-0-ext.unalloc-blkls.uni, the ASCII and Unicode extractions of the unallocated space respectively.

Upon inspection of both files, we were able to retrieve the data prior to the remote wipe. The following images represent the data in its raw form found after analyzing the unallocated space of the /data partition image.

```
97130591  GYfkQoAUX98AAAAAAAACXgUzzwEPSG1V1----------cHME6J6NnOYk1m07YG8SV/photo 03.06.2013 15 15 32.jpgf
97130696  GYfkQoAUX98AAAAAAAACYAUzzwEPSG1V1----------cHME6J6NnOYk1m0CXHrjc/photo 03.06.2013 15 14 58.jpgu
97130801  GYfkQoAUX98AAAAAAAACTAV--0A6AAkaX---3gRgj1P-cHME6sdMsvXSpm833sfPs/camera uploads/2010-11-22 08.55.45.jpg
97130921  GYfkQoAUX98AAAAAAAAB3AV--0A6AAl_1---3gRgj5J-cHME6sdMsvXSpm82aEnfB/camera uploads/2010-11-22 08.55.46 hdr.jpg
97131045  GYfkQoAUX98AAAAAAAAByAV--0A6AAuMX---3gRgjdT-cHME6sdMsvXSpm87dZUVg/camera uploads/2010-11-22 08.55.55.jpg
97131165  GYfkQoAUX98AAAAAAAACNQV--0A6AAuMX---3gRgjdT-cHME6sdMsvXSpm8Dz-aJV/camera uploads/2010-11-22 08.55.55 hdr.jpg
97131289  GYfkQoAUX98AAAAAAAACIAV--0A6AHv_X---3gRhAiL-cHME6sdMsvXSpm804A2Ws/camera uploads/2010-11-22 09.03.35.jpg
97131409  GYfkQoAUX98AAAAAAAACDwV--0A6AI8FX---3gRhB_1-cHME6sdMsvXSpm8BayouR/camera uploads/2010-11-22 09.03.49.jpg
97131575  GYfkQoAUX98AAAAAAAAAIQUzzwEPSG1V1----------cHME60tzNMaVj_74Q3B1w/photos/sample album/boston city flow.jpgq
97131691  GYfkQoAUX98AAAAAAAAAFAUzzwEPSG1V1----------cHME60tzNMaVj_75xqbQF/photos/sample album/costa rican frog.jpgq
97131807  GYfkQoAUX98AAAAAAAAACAUzzwEPSG1V1----------cHME60tzNMaVj_76QQ6Mk/photos/sample album/pensive parakeet.jpgf
97131923  GYfkQoAUX98AAAAAAAACYgUzzwEPSG1V1----------cHME6J6NnOYk1m0-0T7Yg/photo 03.06.2013 15 16 32.jpgf
97132028  GYfkQoAUX98AAAAAAAACYQUzzwEPSG1V1----------cHME6J6NnOYk1m05EcvrJ/photo 03.06.2013 15 15 53.jpgf
97132133  GYfkQoAUX98AAAAAAAACYwUzzwEPSG1V1----------cHME6J6NnOYk1m05mA7t7/photo 03.06.2013 15 16 49.jpgf
97132238  GYfkQoAUX98AAAAAAAACZQUzzwEPSG1V1----------cHME6J6NnOYk1m06Osk03/photo 03.06.2013 15 17 29.jpgf
97132343  GYfkQoAUX98AAAAAAAACXwUzzwEPSG1V1----------cHME6J6NnOYk1m06Qqec-/photo 03.06.2013 15 16 13.jpgf
97132448  GYfkQoAUX98AAAAAAAACZAUzzwEPSG1V1----------cHME6J6NnOYk1m07RwVus/photo 03.06.2013 15 17 12.jpg
97133131 V--0ANeNO0X---3hXdWZ5-cHME6sdMsvXSpm86WBPkV
```

Dropbox files

```
4453112828 1438011848648562675Re: This is a testThis reply will be deleted as well, TEST123This reply will be deleted as well, TEST123
4453112955 On Sun, Jun 16, 2013 at 6:02 PM, Ali Uz <auz097ottawa@gmail.com> wrote:
4453113028 This email will be deleted. Do not mind."Ali Uz" <aliuz1@gmail.com>"Ali Uz" <auz097ottawa@gmail.com>
4453113856 <?xml version='1.0' encoding='utf-8' standalone='yes' ?>
```

Emails from Gmail and MDM client

```
97871999 com.*.sm.documentDocumentImportant document*, Inc.com.*.sm.
documentadb7666148324eec81e2ad296b57cec2e5e38453eb8a4808534bb343cb4bcom.meraki.sm.58209025183763772http://i.imgur.
com/F2Q55Jr.jpgimage/jpeg
97872216 jpegThu, 16 May 2013 17:22:13 GMT/mnt/sdcard/Android/data/com.*.
sm/files/backpack/adb7666148324eec81e2ad296b57cec2e5e38453eb8a4808534bb343cb4b/Important document.doc
```

MDM Client container documents

```
106304940 smtpsmtp.gmail.com
106304961 aliuz1$test26$0
106304991 pop3mailbox.uottawa.can
106305015 auz097test'o
106305939 auz097@uottawa.caauz097@uottawa.ca
```

MDM Email account passwords and Gmail account password

In addition, we implemented the same analysis on an encrypted Android device. We issued a remote wipe command on the encrypted device and then acquired an image of the erased /data partition. By using the same methods of analysis, we were able to identify the magic number of the Android encryption footer. Although we were not able to locate the key itself, as the data in the unallocated space is not in a specific order, we assume that the magic number's existence in the unallocated space is a sign that the encrypted master key is also present in the image.

In the first part of our experiments we have analyzed two cases of an Android device enrolled with an MDM, the first case where the device was compromised and the remote wipe was circumvented and the second where the device was successfully remote wiped. In both cases, we have demonstrated that security measures implemented by the native Android interface and the MDM software, such as remote wipe, were insufficient to prevent an attacker from retrieving sensitive data from the device.

4.3.3 iPhone experiment – Remote Wipe Unsuccessful

Our next two experiments were performed on an iPhone 4 running iOS 4.3. As mentioned in the previous chapter, the iPhone's security is built into its hardware, therefore getting access to the data on a compromised iPhone was much more difficult.

In the first scenario, we tested a screen-locked iPhone for which the remote wipe function was successfully circumvented. Similar to Android, the iPhone's screen lock options are:

- PIN code lock: a 4-digit number
- Pass code lock: an alphanumeric password of custom length

Furthermore, the iPhone can be set to auto-lock after a certain time of inactivity, which automatically enables the lock screen without user intervention. As opposed to the Android operating system, the iOS does not have a USB debugging alternative that enables users to connect via a USB cable. Furthermore, manually brute forcing was not an option as the iPhone can be set to wipe all data after 10 wrong attempts.

However, the option of jail breaking the device and trying to bypass the screen lock was still a viable option. It is important to note that there are two levels of Jailbreak for the iPhone: firmware-level and Bootrom-level.

A firmware level Jailbreak allows the user to acquire root, or administrative privileges, on the iPhone, allowing the user to install third-party applications and attaching functionality not supported by Apple. A firmware level Jailbreak can be patched with a future firmware update, as it exists only on the OS level of the device. On the other hand, a Bootrom jailbreak exploits the iPhone's boot loader and disables signature checks, which can be used to load custom firmware. This type of exploit cannot be fixed by a firmware update, as there is no code that checks the validity of the Bootrom, since it is the lowest level of the entire code signing chain of authentication. A hardware upgrade is necessary to patch this type of exploit.

In order to bypass a PIN or Password lock, a firmware-level Jailbreak would not be enough, as the device would remain locked after the Jailbreak process. However, with a Bootrom exploit, we could load custom firmware in memory that would attempt to brute force the PIN or Pass code during the boot phase. We patched an official iOS 4.3-

firmware file to include SSH access and PIN/Password brute forcing scripts in order to boot it as custom firmware. The SSH access is required to access the emulated terminal on the iPhone in order to input and execute commands. Upon boot, the custom firmware started and spawned a terminal shell for us; we were able to connect to the iPhone via SSH and execute the brute force scripts to reveal the PIN or Password used to unlock the device. However, this method is, currently, only available on the iPhone 4 models and below because Bootrom exploits have not yet been found on newer models, although there are a number of experienced members from the Jail breaking community working on releasing public exploits for newer models.

4.3.4 iPhone experiment – Remote Wipe Successful

Our second scenario involves executing a successful remote wipe on the iPhone and trying to acquire deleted data via forensics tools. In this experiment, we created the same data as we did for the Android scenario and executed a remote wipe from the MDM administration server. The remote wipe took only a couple of seconds and the iPhone was restored to factory conditions. Our forensic analysis did not recover any of the deleted data. The reason for this was because on the iPhone models 3GS and up, the remote wipe function calls an API on the device that erases the encryption keys used to encrypt the class keys. The encryption keys are stored in a special storage area called 'effaceable area' specifically designed by Apple and are securely wiped. Our forensic recovery revealed only an encrypted portion of the user data. Unless special hardware is used to recover the UID and GID keys embedded within the CPU of the device, the encrypted user partition is unrecoverable since the encryption keys are securely wiped. Brute forcing the encrypted partition was not an option as Apple uses AES-256 as their encryption standard, and no known weaknesses have been published regarding this algorithm as of yet.

4.4 Conclusion

We conclude this chapter by summarizing the security of both devices. In our Android scenario, we have presented several methods for bypassing the screen-lock in an unsuccessful remote wipe scenario, and we have also demonstrated that data is not physically deleted from the Flash storage following a remote wipe. Encrypting the Android device was not enough as the magic number was found during our data recovery experiment, meaning that the encryption key itself is not securely wiped from the device. The iPhone, on the other hand, required more effort to bypass the screen-lock through loading custom firmware and brute forcing the PIN or Password. Furthermore, the method presented only works on iPhone models 4 and below and is not valid on the iPhone 4S and up. In the event that the remote wipe is successful, file recovery was not possible in our experiment because of the secure deletion of encryption keys. At the moment, companies that employ BYOD policies should be aware that the remote wipe is not a fool proof solution for the Android as data, emails and passwords were recovered following a remote wipe procedure sent from the MDM administration server. The iPhone, on the other hand, uses hardware encryption to prevent any form of data recovery, which is beneficial for companies allowing the use of iPhones on corporate premises. Although, the potential recovery of the hardware keys from the device would allow an attacker to derive the remaining keys in order to decrypt the encrypted user data. While such a method is not publically known, researchers are continuously analyzing the iPhone from every angle hoping to find vulnerabilities and exploits.

In this chapter, we have demonstrated the use of open-source forensics tools and publically available exploits to thwart MDM device security and access sensitive corporate data. In the next chapter, we further analyze the topic of mobile forensics and how law enforcement and government agencies acquire data from seized devices that are screen-locked and/or encrypted.

Chapter 5

Mobile Forensics

5.1 Introduction

In this chapter, we overview the methodologies used for performing forensic examinations on mobile devices.

The use of mobile devices and smart phones in crime has substantially increased in recent years. Many criminals conduct their illegal business through their mobile phones by placing calls, text messaging, e-mailing and saving information in their device applications regarding their business. The prevalent use of mobile devices to commit crimes makes them incredibly valuable for law enforcement agencies as they contain a trove of valuable information needed to apprehend the criminal and his organization [59].

When a mobile device is seized, the information it contains is extracted and analyzed through the use of a method called mobile forensics. Mobile forensics is a form of digital forensics applied to mobile devices. The ultimate goal of mobile forensics is to seize and analyze a mobile device with the intent of extracting any form of incriminating evidence that can be used for the law enforcement agent's case. Although mobile forensics is a relatively new branch of digital forensics, criminals are already taking measures to prevent access to their devices through various methods like screen-locking the device and encryption. Several tools are available to law enforcement agencies to conduct their forensic analysis on seized devices.

Following device seizure, a series of steps must be performed in order to acquire the seized device's data in a forensically sound manner. Initially, the areas of evidence must be identified, followed by a methodical process for data extraction. In the next step, it needs to be further analyzed in order to produce a timeline of a suspect's actions and whereabouts.

5.1.1 Motivation

Our motivation in writing this chapter is to provide a more complete picture of the area of mobile forensics. In the previous chapter, we demonstrated that mobile forensics could be used maliciously to recover deleted enterprise data from a smart phone. However, in the hands of law enforcement, it can be used to recover evidence that would support a criminal case against a suspected individual. In both cases, the objective is to recover deleted data from mobile devices. A suspected criminal can send a remote wipe command to, or attempt to manually delete data from, the device in an attempt to dispose of evidence that may be used against him. Specially-developed forensics toolkits and software can aid law enforcement in recovering such evidence using advanced methods which are not found in their open-source counterparts.

While most of these methods are generally used to collect evidence against serious crimes such as murder, kidnapping, theft and so forth, they can also be applied to white-collar crimes involving corporate entities such as embezzlement, fraud, and internal trading. Corporations, especially those specializing in finance, are tied to their technology and are shifting to implementing company-wide BYOD policies. Inspecting suspected employee mobile devices could lead to the recovery of substantial evidence. The specialized law enforcement tools described in this chapter allow for the recovery of such data from devices to incriminate suspected individuals.

Most of the software and hardware that is described in the following sections are solely available for purchase by government or law enforcement agencies. These tools simplify the forensic process by automating device identification, preparation and data extraction, letting examiners shift their focus on the relevance of the extracted data and the construction of a timeline based on the extracted data (a chronological analysis of GPS locations, texts and calls placed).

Our demonstration used an open-source tool named The Sleuth Kit (TSK) to extract data from an Android device. It is freely available to anyone but requires a fair bit of technical knowledge in order to set up and use properly to extract relevant data. We chose the open-source route due to budgetary limitations and to the fact that most commercial products are only available to eligible customers (law enforcement). TSK is one of the most popular open-source forensics tools with a vast array of support for modern file systems. However, TSK was not designed to operate on a mobile device directly. We needed to perform a fair amount of work by physically extracting the Flash memory and analyzing it before being able to extract relevant data.

Open-source tools can generally perform at the same level as commercial products do but require manual effort. By using open-source tools we demonstrated successful results on a single device selected from a vast array of Android smart phones, and unsuccessful results on the iPhone. Commercial products provide better support, documentation and a multi-OS compatibility (e.g., Windows, Linux, and OSX). Furthermore, these tools are constantly upgraded with novel forensic methods and techniques that are usually not available in open-source unless a contributor volunteers to develop additional functionality; they also provide support for an extensive number of mobile devices. In addition, open-source tools underperform when the device being analyzed is an iPhone. Certain tools are able to extract data from the models up to the iPhone 4 running iOS version 5.x, but no publically known open-source tool has been able to extract any data from newer models and iOS versions. On the other hand, commercial products are methodically advancing in the area of iPhone forensics claiming to support up to iPhone 5 running iOS 6.x and yet their methods for data extraction are unknown.

In this chapter, our objective is to contrast the capability and success of commercial tools against those of open-source tools in order to provide the reader with a more comprehensive description of the mobile forensics field.

5.2 Evidence Collection

Extracting evidence from a mobile device can be challenging as there are several types of data to look for. Before the release of smart phones, law enforcement would typically extract phone call logs, SMS/MMS texts, e-mail and contact information. With the advancement of mobile device technology and the release of smart phones, the number of types of data has significantly increased with the use of social networking sites and applications that may contain other valuable information. The challenge is to know which types of evidence a mobile device offers before attempting to analyze it [46].

5.2.1 Internal Memory

Recently released smart phones use NAND Flash as their primary storage. Not all mobile devices use Flash storage the same way and can implement different file systems to interact with the physical storage. The internal memory can be examined through the use of Flash forensics [49].

5.2.2 External Memory

The internal memory of mobile devices is fixed in size. Recent devices allow the user to purchase external memory in the form of a standard, micro or mini SD card [29] to increase storage size. Devices can be configured to save application data to external memory. This creates another critical area for evidence collection.

5.2.3 Operator Data

Mobile devices, smart phones in particular, are always connected. They are in constant communication with a GSM operator's cell towers. The operator, if court ordered, can provide the approximate location of the device at the time a certain call was placed or recover text messages through their internal logs and databases. Gathering information from GSM operators is challenging and requires a search warrant along with precise

details of what calls and text messages need to be recovered. However, the information gained from an operator can be of significant value and could lead to an indictment of the suspect. The level of detail of the retained data by operators can vary from country to country as each has its own law on the log information that can and should be retained [80].

5.3 Forensic Examination Process

5.3.1 Preservation

Once each type and area of evidence has been identified, it is necessary for to go through a data extraction process. The data residing on the device should be treated with great care and never modified. Data tampering could result in the criminal case being dismissed since the defendant could argue that falsified data was uploaded to the device in hopes of incriminating the suspect. The following steps will ensure preservation of data and tamper-proof analysis of data that can be relied upon in court [58]:

1) Isolating the device from the network by turning it off. However, this could erase volatile memory and it is important to keep the device in the state it was when seized. To keep the device intact, it can be placed (while still turned on) into a shielded container to prevent outside network access.
2) Using software that is designed for forensic analysis. Most tools acquire data via requests to the OS, making 2-way data transfer inevitable. Forensic tools are specially designed to keep extracted data intact during extraction.
3) Ensuring a secure and reliable connection to the device. Connecting via Bluetooth or WiFi could potentially write data to the device, therefore a cable is the preferred choice in this step as it has the least impact.
4) Planning the examination process to avoid destruction of data that may be pertinent to a case. An examiner should prioritize the areas of the device to be analyzed. For example the removal of the SIM card requires the removal of

the device battery, which may lead to the loss of volatile data. It is essential that the examiner knows beforehand what data is needed before proceeding with the examination itself.

5) Exercising care when dealing with PIN/Pass codes to prevent permanent damage to the device. Entering the wrong PIN/Pass code a certain number of times may result in the device locking up permanently and the data becoming irretrievable.

A seized device should be kept secure and data integrity maintained to prevent modification. If data is significantly modified during the examination, the evidence may be ruled inadmissible by the court due to tampering. Forensic analysis of a mobile device is very challenging and each step should be documented, recorded and carefully logged. Any modification to the data, whether on purpose or inadvertently, should also be logged.

There are four principles of computer-based electronic evidence based on the Association of Chief Police Officers (ACPO) guideline [53]:

1) No action taken by law enforcement agencies or their agents should change data held on a computer or storage media, which may subsequently be relied upon in court.
2) In circumstances where a person finds it necessary to access original data held on a computer or on storage media, that person must be competent to do so and be able to give evidence explaining the relevance and the implications of their actions.
3) An audit trail or other record of all processes applied to computer-based electronic evidence should be created and preserved. An independent third party should be able to examine those processes and achieve the same result.

4) The person in charge of the investigation (the case officer) has overall responsibility for ensuring that the law and these principles are adhered to.

5.3.2 Data Integrity

Ensuring data integrity is a vital part of a mobile forensics examination. Rigorous processes and guidelines have been developed to prevent modifications from occurring during the device preparation and data extraction phases. Several suspects have been convicted with the help of evidence gleaned from their mobile devices; an investigation of a murder suspect's cellphone, by analyzing cell tower transmissions, showed that the suspect had called the victim several times with each call being transmitted closer to the victim's area of residence [77].

Data extracted from a mobile device can provide crucial information about the owner and his whereabouts. However, in order for a court to accept it as hard evidence, there must be proof that the data has not been tampered with. But many of the tools that investigators use to extract evidence are not designed to be forensically sound; put simply, they don't always have built-in features to prevent evidence tampering. A commercial product named Oxygen Mobile Phone Manager is a phone-syncing tool that was used for at least two years by law enforcement to gather evidence. However, it wasn't until April that the company released a tamper-resistant "forensic" version of the software that saves a cryptographic hash of the data it acquires from a mobile device, allowing investigators to later verify that nothing has changed [72].

It is essential for the investigator to pre-examine the tools that will be used to extract data in order for the forensic examination to result in tamper-proof and usable evidence in court. While some tools provide forensically sound data extraction methods, others might change, delete, or write new data to the device, rendering the evidence useless.

5.4 Data Acquisition

Once the device is contained and data integrity has been ensured it can be prepared for the data acquisition phase of the examination. Data acquisition refers to the process of imaging or otherwise obtaining information from the mobile device. Acquisition techniques can be divided into two techniques: logical and physical. A logical acquisition aims at recovering the allocated data from the device, mainly by just accessing the file system. Allocated data is typically associated with data that is not deleted and can be accessed from the file system; this technique does not acquire deleted data from the device with the exception of database files that may contain deleted records. A physical acquisition, on the other hand, targets the physical storage of the device bypassing the file system [61, 62]. Smart phones use NAND Flash as their storage, which retains deleted data even if it's no longer referenced by the file system. A physical acquisition technique has the advantage of extracting a significantly increased amount of data from the device including deleted data. Targeting the physical storage targets allocated and unallocated space and the probability of recovering deleted data is greatly increased. However, performing and analyzing a physical acquisition is far more challenging and time consuming.

It is important to note that for these techniques to work, the forensic analyst should be able to bypass the screen-lock protection, if activated, of the device. However, for critical cases, the device can be sent back to its manufacturer with an accompanying court order requesting that the device be unlocked and encryption be removed [63]. In the case of the Apple iPhone, we can assume that the manufacturer has a method of retrieving the hardware encryption keys and unlocking the data contents.

5.4.1 Logical Acquisition

A logical acquisition is a technique to extract allocated data. Typically, the data is acquired from the logical storage objects (e.g., file system, directories, files). This technique typically creates a replica of the file system structure found on the device that

can be browsed in an offline environment without modifying live data on the device, conserving its integrity. Below we describe the techniques that can be used to perform a logical acquisition. In each case of a logical acquisition, it is assumed that the forensic analyst is able to bypass the screen-lock protection.

5.4.1.1 Device Specific Interface

All brands of smart phones provide users with software that interfaces with the device and allows operations such as synchronization, data backup and firmware updates. In some cases, the operating system of the device may provide several options for interfacing (e.g., Android can be interfaced through ADB, Google play market, Google Cloud and other custom applications designed for logical communication) and in other cases there is a single point of communication (e.g. the iPhone uses iTunes or iCloud for all data transfer operations). In both cases, the software allows for a logical view of the file system and certain contents of the device. In the case of an Android device, for example, the ADB pull command can be used to download the entire data partition (adb pull /data). The success of these methods would depend on the devices' status following seizure. For an Android device, using ADB requires that USB debugging be enabled which is disabled by default. Most common users do not enable USB Debugging as it is a reserved option for developers of applications and other functionalities for the device. For the iPhone, if the device is locked then iTunes will not even recognize the device, preventing further logical investigation of the device. Generally, criminals are wary enough to password protect their devices decreasing the probability of simply connecting the device and examining it and leading the investigator to use more advanced methods.

5.4.1.2 Backup Files

Backup data usually contains a wealth of information significant in value for a forensic examination. Both Android and iPhone smart phones have the ability to backup user data. On the Android, users can use Google's services or custom applications to create a

backup of their user data. The backup data can be stored on external memory, such as an SD Card, and re-synced when needed [7].

On the iPhone, users can backup data to external storage with iTunes or to a cloud-based storage with iCloud [74], which allows backup & restoring iPhone contents over Wi-Fi/3G to or from a cloud with a registered Apple account. iCloud backs up photos, application data, device settings, messages and mail, etc… iCloud services were introduced to provide a computer-free backup solution.

iTunes is used to back up the iPhone to an external storage. When the iPhone is connected to a computer for the first time and synced with iTunes, iTunes automatically creates a folder with the device UDID (Unique device ID – 40 hexadecimal characters long) as the name and copies the device contents to the newly created folder. The iPhone can be synced with iTunes over Wi-Fi or over a USB connection [42].

Since the iPhone does not allow for storage expansion via an SD card or some other form of external storage, the backup files are usually stored on the user's computer or on iCloud.

If the forensic examiner knows the Apple ID and password of the suspect, then he can pull the backup from the cloud; if not, then the suspect's computer would need to be seized as well to analyze the iTunes backup. Another option would be to ask Apple for the suspect's authentication information, provided that a court order can be produced. Although for these methods to result in success, the seizure must include the suspect's personal computer along with his/her mobile device in order to analyze the backup. Getting an answer to a request for the user's personal account information from the manufacturer (e.g., Apple) can take a substantial amount of time as they are inundated with requests from law enforcement from all over the country [51].

5.4.1.3 AFLogical

This tool is only available for Android devices and has been developed by viaForensics [81], a mobile security company specializing in forensic solutions for smart phones. It is an application that is freely distributed to government and law enforcement agencies. By default, the Android security model is designed so that each application operates within its own sandbox and does not share data with other applications. However, application developers can use Content Providers as a means to share data with other applications. A few examples of Content Providers are SMS/MMS, Contacts, Calendar, and E-mail. The AFLogical tool leverages this architecture to gain access to user data being shared between applications. Once the application is downloaded, it can be installed via ADB to the device and executed. Upon execution, the application will automatically start extracting data and store it in the device's external SD Card.

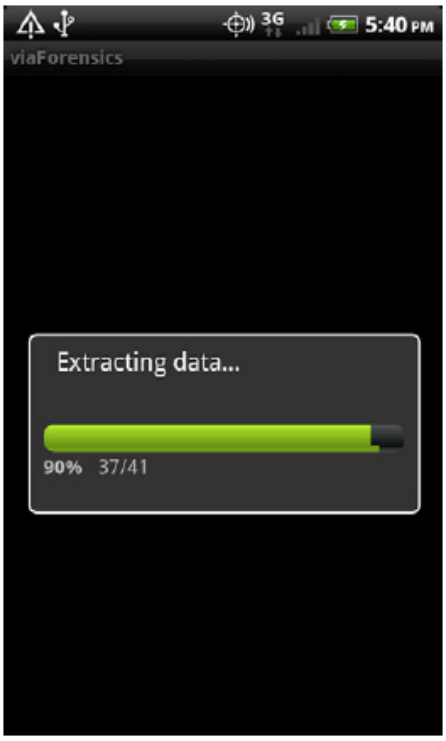

Figure 21. AFLogical extraction

The forensic examiner can then dump the contents of the SD card and analyze them for further investigation. This method is reported to have a 100% success rate although it is

unsure whether future versions of the Android OS will be compatible with this application.

5.4.1.4 Zdziarski Method

Bypassing the iPhone screen-lock is more challenging than bypassing it on the Android and due to the hardware-encrypted nature of the files residing on the device, no single application exists that can simply perform a logical acquisition. However, there are certain steps that can be followed to obtain a logical acquisition of the iPhone. Jonathan Zdziarski, a prominent researcher in the iPhone forensics field, has developed a method that does just this [68]. This tool is free for law enforcement agents and provides a series of scripts for the agent to execute in order. This technique leaves no trail on the device and operates only in the device's RAM extracting data over USB onto a computer. This method only works on iPhone models up to iPhone 4 and running at most iOS version 5.x. Our research has determined that this method fails on any iPhone model greater than the iPhone 4 and on iOS versions 6.x and up.

5.4.2 Physical Acquisition

A physical acquisition is a complete imaging of a mobile device; it involves the bit-by-bit copy of the entire physical storage and has the advantage of resulting in the recovery of deleted files. During a forensics process, an examiner would first attempt a logical acquisition and usually follow with a physical attempt if the information extracted was deemed insufficient for a case to be upheld in court [57].

5.4.2.1 Commercial Tools

Commercial mobile forensics tools are software and hardware that are specially designed to perform logical and/or physical acquisitions on a mobile device. These companies work solely with government or law enforcement agencies in hopes of aiding them by simplifying the forensic extraction of device data.

Some of these tools are self-contained "all-in-one" tool-kits that come pre-packaged with several cables to allow for multiple mobile device models. Other tools are software-based and closed source using undisclosed techniques that allow for physical acquisition on multiple mobile devices [64].

Name	Platform	License	Description
BlackLight	Windows/OSX	Proprietary	iOS Forensics
Cellebrite Mobile Forensics	Windows	Proprietary	Universal Forensics Extraction Device (UFED) – Hardware and Software
Radio Tactics Aceso	Windows	Proprietary	Multiple mobile device forensics
Paraben Device Seizure	Windows	Proprietary	Multiple mobile device forensics – Hardware and Software
SAFT Mobile Forensics	Windows	Proprietary	Android forensics
Oxygen Forensics Suite	Windows	Proprietary	Multiple device forensics – hardware and software
MicroSystemation XRT/XACT	Windows	Proprietary	Specialized in deleted data recovery – hardware and software
Elcomsoft iOS Forensic Toolkit	Windows/OSX	Proprietary	Acquires bit-by-bit copy of iOS data partition
Elcomsoft Phone Password Breaker	Windows/OSX	Proprietary	Brute forces encrypted backups of iPhone and Blackberry devices
MOBILedit! Forensic	Windows	Proprietary	Multiple devices – Software and Hardware
viaForensics viaExtract	Any	Proprietary	Software that runs on VM specialized in Android forensics
iXam	Windows	Proprietary	iOS forensics

Table 2. List of commercial products for mobile forensics

As it can be seen, the list of commercial products specializing in mobile forensics is extensive and each tool focuses on a certain area (i.e., iOS forensics vs. Android forensics). Out of all the tools listed, our research has indicated that Cellebrite is the most popular product among forensics experts with its signature product the Cellebrite Universal Forensic Extraction Device (UFED) Touch Ultimate [13]: this is an all-in-one

mobile forensics tool-kit, only available to law enforcement at a price of $10,000 and up, and designed for optimized and automated data extraction from a smart phone (Figure 22).

Figure 22. Cellebrite UFED Touch Ultimate [13]

It supports approximately 5000 devices along with the most advanced extraction process for Apple iPhones that no other tool has been able to replicate. The Cellebrite UFED toolkit has been used to aid several criminal cases, from theft to homicide, by recovering deleted data from a suspect's smart phone [12].

Cellebrite is among the mobile forensics vendors best known for their support of physical and file system extractions for major smartphone platforms including Apple iOS, BlackBerry, Android, Symbian, and Nokia BB5 [43].

Cellebrite claims to have been the first in the mobile forensics industry to have achieved a number of smartphone forensic breakthroughs. These include physical extraction and decoding of BlackBerry flash memory (going beyond decoding backup files), Android

user/pattern lock bypass for physical extraction and decoding, physical extraction from phones with Chinese chipsets, and other research and development.

Unlike its open-source counterpart, the Cellebrite UFED product comes equipped with an abundance of different cables and connections for every supported device and an easy-to-use graphical interface allowing investigators to micromanage the forensic process. However, the Cellebrite UFED does not guarantee successful results for all supported devices. Although the product has a wide support for iPhone model smart phones, newer iPhone models, such as the iPhone 5S and 5C, are only supported at the logical level: a file system extraction and analysis of unencrypted data such as certain sqlite databases. The company provides a detailed list of devices that are fully and partially supported. A fully supported device's deleted data will be 100% recovered unless it has been tampered with beforehand while a partially supported device's success rate may depend on how much data can be logically extracted.

5.4.2.2 Chip-off and JTAG

Although commercial tools and software are providing extensive support to a great number of devices through innovative data extraction techniques and methods, there will always be new devices that are incompatible with these tools and challenge forensic examiners. The ultimate goal of a forensic examination is to create an image of the internal memory and at one point the examiner will have to directly access the memory chip itself instead of using the file system to extract data.

Additionally, for devices that are damaged or locked with an encryption scheme that is beyond a tool's ability to crack or bypass, the chip-off and JTAG methods prove to be among the alternative solutions for examiners wishing to gain access to the memory. The JTAG (Joint Test Action Group) forensics is an advanced level data acquisition technique whereby one connects to the test access ports (TAP) of a mobile device and instructs the processor to transmit raw data stored on the connected memory chips [70].

The basic steps of performing a JTAG connection and acquisition are the following [50] (example figures are shown for a Samsung Galaxy S4 and taken from [26] with permission):

1) Remove the back panel of the mobile device

2) Identify TAPs by researching documented devices. When TAPs are unknown, inspect the device for potential TAPs.

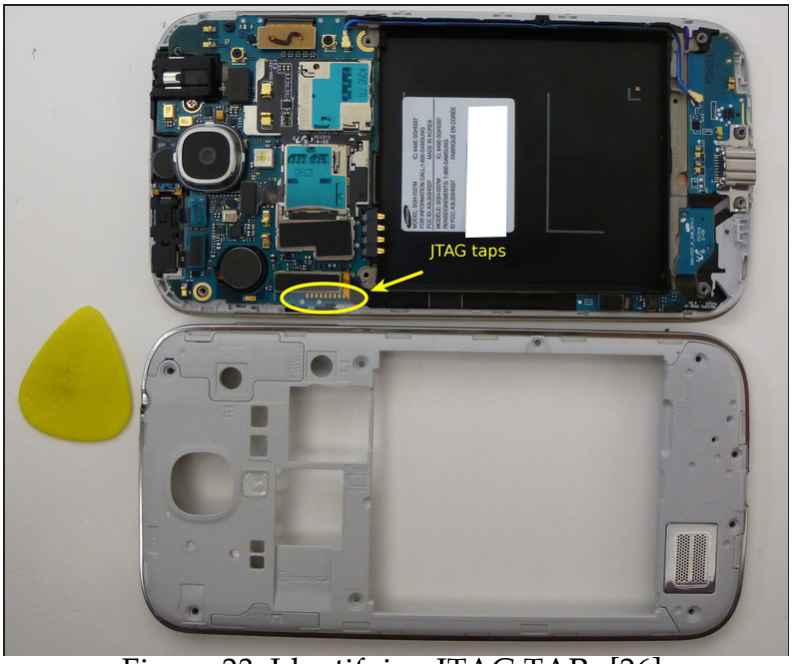

Figure 23. Identifying JTAG TAPs [26]

3) Solder wire leads to the correct connector pins.

4) Connect wire leads to an appropriate JTAG emulator with support for the target device.

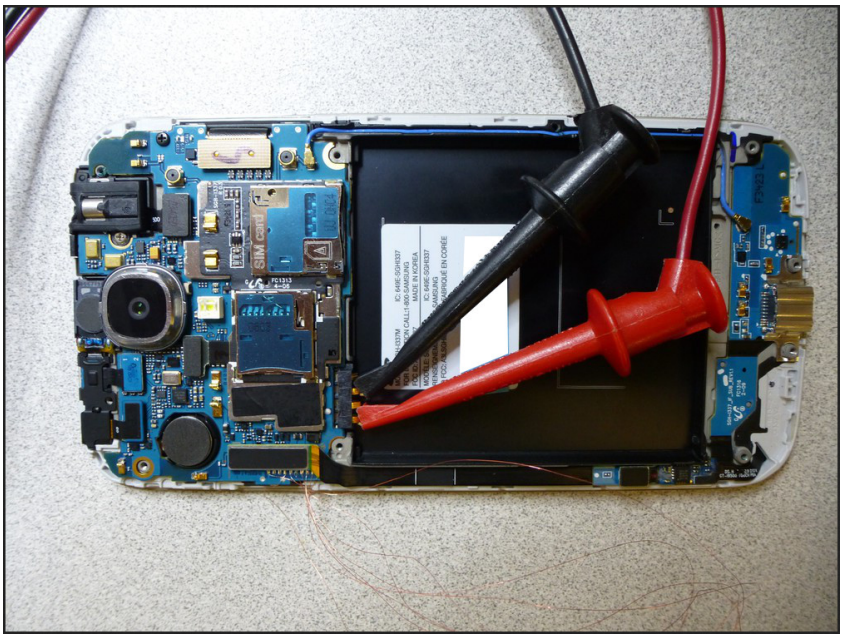

Figure 24. Soldering wires and connecting to JTAG emulator [26]

5) Read the flash memory after selecting the appropriate device.

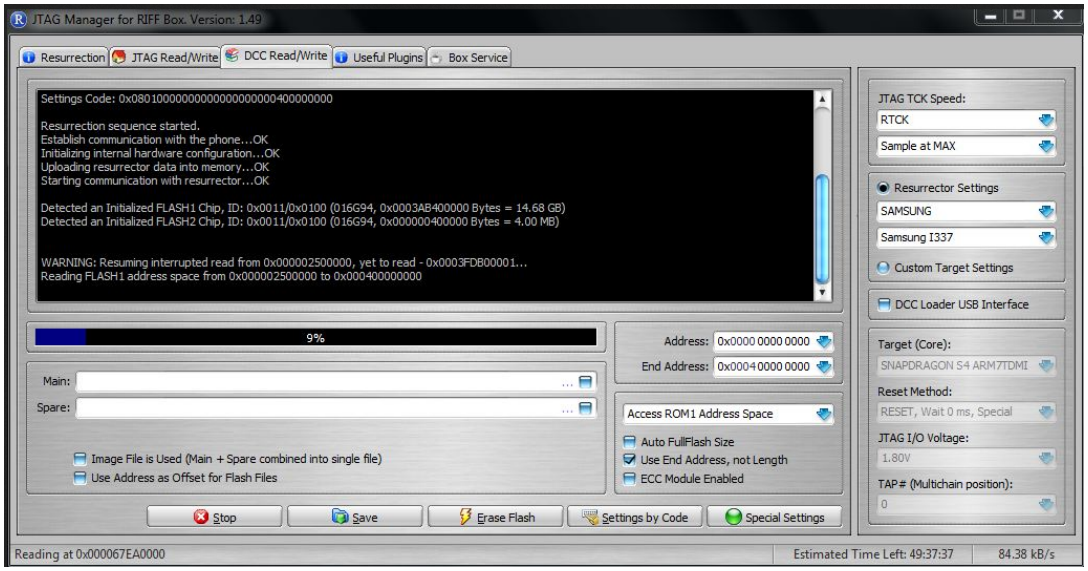

Figure 25. Reading Flash memory using RIFF BOX JTAG software [26]

The chip-off technique, on the other hand, is a much more destructive procedure and is generally used as a last resort technique when all else fails. It consists of removing the NAND Flash memory chip completely and reading it [48]. Following a chip-off method, the device cannot be returned to its original state or examined by a commercial tool. If a device that is being examined has been damaged beyond repair (e.g., the suspect

attempted to destroy the device physically prior to device seizure), then the chip-off technique is the only solution left to extract data.

The chip-off technique involves desoldering the Flash memory chip from the mobile device circuit board and requires a set of specific tools in addition to extreme precision. The risk of damaging the chip is very high if proper care is not taken during the desoldering process. Following removal, the chip then has to be read with specialized chip-reading programs and adapters. The chip removal process is generally done by applying heat, at very high temperatures, on the heat shield protecting the NAND chip, then the adhesive surrounding the chip is removed.

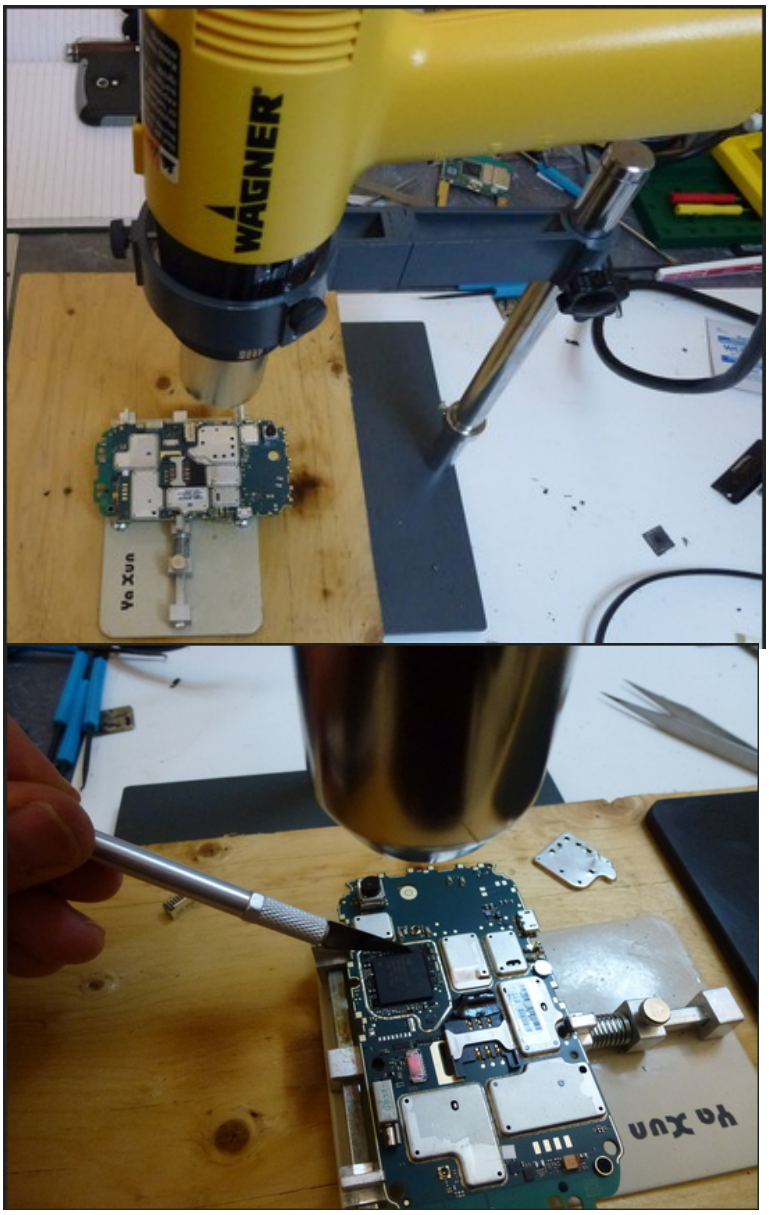

Figure 26. Chip heating and removal [69]

Finally, the chip is removed, cleaned and is prepared to be read [73].

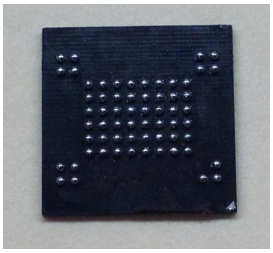

Figure 27. Desoldered NAND Chip ready to be read [69]

Both techniques accomplish the same result, but one is more destructive than the other.

The chip-off technique should only be considered if the JTAG TAPs cannot be located or

the device is damaged beyond repair. These methods access the physical storage directly and have a greater chance at recovering deleted data as it also reads sectors for which the file system does not have access. In both cases, the examiner performing these techniques should possess the skill and expertise necessary to perform them as the device can be damaged and forensic evidence may be destroyed.

5.5 Conclusion

In conclusion, this chapter has examined the various methods that are available for gathering evidence from a mobile device. Mobile forensics methods can be used beneficially to help law enforcement strengthen criminal cases with significant evidence extracted from the suspect's mobile device. In addition, several tools, only available for law enforcement agencies, simplify the forensics process by employing advanced and innovative data extraction methods unavailable in open-source tools. Furthermore, most of these tools and methods are specialized in recovering deleted data, which makes them an ideal choice for use in mobile forensics.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Returning to our initial question of whether remotely wiping a smart phone is an effective defense mechanism that prevents data recovery, we have provided ample evidence that it is not. The native design of mobile devices prevents secure deletion of data following a remote wipe, mainly due to the use of Flash memory for storage.

Remotely wiping or deleting data on mobile devices is portrayed as a secure solution for those that are stolen or lost. But, we have shown that this solution, on its own, is insufficient to keep someone from recovering deleted data. Our use of open-source tools allowed us to recover deleted data from an enterprise-enrolled device and the last chapter introduced several other commercial tools and advanced methods that law enforcement agencies can use to recover deleted data from devices owned by suspected criminals.

A remote wipe command attempts to ensure the security of a mobile device by seemingly deleting sensitive data. However, our methods showed that a remote wipe can be circumvented, or, in the case where the remote wipe is successful, deleted data can be recovered.

Real security demands much more than remote wipe in business settings. Even when it is certain that sensitive data has fallen into enemy hands -- as opposed to merely suspecting it -- it's really too late for any kind of wipe. Compromised data will remain such forever. Remote wipe is a useful solution when it complements more effective techniques such as mobile device encryption. Our experiments with the Apple iPhone have shown that a remote wipe securely deletes the encryption keys used to encrypt the file system on the device. Deleting the encryption keys renders the data extracted from the device unreadable unless one has a method of deriving the encryption key. It is

important to note that this method remains effective as long as the master keys are unrecoverable from the hardware. The mobile forensics field is in a constant state of advancement and we assume it is likely that techniques circumventing mobile encryption will be discovered in the near future. It is also worth noting that mobile encryption can fall prey to a poor or vulnerable implementation, shown by our experiments on the Android device, where the master key is retrievable even after a remote wipe has occurred.

In terms of securing data on a mobile device, a strong encryption key derived from a complex password, a viable implementation of the encryption scheme and a forensically-sound remote wipe function will make data recovery next to impossible. Furthermore, a secure wipe can be triggered by implementing the TRIM command (Chapter 3) after erasing the data. However, in our experiments, we were able to implement TRIM on an Android device by using a UNIX command-line utility named "fstrim" that would mark all deleted data as unused and ready to be erased by the garbage collector. The execution of this command required root privileges, therefore the MDM solution would need root privileges on the smart phone. Further methods would include the manufacturer leveraging Flash storage to securely erase encryption keys following a remote wipe function.

6.2 Future Work

BYOD, remote wipe and mobile forensics are all fairly new fields lacking in substantial academic research. One important area worth researching would be the implementation of a secure and forensically sound remote wipe function for use in the enterprise. A balanced blend of encryption and a secure deletion on Flash memory would lead to an ideal function for enterprises to execute an efficient remote wipe procedure. Furthermore, enterprises may use virtualization technologies to prevent local storage of

sensitive files altogether. Mobile virtualization is still an emerging area of research and would fit ideally in a BYOD enabled enterprise environment.

Our experiments were performed on only two devices which is a fairly small sample for a conclusive assertion of how remote wipe works in general and we were limited to using free MDM software to test our experiments. A much more exhaustive experiment featuring a greater sample size of devices and testing commercial products could lead to different conclusions and a better insight into the MDM and BYOD world of enterprises.

Finally, Flash memory is still not completely understood in the forensics field; the use of the TRIM command might be detrimental to a forensics investigation although conclusive research has not been done in this area. Further experimentation on Flash memory is required to gather conclusive evidence on how it fundamentally works and to ascertain its implications on forensics.

Bibliography

[1] "AFLogical." AFLogical. ViaForensics, n.d. Web. 14 Oct. 2013. <https://viaforensics.com/resources/tools/android-forensics-tool/>.

[2] "Android Debug Bridge." Android Developers. Web. 14 Oct. 2013. <http://developer.android.com/tools/help/adb.html>.

[3] "Android Distribution Numbers: Jelly Bean Rises to 28.4% of Active Devices." Android Distribution Numbers: Jelly Bean Rises to 28.4% of Active Devices. Phandroid, 01 Mar. 2013. Web. 03 Nov. 2013. <http://phandroid.com/2013/05/01/android-distribution-april/>.

[4] "Android Fragmentation Visualized (July 2013)." Android Fragmentation Report July 2013. OpenSignal, July 2013. Web. 30 Sept. 2013. <http://opensignal.com/reports/fragmentation-2013/>.

[5] "Android Security Overview." Android Developers. Android, Web. 30 Sept. 2013. <http://source.android.com/devices/tech/security/>.

[6] "Android, the World's Most Popular Mobile Platform." Android Developers. Android, Web. 30 Sept. 2013. <http://developer.android.com/about/index.html>.

[7] "App Backup & Restore - Android Apps on Google Play." App Backup & Restore - Android Apps on Google Play. Infolife LLC, Web. 14 Oct. 2013. <https://play.google.com/store/apps/details?id=mobi.infolife.appbackup>.

[8] "Beautifully Designed, Uniquely Yours | Windows Phone." Beautifully Designed, Uniquely Yours | Windows Phone (United States). Microsoft, Web. 12 Nov. 2013. <http://www.windowsphone.com/en-us/phones>.

[9] "BlackBerry Enterprise Service 10." BlackBerry Enterprise Service 10 - BES 10 - Mobile Device Management. BlackBerry, Web. 30 Sept. 2013. <http://blackberry.com/business/software/bes-10.html?IID=bb:business:software:enterprise-server:enterprise-server-overview:jan2013:BES10promo>.

[10] "BlackBerry Push APIs: Accessing the Power of BlackBerry Push Technology." Push Data - Whitepaper. BlackBerry, Web. 30 Sept. 2013. <http://us.blackberry.com/developers/javaappdev/BlackBerry_Push_APIs_Whitepaper.pdf>.

[11] "Cell Phones, Smartphones & Mobile Phones from BlackBerry.com." Blackberry.com. Web. 14 Oct. 2013. <http://www.blackberry.com/>.

[12] "Cellebrite - UFED Testimonials." Cellebrite - UFED Testimonials. Web. 3 Nov. 2013. <http://www.cellebrite.com/forensic-resources/ufed-testimonials>.

[13] "Cellebrite - UFED Touch Ultimate." Cellebrite - UFED Touch Ultimate. Web. 03 Nov. 2013. <http://www.cellebrite.com/mobile-forensics/products/standalone/ufed-touch-ultimate>.

[14] "Cellebrite UFED - IPhone Forensics Whitepaper." ViaForensics. Web. 03 Nov. 2013. <http://viaforensics.com/resources/white-papers/iphone-forensics/cellebrite-ufed/>.

[15] "Challenges and Benefits of BYOD - Bring Your Own Device for Enterprise - Accenture." Accenture | Management Consulting, Technology and Outsourcing. Accenture, Web. 25 June 2013. <http://www.accenture.com/us-en/Pages/insight-making-bring-your-own-device-work-enterprise.aspx>.

[16] "Chip-Off Forensics." Forensics Wiki. Web. 12 Nov. 2013. <http://www.forensicswiki.org/wiki/Chip-Off_Forensics>.

[17] "Flash Memory: SSDs, UFS, E.MMC." JEDEC: Global Standards for the Microelectrics Industry. JEDEC, Web. 14 Oct. 2013. <http://www.jedec.org/category/technology-focus-area/flash-memory-ssds-ufs-emmc>.

[18] "Full & Selective Remote Wipe Mobile Devices." Full & Selective Remote Wipe Mobile Devices. Maas360, Web. 30 Sept. 2013. <http://www.maas360.com/products/product-tours/mdm-product-tour/selective-and-full-remote-wipe/>.

[19] "Good Technology." Good Technology. Web. 14 Oct. 2013. <http://www1.good.com/>.

[20] "Home." MobileIron. Web. 14 Oct. 2013. <http://www.mobileiron.com/>.

[21] "ICloud." ICloud. Apple, Web. 14 Oct. 2013. <https://www.icloud.com/>.

[22] "Internet Security Threat Report 2013." Internet Security Threat Report 2013. Symantec, Apr. 2013. Web. 3 Nov. 2013. <http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v18_2012_21291018.en-us.pdf>.

[23] "IOS Security." IOS Security. Apple, Oct. 2012. Web. 30 Sept. 2013. <http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf>.

[24] "IOS: Understanding Data Protection." IOS: Understanding Data Protection. Apple, 23 Sept. 2013. Web. 30 Sept. 2013. <http://support.apple.com/kb/HT4175>.

[25] "JFFS : The Journalling Flash File System." JFFS : The Journalling Flash File System. Web. 14 Oct. 2013. <http://sourceware.org/jffs2/jffs2-html/>.

[26] "JTAG Samsung Galaxy S4 (SGH-I337)." Forensics Wiki. Web. 12 Nov. 2013. <http://www.forensicswiki.org/wiki/JTAG_Samsung_Galaxy_S4_(SGH-I337)>.

[27] "Notes on the Implementation of Encryption in Android 3.0." Android Developers. Android, Web. 30 Sept. 2013. <https://source.android.com/devices/tech/encryption/android_crypto_implementation.html>.

[28] "Oclhashcat-plus Advanced Password Recovery." Hashcat. Web. 14 Oct. 2013. <https://hashcat.net/oclhashcat-plus/>.

[29] "SD Association." About SD Standards. Web. 14 Oct. 2013. <https://www.sdcard.org/home/>.

[30] "SLC vs. MLC NAND." SLC vs. MLC NAND. Cactus Technologies, 3 June 2008. Web. 3 Nov. 2013. <http://www.systronics.ch/files/Application_Note_SLC_vs_MLC.pdf>.

[31] "SSDs – Write Amplification, Trim and GC." SSDs – Write Amplification, Trim and GC. OCZ Technology, Web. 3 Nov. 2013. <www.ocztechnologyforum.com/forum/attachment.php?attachmentid=21721&ei=RsF2UrHLE8motAaRnYGACA>.

[32] "Standard BlackBerry Encryption." Standard BlackBerry Encryption. BlackBerry, Web. 30 Sept. 2013. <http://docs.blackberry.com/en/admin/deliverables/12873/Standard_BlackBerry_message_encryption_193608_11.jsp>.

[33] "TeamWin Projects - TeamWin Recovery Project (twrp) - | TeamWin." TeamWin Projects - TeamWin Recovery Project (twrp) - | TeamWin. Web. 14 Oct. 2013. <http://teamw.in/project/twrp>.

[34] "The Android Open Source Project." Google Git. Web. 14 Oct. 2013. <https://android.googlesource.com/platform/system/vold/+/b87937cdea689594a293979b30b13054e7455dee/cryptfs.c>.

[35] ”The Impact of Mobile Devices on Information Security: A Survey of IT Professionals”. Check Point Software Technologies Ltd., June 2013. Web. 3 Nov. 2013. <http://www.checkpoint.com/downloads/products/check-point-mobile-security-survey-report2013.pdf>.

[36] "The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools." The Sleuth Kit (TSK) & Autopsy: Open Source Digital Forensics Tools. Web. 14 Oct. 2013. <http://www.sleuthkit.org/>.

[37] "Welcome to Xda-developers, a Community Founded for Developers by Developers." Xda-developers. Web. 14 Oct. 2013. <http://forum.xda-developers.com/index.php>.

[38] "Windows Phone for Business | Security | Windows Phone (United States)." Windows Phone for Business | Security | Windows Phone (United States). Microsoft, n.d. Web. 30 Sept. 2013. <http://www.windowsphone.com/en-us/business/security>.

[39] "Yaffs2." Yaffs. Web. 14 Oct. 2013. <http://www.yaffs.net/>.

[40] Andrivet, Sebastian. "The Security of MDM Systems." Hack in Paris 2013. 20 June 2013. Web. 3 Nov. 2013. <https://www.hackinparis.com/sites/hackinparis.com/files/MDM-HIP_2013.pdf>.

[41] Aviv, Adam J.; Gibson, Katherine; Mossop, Evan; Blaze, Matt; Smith, Jonathan M. "Smudge Attacks on Smartphone Touch Screens". 4th USENIX Workshop on Offensive Technologies.

[42] B., Satish. "IPhone Forensics – Analysis of IOS 5 Backups : Part 1." IPhone Forensics – Analysis of IOS 5 Backups. Infosec Institute, 3 May 2012. Web. 14 Oct. 2013. <http://resources.infosecinstitute.com/ios-5-backups-part-1/>.

[43] Blanchou, Marc. "Auditing Enterprise Class Applications and Secure Containers on Android: The Limitations of Mobile Security in the Enterprise." Auditing Enterprise Class Applications and Secure Containers on Android: The Limitations of Mobile Security in the Enterprise. ISEC Partners, Dec. 2012. Web. 3 Nov. 2013. <https://www.isecpartners.com/media/17994/isec_mdm_android.pdf>.

[44] Brodie, Daniel. "Practical Attacks against Mobile Device Management (MDM)." BlackHat 2013. 12 Mar. 2013. Web. 3 Nov. 2013. <https://media.blackhat.com/eu-13/briefings/Brodie/bh-eu-13-lacoon-attacks-mdm-brodie-wp.pdf>.

[45] Carstens, Martin. "The Truth about BlackBerry's Encryption." The Truth about BlackBerry's Encryption. Memeburn, 09 Aug. 2011. Web. 30 Sept. 2013. <http://memeburn.com/2011/09/the-truth-about-blackberrys-encryption/>.

[46] Casey, Eoghan, and Benjamin Turnbull. "Digital evidence on mobile devices." Eoghan Casey, Digital Evidence and Computer Crime. Third Edition. Forensic Science, Computers, and the Internet, Academic Pres (2011).

[47] Chen, Brian X. "Hacker Says IPhone 3GS Encryption Is 'Useless' for Businesses." Wired.com. Conde Nast Digital, 23 July 2009. Web. 30 Sept. 2013. <http://www.wired.com/gadgetlab/2009/07/iphone-encryption/>.

[48] Elder, Bob. "Chip-Off and JTAG Analysis." Evidence Magazine, June 2012. Web. 14 Oct. 2013. <http://www.evidencemagazine.com/index.php?option=com_content&task=view&id=922>.

[49] Fiorillo, Salvatore. "Theory and practice of flash memory mobile forensics." Australian Digital Forensics Conference. 2009.

[50] Forensics Wiki. Web. 14 Oct. 2013. <http://www.forensicswiki.org/wiki/Main_Page>.

[51] Golson, Jordan. "Apple Has Backlog of Requests From Police to Unlock Seized IPhones." Apple Has Backlog of Requests From Police to Unlock Seized IPhones. MacRumors, 10 May 2013. Web. 3 Nov. 2013.

<http://www.macrumors.com/2013/05/10/apple-has-backlog-of-requests-from-police-to-unlock-seized-iphones/>.

[52] Graham, Robert. "Errata Security: Password Cracking, Mining, and GPUs." Errata Security: Password Cracking, Mining, and GPUs. 22 June 2011. Web. 03 Nov. 2013. <http://erratasec.blogspot.com/2011/06/password-cracking-mining-and-gpus.html>.

[53] Guide for Mobile Phone Seizure and Examination. APCO Good Practice Guide for Computer Based Electronic Evidence – Official Release Version 4.0, 45-51.

[54] Hartocollis, Anemona. "When the Trill Of a Cellphone Brings the Clang Of Prison Doors." The New York Times. The New York Times, 16 July 2007. Web. 03 Nov. 2013. <http://www.nytimes.com/2007/07/16/nyregion/16cell.html>.

[55] Henderson, Tom. "How Mobile Device Management Works." How Mobile Device Management Works. ITWorld, 9 May 2011. Web. 3 Nov. 2013. <http://www.itworld.com/mobile-wireless/163465/how-mobile-device-management-works?page=0,%200.>.

[56] Hines, Elise. "What Is BlackBerry Internet Service?" What Is BlackBerry Internet Service? - How BIS Sends Email to Your BlackBerry. Web. 30 Sept. 2013. <http://cellphones.about.com/od/frequentlyaskedquestions/a/what-is-blackberry-internet-service.htm>.

[57] Hoog, Andrew, and Katie Strzempka. iPhone and iOS Forensics: Investigation, Analysis and Mobile Security for Apple iPhone, iPad and iOS Devices. Elsevier, 2011.

[58] Jansen, Wayne, and Rick Ayers. "Guidelines on cell phone forensics." NIST Special Publication 800 (2007): 101.

[59] Janssen, Cory. "Mobile Phone Forensics." Techopedias. Web. 14 Oct. 2013. <http://www.techopedia.com/definition/2956/mobile-phone-forensics>.

[60] Jones, M. Tim. "Anatomy of Linux flash file systems." IBM - United States. Web. 30 June 2013. <http://www.ibm.com/developerworks/library/l-flash-filesystems/>.

[61] Kameka, Andrew. "As Android Rises, Windows Phone Surpasses BlackBerry Market Share." As Android Rises, Windows Phone Surpasses BlackBerry Market Share. 04 Sept. 2012. Web. 30 Sept. 2013. <http://www.mobileburn.com/20421/news/as-android-rises-windows-phone-surpasses-blackberry-market-share>.

[62] Kim, Keonwoo, Dowon Hong, and Jae-Cheol Ryou. "Forensic Data Acquisition from Cell Phones using JTAG Interface." Security and Management. 2008.

[63] Kyle Rankin. 2012. Hack and /: password cracking with GPUs, Part II: get cracking. Linux J. 2012, 214, pages.

[64] Lessard, Jeff, and Gary Kessler. "Android Forensics: Simplifying Cell Phone Examinations." (2010).

[65] M. Bauer, R. Alexis, G. Atwood, B. Baltar, A. Fazio, K. Frary, M. Hensel, M. Ishac, J. Javanifard, M. Landgraf, D. Leak, K. Loe, D. Mills, P. Ruby, R. Rozman, S. Sweha, S. Talreja and K. Wojciechowski, "A multilevel-cell 32 Mb flash memory," Solid-State Circuits Conference, 1995. Digest of Technical Papers. 42nd ISSCC, 1995 IEEE International (1995):132-133, 351.

[66] Makhani, Naveed. "Comparison of Native Mobile Device Management & Container Approaches." Comparison of Native Mobile Device Management & Container Approaches « BeYOnD Endpoints. 20 Feb. 2013. Web. 30 Sept. 2013. <http://www.beyondendpoints.com/comparison-of-native-mobile-device-management-container-approaches/>.

[67] Matthias, Jess. "Logicalis commissions white paper study into BYOD | Logicalis." Global IT Partner | Global IT Services, Solutions from Logicalis Group. Web. 25 June 2013. <http://www.logicalis.com/news-and-events/news/logicalis-white-paper-byod.aspx#.Ucmbmpwzc4m>.

[68] McCullagh, Declan. "How Apple and Google Help Police Bypass IPhone, Android Lock Screens." CNET News. CBS Interactive, 02 Apr. 2012. Web. 14 Oct. 2013. <http://news.cnet.com/8301-31921_3-57408370-281/how-apple-and-google-help-police-bypass-iphone-android-lock-screens/>.

[69] Morrissey, Sean. iOS Forensic Analysis: for iPhone, iPad, and iPod touch. Apress, 2010.

[70] Nasheri, Hedieh. Economic Espionage and Industrial Spying. Cambridge, UK: Cambridge UP, 2005. Print.

[71] Newitz, Annalee. "Courts Cast Wary Eye on Evidence Gleaned From Cell Phones." Wired.com. Conde Nast Digital, 05 July 2007. Web. 03 Nov. 2013. <http://www.wired.com/politics/law/news/2007/05/cellphone_forensics>.

[72] Roberts, David, Taeho Kgil, and Trevor Mudge. "Integrating NAND Flash Devices onto Servers." Communications of the ACM 52.4 (2009): 98-103.

[73] Rosner, Bob. "HR Should Get a Clue Corporate Spying Is Real." HR Should Get a Clue Corporate Spying Is Real. Workforce, 7 Apr. 2007. Web. 03 Nov. 2013. <http://www.workforce.com/articles/hr-should-get-a-clue-corporate-spying-is-real>.

[74] Schwartau, Winn. "BYOD - The Privacy and Compliance Risks from Bringing Your Own Mobile Device to Work." Hack in Paris 2013. 20 June 2013. Web. 03 Nov. 2013. <https://www.hackinparis.com/talk-winn-schwartau>.

[75] Smith, Kent. "Benchmarking SSDs: The Devil Is in the Preconditioning Details." Benchmarking SSDs: The Devil Is in the Preconditioning Details. 17 Aug. 2009. Web. 3 Nov. 2013.
<http://www.flashmemorysummit.com/English/Collaterals/Proceedings/2009/20090811_F2A_Smith.pdf>.

[76] Someren, Laurie Van. "IDEAS List." IDEAS List. Yaffs, 05 Mar. 2008. Web. 03 Nov. 2013. <http://www.yaffs.net/ideas-list>.

[77] Thatcher, Jonathan. "NAND Flash Solid State Storage Performance and Capability -- an In-depth Look." NAND Flash Solid State Storage Performance and Capability -- an In-depth Look. SNIA, 2009. Web. 3 Nov. 2013. <https://www.snia.org/sites/default/education/tutorials/2009/spring/solid/JonathanThatcher_NandFlash_SSS_PerformanceV10-nc.pdf>.

[78] Whittaker, Zack. "Android Accounts for 75 Percent Market Share; Windows Phone Leapfrogs BlackBerry." Android Accounts for 75 Percent Market Share; Windows Phone Leapfrogs BlackBerry. ZDNet, 16 May 2013. Web. 03 Nov. 2013. <http://www.zdnet.com/android-accounts-for-75-percent-market-share-windows-phone-leapfrogs-blackberry-7000015496/>.

[79] Willassen, Svein. "Forensics and the GSM mobile telephone system." International Journal of Digital Evidence 2.1 (2003): 1-17.

[80] Zdziarski, Jonathan. "IPhone Forensic Method FAQ." IPhone Forensic Method FAQ. 17 Sept. 2009. Web. 14 Oct. 2013. <http://www.zdziarski.com/blog/?p=524>.

[81] McNamee, Kevin. "Wireless Week." Wireless Week. Kindsight Security Labs, 01 Mar. 2012. Web. 17 Dec. 2013. <http://www.wirelessweek.com/articles/2012/01/state-mobile-malware>.